programmez!
#198 - Juillet-Août 2016
le magazine des développeurs

Spécial été 2016

100 PAGES

robot | domotique | voiture | jeux | Raspberry

- **Réalité virtuelle avec CardBoard et HoloLens**
- **Coder son jeu vidéo**
- **Arduboy : Arduino se transforme en console**
- **Découvrir les API Haven OnDemand**
- **Créer son langage de programmation**
- **Résumé de Devoxx France 2016**
- **La culture geek au musée**

Hacker une Tesla

© Orion

M 04319 - 198 - F: 6,50 € - RD

Printed in EU - Imprimé en UE - BELGIQUE 7 € - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 € - TOM 1020 XPF - MAROC 55 DH

PROLONGATION JUSQU'AU 18 JUILLET 2016

COMMANDEZ WINDEV MOBILE 21

OU WEBDEV 21 OU WINDEV 21

ET RECEVEZ LE NOUVEL iPhone 6S

Ou choisissez parmi:

- 1x **MacBook Air** 11,6" 128GB
- 1x **iPad Pro** 9,7"WiFi 128GB
- 1x **iPad Pro** 12,9" WiFi 32GB
- 2x **iPhone SE** 16GB
- 2x **iPad Mini 4** Wi-Fi 64GB
- 2x **iPad Air 2** Wi-Fi 16GB

D'autres matériels de marque **SAMSUNG** et **ASUS** sont proposés sur le site www.pcsoft.fr
Aucun abonnement à souscrire.

iPhone 6S **128GB**.
Choix de la couleur sur le site

ou

iPhone 6S Plus **64GB**.
Choix de la couleur sur le site

OPÉRATION POUR 1 EURO DE PLUS

Pour bénéficier de cette offre exceptionnelle, il suffit de commander WINDEV Mobile 21 (ou WINDEV 21, ou WEBDEV 21) chez PC SOFT au tarif catalogue avant le 18 Juillet 2016. Pour 1 Euro de plus, vous recevrez alors le ou les magnifiques matériels que vous aurez choisis. Offre réservée aux sociétés, administrations, mairies, GIE et professions libérales, en France métropolitaine. **L'offre s'applique sur le tarif catalogue uniquement.** Voir tous les détails et des vidéos sur : **www.pcsoft.fr** ou appelez-nous (04.67.032.032).
Le Logiciel et le matériel peuvent être acquis séparément. Tarif du Logiciel au prix catalogue de 1.650 Euros HT (1.973,40 TTC). Merci de vous connecter au site www.pcsoft.fr pour consulter la liste des prix des matériels et les dates de disponibilité. Tarifs modifiables sans préavis.

Elu
«Langage
le plus productif
du marché»

www.pcsoft.fr

Season finale / s18e198, « tes vacances d'été tu coderas !»

L'été est là... Bientôt les vacances... Seul(e), ou presque, au bureau... Mais attention, il ne faut pas perdre les bonnes habitudes. La rédaction de Programmez ! vous a concocté un cahier de vacances dopé aux codes, à la Raspberry Pi, aux robots et à la réalité virtuelle ! Et pour les retardataires, révisions accélérées de gaming en 2D et 3D !

Au programme de ce cahier du très lourd :

- Réalité virtuelle avec Google CardBoard et HoloLens ;
- Programmation de jeux vidéo sur rétroconsole, avec le super moteur BabylonJS (coucou copain David), et même un jeu entier codé sur Arduino (si, si, on peut le faire) ;
- Un peu de robotique ;
- Hacker sa belle Tesla ;
- Comment créer son propre langage de programmation.

Bien entendu, le code n'est pas oublié. Nous vous proposons un compte-rendu de la conférence Devoxx France 2016, du .Net Core, un aperçu de l'outil Consul, la suite d'Android N.
Ceci sans oublier, un focus très sympa sur les bots, la future vedette des apps et des entreprises.

Programmez ! vous souhaite un excellent été, et rendez-vous le 2 septembre avec le n°199...

ftonic@programmez.com

A propos de l'illustration de couverture

Fateh Beroual est un illustrateur, auteur de bandes dessinées connu sous le pseudonyme d'Orion. Né en 1978 et exerçant à Strasbourg, il crée en 2009 le blog BD journaldorion.net dédié à la thématique High-tech et geek.

À lire dans le prochain numéro n°199 / en kiosque le 2 septembre 2016

La santé et le développeur

Comment et pourquoi les développeurs vont révolutionner la santé ?

Java 9

Que faut-il attendre du prochain Java prévu pour mars 2017 ?

Dernière minute

La nuit du hack 2016 :

2 & 3 juillet/Disneyland Paris

Pour la 14ème année consécutive, la Nuit du Hack (NDH) rassemblera les experts internationaux de la sécurité informatique dans le but d'éduquer et d'informer sur l'importance du hacking. Une fois par an à Paris, l'équipe HackerzVoice accueille le grand public pendant 24 heures pour démystifier ces pratiques autour de conférences, d'ateliers et de challenges. L'événement débutera dès le 2 juillet à 10h pour une grande plénière puis des conférences techniques toute la journée. De nombreux ateliers auront lieu à partir de 20h.
Site : https://nuitduhack.com/fr/

septembre

Hardwear.io 2016 :

22 & 23 septembre/La Hague/Hollande

Pour la 2e année, la conférence sécurité matérielle revient. Elle se tiendra en Hollande fin septembre. Avec les multiplications des matériels et logiciels embarqués (voitures, IoT, avions, médecine, dans les maisons, etc.), on oublie parfois un peu vite la sécurité. Trois catégories de sessions seront disponibles : recherches actuelles, nouvelles recherches et les outils. L'objectif est de parler de sécurité offensive et défensive. Les domaines abordés seront très nombreux : processeurs, embarqué, IoT médical, firmware, test de pénétration hardware, etc.
Site officiel : http://hardwear.io

FrenchKit :

23 & 24 septembre/Paris

Une grande conférence développeur dédiée à iOS et macOS se tiendra en septembre à Paris. Xebia et CocoaHeads Paris organisent cet événement. Une quinzaine de sessions sont prévues la première journée et une seconde journée orientée communauté avec du live-coding, des ateliers, du hacking et des démos. + 200 personnes sont attendues.
Site officiel : http://frenchkit.fr

Meetup Big Data & Machine Learning :

22 septembre / Paris

Tout récemment créé, ce meetup parlera big data, machine learning, deep learning. On y abordera les bonnes pratiques, les API, les outils. La première soirée aura lieu de 22 septembre prochain.
Site : http://www.meetup.com/fr-FR/Big-Data-et-Machine-Learning

octobre

Microsoft experiences :

4 & 5 octobre/Paris

Les TechDays sont morts, vive Experiences. Microsoft a décidé de réinventer sa grande conférence annuelle, les MS TechDays qui étaient l'événement technique de l'année. Désormais, l'événement aura lieu en octobre et sur 2 jours uniquement, au lieu de 3 ! Cela signifie moins de sessions. Le salon se découpera en deux parties : journée business et journée technique. La journée technique s'articulera autour de 95 sessions... le choix va être difficile !
Et cette année, Scott Guthrie sera l'invité vedette.
Site officiel : https://experiences.microsoft.fr

D Day 2e édition :

7 octobre/Marseille

La seconde édition de la conférence D Day sera organisée début octobre à Marseille. Le thème central est le DevOps. Il s'agit de la plus grande conférence autour de ce thème dans le Sud de la France. Organisée par Treeptik. Site officiel : http://2016.devops-dday.com

dotGo :

10 octobre/Paris

Le langage Go sera largement à l'honneur à Paris durant la conférence dotGo. Les conférences dot rassemblent les meilleurs speakers du domaine. Cette année, Robert Griesemer, co-créateur du langage, sera là. Conférence uniquement en Anglais. Site officiel : http://www.dotgo.eu

Forum PHP :

27 & 28 octobre/Montrouge

C'est LE rendez-vous des communautés PHP. L'appel à contribution est ouvert jusqu'au 17 juillet. On va sans doute beaucoup parler PHP 7 et lever le voile sur le futur du langage. Site : http://afup.org

novembre

Paris Open World Summit 2016 :

16 & 17 novembre/Plaine Saint-Denis

La nouvelle édition de la grande conférence open source française se tiendra en novembre prochain. Le thème central sera l'innovation qui s'appuie de plus en plus sur l'open source, l'open data, l'open hardware, etc. A cette thématique principale, trois volets seront abordés : la technologie, l'entreprise et la société. Chacun de ces secteurs a des défis propres à relever.
Pour en savoir plus : http://www.opensourcesummit.paris

PROLONGATION JUSQU'AU 18 JUILLET 2016

COMMANDEZ WINDEV MOBILE 21

OU WEBDEV 21 OU WINDEV 21

ET RECEVEZ LE NOUVEAU

SAMSUNG Galaxy S7 edge

- Etanche IP68 (30mn à 1m)
- Ecran 2.560 x 1.440 pixels
- Emplacement carte Micro SD
- Android 6.0
- Batterie à chargement rapide, sans fil
- Capteur photo 12 M
- ...

SAMSUNG
Galaxy S7 edge
32 Go.
Choix de la couleur sur le site

ou

SAMSUNG
Galaxy S7
32+128 Go.
Choix de la couleur sur le site

Ou choisissez parmi:

- 1x **TV SAMSUNG 4K Ultra HD 138cm** WiFi
- 1x Tablette **Galaxy Tab Pro S** Windows 10 Home 12" Full HD+. WiFi. 4Go+SSD 128Go. Clavier.
- 2x **Galaxy Tab S2** 9,7"
- 2x Smartphone **Galaxy A5**

D'autres matériels de marque **APPLE** et **ASUS** sont proposés sur le site www.pcsoft.fr
Aucun abonnement à souscrire.

OPÉRATION POUR 1 EURO DE PLUS

Pour bénéficier de cette offre exceptionnelle, il suffit de commander WINDEV 21 (ou WINDEV Mobile 21, ou WEBDEV 21) chez PC SOFT au tarif catalogue avant le 18 Juillet 2016. Pour 1 Euro de plus, vous recevrez alors le ou les magnifiques matériels que vous aurez choisis. Offre réservée aux sociétés, administrations, mairies, GIE et professions libérales, en France métropolitaine. L'offre s'applique sur le tarif catalogue uniquement. Voir tous les détails et des vidéos sur : www.pcsoft.fr ou appelez-nous (04.67.032.032).
Le Logiciel et le matériel peuvent être acquis séparément. Tarif du Logiciel au prix catalogue de 1.650 Euros HT (1.973,40 TTC). Merci de vous connecter au site www.pcsoft.fr pour consulter la liste des prix des matériels et les dates de disponibilité. Tarifs modifiables sans préavis.

Elu
«Langage
le plus productif
du marché»

DAS :Galaxy S7 DAS pour la tête : 0,406 W/kg | DAS pour le corps : 0,621 W/kg
Galaxy S7 edge DAS pour la tête : 0,264 W/kg | DAS pour le corps : 0,507 W/kg
Le DAS (débit d'absorption spécifique des téléphones mobiles) quantifie le niveau d'exposition maximal de l'utilisateur aux ondes électromagnétiques, pour une utilisation à l'oreille. La réglementation française impose que le DAS ne dépasse pas 2 W/kg.

www.pcsoft.fr

Document non contractuel. Logiciel professionnel.

Genuino MKR1000 : une nouvelle Arduino vraiment intéressante

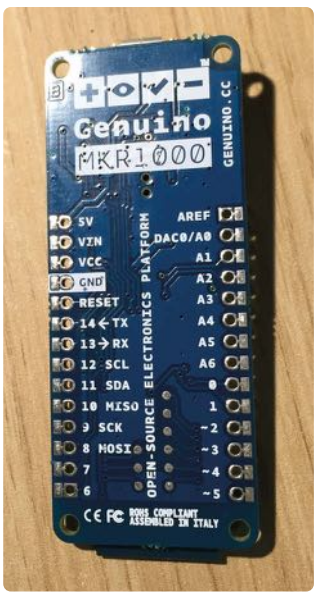
La nouvelle carte Arduino, oups, Genuino, est disponible depuis quelques semaines, la MKR1000. Elle est un peu plus grande qu'une Nano et plus petite qu'une Raspberry Zero. Encore difficile à trouver en France, elle possède des fonctions très intéressantes.

François Tonic
Programmez!

L'ambition de cette mini-carte est de pouvoir créer des objets connectés très facilement, tout en minimisant l'espace nécessaire pour la carte. La board présente une bonne qualité de fabrication. L'absence de hearders soudés facilite son logement dans un espace réduit. En utilisation soutenue et constante, elle chauffe un peu. Il faudra prévoir une aération.

Des fonctions inédites

La MKR1000 est basée sur un SOC Atmel ATSAMW25. Il contient un CPU ARM Cortex 32 bits basse consommation, un module WiFi (802.11 b/g/n) et un inédit module de crypto-authentification (EEC508). La connectique est réduite au minimum : port micro-usb et connecteur LiPo (très pratique !). Par contre, et c'est le principal défaut de cette Arduino, un fonctionnement en 3,3V uniquement. Nous disposons de 7 entrées analogiques, 1 sortie analogique, 8 broches digitales, 1 seule I2C. Le module EEC508 sert à crypter et sécuriser les communications : très pratique quand vous créez un IoT constamment connecté.



La communication avec Cloud Arduino est très simple :

- On déclare le réseau (SSID / mot de passe)
- Puis on identifie Arduino Cloud : userName, thingName, thingId, thingPsw

La programmation est vraiment très simple et rapide à faire : - 10 lignes.

Et comme toujours avec Arduino, la carte est open hardware, les schémas électroniques sont librement accessibles !

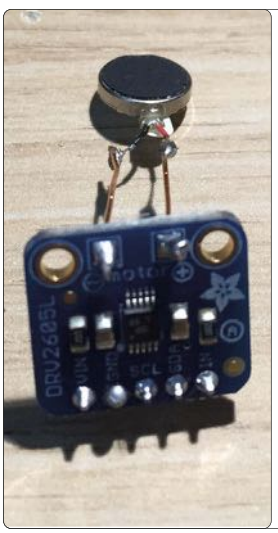
La carte officielle est un peu chère (+40 €), mais des clones devraient arriver dans les prochains mois.

La carte se développe comme les autres Arduino, avec Arduino IDE (dernière version) ou tout IDE compatible. Il vous faudra ajouter les pilotes Arduino SAMD Boards (32-bits ARM Cortex M0+).

Une opportunité pour les makers et développeurs

Pour utiliser au mieux les fonctionnalités de la carte, vous devez au préalable installer les différentes librairies compatibles MKR1000, en vrac : WiFi101, RTCZero, AudioFrequencyMeter, AudioZero, AzureIoTHub et ArduinoCloud.

La disponibilité du module WiFi étend les scénarii d'usages. Le service Cloud Arduino permet de connecter en quelques minutes la MKR1000 au Cloud. Il suffit de charger la librairie et coder les interactions Arduino -> Cloud. Très pratique pour afficher l'état ou des informations sur les capteurs : température, bouton, etc. Cloud Arduino est pour le moment en pré-version et son fonctionnement n'est pas garanti. Nous avons parfois beaucoup de latence réseau. Mais honnêtement, il devient réellement facile, avec un peu de code de créer des prototypages et des petits IoT connectés.



INSTALLER UN SYSTÈME HAPTIQUE

On parle beaucoup de système haptique depuis quelques mois. Il s'agit de petites vibrations pour notifier l'utilisateur d'un événement, d'une action. Les terminaux mobiles (téléphones, tablettes, montres, etc.) utilisent ces systèmes. De petits moteurs sont utilisés.

Pour vos projets, vous pouvez très simplement créer ce genre de système. Vous pouvez acheter de petits vibrateurs en forme de disque. Les plus courants fonctionnent en 3,3V. Vous pouvez l'utiliser directement ou passer par un contrôleur haptique de type Adafruit Haptic Moter Controller, franchement cher, 25 € (Adafruit + moteur). Vous pouvez utiliser un vibrateur + contrôleur intégré tel que le Grove Haptic Motor, environ 20 €. L'avantage d'utiliser des contrôleurs est de pouvoir varier les vibrations et d'utiliser finement le vibrateur.

Si vous cherchez uniquement le vibrateur, vous pouvez trouver des lots de 5 pièces à – 3 €.

LES +

- WiFi
- Module de cryptographie
- Cloud Arduino
- Dimension
- Connecteur LiPo
- Programmation Arduino classique

LES –

- Prix
- Difficile à trouver
- Voltage limité à 3,3V
- Format du micro-USB
- Dégagement de chaleur

DOMAINES | SITES WEB | SERVEURS CLOUD | HÉBERGEMENT

LA PLUS SÛRE DES SÉCURITÉS

TRANSACTIONS 100 % SÉCURISÉES
SSL INCLUS

HÉBERGEMENT SÉCURISÉ ? BIEN SÛR !

1&1 offre le plus haut niveau de protection actuellement disponible. Montrez à vos visiteurs que leur sécurité est votre priorité absolue :

- ✓ Certificat SSL inclus
- ✓ Data centers certifiés
- ✓ Géo-redondance
- ✓ Protection DDoS

CERTIFICAT SSL
SÉCURITÉ MAXIMALE
SEULEMENT CHEZ 1&1 !

1 MOIS POUR ESSAYER

1 CLIC POUR CHANGER DE PACK

1 APPEL UN EXPERT VOUS RÉPOND

0970 808 911
(appel non surtaxé)

1&1

1and1.fr

Conditions détaillées sur 1and1.fr. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

Présentation du Bot Framework

Le Bot Framework est l'une des nouveautés annoncées par Microsoft à la //Build 2016 qui a largement été mise en exergue lors des différentes sessions. L'objectif principal des Bots est clair : interagir de manière intelligente avec vos utilisateurs, en se connectant à eux via la messagerie, les SMS, Cortana, Slack, etc.

Thomas LEBRUN
tlebrun@infinitesquare.com
http://blogs.infinitesquare.com/b/tom

La création d'un Bot est très simple et se passe en plusieurs étapes. Il faut commencer par se connecter au portail pour créer le Bot: https://dev.botframework.com. Ensuite, après avoir ajouté le template de projet correspondant dans Visual Studio, vous pourrez réaliser la mise en place d'un BotConnector : Fig.1. Ensuite, il faut ajouter une référence Nuget au package "Microsoft.Bot.Builder" qui vous permettra d'accéder aux types nécessaires : Fig.2. En regardant le code généré, on se rend compte qu'un BotConnector n'est qu'une simple Web API avec une méthode Post qui sera appelée par le Bot (créé dans le portail) : Fig.3. L'implémentation technique des BotConnectors peut se faire de 2 manières différentes, en fonction de vos besoins :

- Soit votre Bot a besoin de préserver des données utilisateurs entre chaque échange, lui permettant d'avoir un "raisonnement" plus poussé : il faut alors que votre Bot s'appuie sur LUIS (Language Understanding Intelligent Service) afin de lui permettre d'interpréter le texte saisi par l'utilisateur, d'apprendre de lui, etc. C'est le domaine du Machine Learning / Deep Learning !
- Soit votre Bot se contente de poser des questions afin d'avoir un échange avec l'utilisateur.

Dans le premier cas, il est nécessaire de faire une classe (Serializable) qui expose des propriétés :

```
[Serializable]
public class WikipediaSearchRequest
{
  [Prompt("Que souhaitez-vous rechercher ?")]
  public string Query { get; set; }

  [Prompt("Dans quelle langue souhaitez-vous faire votre recherche ? {||}")]
  public Language Language { get; set; }
}
```

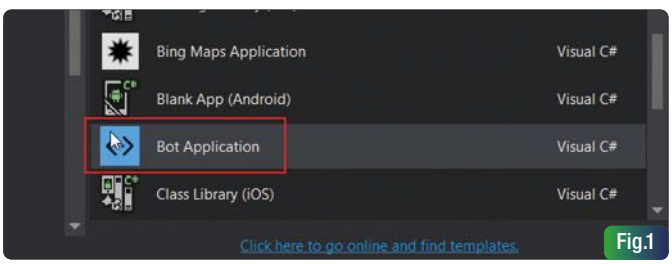


Fig.1

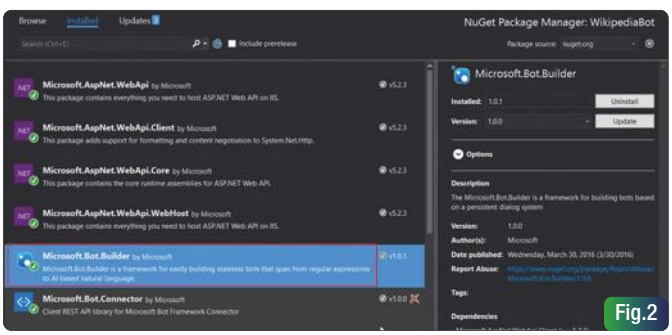


Fig.2

Comme vous pouvez le constater, l'attribut Prompt est utilisé pour indiquer au Bot ce qu'il doit afficher à l'utilisateur. A présent, il est nécessaire de "construire" le formulaire qui sera présenté à l'utilisateur, via une méthode dans cette même classe :

```
public static IForm<WikipediaSearchRequest> BuildForm()
{
  return new FormBuilder<WikipediaSearchRequest>()
      .Message("Bonjour, je suis WikipediaBot, l'agent de recherche Wikipedia.")
      .Build();
}
```

Et le tour est joué ! Pour tester votre Bot, il suffit de lancer le Debug, puis de lancer l'application "Bot Framework Emulator" en se connectant à l'adresse sur laquelle votre API est exposée.
Vous pouvez tester votre bot en saisissant du texte dans la console : Fig.4.
Vous avez également la possibilité "d'intercepter" les échanges réalisés entre votre Bot et l'utilisateur, grâce à la méthode OnCompletionAsync qui vous permettra de renvoyer du texte "personnalisé" à l'utilisateur :

```
public static IForm<WikipediaSearchRequest> BuildForm()
{
  return new FormBuilder<WikipediaSearchRequest>()
      .Message("Bonjour, je suis WikipediaBot, l'agent de recherche Wikipedia.")
      .OnCompletionAsync(async (context, request) =>
      {
        var httpClient = new HttpClient();
        var stringResults = await httpClient.GetStringAsync(string.Format("https://{0}.
wikipedia.org/w/api.php?action=query&list=search&srsearch={1}&utf8&format=json",
request.Language == Language.English ? "en" : "fr", HttpUtility.UrlEncode(request.Query)));

        var message = context.MakeMessage();
        message.Text = stringResults;
```


Fig.3

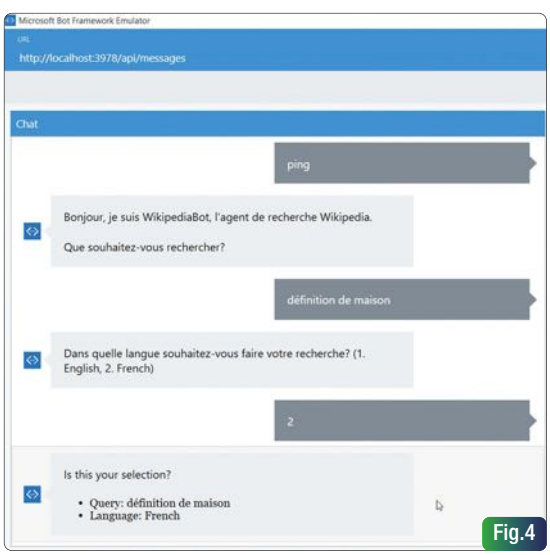


Fig.4

```
            await context.PostAsync(message);
        })
        .Build();
}
```

Astuce : *dans le message que vous renvoyez à l'utilisateur, ne renvoyez pas de JSON car c'est le format utilisé pour la communication entre votre Bot Connector et l'émulateur. Du coup, ce dernier essayera de désérialiser du JSON en JSON ce qui provoquera une erreur non catchée et donc un crash systématique de l'outil.*

Une fois la logique applicative de votre Bot terminée, il ne vous reste plus qu'à le publier à l'emplacement que vous avez indiqué lors de la création sur le portail : Fig.5. De base, le Bot est accessible en mode Web, pour une utilisation sur une page Web. Vous pouvez d'ailleurs le tester directement depuis le portail : Fig.6. Et vous pouvez récupérer le code HTML nécessaire pour l'inclure sur vos pages Web : Fig.7. Mais vous pouvez choisir de configurer votre Bot pour d'autres Channels tels qu'Email, Skype, Slack, etc. : Fig.8. Il est à noter que les configurations respectives des channels sont simples à mettre en œuvre et certaines, telle que celle pour les SMS, reposent sur la création de comptes sur des plateformes tierces. Comme vous pouvez le constater, la création d'un Bot "simple" est aisée grâce au Bot Framework, mais elle offre des scénarios intéressants dans le cadre d'outils d'aide à la vente, de SAV, etc. Voyons à présent comment faire pour rendre notre Bot plus intelligent, notamment grâce à l'utilisation de LUIS.

Actuellement en version Beta, ce service proposé par Microsoft vous permet de disposer d'un moteur capable d'analyser et d'interpréter le texte pour vous permettre de comprendre ce que souhaite votre utilisateur (et ainsi vous permettre de mieux répondre à sa demande).

Pour mieux comprendre, prenons un exemple simple : lorsque votre utilisateur tape "translate France in EN", il souhaite traduire le pays "France" en anglais. Cependant, il existe énormément de pays et de langues différentes, donc vous pouvez difficilement tout gérer à la main dans votre code. Et c'est là que LUIS entre en jeu en faisant en sorte d'être capable de déterminer que lorsque l'utilisateur saisit "translate XX in YY" (ou une phrase similaire qu'il aura appris à reconnaître), il veut traduire le nom d'un pays dans une langue, quel que soit le pays ou la langue.

C'est là toute la force de LUIS: vous permettre de déduire les demandes de l'utilisateur et de les comprendre, en fonction du contexte ! Et pour bien comprendre comment cela fonctionne, faisons un petit tour d'horizon de l'outil, en créant un modèle capable de rechercher dans les contacts Exchange! Une fois connecté au portail (https://www.luis.ai), vous pouvez choisir de créer ou éditer une application déjà existante : Fig.9.

LUIS s'appuie sur un ensemble d'éléments qui vous permettront de construire le moteur idéal :

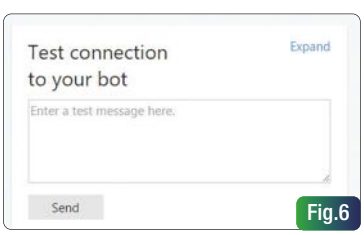


Fig.6

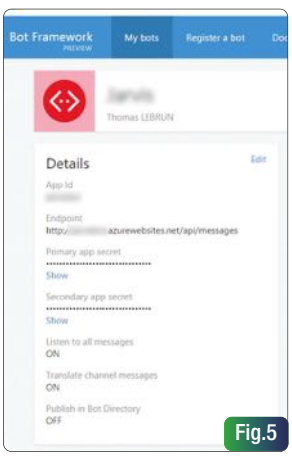


Fig.5

- Intent : il s'agit de "l'intention" que vous recherchez à réaliser (traduire du texte, liste des contacts, filtrer les contacts, etc.) ;
- Entities : il s'agit des entités qui seront reconnues/manipulées. Il peut s'agir d'entités personnalisées ou d'entités prédéfinies (nombre, températures, âge, monnaie, etc.) ;
- Utterances : ce sont les énonciations, autrement dit les patterns de texte, qui seront reconnues et qui vous permettront de déclencher vos intentions.

Pour construire une application LUIS, il est nécessaire d'ajouter des intentions permettant de dire : "Quand je demande XX, exécute l'action Y". Il faut également indiquer l'énonciation qui me permettra de déclencher cette intention, et c'est ce que je peux faire dans l'onglet "New utterances": Fig.10. Ici, LUIS m'indique qu'en saisissant le texte "display contacts", il a reconnu à 97% qu'il s'agissait de l'action "Get all contacts". Comme c'est bien le cas, je peux lui confirmer qu'il s'agit de la bonne intention en cliquant sur Submit. Une fois l'ensemble des énonciations ajoutées, il faut entraîner le modèle en cliquant sur le bouton "Train" situé en bas à gauche de l'écran. Cela permet de déclencher l'entraînement du modèle pour qu'il affiche le nombre d'énonciations associées à chaque intention, ainsi que le nombre d'énonciations correctement prédites. Dans le cas où notre intention nécessite un paramètre, ce cas est également prévu de la part de LUIS. Nous déclarons ce paramètre comme étant "Required", impliquant que l'action associée (l'appel de l'URL) ne se fera que si le paramètre est bien reconnu par LUIS. Dans le cas contraire, le message spécifié ("Please, enter the name of the contact to search") sera renvoyé à l'utilisateur, permettant d'avoir un dialogue avec l'utilisateur ! Etant donné que notre intention repose sur un paramètre, nous devons indiquer à notre modèle qu'il recevra ce paramètre et donc comment le recevoir. Pour cela, il faut l'entraîner en lui indiquant des énonciations possibles : Fig.11.

En saisissant "show simon", le modèle pense qu'il s'agit d'une demande

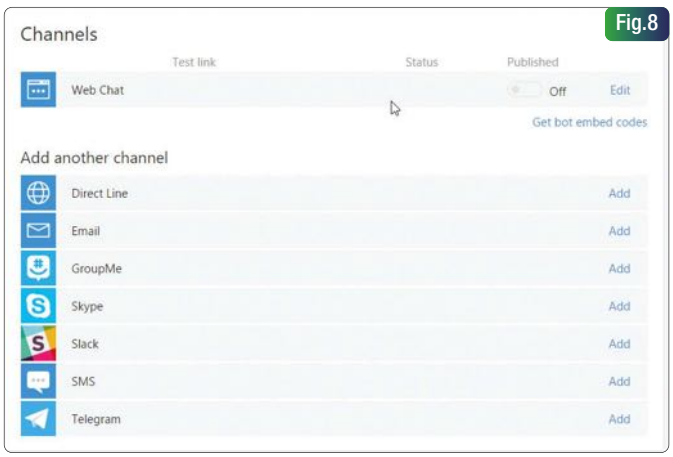


Fig.8

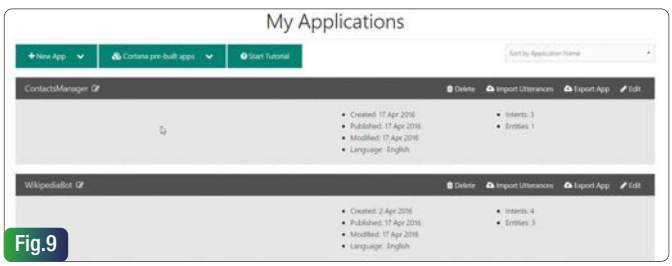


Fig.9

Fig.7

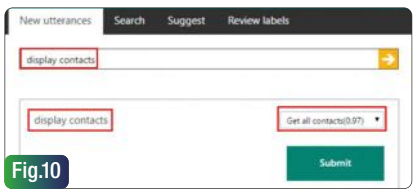


Fig.10

pour lister tous les contacts. Vu qu'il se trompe, je lui indique que "simon" est un nom (donc le paramètre de l'appel à mon Web Service) et qu'il s'agit bien de l'intention "Search contacts".

Mes entités sont définies, les énonciations sont mises en place, le modèle est entraîné et publié: il ne reste plus qu'à l'utiliser! Je peux appeler son URL de publication directement (https://api.projectoxford.ai/luis/v1/application/preview?id=XXXXX&subscription-key=XXXXXX), passer en paramètres le texte nécessaire et interpréter le JSON; ou bien, je peux m'appuyer sur le Bot Framework pour proposer une intégration sur différents canaux. Une fois la connexion entre LUIS et le Bot Framework mise en place, je peux le tester directement.

Une fois le test effectué (et validé) dans la console, il ne reste plus qu'à brancher votre Bot sur un channel, comme Slack par exemple, vous permettant ainsi de faire vos recherches de contacts directement depuis cet outil. Dans l'image précédente, vous pouvez visualiser le principe de dialogue mis en place entre LUIS et Slack : lors de la reconnaissance du texte, LUIS n'a pas compris le nom (paramètre), il a donc affiché le message d'erreur associé. Lorsque l'utilisateur a saisi le texte ("thomas"), LUIS a continué le dialogue : il a identifié qu'il s'agissait du paramètre pour l'intention de recherche de contacts, il a donc déclenché l'action correspondante !

Parfois, votre modèle ne sera pas en mesure de comprendre ce que l'utilisateur lui demandera. Mais vous pouvez l'éduquer, lui apprendre comment interpréter ce type de demande afin qu'il soit en mesure de devenir de plus en plus autonome. Nous allons à présent voir comment créer un bot plus intelligent, s'appuyant sur LUIS pour comprendre le contexte et pour ensuite vous permettre d'avoir une maîtrise à 100% du processus. Comme précédemment, la première étape consiste à créer un Bot, via le modèle de projet associé et à créer une nouvelle classe, héritant de LuisDialog, qui se chargera des interactions/dialogues entre le bot et l'utilisateur :

```
[LuisModel(Constants.LUIS_MODEL_ID, Constants.LUIS_MODEL_SUBSCRIPTION_KEY)]
[Serializable]
public class ContactsManagerBotDialog : LuisDialog
{
  [LuisIntent("")]
  public async Task None(IDialogContext context, LuisResult result)
  {
    string message = "I'm sorry, my responses are limited, you have to ask me the right this.";
    await context.PostAsync(message);

    context.Wait(MessageReceived);
  }

  [LuisIntent("SearchContacts")]
  public async Task SearchContacts(IDialogContext context, LuisResult result)
  {
    EntityRecommendation name;
    if (!result.TryFindEntity("Name", out name))
    {
      context.ConversationData.SetValue<string>("DisplayName", string.Empty);

      string message = "Please, enter the name of the person your are looking for.";
      await context.PostAsync(message);
    }
    else
    {
      context.ConversationData.SetValue<string>("DisplayName", name.Entity);
    }
    context.Wait(MessageReceived);
  }
}
```

Plusieurs points sont à remarquer sur cette classe :

- Elle est décorée de l'attribut LuisModel, qui permet (via le couple Id/Secret) de savoir à quel modèle LUIS cette classe correspond ;
- Les méthodes portent l'attribut LuisIntent qui prend en paramètre le nom de l'intent, défini côté LUIS, qui doit déclencher cette méthode.

Le reste du code parle de lui-même : lorsque l'intent "SearchContacts" est déclenché côté LUIS, alors la méthode SearchContacts est appelée. Cette méthode regarde si, dans ce que le modèle lui renvoie, l'entité "Name" a bien été trouvée, et, dans ce cas, elle ajoute la valeur dans les données de la conversation. Il ne vous reste plus qu'à déployer votre Bot et à régler la configuration (Endpoint) du Bot Framework pour que le reste fonctionne ! Attention, au niveau de la configuration, pensez à bien utiliser une adresse en HTTPS. De plus, pensez à mettre à jour vos identifiants, dans le fichier web.config pour éviter les problèmes d'autorisation. De cette manière, vous avez la possibilité d'avoir la maîtrise complète sur le code de votre Bot et du modèle associé. De plus, cela vous évite d'avoir à formater vos résultats directement au niveau de votre API (comme c'est nécessaire dans la version utilisant uniquement LUIS): vous pouvez (et devez) garder votre API la plus simple possible et faire le rendu et la restitution des données au niveau du Bot. Pour finir, et parce que le Bot Framework le permet, voici le résultat de l'utilisation du Bot, conjointement avec LUIS, dans Skype (possible après avoir eu accès à la preview de bots Skype via https://developer.microsoft.com/en-us/skype/bots/) : **Fig.12.**

Quelques liens

La page d'aide de LUIS : https://www.luis.ai/help ;
La documentation du Bot Framework : http://docs.botframework.com ;
Un exemple de Bot qui se connecte à VSTS pour faire des requêtes : https://github.com/jeffhollan/BotFrameworkSample ;
Le blog de Patrice Lamarche : http://www.sortirduchaos.net.
Pour information, la documentation est accessible ici : http://docs.botframework.com/. Petit conseil : même si vous êtes développeur C#, regardez la documentation du Bot Builder en version Node.js, qui est un peu plus étoffée sur certaines parties !

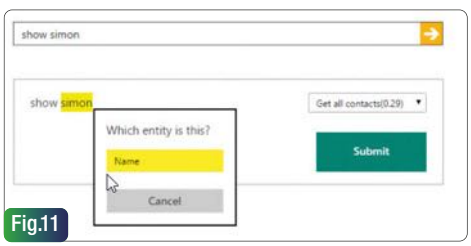


Fig.11

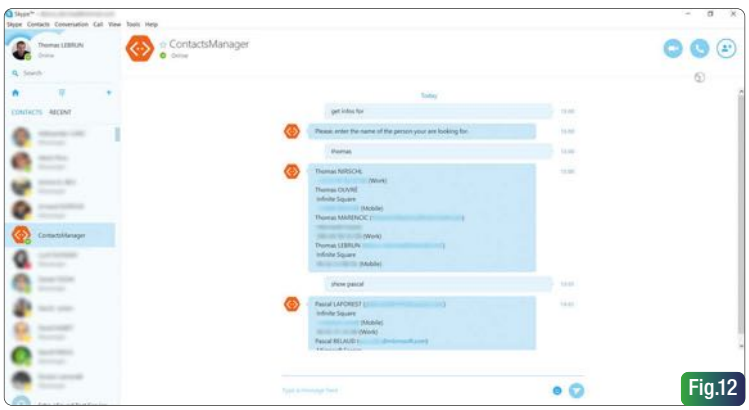


Fig.12

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

Ou encore sur votre tablette

Devoxx France 2016 : passion développeur 1ere partie

Que serait le printemps sans une grande conférence développeurs ? Devoxx a une nouvelle fois offert de très nombreuses sessions techniques sur de nombreux sujets. DevOps et Docker ont été deux sujets très présents, tout comme Java (ouf !) et JavaScript. C'est toujours une occasion de découvrir des technologies et des outils que l'on ne connaît pas vraiment, de savoir que telle fonction existe ou tout simplement de croiser les communautés, toujours nombreuses.
Impossible de parler de toutes les sessions, comme chaque année, nous avons dû faire un choix... Bonne lecture !

Modularisez votre JavaScript avec JSPM et SystemJs

En préparant ma venue au Devoxx 2016, une conférence a particulièrement retenu mon attention : « Modularisez votre JavaScript avec JSPM et SystemJs », présentée par Benoit LEMOINE (@benoit_lemoine) et Samuel VAILLANT (@samouss_28).
Ce talk m'a donné un aperçu des possibilités de JSPM et SystemJs.

Emilien MURATON
Développeur chez **PALO IT France**

Emilien développe depuis plus de 2 ans des applications en JavaScript et Java. Il utilise des frameworks tels que ReactJS, Webpack, Spring et Guava. Ses différentes experiences l'ont amené à travailler au sein d'équipes Agiles. Il a récemment développé une application Web s'appuyant sur le langage JavaScript natif dans un contexte international, ce qui lui a permis de renforcer ses connaissances en UX Design.

L'approche traditionnelle du JavaScript est d'utiliser beaucoup, beaucoup, mais alors beaucoup de balises scripts pour récupérer des librairies.

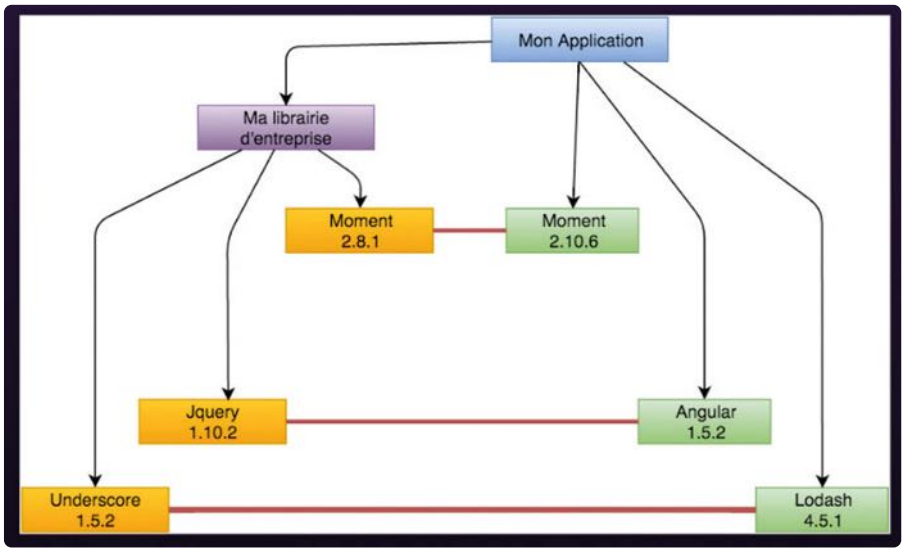
```
<script src="vendor/moment.js"></script>
<script src="vendor/lodash.js"></script>
<script src="vendor/query.js"></script>
<script src="src/app.js"></script>
```

LES PROBLEMES RECURRENTS

Cet enchaînement de balises pose de nombreux problèmes :

- Variables globales : prenons un exemple simple, si on importe lodash et underscore, ces 2 librairies définissent une variable globale avec le

même nom (le caractère underscore). Mais comment savoir quelle librairie on utilise lorsque l'on se sert de la variable _ ? ;
- Conflits de versions ;
- Ordre d'exécution des scripts (attention ! un mauvais ordre d'appel des scripts casse l'application.) ;
- Dépendances transitives : quand deux librairies ont comme dépendances la même librairie mais avec deux versions différentes. Voici un exemple concret présenté lors de la conférence qui résume ces problématiques :



QUE FAIRE ?

Il faut utiliser un système de modules. Historiquement, on retrouve AMD (Asynchronous Module Definition), une spécification qui permet d'organiser le code en modules, afin de pouvoir les charger à la demande. Cela évite d'avoir un seul gros fichier ou plusieurs fichiers à inclure dans le HTML dans un ordre précis. Son implémentation la plus connue est RequireJS.

Une autre spécification de module est CommonJs, plus orientée serveur (contrairement à AMD qui s'intègre très bien dans les navigateurs) et notamment utilisée dans Node JS. CommonJS charge les modules de façon synchrone. Lors de ce talk, les speakers nous ont plutôt conseillé d'oublier ces deux systèmes de modules afin de privilégier les imports d'ECMAScript2015 (es6), un format créé pour réunir les utilisateurs de CommonJS et d'AMD. Malheureusement, les imports es6 ne sont pas encore implémentés nativement par les navigateurs. C'est pourquoi les speakers nous ont orienté vers SystemJS, un polyfill ou shim des imports es6, qui comprend aussi les modules CommonJS et AMD.

Exemple system JS

```
var System = require('systemjs');
System.import('lodash');
System.import('https://code.jquery.com/jquery.js');
```

Nous voyons clairement ici que SystemJS permet de répondre de manière élégante aux problématiques d'imports successifs couramment rencontrés sur les projets legacy, via un système de modules. Pour aller un peu plus loin, et avoir un workflow plus sympa, on peut utiliser JSPM. Mais qu'est-ce que JSPM ? JSPM (JavaScript Package Manager) est un gestionnaire de paquets qui utilise SystemJS pour gérer les packages et leurs dépendances. Il a pour objectif de devenir LE module manager front-end et supporte le HTTP/2. Pour une installation facile de votre projet, jspm fournit un registry (lien : http://kasperlewau.github.io/registry/#/) des librairies les plus communément utilisées.

COMMENT L'UTILISER ?

Importons jspm :

```
npm install jspm –g
```

Créons la configuration de notre projet.

```
jspm init
Package.json file does not exist, create it? [yes]:
Would you like jspm to prefix the jspm package.json properties under jspm? [yes]:
Enter server baseURL (public folder path) [.]:
Enter jspm packages folder [./jspm_packages]:
Enter config file path [./config.js]:
Configuration file config.js doesn't exist, create it? [yes]:
Enter client baseURL (public folder URL) [/]:
Which ES6 transpiler would you like to use, Traceur or Babel? [traceur]:
```

Cela va créer un fichier nommé config.js qui contient les chemins des dépendances et sera mis à jour lors de l'utilisation des commandes avec jspm. Comme avec npm, on peut choisir d'enregistrer les paquets en tant que devDependencies (uniquement pour l'environnement de dev).

Il ne nous reste plus qu'à éditer ce fichier ou utiliser la commande jspm install pour gérer les dépendances Javascript de notre application.

ATTENTION !

Par défaut, jspm transpile l'es6 en es5 au runtime avec Babel, ralentissant énormément l'application. Ainsi il est conseillé de créer un bundle avec le CLI de JSPM :

```
jspm bundle clock-component --watch
```

Voici une comparaison des temps d'exécution entre une exécution avec transpilation au runtime, et une exécution avec notre bundle. Sans surprise on voit une énorme différence de temps entre les deux exécutions.

CONCLUSION

La combinaison de JSPM et SystemJS procurent une façon unifiée d'installer et de charger simplement et rapidement les dépendances. Elle permet aussi d'utiliser le futur des systèmes de gestion de modules dès aujourd'hui. Enfin JSPM est capable d'installer des paquets non seulement de NPM, mais aussi de Bower, ou directement de GitHub.

Source : Présentation Devoxx France 2016 : http://blemoine.github.io/jspm-slides/.

De l'IoT, des timeseries et de la prédiction avec Android, Cassandra et Spark

A l'heure des objets connectés, du Big Data et du machine Learning, nous pouvons voir que la pleine exploitation de ces technologies est encore timide et que celles-ci peuvent être mal articulées entre-elles. Aussi quand est paru le programme de Devoxx 2016, la conférence animée par Amira Lakhal, représentante des Duchess France et également développeuse agile Java et Scala, ne pouvait être qu'enthousiasmante.

Audrey COTON
Ingénieur Etude & Développement chez **Netapsys Atlantique**

Le pitch et la promesse de ces 45 minutes :

> *« Le nombre des objets connectés ne cesse d'augmenter de jour en jour. Nul ne peut ignorer leur potentiel et surtout l'impact que ces objets pourraient avoir sur notre quotidien.*
> *Je vais vous présenter un exemple d'objet connecté et de quelle façon analyser ses données. L'objectif est de voir comment collecter les données depuis un capteur du smartphone, les stocker dans Cassandra, et, enfin, les analyser pour prédire notre activité avec Spark.»*

Nous avons donc 3 étapes à réaliser : le recueil de données, l'analyse de celles-ci et l'interprétation que nous leur donnons.

Le recueil

A partir d'un Smartphone et d'une équipe de cobayes, Amira a recueilli un échantillon de données complet. Pour réaliser un recueil de données, une simple application ***Android*** suffit. Nous nous appuyons sur l'accéléromètre du téléphone afin de récupérer la donnée en temps réel, avec une composante essentielle : le temps. En effet toutes les données émises par les objets connectés sont horodatées. L'accélération de l'accéléromètre permet de récupérer les mesures dans les trois dimensions spatiales, X, Y et Z. Ici, cette simple application a été spécialement customisée pour les cobayes afin d'indiquer quelle activité était en cours : Standing, Sitting, Walking ou Running.

L'ingéniosité de la manœuvre tient surtout dans l'utilisation de ***Cassandra***.

Cette base de données (SGBD) de type NoSQL conçue pour gérer des quantités massives de données est une base orientée colonnes.

En utilisant le Time Series Data Model, un type de donnée particulier et en y associant le user concerné et la date du jour, Amira contourne la problématique de création de SSTable, à chaque insertion de nouvelles données ainsi que la limite du nombre de colonnes pour une même table.

En effet même si cette limite est impressionnante, un capteur qui établit une mesure toutes les secondes, c'est plus de 31,5 millions de valeurs à la fin de l'année ! Je vous laisse imaginer le volume pour des enregistrements toutes les millisecondes...

Nous avons donc maintenant une série d'enregistrements X, Y, Z, de timestamp associés par un user sur une activité définie Fig.1.

L'analyse

Une fois ces échantillons de valeurs stockés, l'étape d'analyse commence et le machine Learning entre en jeu. En effet la participation de chacun des cobayes sert ici à éditer un modèle de données.

Quelles sont les données représentant la course, l'arrêt, la marche, la position assise ?

Et c'est sur ***Spark***, framework de traitements Big Data open source construit pour effectuer des analyses sophistiquées et conçu pour la rapidité et la facilité d'utilisation, que continue la démonstration d'Amira. Il existe déjà des connecteurs entre Cassandra et Spark, ***Spark Streaming*** semblant idéal. Amira l'adapte donc en implémentant les méthodes onStart et OnStop, en y définissant son contexte Spark et surtout en délimitant la durée des batchs qui seront analysés.

L'interprétation

Le modèle de classification est quant à lui confié à une autre API de Spark, ***MLlib*** et son algorithme Random Forest sur la base des informations de référence enregistrées en amont. Eh oui, les cobayes n'ont pas couru, marché, pour rien.

Et ***MLlib*** permet d'exécuter de manière distribuée la plupart des algorithmes courants de Machine Learning dont l'algorithme Random Forest.

Une fois le training effectué et quelques étapes de transformation plus tard – Feature Engineering – un modèle réutilisable est maintenant exploitable. Mise en service sur un service REST, l'application peut enfin prédire/interpréter en temps réel l'activité de son utilisateur.

La démonstration en live d'un volontaire courant, marchant ou se tenant simplement immobile lors de cette conférence reste un temps fort et un moment original de la présentation d'Amira Lakhal

Pour conclure

Nous avons pu assister en live à l'enregistrement des données du volontaire, la transformation par batch avec Spark Streaming, l'application du modèle et la prédiction en temps réel de l'activité menée. La promesse de cette conférence fut tenue haut la main !

Amira Lakhal ayant open-sourcé l'ensemble de son projet, je ne peux que conseiller d'aller jeter un coup d'œil sur son gitHub https://github.com/MiraLak/

Je nuancerai cette démonstration par une précision : on assiste ici à une remarquable « démonstration » d'analyse en temps réel mais nous sommes encore loin de l'analyse prédictive. Mon smartphone peut détecter si je marche, si je cours mais pas encore si je vais marcher ou courir dans les minutes à venir ! A suivre donc, et de très près.

```
cqlsh:actitracker> SELECT * FROM users WHERE user_id = 8 AND activity = 'Standing' ORDER BY timestamp asc LIMIT 10;

 user_id | activity | timestamp       | acc_x | acc_y | acc_z
---------+----------+-----------------+-------+-------+-------
       8 | Standing | 112578291481000 | -5.13 |  8.16 |  1.31
       8 | Standing | 112578334541000 | -5.05 |  8.16 |  1.31
       8 | Standing | 112578411384000 | -5.41 |  8.16 |  1.38
       8 | Standing | 112578451484000 | -5.48 |  8.16 |  1.31
       8 | Standing | 112578491615000 | -5.33 |  8.16 |  1.18
       8 | Standing | 112578542701000 | -5.28 |  8.12 |  1.18
       8 | Standing | 112578581489000 | -5.18 |  8.12 |  1.23
       8 | Standing | 112578661476000 | -5.33 |  8.16 |  1.33
       8 | Standing | 112578701515000 | -5.37 |  8.12 |  1.33
       8 | Standing | 112578741462000 | -5.33 |  8.16 |  1.33
```

Fig.1

La Blockchain dans tous ses états

J'ai choisi d'assister à l'intervention de François Galilee et de Heykel Jelassi intitulée « Blockchain as a Trust Machine ? Vers une économie programmable » mais également au quicky de Philippe Antoine intitulé « Développez votre première application décentralisée sur la blockchain avec Ethereum et Embark ». Comme vous l'aurez compris, la Blockchain était le sujet de prédilection cette année.

Robert DEGRET
Architecte chez **PALO IT France**

Robert travaille depuis plus de 8 ans sur le développement d'applications requérant performance et haute disponibilité. Son expérience dans le domaine bancaire lui a permis d'appréhender des flux temps réel comportant des millions de données. Il a également développé en méthode Agile une application Web. Après avoir travaillé chez PALO IT Singapour pendant 4 ans, il est désormais basé au bureau parisien.

Blockchain

Bitcoin a apporté une solution concrète au « problème des généraux byzantins ». Ce problème classique de l'informatique distribuée dans un environnement non fiable peut s'expliquer par la métaphore suivante : dans le schéma, l'armée A fait le siège de l'armée B et C. B et C doivent attaquer ensemble pour gagner contre A. Pour cela, ils doivent échanger des messages : d'abord de B vers C pour proposer le moment et l'heure de l'attaque, mais C peut douter du message qu'il recevra de B (A en a eu connaissance, et peut-être l'a-t-il modifié, etc.). De même B attend le message de confirmation de C pour ne pas se retrouver à attaquer seul. Mais quand il reçoit ce message, il peut lui aussi douter de son origine et de son contenu.

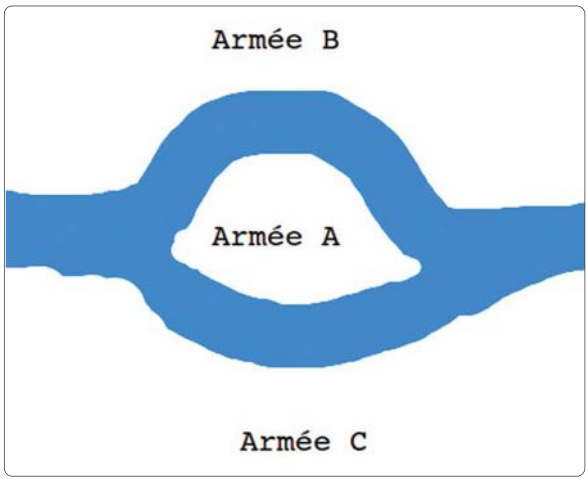
Satoshi Nakamoto [1] donne les principes de sa solution pour résoudre les problèmes du double paiement et des généraux byzantins :

> *« Nous proposons une solution au problème de double-dépense en utilisant un serveur horodaté distribué en pair-à-pair pour générer des preuves informatiques de l'ordre chronologique des transactions. Le système est sécurisé tant que des nœuds honnêtes contrôlent ensemble plus de puissance de calcul qu'un groupe de nœuds qui coopéreraient pour réaliser une attaque. »*

La blockchain représente simultanément un stockage d'information distribué en pair à pair et le mécanisme enrichit toutes les 10 minutes ce livre de compte ouvert à tous. Elle permet l'automatisation de la transaction en supprimant les tiers.

Contrats

Jusque-là nous n'avons parlé que de faits mais une blockchain peut aussi exécuter des programmes. Ils sont stockés avec les faits : chaque nœud l'exécute lorsqu'il reçoit le fait. Dans Bitcoin, c'est le mécanisme utilisé pour rendre certaines transactions conditionnelles. Par exemple, une transaction ne sera valide qu'à une certaine date.



D'autres blockchains sont d'ailleurs conçues autour de contrats bien plus élaborés. Chaque transaction Ethereum est accompagnée d'une mini-base de données avec des méthodes pour modifier les données. Comme les contrats sont diffusés sur tous les nœuds, les bases de données le sont également. A chaque modification des données par un utilisateur, le changement est répliqué et rejoué sur tout le réseau.
La notion de conditions préprogrammées, reliées au monde réel, diffusées à tous, constitue un "smart contract" [3]. Un contrat est une promesse qui lie 2 parties de manière légale. Un "smart contract" est la même chose, en remplaçant le mot légal par "technique". Il n'y a plus besoin de juge.

Effet de masse

Le mouvement est lancé et prend de l'ampleur. La communauté grandit et bouillonne d'idées. L'accès à la blockchain est de plus en plus simple, la blockchain en C++ de bitcoin a été portée dans de nombreux langages. Embark [2][4] permet par exemple de créer sa propre blockchain privée en moins de deux minutes. Des sites répertorient les dizaines d'applications "smart contract". Vitalik Buterin, le jeune créateur d'Ethereum, saute d'avion en avion pour donner des conférences à travers le monde. 40 grandes banques se sont ralliées autour de la startup R3 afin de créer une blockchain car elles sentent le danger.

Limites

Bitcoin a déjà rencontré quelques embûches : les fortes fluctuations de valeur, des délais dans les transactions, la fermeture de certaines plateformes d'échanges sous la menace des autorités monétaires dans le monde, etc. La blockchain sera testée lorsqu'il y aura besoin d'arbitrer des contrats réels. Une solution [5] proposée par Vitalik Buterin est d'avoir une cour décentralisée avec des "juges" payés par le réseau.

Liens :

[1] http://www.bitcoin.fr/pages/Bitcoin-expliqu%C3%A9-par-son-inventeur
[2] http://cfp.devoxx.fr/2016/talk/VZJ-1808/Developpez_votre_premiere_application_decentralisee_sur_la_blockchain_avec_Ethereum_et_Embark
[3] http://cfp.devoxx.fr/2016/talk/JGT-7450/Blockchain_as_a_Trust_Machine_%3F_Vers_une_economie_programmable_
[4] http://iurimatias.github.io/embark-framework/
[5] https://www.reddit.com/r/ethereum/comments/4gigyd/decentralized_court/?ref=readnext

La migration d'AngularJS à Angular 2

Je vous propose dans cet article de revenir sur une des conférences de Devoxx France 2016 dont le thème était « D'AngularJS à Angular 2 : préparez-vous dès maintenant à la migration » présentée par Benoit Lemoine.

Anthony Caillaud
Ingénieur Etude & Développement chez **Netapsys Atlantique**

L'annonce de la sortie d'Angular 2 a été faite en septembre 2014 et cette version 2 n'a pas grand-chose à voir avec AngularJS. Cependant, aucun chemin de migration n'a été prévu et cela a provoqué le « *Angular Bashing* », c'est-à-dire le rejet d'Angular en 2015. Suite à ce constat, la mise en place d'un chemin de migration est devenue indispensable.
Les grandes différences entre AngularJS et Angular 2 sont les suivantes :

- Javascript -> Typescript ;
- Orienté Modèle-Vue-* -> Orienté composant ;
- Double databinding -> Databinding *presque* unidirectionnel ;

Plusieurs étapes de migration sont donc nécessaires pour gérer ces différences.

Etape n°1 : suivre les bonnes pratiques

Tout d'abord, il faut organiser son code en modules fonctionnels et non techniques. Il est donc préconisé de regrouper le service, le controller, le template HTML et le fichier CSS nécessaires pour un module fonctionnel et non pas de regrouper ensemble tous les controllers, tous les services, etc. Ensuite, il est préférable d'utiliser les services d'Angular et d'éviter les factory. Au niveau des controllers, il est conseillé de bannir l'utilisation de $scope et d'utiliser controllerAs au niveau du routeur.

Etape n°2 : les composants

Un des grands changements cités ci-dessus est le passage d'un modèle orienté Modèle-Vue-* vers un modèle orienté composant. Ce modèle, très répandu côté JavaScript avec l'arrivée de ReactJS, a été choisi pour Angular 2. Pour faire simple, dans ce modèle, tout est composant : de l'input au formulaire, en passant par une page et l'application en elle-même, chaque élément est un composant. L'application peut donc être représentée sous la forme d'un arbre de composants dont la racine serait l'application elle-même.
À partir de cette observation, un nouveau routeur orienté composant a été développé afin de permettre le routage entre les différentes briques de l'application.

Etape n°3 : le système de modules dans Angular 1

Angular 2 et Typescript utilisent la notion de modules pour l'isolation du code de chaque composant. Les mots-clés *import* et *export* permettent l'utilisation des composants dans d'autres en gardant des composants cloisonnés. Plusieurs systèmes de modules sont supportés tels que AMD, UMD, CommonJS, ES6 ou encore SystemJS. C'est ce dernier qui est utilisé par défaut dans Angular 2.
En matière de package manager, Benoit Lemoine a abordé le dernier d'entre eux : JSPM. Sa force consiste à charger n'importe quels types de modules cités précédemment et depuis n'importe quel repository comme npm, github, bitbucket, etc. De plus, il a une intégration forte avec SystemJS. Un autre grand changement d'Angular 2 est l'utilisation de Typescript. Ce langage créé par Microsoft en 2012, open-source et qui transpile vers Javascript est un sur-ensemble d'EcmaScript 2015 ou EcmaScript 6. Il est statiquement typé avec un typage structurel et graduel. En voici un exemple : Fig.1. Typescript est décrit comme étant simple d'apprentissage et facile à migrer grâce à une complétion dans les IDE, un refactoring simplifié et des types qui servent aussi de documentation. Typescript supporte aussi des librairies tierces par des fichiers d'en-tête (*.d.ts) grâce à un autre package manager qui est Typings. En ce qui concerne le typage, plusieurs bonnes pratiques sont à suivre. Il est important de ne plus utiliser de *var* mais des *let* ou *const* lors de la définition de variables. L'usage du *const* est conseillé si on veut avoir une application avec un maximum d'immutabilité. Il existe le mot-clé *any* mais celui-ci est à bannir. Il faut maintenant typer explicitement les entrées et les sorties et éviter les appels dynamiques. Une autre différence entre Angular 1 et Angular 2 correspond à l'utilisation de $watch. Celle-ci n'est plus possible dans Angular 2, Benoit Lemoine nous montre donc sa solution à l'aide de getter et setter : Fig.2. L'utilisation de décorateurs sur les méthodes est aussi une nouveauté. Ils permettent d'ajouter du comportement transverse aux méthodes à l'aide d'annotations telles que throttle, inject ou encore memoize. Ce dernier permet par exemple de renvoyer une fonction d'une valeur identique au

Fig.1

```
interface User {
  firstName:string
  lastName:string
}

class Room {
  constructor(private users:Array<User> = []) {}

  addUser(user:User) {
    this.users.push(user);
  }
}

const room = new Room();
room.addUser({firstName:'Georges', lastName:'Abitbol'});
```

Fig.2

```
class TurtlesCtrl {
  public turtles:Array<Turtle> = []

  get firstTurtleName():string {
    return this.turtles[0] && this.turtles[0].name;
  }
  set firstTurtleName(firstTurtleName:string) {
    if (!this.turtles[0]) {
      this.turtles[0] = {name:'', color:''}
    }
    this.turtles[0].name = firstTurtleName;
  }
}
```

Fig.3

```
const MyAngular1Directive =
  upgradeAdapter.upgradeNg1Component('myAngular1Directive');

@Component({
  selector:'MyAngular2Directive',
  template:`
  <div>
    <My-Angular1-Directive my-params="2" />
  </div>`,
  directives:[MyAngular1Directive]
})
export default class MyAngular2Directive {

};
```

Fig.4

```
import angular from 'angular';
import upgradeAdapter from 'upgrader';
angular.module('turtles.app').directive('turtlesShow',
        <IDirectiveFactory>upgradeAdapter.downgradeNg2Component(turtlesShow2))
```

précédent appel dès lors que cette fonction a été appelée avec les mêmes arguments. Cela engendre un gain de performances à l'aide d'une seule annotation @memoize sur la fonction.

Etape n°4 : Angular 2 dans Angular 1

Pour commencer, il faut installer Angular 2 à l'aide du package manager choisi. Pour que votre application fonctionne, il faut récupérer au même moment rxjs et reflect-metadata.

Pour utiliser Angular 2 dans une application Angular 1, peu importe l'ordre et la position des modules Angular 2 dans l'arbre définissant l'application hybride, celle-ci fonctionnera correctement tant que la racine restera un module Angular 1. Ensuite, pour ugrader des composants et services Angular 1 ou downgrader des composants et services Angular 2, nous devons utiliser *UpgradeAdapter.* Dans le premier cas, nous souhaitons intégrer une directive Angular 1 dans un composant Angular 2. Voici comment l'utiliser : Fig.3.

Dans le second cas, nous souhaitons utiliser un composant Angular 2 dans notre application Angular 1. Pour cela, la méthode de downgrade de l'UpgradeAdapter va permettre de créer une directive Angular 1 à partir du composant Angular 2. Fig.4. Une autre solution qui s'appelle Ng-forward consiste à écrire du code avec les conventions Angular 2 avec du Angular 1.3+. Cette solution peut être une première étape avant d'écrire du vrai code Angular 2.

En conclusion, pour réaliser une bonne migration d'une application Angular 1 vers Angular 2, il faut suivre les bonnes pratiques Angular, utiliser un système de modules avec Typescript, migrer les « feuilles » de votre arbre de composants en finissant par le routage et le composant le plus haut.

Domain Driven Design

J'ai pu découvrir cette année le Devoxx France. C'était une grande première pour moi, et je dois dire que je n'ai pas été déçu. Un planning qui ne connaît aucun temps mort et des conférences plus intéressantes les unes que les autres : je décrirais le Devoxx comme le « Disneyland des Développeurs » ! J'ai choisi de donner mon retour sur une conférence intitulée « DDD : et si on reprenait l'histoire par le bon bout ? » par Thomas Pierrain et Jérémie Grodziski.

Stéphane LOPES NEVES
Développeur chez **PALO IT France**
Au cours de ses études, Stéphane a travaillé en tant que Développeur au sein d'un cabinet de conseil, mais également chez différents leaders de marché comme Airbus Helicopters, où il a conçu, entre autres des applications de gestion en .NET et Java JEE. Il a rejoint PALO IT.

"Le DDD, mais qu'est-ce que cela peut être ?" Les paris sont lancés ! Que peuvent bien signifier ces fameux « D » ? « Driven Development » ? Framework, pattern, techno ?

Le gagnant de l'acronyme du jour n'est autre que le "Domain Driven Design". Il y avait bien "Driven", je n'étais pas si loin de la vérité ! Cachée derrière ces trois mots, une vraie philosophie de développement plus qu'un pattern.

L'objectif est de faire de l'objet en restant le plus proche possible du langage des utilisateurs. Concrètement, il m'arrive de faire face à ce genre de situation où l'on me demande : « Pourquoi certaines factures en attente ne sont pas valides ? Les spécifications fonctionnelles de cette demande sont-elles facilement accessibles? » Si c'était le cas on ne viendrait pas me poser la question. Résultat, je lance l'application que j'exécute pas à pas jusqu'à tomber sur la méthode suivante :

```
public Boolean validate(Invoice invoice) {
    return !"14".equals(invoice.getStatus())
        && LocalDate.now().getDayOfMonth() <= 25
        && invoice.getPrice().compareTo(new BigDecimal(500)) < 0;
}
```

Je réponds en plissant les yeux dans un français que je pense correct : *"Pour être valide, une facture ne doit pas avoir le statut 14, être validée jusqu'au 25 et avoir un prix inférieur à 500."*

Si on me répond "Aaah, merci !", dans ce cas pas de problème ! Mais je peux être également confronté à ce type de questions "c'est quoi le statut 14 ?", "pourquoi le 25 ?" ou encore "pourquoi 500 ?". Toute la logique fonctionnelle est masquée par des nombres magiques qui n'ont pas forcément de sens pour le métier. Cela a pour conséquence de complexifier le dialogue entre développeurs et utilisateurs. Je reprends cette même méthode écrite de la manière suivante :

```
public Boolean validate(Invoice invoice) {
  return !InvoiceStatus.ABANDONNEE.equals(invoice.getStatus())
      && invoice.beforeLastDayOfMonthToValidateInvoice()
      && invoice.lowerThanTheMaximumPriceAllowed();
}
```

Simplement en lisant, je sais qu'une facture est valide si :

- Elle n'a pas le statut abandonnée ;
- La validation doit être faite avant un jour précis ;
- Qu'elle doit avoir un montant inférieur au prix autorisé.

Derrière ces différentes méthodes se cachent le même code. Je connais juste la signification de l'utilisation de données et je suis capable de fournir une information à la personne qui me pose la question. C'est simple à mettre en place, non ? Il existe un ouvrage que les adeptes appellent le "Blue Book" d'Eric J. Evans, considéré comme la référence pour comprendre le DDD. Sur le papier, ça n'a vraiment pas l'air d'être un livre auquel on s'attaque pendant les trajets de bus. C'est d'ailleurs un point qui a été souligné par les conférenciers. Ils nous ont également expliqué que le meilleur moyen de faire du DDD, c'est de ne pas dire que l'on en fait, car c'est souvent complexe et difficile d'accès. Afin de démocratiser cette pratique, ils optent plutôt pour une vulgarisation des différents concepts. Ils préconisent que l'on procède par des améliorations successives du code en fonction de ses connaissances du DDD, plutôt que d'essayer de tout assimiler et de tout remettre en question de manière radicale.

Conclusion

L'adoption de cette philosophie amène les développeurs à comprendre les besoins fonctionnels auxquels ils doivent répondre quotidiennement au travers des applications sur lesquelles ils travaillent. Cela a pour conséquence de faciliter leur communication avec les utilisateurs qui se sentent alors compris et seront plus enclins à partager des informations. C'est un exercice qui peut être fait sur le code dans lequel vous travaillez sans compromettre le reste de l'application. Bien sûr, le sujet est bien plus large que ce que j'ai présenté mais cette conférence m'a donné envie de creuser le sujet pour comprendre les différents concepts cachés derrière ces trois lettres.

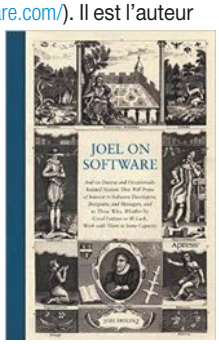
Interview de Joel Spolsky (Stack Overflow)

Aurélie Vache
Développeuse Web Full-Stack chez **atchikservices**
Duchess France Leader

Amira Lakhal
Développeuse Agile en Java et Scala chez **Valtech**
Duchess France Leader

Jeudi 22 avril, à l'occasion de Devoxx France 2016, nous avons interviewé Joel Spolsky, créateur de Trello, CEO de Stack Overflow et auteur de plusieurs livres. Retour sur l'interview d'une personne très inspirante et passionnée par son travail.

Joel a étudié l'informatique à l'université de Yale, il y obtient son diplôme en informatique en 1991. Il a ensuite travaillé chez Microsoft sur les logiciels Excel et Visual Basic for Applications (VBA). En 2000, il a fondé la compagnie Fog Creek Software et a créé le blog Joel on Software (http://www.joelonsoftware.com/). Il est l'auteur de plusieurs livres publiés entre 2001 et 2005. Il a créé en 2008 StackOverflow et en 2011 il a fondé Trello.



Stack Overflow

Avant Stack Overflow, il y avait un site appelé Experts Exchange, où les développeurs pouvaient partager des réponses.

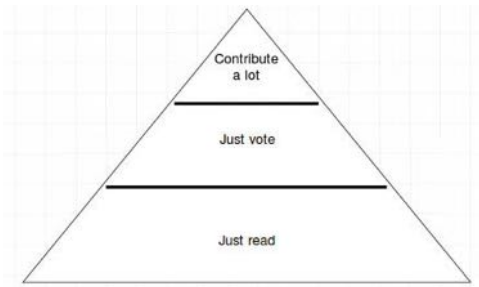
Et un jour, en pleine apogée de la bulle Internet, la société derrière Experts Exchange ne pouvait plus payer les factures. Ils avaient tous ces développeurs qui pouvaient résoudre leurs problèmes sur le site Internet, mais la société n'arrivait pas à résoudre le sien, faire de l'argent, donc, à un moment donné, ils en ont manqué (d'argent, pas de développeurs). Ils ont presque fermé le site, mais des nouveaux propriétaires sont arrivés, et ont résolu ce problème financier en mettant en place un paywall.
De ce fait, si les développeurs voulaient lire les réponses, ils devaient être membres du site moyennant le faible prix de 20 dollars par mois, en devise américaine ou équivalent. Cette modification a mis Joel en colère ! Pourquoi les développeurs devaient payer pour accéder à des réponses données en toute bonne foi ? C'est à ce moment là, en Avril 2008, qu'il a commencé Stack Overflow avec Jeff Atwood.

Ils ont commencé Stack Overflow, dans le but de rendre la vie meilleure pour les développeurs en refusant de devoir faire payer ces derniers pour utiliser la plateforme. Au lieu de cela, ils ont réalisé qu'ils pourraient exiger des employeurs de diffuser leurs offres d'emplois sur la plateforme - avec d'excellents résultats ! Le premier résultat, est que les développeurs seraient effectivement en mesure de voir les offres d'emplois qui les intéressent. Des emplois basés en fonction de leurs compétences en programmation, et géo-localisées à leur emplacement. Et d'autre part, que Stack Overflow pourrait continuer à croître, grâce à un flux de revenus.


Jeff Atwood

Contributeurs

S'il y a un chiffre à retenir c'est qu'environ 26-27% des utilisateurs postent environ 5 questions et/ou réponses par mois.
Joel nous a expliqué qu'ils ont quelques métriques sur les utilisateurs et l'utilisation de Stack Overflow :
La principale connexion du site se produit sans authentification.
Il y a une catégorie des utilisateurs qui s'authentifient et lisent des réponses. Ensuite il y a les utilisateurs authentifiés, qui lisent les réponses, votent pour les meilleures réponses et donnent rarement d'autres indices. Viennent enfin ceux qui s'authentifient et contribuent régulièrement en répondant aux différentes questions.
Ces différences d'utilisation peuvent être représentées sous forme d'une pyramide.



Avez-vous des statistiques concernant le nombre de contributrices sur StackOverflow ?

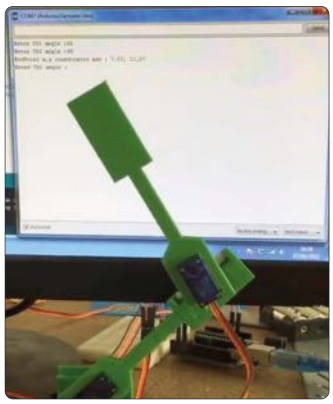
Il n'y a pas de statistiques du nombre de contributrices sur le site. Joel aimerait que le nombre de femmes sur Stack Overflow soit plus important, pour inspirer plus de jeunes femmes à contribuer a leur tour. Selon lui " nous avons besoin de rôles modèles féminins, pour inspirer une génération future " et nous sommes bien d'accord avec lui !

Nouveautés sur Stack Overflow

Developer Story est une fonctionnalité de Stack Overflow actuellement en Beta (http://meta.stackoverflow.com/questions/313960/introducing-the-developer-story). Même si cette fonctionnalité se nomme Developer Story, elle n'est pas disponible uniquement pour les devs mais pour toute personne utilisant SO, mais également les sysadmins, et les chefs de projets par exemple. Joel croit que la valeur des développeurs ne se reflète pas de manière adéquate dans un Curriculum Vitae (CV). Au lieu de cela, il pense qu'il est préférable de raconter votre histoire : les livres que vous avez lus, les projets auxquels vous avez contribué, les questions que vous avez posées ou auxquelles vous avez répondu sur Stack Overflow, les langues que vous avez apprises... La valeur d'un développeur ne se résume pas aux entreprises dans lesquelles il a travaillé et à l'école qu'il a fréquenté, c'est bien plus que cela !

Nous remercions Joel de nous avoir consacré un peu de son temps et pour cette rencontre très enrichissante ! :-)

Les cinématiques avec Robobox



Une des premières étapes dans la construction d'un robot consiste à pouvoir déterminer le comportement des membres de notre machine. Le bras articulé constitue une très bonne entrée en matière car il permet d'introduire la notion de cinématique, c'est-à-dire le lien entre l'orientation de nos moteurs et la position finale de notre robot.

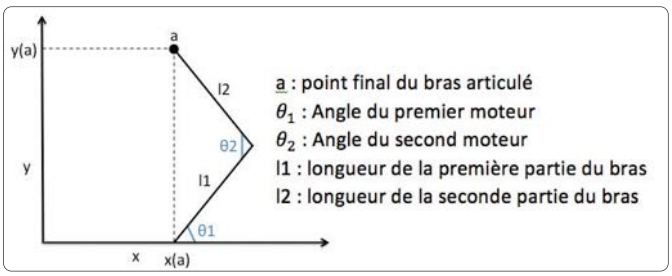
Jonathan Trevier
CEO - Robobox
robobox.fr

Lorsque nous voulons modéliser le comportement d'un bras articulé, deux scénarios s'offrent à nous :

- Nous voulons déterminer la position finale du bras du robot à l'aide des angles des moteurs, on utilisera alors les règles de cinématique avant.
- Nous cherchons à définir les angles requis pour arriver à une certaine position, nous parlons alors de cinématique inverse.

Les cinématiques avant

Il est souvent plus simple de définir la position d'un bras articulé grâce aux angles des moteurs que de trouver les angles requis pour une position donnée. Pour calculer la position d'un bras articulé nous faisons appel aux règles de trigonométrie :



Dans le cas d'un bras n'ayant qu'un seul moteur, nous calculerions les coordonnées de bras comme ceci :

$$Xa = l1 * \cos\left(\theta_1 * \frac{\pi}{180}\right) \qquad Ya = l1 * \sin\left(\theta_1 * \frac{\pi}{180}\right)$$

Pour deux bras, l'équation ne se complique que légèrement

$$Xa = l1 \cdot \cos\left(\theta_1 \cdot \frac{\pi}{180}\right) + l2 \cdot \cos\left((\theta_1 + \theta_2) \cdot \frac{\pi}{180}\right)$$

$$Xa = l1 \cdot \sin\left(\theta_1 \cdot \frac{\pi}{180}\right) + l2 \cdot \sin\left((\theta_1 + \theta_2) \cdot \frac{\pi}{180}\right)$$

Comme vous pouvez le constater la simulation est très proche de la réalité et le code de notre module Arduino est assez simple :

Code : https://gist.github.com/RoboboxFR/49dd0aa14cdd29f3ce8757790ed4892a

Dans un premier temps nous importons la bibliothèque « Servo » puis nous ajoutons deux objets « servo moteurs ». Nous initialisons ensuite des variables qui nous permettront d'enregistrer un certain nombre de données comme la position des moteurs, l'angle inscrit par l'utilisateur ou encore la longueur des membres du bras (ici définis à 10 cm). Dans un second temps, à l'intérieur de la fonction « setup » nous informons le microprocesseur que les moteurs sont attachés sur les ports 2 et 4 de notre carte. Puis nous initialisons la conversation avec le port USB et écrivons dans le terminal une requête pour entrer le premier angle désiré « Enter Th1 angle : ». Nous nous attaquons enfin à la boucle sans fin. Une boucle sans fin, de la forme While(TRUE) ou void loop() est typique des projets de robotique dans lesquels on demande à ce qu'un programme soit exécuté continuellement.

L'objectif de notre boucle est :

- De demander à l'utilisateur l'angle souhaité pour le moteur numéro 1 ;
- D'actionner le moteur pour qu'il atteigne cet angle ;
- De demander à l'utilisateur l'angle souhaité pour le moteur numéro 2 ;
- D'actionner le moteur pour qu'il atteigne cet angle ;
- De réinitialiser le programme.

Un des challenges de ce code est que les informations transmises à travers le moniteur ne peuvent dépasser un caractère ou un entier, or nous voulons entrer un angle entre 0 et 90°. Pour remédier à ce problème nous créons un array de taille 2 [0,0] puis incorporons l'input de l'utilisateur un élément à la fois. Une autre complication est que l'information entrée dans le moniteur est interprétée en ASCII, l'entrée 0 correspond donc à l'entrée 48 de la bibliothèque ASCII, le 1 correspond à l'entrée 49, et ainsi de suite. Nous devons donc soustraire 48 à chaque entrée pour traduire la valeur en ASCII en nombre entier. Lorsque les deux éléments de l'array sont renseignés, soit quand i == 2 dans notre code, nous procédons à la commande de notre bras articulé grâce à la commande servo.write() puis répétons le même processus pour le second moteur. Enfin, une fois que la position voulue des deux moteurs a été renseignée, nous précisons à l'utilisateur la position finale du bras sur l'angle (x, y) grâce aux équations détaillées plus haut.

Les cinématiques inverses

Les cinématiques inverses permettent de plus nombreuses applications mais sont beaucoup plus complexes à modéliser que les cinématiques avant. Grâce aux cinématiques inverses, nous pouvons demander à notre bras de se placer dans une position particulière, d'éviter un obstacle et d'effectuer n'importe quelle trajectoire.

Le calcul des cinématiques inverse se fait à partir de la position finale du bras (x,y) et a comme θ_1 et θ_2. Dans le cas d'un bras ayant deux articulations, et donc deux inconnues, on peut approcher la valeur des angles grâce aux équations ci-dessous :

$$\theta_2 = cos^{-1} \cdot \frac{(x^2 + y^2 - l_1^2 - l_2^2)}{2 \cdot l1 \cdot l2}$$

$$\theta_1 = \operatorname{atan} 2(y, x) - \operatorname{atan2}(l1 + l2 \cdot \cos(\theta_2), l_2 \cdot \sin(\theta_2))$$

Une fois implémentées dans notre programme Arduino, ces formules nous offrent de très nombreuses possibilités ! Nous pouvons par exemple demander à notre bras de dessiner sur une feuille, de danser ou encore d'éviter un obstacle à l'approche !

Code : https://gist.github.com/RoboboxFR/79af4cf33e04ddc67ce7e39adf04f0be

Nous avons a présent légèrement altéré notre code pour lui permettre de calculer les cinématiques inverses.

Alors que le code précédent demandait les angles theta 1 et theta 2, notre nouveau code demande les angles X et Y du point visé A. Une fois ces deux points renseignés nous calculons grâce aux formules ci-dessus les angles requis. Les instructions ne sont envoyées aux moteurs qu'après le renseignement des deux coordonnées désirées. En effet, le système ayant deux inconnues (θ_1 et θ_2), nous avons besoin de connaître les deux points pour pouvoir modéliser les angles.

Développement d'une application mobile de RV avec le Cardboard SDK

Wajdi Ben Rabah
Ingénieur en informatique spécialisé en technologies mobile et multimédia qui travaille actuellement avec le leader dans la communauté technologique : Sfeir. Sa passion pour les nouvelles technologies de réalité parallèle (VR/AR) n'a pas de limite.
Email: benrabah.wajdi@gmail.com
Web: www.wajdibr.com
Twitter: @WajdiBenRabah

Cet article est pour qui?

Si la réalité virtuelle vous intéresse, si vous voulez savoir comment ça marche, si vous voulez créer votre propre expérience, cet article est pour vous. Je suis persuadé que même si vous n'êtes pas programmeur, cet article va vous guider pas à pas dans la création d'une application de RV avec Google Cardboard sans connaissance antérieure. L'idée est de créer une application qui détecte le sens du regard et applique une nouvelle couleur au bloc de texte pointé. Vous pouvez utiliser ce projet comme un "starter" pour adapter l'action à effectuer dès la détection du regard de l'utilisateur.

La réalité virtuelle :

La RV ? Kézako?

La RV est généralement un environnement en 3D généré par une machine et qui semble bien réel à la personne qui l'expérimente. La RV est généralement testée en employant un équipement électronique spécifique (habituellement un casque). L'objectif est donc de parvenir à avoir une expérience immersive au point d'avoir le sentiment d'avoir une forte présence dans le monde virtuel.

Historique

La RV n'est absolument pas un nouveau concept. Elle existe depuis un bon moment mais cachée dans les laboratoires de recherche et dans le domaine militaire. La toute première tentative de commercialisation d'un dispositif de RV a été faite en 1966 par Ivan Sutherland. Toutes ses tentatives ont connu un échec. C'était un dispositif très encombrant, et très cher.
En 2012, Palmer Luckey le fondateur d'Oculus VR LLC a fait une démonstration d'un dispositif à John Carmack, bien connu pour avoir développé Doom, Quake et bien d'autres jeux. Ensemble, ils ont lancé le premier kit de développement appelé Oculus Rift Development Kit 1.
Leur succès a attiré les investisseurs, notamment Mark Zuckerberg, le fondateur du réseau Facebook, qui a racheté l'entreprise en 2014 pour 2 milliards de dollars. Tout ce mouvement était juste une promesse dans un premier temps, sans marché, ni demande, et même pas un produit fini ! Très peu de temps après, plusieurs ont suivi, dont Google, Sony, Samsung et Steam.
Cette technologie peut être employée dans plusieurs domaines à savoir la recherche scientifique, la simulation, dans le domaine médical (surtout pour le traitement des phobies), dans le domaine militaire et surtout dans le domaine des jeux vidéo.

Types de casques de RV

Actuellement on dispose de deux types :

- *Les casques Desktop :*

Le casque dans ce cas est un périphérique connecté d'une manière câblée généralement à une machine qui s'occupe du rendu graphique (un processus bien lourd). La machine peut être un ordinateur (Linux, Windows ou Mac) ou encore une console de jeux vidéo (La playstation par exemple). Le casque dans ce cas est un dispositif d'affichage (Head-mounted display) qui dispose de capteurs sensoriels.
Dans cette catégorie on peut citer l'Oculus Rift, ainsi que le HTC Vive, ou encore le Playstation VR.

- *Les casques mobiles :*

Le casque englobe tout simplement deux lentilles et un espace pour un téléphone mobile. Ce dernier adapte l'affichage pour avoir un rendu stéréographique. On dispose d'un capteur pour le suivi de la rotation de la tête mais pas de la position. Le rendu est assez limité pour les environnements qui nécessitent un calcul important vu que cette tâche est prise en charge par le téléphone en question (à l'encontre des casques Desktop qui délèguent cette tâche au processeur graphique d'un ordinateur/console).
Dans la catégorie des casques mobiles, on cite principalement le Samsung Gear VR et le Google Cardboard qu'on va utiliser dans cet article.
Google fournit des spécifications open source pour le Cardboard ainsi que des exemples riches pour guider les développeurs et les aider à bien commencer. Vous pouvez vous procurer un casque cardboard pour 15€ à 20€ généralement. Le SDK est aussi disponible en téléchargement, en fonction de la plateforme de développement, gratuitement à l'adresse suivante : https://developers.google.com/cardboard/overview.

La différence entre RV et RA :

La réalité augmentée est une technologie soeur à la réalité virtuelle. Elle est caractérisée par l'intégration de l'information numérique avec l'environnement de l'utilisateur en temps réel. Contrairement à la réalité virtuelle, qui crée un environnement totalement artificiel, la réalité augmentée utilise l'environnement existant et le recouvre de nouvelles informations. Les applications mobiles de la RA utilisent généralement le système de positionnement global (GPS) pour localiser l'emplacement de l'utilisateur et son compas pour détecter l'orientation de l'appareil.
La forme la plus avancée de nos jours sur le marché est le produit de Microsoft (HoloLens) ainsi que le Magic Leap (Google a investi 542 millions de dollars dans cette startup).

Développer une application pour le Google Cardboard :

Pré-requis :

Faisons le tour des outils nécessaires (logiciels et hardware) pour débuter l'application.

1. Logiciels

– La dernière version de Unity3d, qui est actuellement la v5. Fig.1.
– Le SDK de la plateforme mobile cible, qui sera Android pour cet

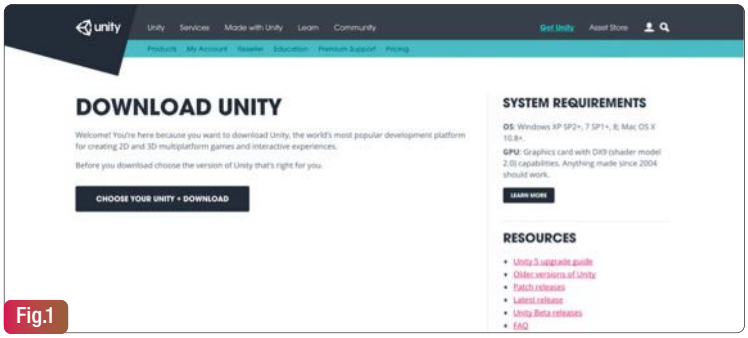


Fig.1

exemple. Télécharger le dernier SDK à cette adresse, choisissez « Stand-Alone SDK ». Fig.2.

– Le SDK de Google Cardboard pour Unity3d. Choisissez le fichier d'extension « unitypackage » afin de l'importer facilement sur Unity3d. Fig.3.

2. Hardware

- Un terminal mobile Android assez récent (disposant de la version 5 ou 6 d'Android).
- Le Google Cardboard. Fig.4.

Etapes :

Dans cette section, nous allons découvrir les premières étapes nécessaires au développement d'une application de RV sur Unity3D.

Préparation de l'environnement

- Lancez Unity3D et connectez-vous. Si vous ne disposez pas d'un compte, créez un compte utilisateur sur cette adresse : https://accounts.unity3d.com/
- Connectez-vous sur l'interface de Unity afin d'avoir cette interface (list de vos projets) : Fig.5.

Fig.4

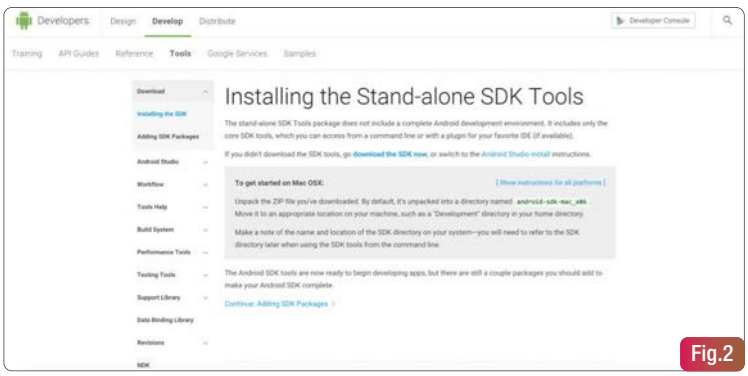


Fig.2

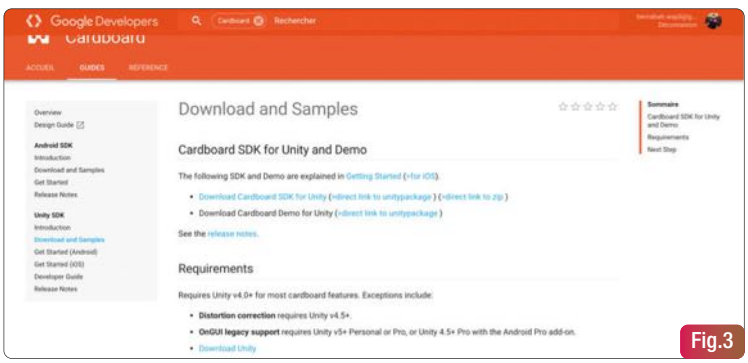


Fig.3

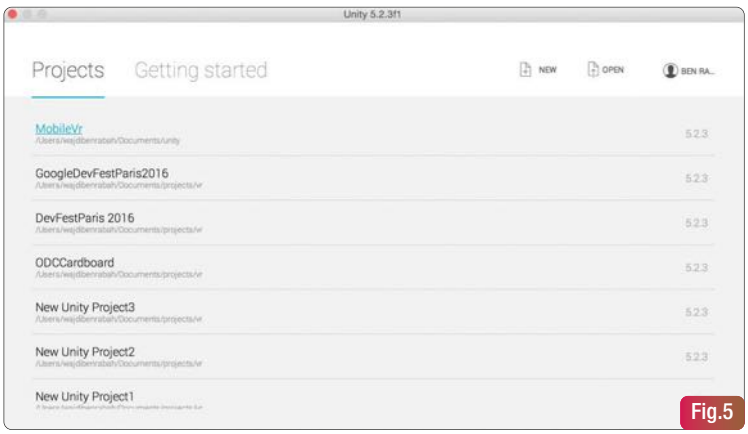


Fig.5

- Selectionnez Nouveau et mentionnez un nom pour votre projet. Sélectionner 3D et confirmer en demandant la création du projet. On ne va importer aucun "unity package" pour l'instant. Fig.6.
- Une fois Unity lancé, importez le SDK de Google Cardboard en lançant le fichier téléchargé (.unitypackage).
- Si vous utilisez une version de Unity inférieure à la v4.5 : sélectionnez tous les dossiers à importer.
- Si vous utilisez la version 4.5 de Unity ou plus (nous utilisons la version 5.3.4.f1 dans cet article) : désélectionnez le dossier « legacy ».

Scène:

Afin de garder les choses propres, enregistrons notre scène comme suit :

- Créez un dossier **scenes** sous le dossier Assets pré-créé par Unity.
- Entrez dans le dossier, et allez dans Files->Save Scene. Choisissez un nom (on l'a nommé main dans notre exemple). Fig.7.

Création de terrain:

Game Object 3D object Terrain. Fig.8.

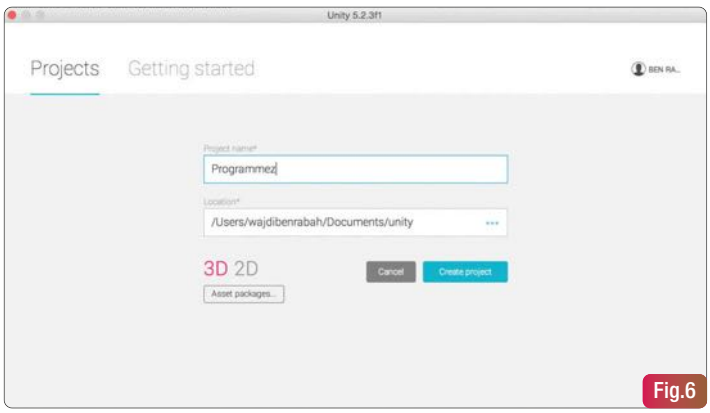


Fig.6

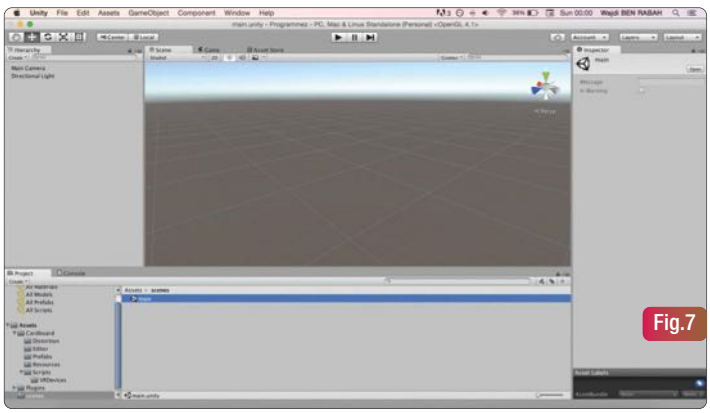


Fig.7

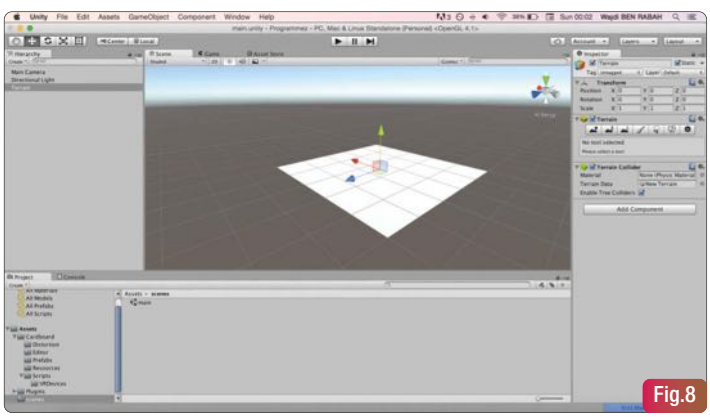


Fig.8

Appliquez une texture :

Sous le dossier Assets, faites un clic droit et sélectionner “import package”-> environment.

Une fois l’importation finie, sélectionnez le terrain depuis le menu hierarchy, puis rendez-vous sous le menu « Inspector » et choisissez « Paint Texture », puis « Edit Texture ».

Maintenant choisissez « Add Texture », et, dans la première case, sélectionnez l’image -> Add.(pour notre exemple nous avons opté pour la texture GrassHillAlbedo).

Personnalisez votre terrain :

Hiérarchie ➔ sélectionner le terrain.

Menu Inspector ➔ Sélectionnez « Raise / Lower Terrain » ➔ choisissez une brosse (optionnel).

Revenir sur la scène ➔ Appuyez pour créer des montagnes. Fig.9.

Ce visuel vous plaît-il ? Il est temps d’inclure une touche de réalité virtuelle !

Google Cardboard SDK :

Si vous vous rappelez bien, nous avons déjà importé le package Google Cardboard pour Unity dans notre projet.

- Maintenant importez l’objet Cardboard main comme suit :
- Menu « Projet » -> Cardboard -> Prefabs -> CardboardMain.
- Sélectionnez ce fichier et déplacez le sous le menu “hierarchy”.

Supprimez l’objet caméra créé par Unity (vous allez le trouver sous le nom de Main Camera généralement sous le menu Hierarchy) car le prefab de Google Cardboard inclut déjà une caméra avec vision stéréoscopique.

- Ajustez la vue et le rendu final en vous référant à la vue « camera preview » en bas à droite ou dans l’onglet « Game ».

Pour ce faire, voilà les raccourcis :

W : Activer le déplacement des objets.

E : Activer la rotation.

R : Activer le redimensionnement des objets.

F : Centrer la caméra sur l’objet sélectionné.

Attention : pour manipuler la caméra de Google Cardboard, il faut sélectionner tout l’objet « CardboardMain » dans la hiérarchie. La sélection dans la scène peut prendre seulement une partie du prefab et peut donc fausser votre rendu.

Voilà un rendu de l’état actuel de notre scène. Veuillez noter qu’on est en mode vision du jeu, afin d’avoir une idée sur le rendu attendu.

Tester le rendu :

Vous pouvez à tout moment appuyer sur la touche “play” depuis l’éditeur de Unity3D afin de voir à quoi ressemble le rendu final du jeu. Si vous nous avez suivi, vous auriez un rendu à peu près similaire à ceci : Fig.10.

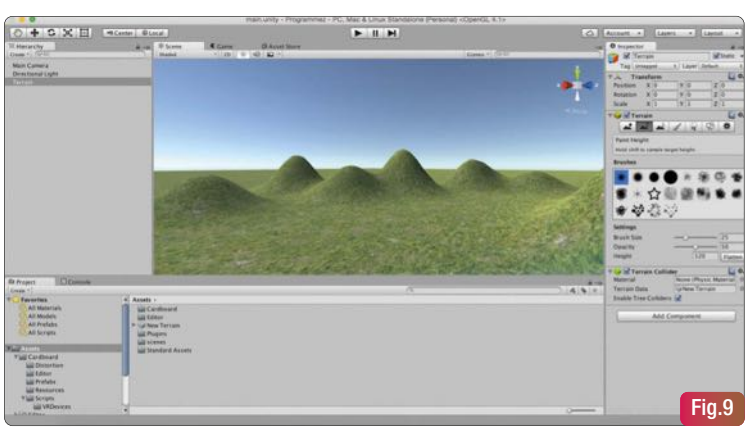


Fig.9

Exploiter le GazeInput (intermediaire) :

Nous avons un “hello world” sur Google Cardboard, mais si vous voulez ajouter un peu d’interactivité je vous invite à mettre les mains un peu plus dans le scripting.

Unity3D nous offre deux possibilités pour ce volet, soit le langage Javascript, soit le langage de Microsoft, C#. Pour cet article nous allons utiliser du C#.

Je voudrais bien attirer votre attention sur le concept d'interaction dans l’environnement VR car, comme vous le savez, nous ne disposons pas du même nombre d’entrées (par exemple pour le cardboard juste une gachette). Je vous propose qu’on ajoute un texte qui change de couleur lorsqu’on le fixe des yeux. Ca vous tente ?

Partie UI:

Créez un texte en 3D :

Menu “game object” -> “3D object”-> “3D Text”.

Positionnez-le devant la camera, et changez le texte.

Maintenant ajoutez un composant “collider” à votre texte pour permettre l’entrée en contact.

Sélectionnez le texte dans le menu “hierarchy”, puis allez au menu component en haut ou dans le menu “inspector” ->physics->box collider.

Voilà ma version actuelle du projet : Fig.11.

Partie scripting :

Pour commencer, je vous invite à préparer le terrain pour votre script.

Sous le dossier “assets”, créez un dossier portant le nom de “scripts” afin d’arranger nos fichiers sources.

Créez maintenant un fichier et nommez le “CardboardCollider”.

Double cliquez afin d’ouvrir le fichier avec l’éditeur MonoDevelop.

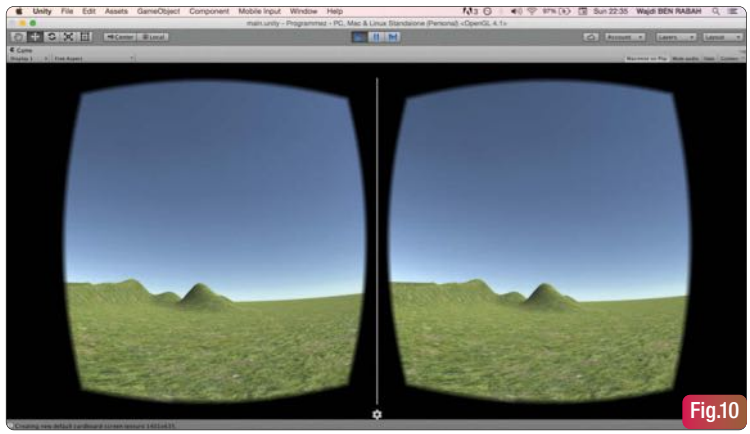


Fig.10

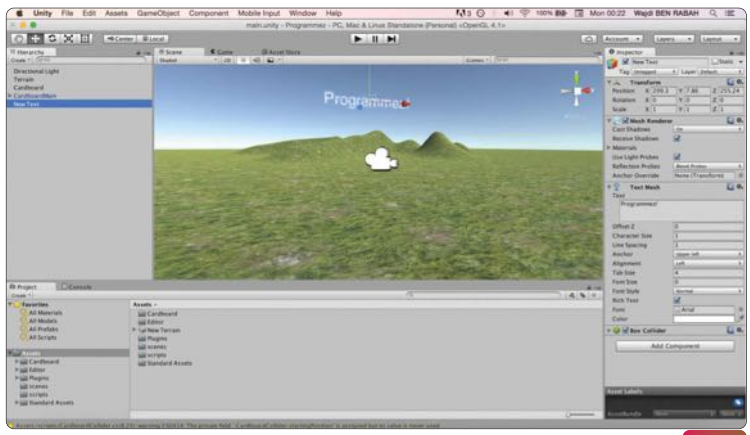


Fig.11

Script à définir :

Afin de faciliter la compréhension, je vais partager le code source, et on va le détailler ensemble.

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(Collider))]
public class CardboardCollider : MonoBehaviour {

  private CardboardHead head;
  private Vector3 startingPosition;

  void Start() {
    head = Camera.main.GetComponent<StereoController>().Head;
    startingPosition = transform.localPosition;
  }

  void Update() {
    RaycastHit hit;
    bool isLookedAt = GetComponent<Collider>().Raycast(head.Gaze, out hit, Mathf.Infinity);
    GetComponent<Renderer>().material.color = isLookedAt ? Color.green : Color.red;

  }

}
```

Explication :

```
using UnityEngine;
using System.Collections;
```

Librairies utilisées pour notre script, comme math.h par exemple pour le langage C si on a besoin d'opérations de calcul dans notre programme.

```
[RequireComponent(typeof(Collider))]
```

Oblige le composant qui va porter le script à avoir un collider (donc si vous l'oubliez, le script ne pourra même pas coller sur votre composant et Unity vous affiche un message d'erreur)

```
private CardboardHead head;
private Vector3 startingPosition;
```

head : pour garder un suivi sur le mouvement de la tête avec le cardboard.
StartingPosition: la position de départ.

```
void Start() : C'est la fonction qui est appelée dès que le jeu se lance. Elle est appelée une fois.

head = Camera.main.GetComponent<StereoController>().Head;
Head prend le head de la caméra du Cardboard.
startingPosition = transform.localPosition;
On initialise la startingPosition du texte à la position initiale du composant.

void Update() { : Cette fonction est appelée pour chaque frame du jeu.
RaycastHit hit : définition d'un rayon invisible.
bool isLookedAt = GetComponent<Collider>().Raycast(head.Gaze, out hit, Mathf.Infinity);
Tester si notre collider (qui réside dans le composant texte 3D) est en contact avec le
composant gaze (regard) du « head » de la caméra du cardboard.

GetComponent<Renderer>().material.color = isLookedAt ? Color.green : Color.red;
```

Cette ligne définit le material de notre texte 3D. Si isLookAt est vrai alors changer la couleur du texte en vert, sinon on affiche le texte en rouge.
Enregistrez le script et revenez sur Unity3D.

Assemblage :

Sélectionnez le texte depuis le menu "hierarchy" et glissez le script "CardboardCollider" en tant que composant au texte sous le menu "inspector", comme suit : Fig.12.
Enregistrez l'avancement sur la scène et essayez de lancer le projet.
Si vous avez tout suivi, vous allez avoir cet écran : Fig.13.
Pour émuler le Cardboard vous pouvez vous servir de "Alt" et la souris pour faire bouger de la tête. Vous pouvez aussi vous servir de "ctrl" et la souris pour une rotation dans les deux sens.
Essayez maintenant de fixer le texte des yeux et comme par magie, il change de couleur.
Si vous voulez tester le rendu sur un téléphone, rendez-vous dans la prochaine section.

Spatial Audio :

Avec le nouveau SDK du cardboard, les développeurs n'ont plus à développer des audio listener des deux côtés (droite et gauche) afin de créer un son immersif. Maintenant il suffit d'ajouter le composant GvrAudioSource à votre GameObject. Si vous disposez d'un Audio source, veuillez le remplacer par ce dernier.
Vous pouvez également utiliser le composant GvrAudioRoom afin de simuler les effets audio les plus sophistiqués dans une chambre. Pour ce faire il faut ajouter le préfab GvrAudioRoom à votre scène. Ça consiste en un GameObject avec un composant GvrAudioRoom attaché à ce dernier. Mais si vous avez déjà votre GameObject, vous pouvez directement atta-

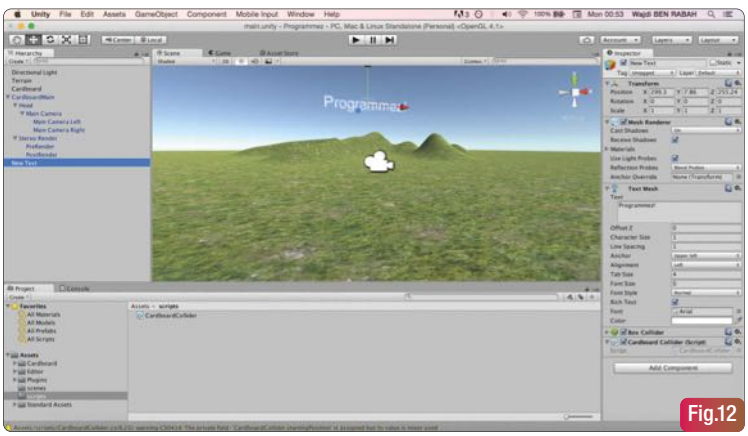


Fig.12

Fig.13

cher le composant à celui-ci. Vous pouvez régler les propriétés sonores, ou le type de chambre, la réflectivité ou beaucoup d'autres propriétés.
Si vous voulez l'ajouter dans votre projet, je vous invite à consulter cette page https://developers.google.com/vr/unity/spatial-audio .

Build et lancement sur un vrai device :

Je vais procéder au test sur un téléphone Android. Vous pouvez tester le rendu sur d'autres téléphones (ios ou autre) en respectant les dépendances de chaque plateforme.
Si vous êtes satisfait de ce rendu, vous pouvez procéder au « build ».
File ➔ « Build Settings » ou encore le raccourci : CMD SHIFT b sur Mac (CTRL SHIFT b sur PC).
Dans le menu « Scenes In Build » ajoutez votre scène en cliquant sur « Add open scene».

Exporter le projet pour Android :

- Choisissez « Android » sous la liste « plateforme » ;
- Cliquez sur « Switch Platform » ;
- Cliquez sur « Player Settings » ;
- Remplissez les informations suivantes :
  - Company Name : programmez
  - Product Name : VR Cardboard
  - Default icon : importez une image (drag & drop dans le dossier « assets » puis drag & drop pour définir une icône à votre jeu) (c'est optionnel).
- Sous le menu « Other Settings » remplacez « bundle identifier » par com.programmez.vrcardboard (c'est l'identifiant unique qui servira de différencier votre application sur le store) ;
- Sous le menu "resolution and presentation" -> "Orientation"-> Default orientation : "Landscape left" ; Fig.14.
- Sous le menu « Publishing Settings », si vous disposez d'une keystore Android, alors choisissez « browse keystore » et sélectionnez-la ;
- Sinon, choisissez « Create New Keystore » tapez un mot de passe et confirmer ;
- Key :
  - Create new key -> remplir le formulaire -> valider.
  - Si vous avez tout configuré, le bouton « Build and run » doit devenir fonctionnel.
- Branchez votre téléphone Android à votre Mac ;
- Cliquez sur le bouton « Build and run » ;
- Unity va vous demander de préciser le dossier du SDK Android. Choisissez donc l'emplacement adéquat.

Il vous demandera également un nom pour votre APK ; "Programmez" est ce qu'on a choisi.

Rendu final :

- Le visuel doit apparaître sur votre téléphone.
- Mettez-vous en mode « landscape » et enfilez le cardboard.

Problèmes potentiels :

Problème de mise à jour sdk Android : Fig.15.
Solution : Tout simplement mettre à jour le SDK Android.

Problème d'émulation :

Unity peut vous demander de brancher votre téléphone. Si vous l'aviez fait, essayez de changer de port USB ou de télécharger les pilotes de votre téléphone.
Si vous n'avez pas de téléphone, vous pouvez utiliser un émulateur comme "genymotion" (https://www.genymotion.com) par exemple afin de voir le rendu de votre scène.

Conclusion :

La technologie qui permet de créer des mondes virtuels progresse tous les jours, pour devenir de plus en plus accessible. Ceci est essentiellement possible grâce aux progrès dans le monde mobile. Pourtant, la route est encore longue : créer des univers très réalistes demande des ressources importantes et il existe encore beaucoup de limitations ou effets secondaires désagréables pour certains utilisateurs (vertiges, troubles de l'équilibres, mal de mer, etc.). D'ici vient l'initiative du groupe Google/Alphabet avec le nouveau projet DayDream présenté lors de la Google i/o édition 2016. Le géant du numérique souhaiterait offrir une expérience plus immersive que son successeur : le Google Cardboard, tout en misant toujours sur l'évolution des smartphones. Si Samsung VR repose sur la puissance des Galaxy S6 et le S7, Google supporterait les smartphones sous Android N. Google a donc pensé non seulement à un casque léger, pratique et très confortable, mais aussi à une expérience loin d'être passive, en présentant un petit contrôleur tactile qui sera notre vis à vis du monde virtuel. Le projet Daydream devrait être lancé d'ici fin de l'année 2016.

Références :

- https://developer.vuforia.com/library/articles/Solution/Integrating-Cardboard-to-the-ARVR-Sample
- https://www.udemy.com/vrcourse/learn/
- https://developers.google.com/cardboard/android/get-started
- http://www.goglasses.fr/
- http://www.vrs.org.uk/virtual-reality-games/future-expectations.html
- https://www.google.com/atap/project-tango/
- http://answers.unity3d.com/
- https://www.assetstore.unity3d.com/en/#!/content/6332

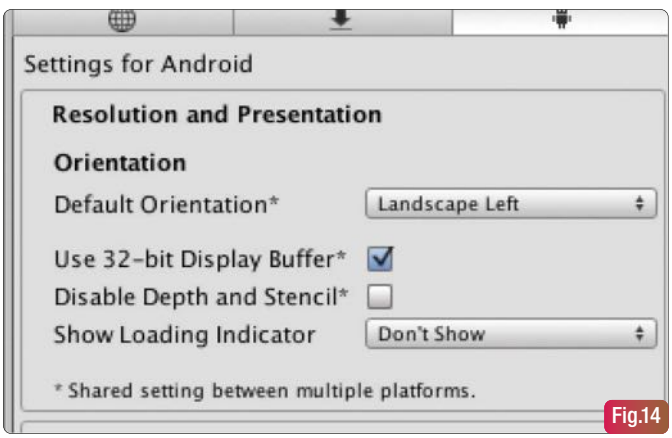


Fig.14

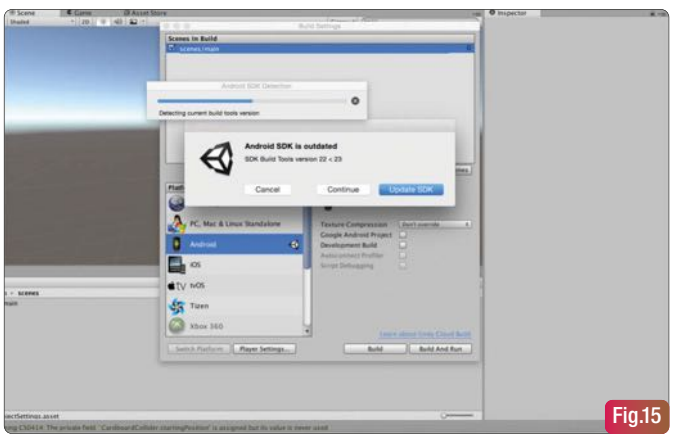


Fig.15

Abonnez-vous à programmez!

le magazine des développeurs

1 an ..... 49€ + 1€ = **50€***
11 numéros + un livre au choix

2 ans ..... 79€ + 1€ = **80€***
22 numéros + un livre au choix

Etudiant ..... 39€ + 1€ = **40€***
1 an - 11 numéros + un livre au choix

PDF ..... **30€***
1 an - 11 numéros + une vidéo au choix

Pour 1€ de + un livre numérique des Editions ENI au choix

valeur : 29,26 €

ou

valeur : 40,50 €

JavaScript
Développez un client Web en Full JavaScript

valeur : 29,99 €

ou

Applications mobiles multiplateformes
Technologie et contexte d'utilisation

valeur : 29,99 €

© Orion

Vous souhaitez abonner vos équipes ? Demandez nos tarifs dédiés* : redaction@programmez.com
(* à partir de 5 personnes)

(*) durée de l'offre du 31 mai au 30 septembre
Tarifs France métropolitaine

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex.

- ☐ ***Abonnement 1 an au magazine :*** 49 €
- ☐ ***Abonnement 2 ans au magazine :*** 79 €
- ☐ ***Abonnement étudiant 1 an au magazine :*** 39 €
  Photocopie de la carte d'étudiant à joindre
- ☐ ***Abonnement 1 an + Livre numérique*** ☐ ***AngularJS ou*** ☐ ***Git :*** 50 €
- ☐ ***Abonnement 2 ans + Livre numérique*** ☐ ***AngularJS ou*** ☐ ***Git :*** 80 €
- ☐ ***Abonnement étudiant 1 an + Livre numérique*** ☐ ***AngularJS ou*** ☐ ***Git :*** 40 €
  Photocopie de la carte d'étudiant à joindre

☐ M. ☐ Mme Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

email indispensable pour l'envoi d'informations relatives à votre abonnement

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !

☐ Je souhaite régler à réception de facture

** Tarifs France métropolitaine*

PROG 198

***Offre limitée**, valable jusqu'au 2 septembre 2016*

Développer pour le casque HoloLens

En janvier 2015, Microsoft annonçait lors d'une conférence de presse ses HoloLens. Cette paire de lunettes au design un peu étrange permet de se retrouver au beau milieu d'un film de science-fiction rempli d'hologrammes.

Jonathan ANTOINE
Consultant
Infinite Square
http://blogs.infinitesquare.com/b/jonathan

Le projet HoloLens, originellement appelé projet Baraboo, remonte à fin 2007. Il est issu des mêmes bureaux que les Kinect qui accompagnent nos consoles Xbox. Il s'agit donc d'un projet de longue haleine de Microsoft pour nous plonger dans le monde holographique. On ne parle plus, ni de réalité virtuelle, ni de réalité augmentée, mais de réalité mixte.

Les différents types de réalité

Il existe déjà depuis plusieurs années des casques (Occulus, HTC Vive, etc.) nous permettant de nous plonger dans une réalité virtuelle. Cette dernière nous coupe complètement du monde réel en nous plaçant dans un monde entièrement fictif. Il est difficile d'interagir avec les personnes autour de nous ou l'environnement qui nous entoure. Ces solutions sont de plus orientées *jeux-vidéos* et reliées à un PC (ou une console) afin de bénéficier de sa carte graphique pour un rendu 3D excellent. Les interactions utilisateurs seront faites avec un clavier/souris ou des contrôleurs dédiés à cet usage.

Google a fait le buzz en présentant les Google Glass il y a quelques années : des lunettes de réalité augmentée. Il s'agissait bien de périphériques mobiles mais les informations projetées à l'utilisateur restaient sur un seul plan, celui des verres des lunettes. L'utilisateur continue de voir le monde réel et peut interagir avec. Il est donc possible de placer des panneaux virtuels devant les yeux de l'utilisateur mais pas de créer d'hologrammes.

Les HoloLens proposent quant à elles de la réalité mixte : il est possible de placer des modèles 3D (que l'on appelle donc communément des hologrammes) au sein même du monde réel qui nous entoure et d'interagir avec. Les HoloLens sont dotés d'une technologie de cartographie de l'espace ultra-performante permettant de donner une consistance physique aux hologrammes : « je peux placer une sphère sur cette table sans qu'elle ne passe au travers ou bloquer son déplacement si une boîte de sucre d'Erstein est positionnée sur la table ». Un objet placé dans le monde réel aura ainsi une présence physique réelle et il est alors possible de tourner autour pour le découvrir sous tous ses aspects. Aussi, HoloLens est un PC complètement mobile tournant sous Windows 10 : il n'y a donc pas besoin d'être connecté à un autre PC pour l'utiliser. L'interaction avec les hologrammes se fait sans nécessiter aucun matériel particulier (plus sur ce sujet plus loin dans l'article).

HoloLens est à ce jour le premier outil mobile de réalité mixte permettant de placer des hologrammes dans le monde réel.

Le matériel

À ce jour, les HoloLens sont disponibles sous la forme d'un kit de développement haut de gamme. On ne parle donc pas encore du modèle grand public mais il s'agit quand même d'un bel objet. Fig.1.

Voici quelques informations sur sa configuration :

- Quad-Core à 1,04GHz 32bits

Fig.1

- 2 Go de RAM
- 64 Go de disque dur
- HPU spécifique pour effectuer le rendu graphique et traiter les informations en entrée
- Bluetooth, Wifi, micro USB
- Appareil vidéo et photo frontal
- Son HD Spatial : les hauts parleurs sont les « onglets rouges » sur chaque côté.
- Windows 10
- 579 grammes

Il s'agit donc bien d'un véritable PC multi-cœurs sur lequel il est possible de faire du développement sérieux même si l'on reste sur une configuration « mobile ». L'autonomie que nous avons pu constater sur nos paires d'HoloLens est d'environ 5 heures d'utilisation non-stop ce qui est en soit très suffisant ! La recharge se fait via un câble MicroUSB.

Les différents types d'interactions

Microsoft introduit dans les HoloLens le concept de Gaze Gesture Voice (GGV) pour permettre l'interaction de l'utilisateur.

Gaze - le regard de l'utilisateur

Il s'agit simplement de la direction dans laquelle regarde l'utilisateur, indiquée par l'orientation du casque. Un pointeur (cercle blanc personnalisable) permet à tout moment de centrer son regard sur un objet en particulier.

Gesture – des gestuelles prédéfinies

Les HoloLens (parenté à Kinect oblige) suivent le mouvement des mains dans son champ de vision (appelé gesture frame) et détectent leurs positions mais aussi leurs mouvements. Il existe 3 gestures qu'il n'est pas possible de compléter pour le moment :

- AirTap : on positionne une main devant soi et l'on effectue un mouvement de « clap de cinéma » devant soi. Il s'agit de l'équivalent d'un click.
- Drag : le même mouvement que précédemment mais en maintenant ses doigts appuyés pour les déplacer de la même manière que si l'on souhaitait attraper un objet virtuel.
- Bloom : serrer son point et ouvrir les doigts en les remontant vers le haut. Il s'agit de l'équivalent du bouton Windows qui permet de revenir dans le ... menu démarrer – scène principale de l'HoloLens. Fig.2.

Au premier abord il est difficile d'associer notre regard à la gestuelle (on a tendance à essayer d'effectuer la gestuelle sur l'objet en question) mais l'usage est finalement acquis assez rapidement.

Il est aussi à noter que les gestuelles multi-mains ne sont pas encore possibles – pas de redimensionnement en étirant les objets pour le moment donc !

Voice – la voix

Windows 10 oblige, Cortana est intégrée de base dans HoloLens ; il est possible de l'utiliser très simplement dans vos applications. Les APIs nécessaires pour enregistrer des commandes vocales sont disponibles et certaines commandes de base sont en place. Il est ainsi possible de regarder un HoloGramme et de lui demander de nous regarder (« face me ») ou de s'agrandir (« make bigger »). Fig.3.

Accessoires

Finalement, lorsque ces moyens de contrôles ne sont plus suffisants (ou que la peur du ridicule vous accable) ,il est possible de brancher en Bluetooth claviers et souris mais aussi le « Clicker », accessoire créé spécialement pour l'HoloLens. Fig.4.

Les différents types de développements possibles

Maintenant que nous en savons un peu plus sur cette magnifique machine, la question que se posent tous les développeurs est : que peut-on coder ? Pour faire rapide : il est possible de déployer toutes vos applications universelles (UWP) sur les HoloLens ! Les choix disponibles sont alors :

- Application UWP « classiques » développées en C#/C++ et XAML ou encore en HTML5/CSS3 à l'aide de WinJS ;
- Application UWP 3D avec Direct X ;
- Application UWP 3D avec Unity.

Les outils nécessaires

Afin de développer une application HoloLens certains outils sont nécessaires :

- Visual Studio 2015 Update 2. La version Community de Visual Studio gratuite est tout à fait suffisante ;
- Au minimum, la version 1.3.1 des outils de développement UWP ;
- Le SDK UWP dans sa version 10.0.10586 minimum ;
- Optionnellement, une version d'Unity 5.4 (encore en beta) spéciale pour cibler HoloLens (avant une intégration dans la branche principale) peut être trouvée sur http://aka.ms/HoloLensUnity .

Microsoft met aussi à disposition un émulateur lorsque l'on n'a pas les lunettes sous la main qui est téléchargeable sur le http://dev.windows.com. Fig.5. Le déploiement sur un périphérique HoloLens est des plus simple, un rêve de développeur :

- Depuis Visual Studio en remote via le WIFI (un peu long certaines fois) ;
- Depuis Visual Studio en déploiement direct via USB ;
- Et même depuis un portail Web (hébergé sur les lunettes) en uploadant les appx ! Bien sûr dans cette configuration, on se prive du débugger Visual Studio.

Applications UWP classiques

Toute application UWP développée en XAML/C#/C++ ou HTML5/CSS3 fonctionne sur les HoloLens. Précisons que cela inclut bien sûr les applications utilisant Cordova ou Xamarin. Fig.6.

Les applications sont exécutées sous la forme de projection 2D dans une zone d'un ratio de 16:9 avec une taille en pixel effectif de 853x840 pixels. Depuis la première mise à jour, l'utilisateur peut désormais décider d'agrandir verticalement les applications. Il est donc important de vérifier le comportement visuel de votre application à cette taille qui reste assez atypique du fait de son manque de hauteur. Un bon moyen de tester l'affichage et d'assigner une hauteur dans le designer d'interface uniquement :

```
<Page x:Class="AppHoloLens.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    d:DesignHeight="480"
    d:DesignWidth="853"
    mc:Ignorable="d">

</Page>
```

L'intégralité du framework graphique est déjà pensée pour fonctionner à l'aide des composantes GVV :

- Les boutons *s'allument* lorsque l'utilisateur les regarde ;
- Le scroll fonctionne lorsque l'utilisateur effectue un drag ;
- L'air-tap déclenche bien sûr un clic.

Certaines fonctionnalités du SDK ne sont pas encore disponibles pour le moment (faute de temps sans-doute !) mais le seront, je pense, dans une version ultérieure. Attention donc si des fonctionnalités cruciales de votre application reposent sur :

- Toasts : pas possible d'afficher une notification Toast, il faudra donc


Fig.2

Fig.3

Fig.4

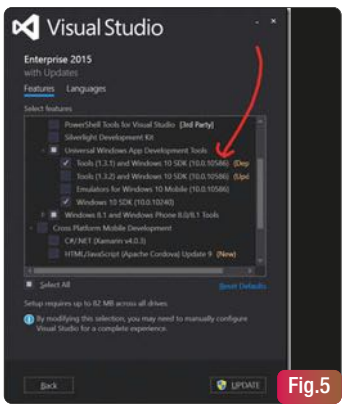


Fig.5

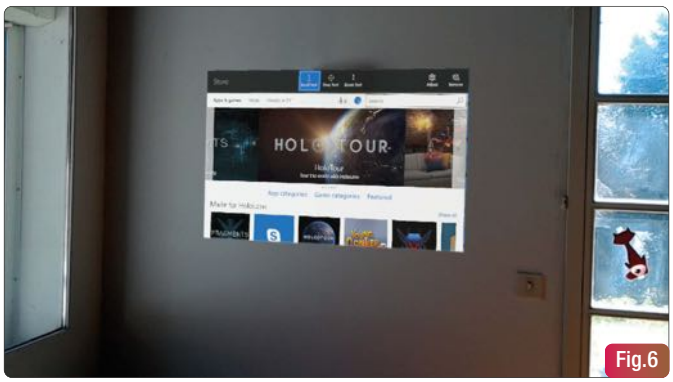


Fig.6

passer par un autre mécanisme pour afficher des messages à vos utilisateurs ;
- Partage : le partage (le contrat et l'interface de partage) n'est pas encore disponible. L'application Photo/Vidéo implémente cependant sa propre interface ;
- AppService : ce fabuleux moyen de faire communiquer les applications entre-elles n'est pas encore disponible (mais le sera prochainement).

La plus grosse limitation consiste au fait de ne pas pouvoir créer d'hologrammes 3D à partir de code XAML. Il faudra pour cela passer à une vue 3D Unity ou Direct X (la technique complète est décrite sur le blog Infinite Square). Finalement, afin de rendre votre application disponible sur le store HoloLens, il faudra penser à sélectionner cette plateforme lors de votre soumission sur le Store.

Applications UWP avec DirectX

Une application UWP utilisant DirectX dans sa version 11 peut être déployée sur HoloLens de manière assez traditionnelle. Pour cela on développera directement en C++/CX ou pourra aussi utiliser (non, non ce n'est pas de la triche !) le wrapper managé (C#) SharpDX. Un bon point de départ consiste à utiliser le template par défaut de Visual Studio qui permet l'affichage très rapide (60 fois par seconde !) d'un cube multicolore. Attention, vous n'aurez cependant pas de moteur de jeu et cela sera à vous de construire entièrement celui-ci. Cela reste une très bonne solution si vous possédez un moteur de jeu existant que vous souhaitez utiliser sur HoloLens. Fig.7.

En partant sur ce chemin il est possible de créer des hologrammes 3D qui ne seront au final *que* des modèles affichés dans une scène 3D. Les APIs nous permettent de récupérer simplement la direction ciblée par chaque œil mais surtout de bénéficier d'un algorithme pré-codé de stabilisation de l'image : pratique pour ne pas avoir d'affichage tremblotant !

La principale différence vis-à-vis d'un développement classique est au niveau des Shaders pour une utilisation sur les HoloLens. En effet, il est nécessaire de prendre en compte l'affichage stéréoscopique produisant des matrices de projection spécifiques à chaque œil.


Fig.7


Fig.8

Applications UWP avec Unity

Unity, le moteur et l'éditeur d'applications 3D multiplateformes (plus d'une vingtaine supportée !) a travaillé main dans la main avec Microsoft pour proposer une version compatible avec les HoloLens. Cette version spécifique permet d'utiliser les APIs spécifiques au monde holographique mais aussi de créer un player UWP compatible avec un déploiement sur HoloLens. Le sérieux de la solution est tel que Skype pour HoloLens est conçu avec Unity ! Fig.8.

Au niveau du développement cela reste, comme pour DirectX, assez classique : Unity s'assure que la caméra principale conserve à tout moment la position de la tête de l'utilisateur et la direction de son regard. Il ne reste alors qu'à créer une scène 3D qui sera reproduite dans votre espace réel grâce aux HoloLens. La seule configuration initiale à avoir est de choisir une couleur unie noire pour le fond de la caméra (clear flag). Cela permet de profiter de l'affichage additif spécifique aux HoloLens : le noir ne peut pas être ajouté (toutes ses composantes couleur sont à zéro) et il apparait donc comme transparent. Fig.9.

Afin de vous montrer, à titre d'exemple, car cet article n'est qu'une introduction, la simplicité d'utilisation des APIs, voici le code permettant de détecter que l'utilisateur vient d'effectuer un air-tap :

```
void Awake()
{
  NavigationRecognizer = new GestureRecognizer();

  NavigationRecognizer.SetRecognizableGestures(GestureSettings.Tap);

  // 2.b: Register for the TappedEvent with the NavigationRecognizer_TappedEvent function.
  NavigationRecognizer.TappedEvent += NavigationRecognizer_TappedEvent;
}

private void NavigationRecognizer_TappedEvent(InteractionSourceKind source, int tapCount, Ray ray)
{
  Debug.Log("Air tap !");
}
```

Afin de gagner en productivité, Microsoft propose un ensemble d'helpers sous la forme d'un package Unity. Il suffit d'aller télécharger le dossier Assets sur le repository GitHub consacré (https://github.com/Microsoft/HoloToolkit-Unity) pour bénéficier :
- D'un curseur personnalisé qui se pose sur les objets de la scène 3D et affiche un état personnalisé ;
- D'un script permettant de trouver l'objet regardé par l'utilisateur ;

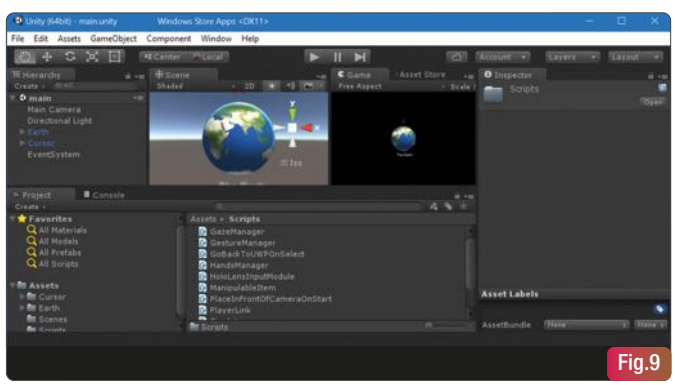


Fig.9

- Des scripts nécessaires à la réalisation de session partagée entre plusieurs utilisateurs d'HoloLens ;
- Des scripts/shaders nécessaires à l'affichage de l'environnement sous la forme d'un maillage. L'environnement détecté étant fourni sous la forme d'un Mesh dynamique (il se met à jour lorsque le code le demande) ;
- Des scripts nécessaires à la détection de plans (sols, plafond, murs, tables, etc.) dans l'environnement ;
- Des outils/scripts nécessaires à la mise en place de son spatial. Très pratique pour indiquer la position d'un objet lorsque l'utilisateur ne l'a pas dans son champ de vision.

Voici par exemple l'affichage d'un cube et du maillage de l'environnement détecté par les HoloLens : Fig.10.

Que permettent les APIs ?

L'objectif de cet article n'est pas de rentrer dans le détail des possibilités mais voici les classes principales que propose le SDK à date :

- HolographicCamera : classe de base pour effectuer le rendu via DirectX (une par œil) ;
- HolographicFramePrediction : classe de base pour récupérer les informations de vision de l'utilisateur après stabilisation ;
- SpatialAnchor : représentation d'une position physique dans le monde réel et des informations permettant de la retrouver. Cette classe peut être transférée via réseau ou stockée sur disque en utilisant la classe SpatialAnchorStore ;
- SpatialSurfaceMesh : représentation de l'environnement sous la forme de Mesh dynamique. À utiliser en conjonction de la classe SpatialSurfaceObserver ;
- SpatialGestureRecognizer : la classe permettant d'obtenir les gestuelles effectuées par l'utilisateur (air-tap, drag, etc.).

Les différentes APIs sont dans des namespaces différents du SDK UWP, mais elles sont toutes dans des « sous-namespaces » nommé « Spatial » comme par exemple : Windows.UI.Input.Spatial

Petite astuce, pour savoir si l'on est en cours d'exécution sur HoloLens dans une application UWP 2D, il suffit de regarder si le SpatialLocator est null :

```
private bool? _isOnHoloLens;

public bool IsOnHoloLens()
{
    if (!_isOnHoloLens.HasValue)
    {
        var locator = Windows.Perception.Spatial.SpatialLocator.GetDefault();
        _isOnHoloLens = locator != null;
    }

    return _isOnHoloLens.Value;
}
```


Fig.10

Comment aller plus loin

Avant d'attendre nos prochains articles sur le sujet (vous pouvez aussi suivre nos aventures sur le blog Infinite Square), Microsoft met déjà beaucoup de ressources à notre disposition :

- Un site complet et dédié à présenter HoloLens et la technologie sous-jacente : https://www.microsoft.com/microsoft-hololens/ ;
- Holographic Academy : des tutoriaux très bien construits vous accompagnent pour la création de votre première application « Holo World » mais aussi sur des sujets complexes (détection de plans, etc.) : https://developer.microsoft.com/en-us/windows/holographic/academy ;
- Le code source du toolkit Unity est disponible sur GitHub : très bon endroit pour savoir comment fonctionnent les choses : https://github.com/Microsoft/HoloToolkit-Unity ;
- Microsoft a construit une application très aboutie, Galaxy Explorer, qui est disponible elle aussi en open-source sur GitHub : https://github.com/Microsoft/GalaxyExplorer .

Quelques limitations matérielles du kit de développeur

Il s'agit d'un kit de développement, et il est important de connaître quelques limitations de celui-ci sachant qu'elles seront corrigées, on l'espère fortement, dans les futures versions :

- Le champ de vision est limité : bien que cela ne soit pas forcément très gênant, on ne peut pas voir la scène en entier mais uniquement une zone rectangulaire ;
- Pas de scènes complexes : nous restons sur un périphérique mobile et il faut faire avec moins de ressources qu'un PC de gamer !
- Le noir apparaît comme transparent ;
- Pas de « multi-touch » actuellement : les gestures à une seule main uniquement sont disponibles pour le moment. Fig.11.

Conclusion

En recevant un kit de développement on peut s'attendre à avoir une sorte de prototype pas complétement abouti... Ce n'est pas du tout le cas avec les HoloLens ! Le matériel est de haute qualité, la réalité augmentée fonctionne plus que bien et les outils de développements (SDK et éditeur) sont déjà fonctionnels sans problème particulier. Microsoft nous propose avec ces lunettes de faire un pas vers le futur en imaginant les scénarios de demain et il le fait bien ! Fig.12.


Fig.12

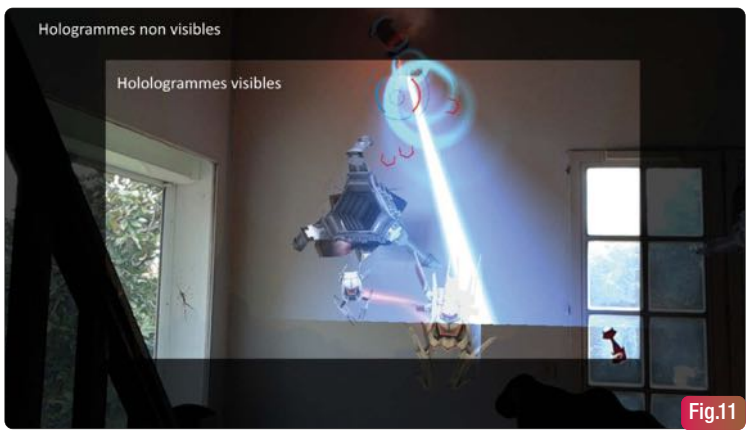


Fig.11

Réaliser un jeu 3D avec Babylon.JS

Aujourd'hui dans le domaine des jeux vidéo jouables dans un navigateur Web, deux technologies de développement règnent en maître : Flash et Unity3D. Ces deux outils ont un point commun ; leurs performances liées à un accès au GPU viennent de l'utilisation d'un logiciel tiers (plugin), l'accélération matérielle étant inaccessible directement depuis un navigateur.

Julian Chenard
Chef de projet chez **Keyveo**
et membre de l'équipe Babylon.js
www.keyveo.com
@Temechon

Julien Moreau-Mathis
développeur 3D et membre de l'équipe Babylon.js -
Alternant chez
Microsoft
@Luaacro

Fig.1 *Le jeu final*

Depuis 2009, ce n'est plus tout à fait exact : le Kronos Group a standardisé une interface de programmation 3D, basée sur OpenGL ES 2.0 : le WebGL. Celui-ci est en train de s'imposer dans le jeu vidéo et sur le Web en général : Flash se meurt petit à petit suite à son abandon par les majors de l'IT, et Unity3D travaille sur sa migration en WebGL. Cependant, si cette API permet d'afficher de la 3D (et de la 2D) en ayant accès au GPU, son utilisation est relativement complexe : calcul vectoriels et matriciels, création de shaders, gestion de la mémoire... Bref, beaucoup de maths et de techniques d'optimisations sont indispensables à connaître pour avoir un résultat satisfaisant. Une librairie permettant d'abstraire tout ce contenu bas niveau serait l'idéal pour créer un jeu rapidement... Et c'est exactement ce que propose Babylon.js !

Babylon.js est un moteur 3D gratuit et open-source créé par quatre français (David Catuhe et David Rousset, Michel Rousseau et Pierre Lagarde) en 2013. Il est développé en TypeScript, est hébergé sur Github (https://github.com/BabylonJS/Babylon.js) et est clairement orienté gaming de par ses fonctionnalités. Le concept du moteur est "simplicité - efficacité" : beaucoup de propriétés sont très simples à utiliser, bien que leur utilisation en profondeur soit possible. Les lead-développeurs et la communauté travaillent d'arrache-pied pour que le moteur soit toujours plus stable, performant et compétitif, et la communauté (regroupée sur le forum officiel - http://www.html5gamedevs.com/forum/16-babylonjs/) est très réactive. Dans ces pages, vous apprendrez à créer un jeu complet en 3D avec Babylon.js, de A à Z. Ce développement permettra d'aborder beaucoup de notions liées à la 3D, à Babylon.js et à la création de jeu en général. Notre jeu comportera une vraie scène 3D, ainsi qu'un gameplay relativement simple : le joueur doit parcourir la ville, et éliminer les 20 cibles disséminées un peu partout en le moins de temps possible. Une version jouable est disponible ici : http://pixelcodr.com/games/babylonjs-fps

Nous verrons d'abord comment créer une scène basique avec Babylon.js, puis nous appliquerons ces premiers éléments de départ à la scène de notre jeu. Nous verrons ensuite comment implémenter les contrôles du joueur, et nous ferons un focus important sur les matériaux, concept très vaste qui mérite qu'on s'y attarde. Nous verrons enfin le développement du gameplay, l'intégration de sons, les animations...

Dans le but d'exploiter au maximum ce dossier, vous devez avoir quelques connaissances en TypeScript (si vous connaissez déjà JavaScript, cela ne devrait pas être très compliqué). Les sites importants à connaître sont la documentation officielle (la bible Babylon.js, très bien faite et très complète : doc.babylonjs.com), et le forum officiel (html5gamedevs.com/forum/16-babylonjs). Dernier point non négligeable : ce jeu sera basé sur la version 2.3 de Babylon.js, dernière version stable du moteur à la date d'écriture de cet article. Si vous êtes prêt, mettez-vous un petit fond musical et faites-vous couler un bon café : c'est parti ! **Fig.1.**

Hello World en 3D

Le "Hello World", dans le monde de la 3D, c'est un cube qui tourne sur lui-même. Rien de très sexy, mais cela va nous permettre de voir quelques fonctions basiques de Babylon.js. Nous avons besoin pour cela d'une page HTML très simple, composée d'un seul élément important : le canvas. index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8"/>
  <title>Hello World in Babylon.js</title>
  <script src="http://cdn.babylonjs.com/2-3/babylon.js"></script>
  <script src="helloWorld.js"></script>
  <style>
   html, body {
     margin  : 0;
     padding  : 0;
     overflow : hidden;
     width : 100%;
     height  : 100%;
   }
   #gameCanvas {
     width  : 100%;
     height  : 100%;
   }
  </style>
</head>
<body>
  <canvas id="gameCanvas"></canvas>
</body>
</html>
```

Cette page très simple ne contient que l'élément canvas (qui va servir à afficher le contenu 3D) et quelques règles de style CSS. La version de Babylon.js est récupérée depuis le CDN officiel, mais peut-être également téléchargée en local. Rien de très compliqué ici, car toute la magie de notre première application se trouve dans le fichier Javascript "helloWorld.js".
helloWorld.js

```
window.addEventListener("DOMContentLoaded", function() {

 // Création de l'engine
 var canvas = document.getElementById("gameCanvas");
 var engine = new BABYLON.Engine(canvas, true);

 // Création de la scène
 var scene  = new BABYLON.Scene(engine);

 // Création de la caméra
 var camera = new BABYLON.FreeCamera("camera", new BABYLON.Vector3(0,0,-5), scene);
 camera.setTarget(BABYLON.Vector3.Zero());

 // Création d'une lumière
 var light = new BABYLON.HemisphericLight("light", new BABYLON.Vector3(0,1,0), scene);

 // Le cube !
 var cube  = BABYLON.Mesh.CreateBox("myBox", 1, scene);

 // La rotation
 cube.registerBeforeRender(function() {
  cube.rotation.y += 0.01;
 });

 // La boucle de rendu
 engine.runRenderLoop(function() {
  scene.render();
 });

}, false);
```

Vous remarquez que l'ensemble des éléments est créé à la réception d'un évènement DOMContentLoaded : WebGL a besoin d'un canvas créé, initialisé et prêt à être utilisé. Une fois celui-ci récupéré par son identifiant, un objet BABYLON.Engine peut être créé. C'est cet objet qui va assurer la communication avec le GPU (et s'occuper de toute la partie mathématique pour vous, de façon totalement transparente).
A partir de l'engine, une scène 3D est créée (l'objet BABYLON.Scene) pour contenir tous les éléments 3D : objets, lumières, matériaux... Dans le cas de Babylon.js, la classe Scene est un graphe de scène : c'est cet objet qui va stocker de façon logique l'ensemble des éléments composant la scène affichée. Le point de vue du joueur est représenté par une caméra : dans cet exemple, nous créons une instance de FreeCamera qui joue le rôle d'une vue à la première personne. De la même façon, une lumière et un cube (*box* en anglais) sont ajoutés à la scène 3D. Enfin, il est gentiment demandé à l'engine d'afficher la scène résultat à 60 images par secondes s'il-vous-plait. Rappelons à cette occasion le principe de base d'affichage d'un jeu vidéo : pour avoir un résultat fluide et naturel, le moteur 3D doit "calculer" (créer) toutes les images composant l'animation du jeu. Ce calcul est lancé dans une boucle de rendu qui tourne en permanence, avec pour objectif idéal de créer 60 images par secondes. Plus la scène 3D est complexe (beaucoup de polygones, beaucoup de textures...), et plus le

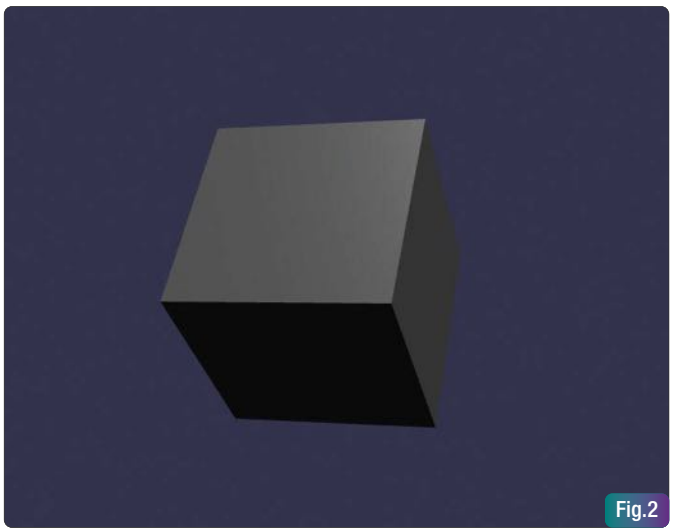


Fig.2

temps de calcul d'une image est long : un sentiment de ralentissement se créé (c'est le lag).
Babylon.js dispose de plusieurs fonctions permettant de créer ce type d'objets de façon très simple : nous verrons plus tard qu'un jeu vidéo demande des ressources graphiques plus poussées que de simples cubes, mais que leur intégration ne demande pas beaucoup plus de code.
Chaque objet 3D (mesh en anglais) peut être manipulé par trois attributs : position, rotation, scaling. Ces attributs ont chacun des propriétés x, y et z qui modifient l'objet en conséquence dans le monde 3D. Changer la rotation selon l'axe Y du cube dans la boucle de rendu a donc pour effet de le faire tourner d'un tout petit angle 60 fois par secondes.
Sans plus de suspens, ce (très) simple code affiche donc le résultat suivant, qui ne présente que peu d'intérêt si ce n'est deux choses : d'abord montrer la simplicité d'utilisation de Babylon.js, et ensuite préparer les bases de notre jeu ! **Fig.2.**

Initialisation de la scène de départ

Commençons par créer la ville en 3 dimensions, qui constituera l'environnement de notre jeu dans lequel le joueur évoluera. Pour trouver des ressources graphiques, il y a plusieurs possibilités, de la moins chère à la plus onéreuse : les faire soi-même en utilisant un logiciel de création 3D (3DSMax, Blender, Maya, Sketchup...), utiliser des modèles 3D gratuits disponibles sur internet (yobi3D.net, turbosquid.com, tf3dm.com,...), acheter des modèles 3D déjà tout faits, et enfin demander à un infographiste professionnel de réaliser vos propres modèles sur mesure. Ici, un ami infographiste a créé pour nous les modèles que nous allons utiliser dans ce dossier. Vous pouvez les retrouver sur Github (https://github.com/Temechon/Babylon.js-FPS) et vous en servir pour réaliser ce jeu. Ces ressources ont toutes été réalisées avec le logiciel 3DSMax. Pour les utiliser avec Babylon.js, il faut passer par un outil permettant d'exporter ces ressources : Babylon.js en propose un pour les logiciels Blender, Unity3D, Clara.io et 3DSMax.
Dans le cas de 3DSMax, il suffit de copier des fichiers dans le répertoire d'installation du logiciel et d'utiliser la commande "Exporter" nouvellement ajoutée (les exporteurs de Blender et Unity ne sont pas plus compliqués à installer, et Clara.io fournit de façon native l'export vers Babylon). Les objets, matériaux, caméras, lumières et animations seront enregistrés dans un fichier dont l'extension est ".babylon". Cela nous permet côté code de récupérer ces informations (et d'autres, décrites dans la documentation officielle) et ainsi de gérer leurs interactions. **Fig.3.**
La scène fournie contient : un environnement, un ensemble de 20 cibles dispersées dans la ville, une caméra, une lumière hémisphérique et un modèle 3D représentant l'arme du joueur. Nous allons dans un premier


Fig.3

Notre environnement sous 3DSMax

temps créer une classe Game, qui se chargera d'initialiser, de créer et de gérer le jeu : d'abord en chargeant la scène 3D réalisée sous 3DSMax, puis en créant les différents éléments d'interactions du jeu (joueur, ennemis, PNJ,...). Le constructeur de la classe Game va juste créer l'engine, et l'objet scène vu précédemment sera automatiquement créé par Babylon.js au chargement du fichier map.babylon. Game.ts

```
constructor(canvasId:string) {
    let canvas : HTMLCanvasElement = <HTMLCanvasElement> document.getElement
ById(canvasId);
    this.engine      = new BABYLON.Engine(canvas, true);
    // Contient l'ensemble des assets du jeu autres que l'environnement
    this.assets      = [];
    // La scène 3D du jeu
    this.scene       = null;
    // On resize le jeu en fonction de la taille de la fenetre
    window.addEventListener("resize", () => {
      this.engine.resize();
    });
    this.run();
  }
```

La méthode run va charger la scène, initialiser le jeu et lancer la boucle de rendu. Pour charger la scène, nous utilisons la méthode "SceneLoader. Load", qui prend en paramètre le chemin vers le dossier du fichier à charger, puis son nom, et qui nous fournit une nouvelle scène configurée et prête à l'emploi :

```
private run() {
BABYLON.SceneLoader.Load("assets/","map.babylon", this.engine, (scene) => {
  // BABYLON.SceneLoader.Load créé une scène pour nous
  this.scene = scene;
// Initialisation de la scène 3D
        this.initScene();
  // Maintenant, il suffit de lancer la boucle de rendu une fois que
  // la scène est complètement initialisée (textures, modèles 3D, etc.)
  this.scene.executeWhenReady(() => {
    // Boucle de rendu
    this.engine.runRenderLoop(() => {
      // On rend (dessine) la scène dans le canvas
      this.scene.render();
    });
});
// Une fois la scène créée, initialisons simplement le jeu
this.initGame();
});
}
```

Le chargement de la scène est réalisé de façon asynchrone : pendant que le fichier babylon est récupéré (via un appel AJAX), le framework ajoute automatiquement un écran de chargement par défaut. Celui-ci est d'ailleurs complètement configurable si besoin (comme l'ensemble des éléments du framework d'ailleurs). Une fois la scène chargée, deux méthodes vont prendre le relais : initScene et initGame. La fonction initScene va nous permettre de modifier quelques paramètres relatifs à la scène : d'abord, nous modifierons les paramètres de caméra pour pouvoir se déplacer en appuyant sur les touches Z,Q,S,D. Nous ajouterons ensuite quelques composants pour renforcer l'immersion du joueur : un ciel (une skybox en anglais), des ombres, la gestion des collisions, et le plein écran.

```
private initScene() {
    // On récupère la caméra créée dans 3DSMax
    let cam = <BABYLON.FreeCamera> this.scene.activeCamera;
    // On active le contrôle clavier - souris
    cam.attachControl(this.engine.getRenderingCanvas());
    // On associe les touches Z, Q, S et D aux mouvements de caméra.
    cam.keysUp.push(90);
    cam.keysDown.push(83);
    cam.keysLeft.push(81);
    cam.keysRight.push(68);

    // On active le plein écran avec la fonctionnalité prévue à cet effet
    let setFullScreen = () => {
      this.engine.switchFullscreen(true);
      window.removeEventListener('click', setFullScreen);
    }
    window.addEventListener('click', setFullScreen);

    // On ajoute un ciel à notre scène
    var skybox = BABYLON.Mesh.CreateSphere("skyBox", 32, 1000.0, this.scene);
    skybox.position.y = 50;
    var skyboxMaterial = new BABYLON.StandardMaterial("skyBox", this.scene);
    skyboxMaterial.backFaceCulling = false;
   skyboxMaterial.reflectionTexture = new BABYLON.CubeTexture("skybox/TropicalSunny
Day", this.scene);
    skyboxMaterial.reflectionTexture.coordinatesMode = BABYLON.Texture.SKYBOX_MODE;
    skyboxMaterial.diffuseColor = new BABYLON.Color3(0, 0, 0);
    skyboxMaterial.specularColor = new BABYLON.Color3(0, 0, 0);
    skyboxMaterial.disableLighting = true;
    skybox.material = skyboxMaterial;

    // Gestion des ombres
    //...

    // Gestion des collisions
    //...
  }
```

La création du ciel a l'air relativement complexe, mais beaucoup de choses sont en fait effectuées de façon très simple : une énorme sphère (représentant le ciel et entourant la scène entière) est créée, et un matériau est ensuite appliqué à la sphère. Le concept de matériau est très important dans Babylon.js : si l'objet 3D représente la géométrie et la forme d'un objet, son matériau représente sa couleur. Plus loin dans ce dossier, nous verrons comment créer et utiliser des matériaux (un peu de patience :)).
Un autre élément renforçant le réalisme et l'immersion du joueur est l'ajout d'ombres dans notre scène. Pour cela, il suffit de rajouter une lumière

directionnelle (qui va simuler un soleil virtuel) et un générateur d'ombres (shadow generator en anglais). Ajoutez ce bloc à la méthode initScene:

```
// Récupérer la lumière hémisphérique créée à partir de 3ds Max
let defaultLight = this.scene.getLightByName('Default light');
defaultLight.intensity = 0.5;
// On créé une lumière directionnelle
let dir = new BABYLON.DirectionalLight('dirLight', new BABYLON.Vector3(-0.5,-1,-0.5), this.scene);
dir.position = new BABYLON.Vector3(40, 60, 40);
// On créé le générateur d'ombres, avec une qualité de 1024 (qualité suffisante)
let shadowGenerator = new BABYLON.ShadowGenerator(1024, dir);
// On active le mode qui rend les ombres plus réelles
shadowGenerator.useBlurVarianceShadowMap = true;
// Application des ombres aux maisons et arbres
this.scene.meshes.forEach((m) => {
  if (m.name.indexOf('house') !== -1 || m.name.indexOf('arbre') !== -1) {
    shadowGenerator.getShadowMap().renderList.push(m);
    m.receiveShadows = false;
  } else {
    m.receiveShadows = true;
  }
});
```

Essayons de décomposer ce code. Comme la scène 3D est un graphe de scène, nous pouvons récupérer n'importe quel objet 3D à partir de son nom (ici défini dans 3DSMax). Nous récupérons ainsi la lumière par défaut pour en baisser l'intensité. Nous créons ensuite une lumière directionnelle (dont les rayons suivent une direction descendante légèrement oblique), puis un générateur d'ombre : il va prendre en paramètre cette lumière directionnelle et calculer une "shadow map" (dont la taille est le deuxième paramètre dans le constructeur). Dans notre exemple, nous ajoutons tous les objets maisons et arbres à cette shadow map : uniquement ceux-là auront une ombre projetée, tandis que les autres objets ne feront qu'afficher ces ombres. Attention, les ombres projetées sont relativement coûteuses à calculer, et dépendent évidemment de la taille de la shadow map. Il est préférable de privilégier des ombres pré-calculées sur les textures (on parle de texture "bakées", à faire réaliser par votre infographiste préféré) ou à défaut d'utiliser une petite taille de shadow map. Fig.4.

Les collisions

Dans notre jeu, nous allons également utiliser un mécanisme déjà inclus : il s'agit des collisions. Autrement dit, comment configurer notre caméra pour qu'elle ne passe pas à travers les murs. Babylon.js nous propose un système complètement intégré et transparent qui va nous permettre de configurer ces collisions :

```
// On peut récupérer la caméra active depuis la scène facilement
let camera = this.scene.activeCamera;
// On active les collisions sur la caméra
camera.checkCollisions = true;
// On définit la distance minimale entre la caméra et les objets de la scène
camera.ellipsoid = new BABYLON.Vector3(1, 1, 1);
// On active la gravité sur la caméra
camera.applyGravity = true;
// On paramètre la gravité sur la scène
scene.gravity = new BABYLON.Vector3(0, -0.91, 0);
// Pour finir, on active les collisions sur tous les objets de la scène
this.scene.meshes.forEach((m) => {
  m.checkCollisions = true;
});
```


Fig.4

Et voilà notre ville en temps réel dans Babylon.js !

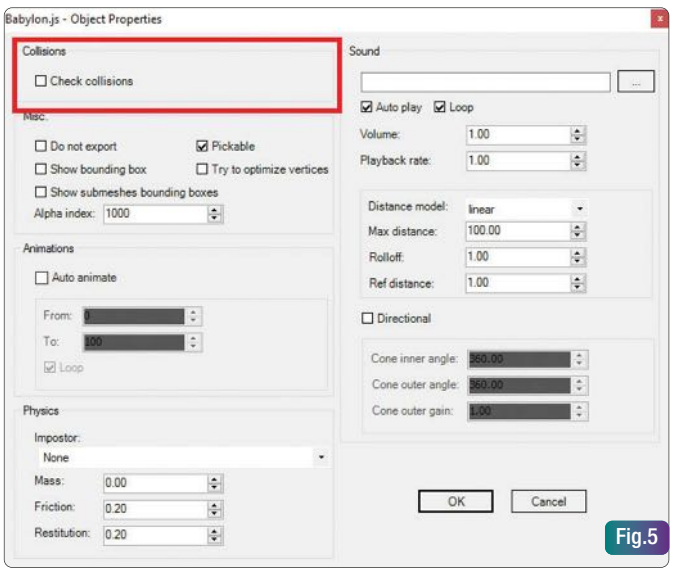


Fig.5

Gérer les collisions est parfois complexe dans un jeu vidéo 3D. Ici, Babylon.js nous permet de les configurer en activant simplement les collisions sur les objets (la propriété "checkCollisions"). Par la suite arrivent des propriétés supplémentaires qui permettent d'affiner au mieux les collisions en fonction de la scène, comme l'ellipsoid. L'ellipsoid permet de définir, sur les 3 axes (X, Y ,Z), quelle est la distance minimum avant d'appliquer la collision et ainsi bloquer la caméra pour qu'elle ne passe pas à travers l'objet.

Pour finir, le système de collisions de Babylon.js nous permet d'appliquer simplement la gravité sur la caméra et ainsi, par exemple, de très facilement monter ou descendre des escaliers.

Heureusement, cette partie peut être configurée sans une seule ligne de code par votre artiste préféré dans son outil préféré (ici 3DS Max). Quand votre artiste configure la scène 3D, il peut avoir accès à des propriétés propres à Babylon.js qui se sont immiscées via le plugin précédemment installé. Dans 3DS Max, il suffit de faire un clic droit sur une caméra, un mesh ou une light, pour voir apparaître un petit menu "Babylon..." qui permet de paramétrer ces propriétés spéciales Babylon.js, y compris les collisions :).

Les propriétés sur un mesh : Fig.5.

Les propriétés sur une caméra : Fig.6.

Et pour finir les propriétés sur la scène (clic droit dans le vide) pour configurer la gravité : Fig.7.

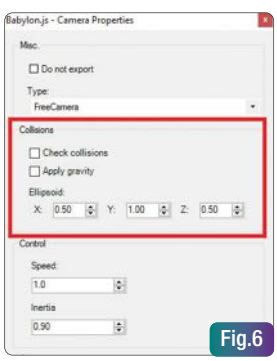


Fig.6

Fig.7

C'est-à-dire que tous ces paramètres peuvent être contrôlés facilement par votre artiste 3D et nul besoin d'écrire une quelconque ligne pour configurer les collisions côté développeur.
Si vous n'êtes pas graphiste, pas de problème! Ces propriétés peuvent être gérées entièrement dans le code, comme le montre le code suivant :

```
// Scene
scene.gravity = new BABYLON.Vector3(0, -9.81, 0);
scene.collisionsEnabled = true;
// Camera
camera.checkCollisions = true;
camera.ellipsoid = new BABYLON.Vector3(1.5,1,1.5);
camera.applyGravity = true;
// Objets
object.checkCollisions = true;
```

Initalisation du jeu

La fonction initGame va initialiser toutes les parties liées à la scène. Typiquement, gérer l'arme du joueur, voire même initialiser les personnages et leur intelligence associée. Par exemple, configurons l'arme pour qu'elle suive la caméra, comme un jeu FPS, caméra qui est notre point de vue sur la scène 3D :

```
private initGame() {
  // Récupérons l'arme avant de pouvoir la configurer
let blaster = this.scene.getMeshByName("blaster");

// Pour placer un mesh dans l'espace, il suffit de modifier sa propriété
// ".position", qui est une instance de BABYLON.Vector3(x, y, z)
blaster.position = new BABYLON.Vector3(0.05, -0.1, 0.4);

// Finalement, pour que l'arme dépende des coordonnées de notre caméra,
// il suffit de définir le parent de l'arme comme étant la caméra.
blaster.parent = this.scene.activeCamera;
}
```

Le blaster sera ainsi lié à la caméra : si la caméra bouge, le blaster bougera avec. De même, la position de l'enfant dans le monde 3D est calculée à partir de la position du parent. Le (petit) vecteur (0.05,-0.1,0.4) est ainsi la distance entre la caméra (le parent) et l'arme (l'enfant).
Toute la logique de notre gameplay sera placée dans cette classe "Game" (dans le fichier "Game.ts"), où typiquement la fonction "initGame" initialisera toutes les entités nécessaires au gameplay, et où la fonction "run", elle, chargera la scène, et traitera plus tard de la logique des personnages, de la gestion des tir, etc.

Les matériaux

Comme vous pouvez le constater dans ces premières parties, mettre en place un jeu vidéo 3D, ou tout simplement une scène 3D avec Babylon.js, est loin de faire appel à des connaissances mystiques en mathématiques réservées aux spécialistes de la 3D. Avec ces quelques lignes de code et la simplicité de Babylon.js, nous pouvons nous déplacer, gérer la gravité, ne pas traverser les murs, afficher un blaster qui suit la caméra, prendre en compte l'environnement comme le ciel et même gérer les ombres en temps réel !
Maintenant, focalisons-nous sur la fameuse notion de matériaux, notion importante directement liée à la 3D (et pas nécessairement compliquée) pour ainsi mieux comprendre comment les objets sont dessinés dans le canvas. Fig.8.

Dans cette petite partie, nous allons créer une simple sphère pour représenter le soleil, auquel nous ajouterons une couleur jaune. Dans la méthode initGame, commençons par créer un mesh de type sphère :

```
var soleil = BABYLON.Mesh.CreateSphere('soleil', 16, 10, this.scene);
```

Comme dit plus haut, nous pouvons manipuler la position de la sphère en lui affectant un objet de type Vector3.

```
soleil.position = new BABYLON.Vector3(0, 100, 0);
```

De la même façon que l'attribut position, les objets 3D disposent d'un attribut de type 'material', permettant au système de lui affecter une couleur. Créons un matériau de couleur jaune pour notre soleil :

```
let soleilMaterial = new BABYLON.StandardMaterial('soleilMaterial', this.scene);
soleil.material = soleilMaterial;
```

Comme les objets 3D, un matériau est un noeud du graphe de scène (c'est pour cela qu'il prend en paramètre de constructeur l'objet Scene) : il peut ainsi être affecté à plusieurs objets sans se dupliquer, l'objectif étant d'avoir le moins de références de noeuds en mémoire, pour gagner en performance.
Vous avez surement remarqué que créer ce matériau ne change en rien à la couleur de la sphère... et c'est tout à fait normal ! Il suffit simplement de lui ajouter une couleur...
Les couleurs sont représentées par deux types d'objets : BABYLON.Color3 et BABYLON.Color4. Chaque couleur prend donc en paramètre trois valeurs : rouge, vert et bleu. Le dernier paramètre d'une Color4 est la valeur alpha, représentant la transparence. Ces couleurs peuvent être affectées à plusieurs paramètres d'un matériau. Les paramètres intéressants sont : diffuseColor (la couleur d'un objet éclairé), emissiveColor (la couleur émise par l'objet), et specularColor (la couleur de la lumière renvoyée par l'objet). Dans notre exemple, il suffit d'affecter la couleur émissive du soleil à Color3.Yellow() et de spécifier la couleur spéculaire à Color3.Black(), et le tour est joué !

```
soleilMaterial.emissiveColor = BABYLON.Color3.Yellow();
soleilMaterial.specularColor = BABYLON.Color3.Black();
```

Toutes les propriétés des matériaux pourraient faire l'objet d'un dossier

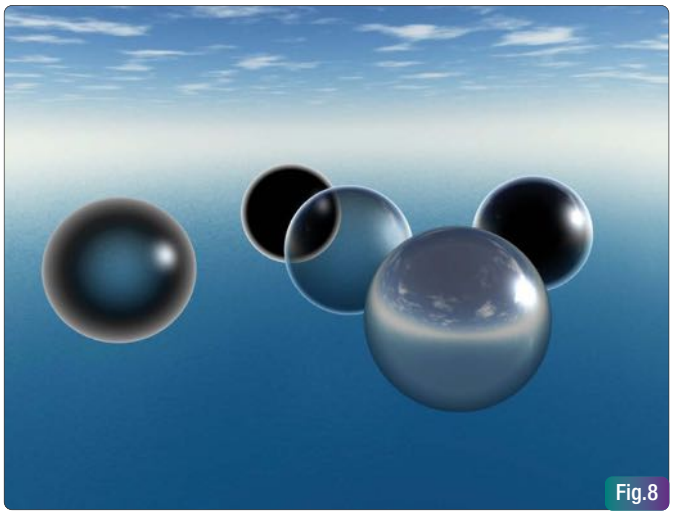


Fig.8

Exemple de matériaux réalisés avec Babylon.js

complet tant ce domaine est complet dans Babylon.js (textures, réflexion, réfraction, paramètres Fresnel, PBR, shaders...). Pour en savoir plus, je vous invite à consulter le forum et la documentation officielle. Fig.9.

Les animations

Dans cette partie, nous allons animer les cibles pour qu'elles tournent sur elles-mêmes. Pour cela, il y a plusieurs façons. La première est de les animer directement dans 3DSMax ou Blender, puis de les exporter dans votre scène. La deuxième façon est de créer un objet animation, qui va prendre en compte un ensemble de clés et de valeurs : entre chaque clé de l'animation, le système va interpoler et affecter une bonne valeur à l'objet animé. Pour cela, il suffit de créer un objet BABYLON.Animation, comme ceci :

```
let animation = new BABYLON.Animation(name, property, fps, dataType, loopMode);
```

Le paramètre property correspond à la propriété de l'objet à animer (dans notre cas, il peut s'agir de "rotation.y" par exemple).
Il faut ensuite spécifier l'ensemble des valeurs clé, et les affecter à l'animation. Il s'agit d'un simple tableau clé-valeur :

```
let keys = [{frame:0, value:0}, {frame:100, value:2*Math.PI}];
animation.setKeys(keys);
```

Enfin, il faut affecter l'animation à l'objet.

```
target.animations.push(animation);
```

Enfin, il existe un troisième mode d'animation, via la boucle de rendu. Cela correspond par exemple à incrémenter une valeur toutes les frames - idéal pour faire tourner toutes les cibles !
Il faut commencer par récupérer toutes les cibles de l'environnement 3D venant de 3DSMax :

```
// Active toutes les cibles de la scène
    this.scene.meshes.forEach((m) => {
      if (m.name.indexOf('target') !== -1) { // Si le nom correspond à 'target'
        m.isPickable = true; // Pour pouvoir les détruire par la suite
       m.rotationQuaternion = null; // On reset la rotation utilisé par le moteur physique
        this.targets.push(m);
      }
    });
```


Fig.9

Exemple de matériaux PBR (Physically Based Rendering)

Ensuite, il suffit d'incrémenter la rotation selon l'axe Y de chaque cible à chaque frame :

```
// Rotation infinie de toutes les cibles
 this.scene.registerBeforeRender(() => {
    this.targets.forEach((target) => {
       target.rotation.y += 0.1*this.scene.getAnimationRatio();
    });
});
```

L'objet Scene comprend la méthode getAnimationRatio, qui permet d'ajuster la vitesse de l'animation en fonction des capacités matérielles du joueur (pour la rendre indépendante des FPS).
Les animations vont permettre d'ajouter un peu de dynamisme dans notre jeu relativement statique. Attaquons nous maintenant à la partie destruction des cibles :) Fig.10.

Le gameplay

Comme notre jeu sera en plein écran et que le pointeur de la souris sera caché par le système, il nous faut rajouter un viseur. En attendant la prochaine grosse fonctionnalité de Babylon.js (version 2.4) qui permettra d'ajouter une interface 2D à notre jeu, il nous faut faire notre viseur autrement. Une solution simple et rapide est de l'ajouter en CSS et HTML : dans notre cas, comme ce viseur a juste un but informatif pour le joueur, c'est parfait.

```
<img id="gunsight" src="assets/viseur.png" />

#gunsight {
  position:absolute;
  top:50%;
  left:50%;
  margin-top:-37px;
  margin-left:-37px;
}
```

Une fois en plein écran, et sans indication de pointeur de souris, c'est ce viseur qui indiquera au joueur où il tirera. Ajoutons maintenant de l'interactivité en deux étapes. L'objet Scene (et son attribut onPointerDown) permet d'ajouter un comportement particulier quand le joueur clique (ou touche) la scène : il suffirait de récupérer l'objet sélectionné, et de le supprimer s'il s'agit d'une cible. Essayons cela :

```
// Active le tir
 this.scene.onPointerDown = (evt, pr) => {
```

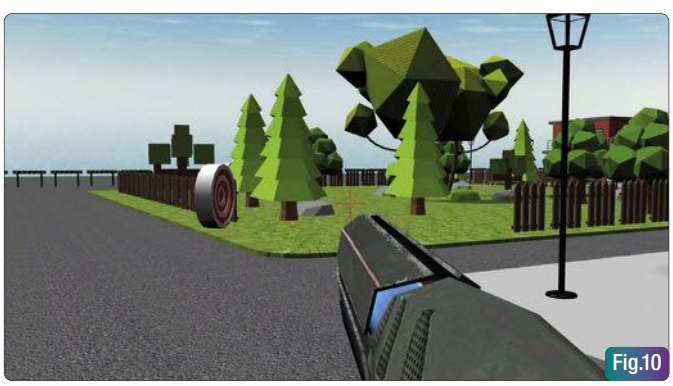


Fig.10

Les cibles tournent sur elles-mêmes

```
    if (pr.hit) {
      this.destroyTarget(pr.pickedMesh);
    }
}
```

La fonction de callback onPointerDown va donc simuler un clic dans la scène à l'endroit de la souris, et renvoie deux paramètres à notre comportement spécifique : l'évènement souris retourné par le navigateur, et un objet de type PickingInfo. Cet objet a deux attributs intéressants : hit, qui est mis à true si l'utilisateur a sélectionné un objet, et pickedMesh qui correspond à l'objet sélectionné. Ici, c'est presque parfait : il ne reste plus qu'à prendre en compte la position du pointeur. Modifions donc notre fonction :

```
this.scene.onPointerDown = (evt, pr) => {
        var width = this.scene.getEngine().getRenderWidth();
        var height = this.scene.getEngine().getRenderHeight();
        var pickInfo = this.scene.pick(width/2, height/2);
        if (pickInfo.hit) {
          this.destroyTarget(pickInfo.pickedMesh);
        }
      }
```

Il suffit d'utiliser la méthode scene.pick en lui spécifiant des coordonnées écran (ici la moitié en hauteur et en largeur). Bien sûr, nous pouvons ajouter ici d'autres détails : le recul de l'arme (une simple animation de la rotation de l'objet), de la dispersion, des particules... Vous pouvez laisser libre cours à votre imagination ! Pour détruire un objet, il faut appeler la méthode dispose : de ce fait, la méthode destroyTarget est très simple :

```
private destroyTarget(target) {
      var index = this.targets.indexOf(target);
      if (index > -1) {
        this.targets.splice(index, 1);
        target.dispose();
        // Mise à jour de l'interface
        this.targetGui.innerHTML = String(this.targets.length);
        if (this.targets.length == 0) {
          // Le jeu est fini !
        }
      }
  }
```

La partie son

Pour ajouter un peu d'ambiance, rien de tel que de la musique et des effets sonores. Babylon.js comprend un moteur de son basé sur la spécification Web Audio (sans aucun fallback : si votre navigateur n'est pas compatible avec le tag <audio>, aucun son ne sera joué).
Dans cette courte partie, nous allons ajouter un son quand le joueur utilise son arme. Et comme d'habitude avec Babylon.js, il suffit d'une ligne de code dans la méthode initScene :

```
var gunshot = new BABYLON.Sound("gunshot", "assets/sounds/shot.wav", this.scene, null,
 { loop: false, autoplay: false });
this.assets['gunshot'] = gunshot;
```

Il suffit ensuite de jouer le son quand on le souhaite :

```
this.assets['gunshot'].play();
```

Le moteur de son est très complet : il est possible de jouer de la musique d'ambiance, des sons spatialisés et des sons directionnels (via des fichiers distincts ou des soundtracks), pour ensuite les attacher à des objets du monde 3D. Une fois encore, la documentation sera votre meilleure alliée pour aller plus loin.

Fig.8

La mini-carte

Il manque une mini-carte à notre FPS : créons-en une en utilisant en deuxième caméra, qui sera en mode orthographique. Ce mode permet de représenter les objets 3D en 2 dimensions, et donnera à notre scène un aspect 2D : parfait pour la mini-carte !

```
let mm = new BABYLON.FreeCamera("minimap", new BABYLON.Vector3(0,100,0), this.scene);
mm.setTarget(new BABYLON.Vector3(0,0,0));
mm.mode = BABYLON.Camera.ORTHOGRAPHIC_CAMERA;
mm.orthoLeft = mm.orthoBottom = -500/2
mm.orthoRight = mm.orthoTop =  500/2;
mm.rotation.x = Math.PI/2;
```

Maintenant que la minimap est créée, il faut lui spécifier un viewport : c'est la partie de l'écran où notre caméra sera affichée.
Ici, on la veut dans le coin supérieur droit de l'écran : les différents paramètres doivent être entre 0 et 1.

```
var xstart = 0.8, ystart = 0.75;
var width = 0.99-xstart, height = 1-ystart;
mm.viewport = new BABYLON.Viewport(xstart, ystart, width, height);
```

Enfin, il faut ajouter un modèle 3D (une simple sphère rouge) représentant le joueur, et jouer avec les layerMask pour que celle-ci ne soit pas visible à la caméra principale. Les layerMask sont une simple opération AND logique entre une caméra et un mesh : si le résultat est différent de 0, le mesh est visible à la caméra.

```
mm.layerMask = 1;
this.scene.activeCamera.layerMask = 2;
playerRepresentation.layerMask = 1; // Ce mesh ne sera pas visible sur la caméra principale
```

Conclusion

Comme vous avez pu le voir, la simplicité est vraiment au coeur de la philosophie de Babylon.js. La plupart des fonctionnalités importantes (création de la scène, initialisation du jeu, animations, collisions, matériaux...) sont très facilement utilisables, mais aussi très complètes : il est évidemment possible pour le développeur perfectionniste de paramétrer et configurer l'ensemble pour répondre exactement à ses besoins.
Ce dossier ne permet pas de lister l'ensemble des possibilités du framework : il faudrait plusieurs centaines de pages pour cela. Si le sujet vous intéresse, je vous encourage à étudier la documentation et le forum, ainsi que les livres à ce sujet.

Console Arduboy : créer un jeu vidéo

Il fait beau, il fait chaud et pour certains ça sent déjà bon le sable chaud... On y est enfin, c'est l'été ! Durant cette période propice à la détente, une boisson fraîche dans une main et un clavier dans l'autre, quoi de plus fun que de s'écrire un petit jeu vidéo ! Et, tant qu'on y est, pourquoi ne pas l'écrire sur l'un des objets les plus cool du moment : l'Arduboy !

Jérôme Perrot
Initgraph

La carte de visite la plus originale du monde, c'est ainsi que Kevin Bates, le créateur de l'Arduboy aime définir sa petite console. Et question originalité, il n'a pas fait les choses à moitié : il a tout simplement créé une console de jeux de la taille d'une carte de crédit (**Fig.1**). Si on voulait résumer un peu « grossièrement » l'Arduboy, on pourrait dire que c'est un micro-contrôleur Arduino avec un écran OLED, un interrupteur, une batterie et 6 boutons. Si on enlève le côté miniaturisation, ce n'est pas très loin de la vérité et, Arduino étant une plateforme ouverte, c'est tout naturellement que cette petite « Gameboy » commence à se faire une place de choix dans le monde des Makers et du retro-gaming. L'Arduboy est donc l'objet idéal pour créer notre jeu et par la même occasion découvrir le développement sur Arduino si vous ne connaissez pas.

L'environnement de développement

Même si il existe aujourd'hui plusieurs IDE pour Arduino, nous nous intéresserons ici à Arduino IDE qui est l'outil officiel.

Pour installer cet environnement, rien de plus simple, il suffit de se rendre à l'adresse https://www.arduino.cc/en/Main/Software et de récupérer le programme d'installation correspondant à votre système d'exploitation. Au niveau de l'installation, il n'y a rien de compliqué, « c'est du Suivant-Suivant » comme on le dit souvent en informatique. Une fois l'IDE installé, lorsque nous l'exécutons, nous voyons apparaître une page avec 2 fonctions **setup()** et **loop()** à l'intérieur (**Fig.2**). Comme leur nom le laisse suggérer, la fonction **setup()** est exécutée une seule fois au lancement du programme et la fonction **loop()** est une boucle sans fin. Vous l'avez peut-être remarqué sur la barre de titre, un programme sous Arduino se nomme « Sketch » (ou « Croquis » en Français), ne soyez pas donc pas surpris de rencontrer cette nomenclature par la suite. La présentation de l'IDE effectuée, il est temps de le paramétrer pour notre Arduboy. Commençons par allumer et relier notre Arduboy à notre ordinateur à l'aide d'un câble micro-USB afin de l'installer sur notre système d'exploitation. Bonne surprise, l'installation se fait automatiquement et il n'est pas nécessaire d'ajouter un pilote manuellement. Retournons à notre IDE et indiquons-lui que nous allons compiler pour un circuit Arduino de type « Leonardo » à l'aide du menu **Outils -> Type de carte -> Arduino Leonardo**, l'Arduboy étant bâtit autour de cette architecture. Notre périphérique ayant été préalablement installé sur notre système d'exploitation, il nous est maintenant possible de le sélectionner

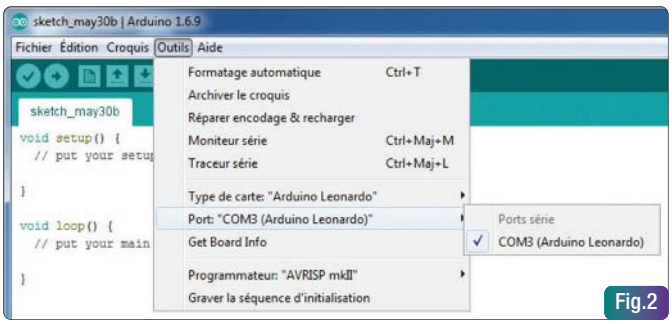


Fig.2

Fig.1

via le menu **Outils -> Port** (**Fig.2**). Pour terminer le paramétrage de l'IDE, il nous reste à installer la bibliothèque de développement de l'Arduboy. La méthode la plus simple pour intégrer cette dernière est de passer par le gestionnaire de bibliothèque accessible via le menu **Croquis -> Inclure une bibliothèque -> Gérer les bibliothèques**. Une simple recherche du mot clé « arduboy » en haut à droite du gestionnaire permet de trouver et d'installer la bibliothèque en question. L'installation de cette bibliothèque effectuée, nous pouvons dorénavant l'ajouter à notre croquis via le menu **Croquis -> Inclure une bibliothèque -> Arduboy**. Effectivement cette action fait apparaître le code **#include <Arduboy.h>** en haut de notre source.

Hello World

Quand on veut découvrir une plateforme, une étape est presque indispensable : l'affichage d'un « Hello World » ! Mais avant d'écrire nos premières lignes de code, il faut que l'on voie comment fonctionne l'affichage sur l'Arduboy et sur la plupart des périphériques vidéo. Afin d'éviter un scintillement disgracieux lorsque nous affichons des animations sur l'écran, rien n'est écrit directement sur ce dernier. A la place, tout est écrit dans un tampon de la taille de l'écran que l'on nomme généralement « Screen Buffer ». Ce tampon est ensuite affiché de manière synchronisée avec la fréquence de rafraîchissement de l'écran. Je ne vais pas rentrer dans les détails ici, vous pourrez trouver plus d'infos à ce sujet sur Internet si vous le souhaitez. Cet éclaircissement effectué, ajoutons le code ci-dessous à notre fonction **setup()** :

```
Arduboy arduboy;

void setup() {

arduboy.begin();

arduboy.clear();

arduboy.setCursor(10, 10);
arduboy.print("Hello World ! »);

arduboy.display();
}
```

Remarquez les fonctions **clear()** et **display()**. Conformément à ce que nous venons de dire, la première efface le « Screen Buffer » et la seconde

l'affiche. Il est également important de noter que la bibliothèque doit être initialisée via la fonction **begin()** avant d'être utilisée.

Cliquons maintenant sur **Croquis -> Vérifier/Compiler** afin de contrôler notre code avant de l'envoyer sur notre machine et observons le résultat (**Fig.3**). Visiblement tout s'est bien passé car nous ne voyons pas d'horribles messages orange. Attardons-nous quand même un petit peu sur ce que nous dit le message en blanc : il s'agit en fait de l'état de la mémoire de notre console de jeux. Nous remarquons qu'il existe 2 types de mémoire : l'une appelée « espace de stockage de programmes » d'une taille de 28 672 octets et l'autre appelée tout simplement « mémoire dynamique » d'une taille de 2 560 octets. Non vous ne rêvez pas, la mémoire dynamique est bel et bien la RAM dont nous disposons pour nos variables, il va falloir faire attention à ce que nous faisons. L'espace de stockage de programmes est quant à lui de la mémoire flash, nous pourrons y stocker des données en lecture seule comme par exemple des images. Nous voilà averti, il est temps d'envoyer notre programme sur notre périphérique en cliquant sur **Croquis -> Téléverser.**

Eureka ! Nous voyons un beau « Hello World » sur l'écran ! Nous venons d'écrire notre premier programme sur l'Arduboy !

Un « Hello World » c'est bien mais un jeu c'est beaucoup mieux !

Vous vous en doutez, nous n'allons pas écrire le dernier « Call of Duty » mais un petit jeu beaucoup plus simple pour commencer. Je me suis demandé quel serait le meilleur jeu pour cela et je me suis dit qu'un jeu de type « Snake » ferait bien l'affaire même si, rassurez-vous, l'Arduboy peut aller beaucoup plus loin en matière de jeux vidéo.

Pour ceux qui se demandent encore ce qu'est un jeu de type Snake (vraiment ?), rappelez-vous le jeu qui connut un grand succès sur les téléphones Nokia, je suis sûr que ça vous revient maintenant ! Rappelons quand même le principe de ce jeu :

- De la nourriture est placée aléatoirement sur l'écran ;
- Le joueur dirige un serpent vers cette nourriture pour gagner des points ;
- A chaque fois que le serpent mange de la nourriture, il gagne un point mais en contrepartie il s'allonge et devient plus rapide ;
- Si le serpent touche le bord de l'écran ou sa queue, la partie est perdue.

A la vue de ce petit cahier des charges, essayons de définir l'architecture que nous pouvons adopter pour modéliser tout cela.

- Nous voyons 2 acteurs principaux : le serpent et la nourriture, nous pouvons donc déjà imaginer une classe nommée **Snake** et une classe nommée **Food** ;
- De la nourriture est placée aléatoirement sur l'écran : on pourrait par conséquent utiliser une méthode **generate()** qui générerait les positions **x** et **y** de cette dernière ;
- Le serpent va manger de la nourriture : nous devrons donc savoir si le serpent touche cette dernière en utilisant par exemple une méthode **checkCollisionWithFood()** ;
- Le joueur gagne des points quand le serpent mange de la nourriture : nous devrons donc stocker le nombre de repas mangés par ce dernier pour pouvoir afficher un score. Une variable **foodEaten** pourrait faire l'affaire pour cela ;
- La partie est perdue si le serpent touche les bords de l'écran ou lui-même : on peut donc imaginer deux méthodes **checkCollisionWithBorder()** et **checkCollisionWithItself()** ;
- Le joueur dirige un serpent à l'écran : il faudra donc mémoriser dans quelle direction va ce dernier et à quelle position il se trouve. Nous pourrions utiliser des variables **x**, **y** et **direction** pour cela. Nous pouvons également prévoir d'ajouter une méthode **move()** dans le but de gérer ces 3 variables.

Vu qu'il faudra initialiser et afficher tous les éléments que nous venons de voir, il faudra prévoir une méthode **init()** et **draw()** pour chaque classe.

Avant de passer à la modélisation de nos classes, réfléchissons un peu à l'élément essentiel de notre jeu : comment faire avancer notre serpent ? En guise de solution nous pourrions assembler notre serpent à l'aide de petits blocs que nous pourrions ajouter ou supprimer pour ajuster sa taille ou le faire avancer. En effet, si nous supprimons le dernier bloc de sa queue et que nous ajoutons un bloc à sa tête, nous aurons l'illusion que le serpent se déplace. Astucieux non ?

Puisqu'il va falloir positionner un ensemble de blocs à l'écran pour représenter notre serpent, nous pourrons donc utiliser un tableau nommé **parts** où chaque élément sera un bloc contenant les coordonnées **x** et **y** de sa position. Ce bloc pourra être un objet d'une classe qu'on peut nommer **SnakePart** et être initialisé via un constructeur **SnakePart(x, y)** où **x** et **y** sont les coordonnées de sa position.

Quand on parle de tableaux sous Arduino, il faut savoir qu'il est recommandé d'utiliser des tableaux statiques et non dynamiques. Notre tableau **parts** devra donc être initialisé avec une taille fixe qu'on peut par exemple fixer à 250. Nous pourrons stocker cette valeur dans une constante **MAX_SNAKE_PARTS** afin d'améliorer la lecture de nos calculs.

Mais alors, comment supprimer un élément ou en ajouter un si le tableau est statique ? La réponse est simple, nous ne sommes pas obligés de le faire. A la place nous pourrons utiliser une variable **partsStart** représentant l'élément du tableau contenant la dernière partie de la queue du serpent et une variable **partsCount** contenant le nombre de blocs que ce dernier contient. Pour faire avancer notre serpent, il nous suffira simplement d'ajouter la nouvelle position de sa tête à l'emplacement **partsStart + partsCount** et d'augmenter la valeur de **partsStart** de 1 pour prendre en compte la nouvelle position du dernier élément de sa queue. Par contre, qu'est ce qui se passe si la valeur **partsStart** ou la somme **partsStart + partsCount** atteint la fin du tableau ? C'est un cas que nous devons prendre en compte, il faut donc que nous indiquions que si on atteint la valeur 250 (le tableau commence à l'élément 0 et se termine à l'élément 249), il faut recommencer à partir de l'élément 0. Nous pourrions utiliser un simple **if** pour cela mais une astuce bien connue des programmeurs consiste à utiliser un modulo pour ce genre d'opérations. Si vous avez bien suivi le raisonnement (félicitation), le déplacement de notre serpent peut se résumer finalement à ces deux lignes de code :

```
parts[(partsStart + partsCount) % MAX_SNAKE_PARTS] = SnakePart(x, y);
partsStart = (partsStart + 1) % MAX_SNAKE_PARTS;
```

Puisque qu'un bon dessin vaut mieux qu'un long discours, j'ai résumé le déplacement du serpent dans un schéma (**Fig.4**).

Simple finalement, non ? Je vous rassure, c'est la partie la plus délicate à comprendre. Nous avons donc maintenant tous les éléments pour créer

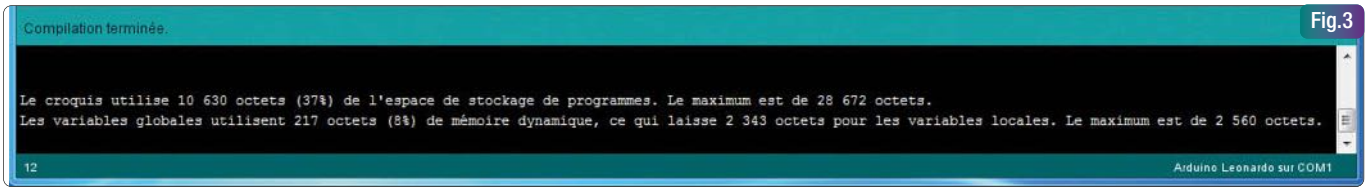


Fig.3

nos classes. Pour plus de clarté, notre projet Arduino peut être organisé de la manière suivante :

- **ArduSnake.ino** : croquis Arduino pour notre jeu. Ce dernier contiendra l'objet global **arduboy**, un objet global **snake** issu de la classe **Snake** et les fameuses fonctions **setup()** et **loop()** ;
- **Constants.h** : fichier contenant les constantes ;
- **Snake.h** : déclaration des classes **SnakePart** et **Snake** ;
- **Snake.cpp** : méthodes des classes **SnakePart** et **Snake** ;
- **Food.h** : déclaration de la classe **Food** ;
- **Food.cpp** : méthodes de la classe **Food**.

Maintenant que nous avons le contenant, ajoutons-y le contenu en commençant par créer les 2 classes de notre serpent :

```
class SnakePart {

 public:

  uint8_t x;
  uint8_t y;

  SnakePart();
  SnakePart(uint8_t positionX, uint8_t positionY);
};
```

```
class Snake {

 public:

  uint8_t x;
  uint8_t y;
  uint8_t direction;
  Food food;
  uint8_t foodEaten;
  bool isDead;

  Snake();
  void init(Arduboy *adb);
  void move();
  void draw();

 private:

  uint8_t partsCount;
  uint8_t partsStart;
  SnakePart parts[MAX_SNAKE_PARTS];
  Arduboy *arduboy;

  bool checkCollisionWithFood();
  bool checkCollisionWithBorder();
  bool checkCollisionWithItself();
};
```

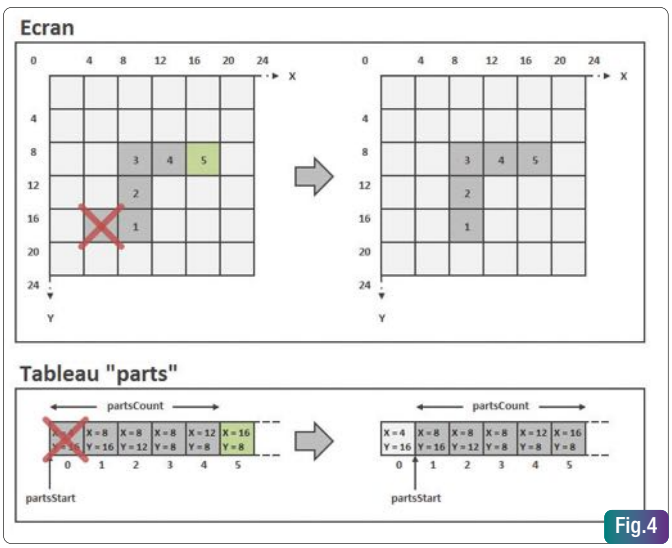


Fig.4

Afin d'améliorer la lisibilité du code et dans un soucis de gestion de la mémoire, vous remarquerez que j'ai utilisé un type uint8_t au lieu d'un traditionnel unsigned char ou unsigned byte. Remarquez également le paramètre Arduboy *adb de la méthode init() de la classe Snake. En effet, vu que l'objet arduboy est global et déclaré au sein du croquis Arduino, il sera en dehors de toute classe, nous avons donc besoin de connaître son emplacement mémoire pour pouvoir l'utiliser au sein de notre classe Snake.

```
class Food {

 public:

  uint8_t x;
  uint8_t y;

  Food();
  void generate();
  void init(Arduboy *adb);
  void draw();

 private:

  Arduboy *arduboy;
};
```

Nos classes maintenant créées, concentrons nous maintenant sur le croquis principal de notre jeu. La première fonction qui sera appelée par notre programme est la fonction **setup()**, elle se contente simplement d'initialiser la bibliothèque de l'Arduboy et notre objet **snake** issu de la classe **Snake** :

```
void setup() {

 arduboy.begin();
 arduboy.setFrameRate(1);

 snake.init(&arduboy);
}
```

La fonction **setFrameRate()** mérite une petite explication. Elle permet simplement de définir combien de fois par seconde nous souhaitons mettre à jour l'affichage des éléments. Une valeur de 1 va donc entraîner une mise à jour toutes les secondes. Cette fonction ne s'utilise pas seule, il faudra lui associer l'expression **if (!arduboy.nextFrame()) return;** au sein de la fonction **loop()** pour pouvoir maintenir la bonne fréquence. Il est important de noter qu'à l'heure actuelle le timing de la méthode **setFrameRate()** comporte quelques bugs, il faut donc tenir compte de ce petit désagrément lorsqu'on utilise cette fonction même si cela risque d'être corrigé rapidement. Comme nous venons de le voir, la fonction **setup()** appelle la méthode **init()** de la classe Snake. Cette méthode se contente simplement d'initialiser les variables de sa classe :

```
void Snake::init(Arduboy *adb) {
...
 direction = SNAKEDIR_RIGHT;

 partsCount = 5;
 partsStart = 0;
 parts[0] = SnakePart(56, 32);
 ...
 parts[4] = SnakePart(72, 32);
...
}
```

Le serpent est ainsi initialisé par défaut avec 5 blocs et la variable direction est initialisée avec une constante **SNAKEDIR_RIGHT** pour plus de lisibilité. Nous devrons donc ajouter une constante pour chaque direction que prendra le serpent dans le fichier **Constants.h**. Passons maintenant au coeur de notre jeu, la méthode **loop()**, nous pouvons l'écrire pour l'instant comme ceci :

```
void loop() {

 // Si le serpent est mort, on ne fait rien pour l'instant
 if (snake.isDead) {}

 // Si le serpent est en vie, on le fait avancer
 else {

  if (!arduboy.nextFrame()) return;

  snake.move();
  if (snake.isDead) return;

  arduboy.clear();

  // Affichage d'une bordure
  arduboy.drawRect(2, 2, 124, 60, 1);

  snake.draw();

  arduboy.display();
 }
}
```

Rien de bien compliqué en somme. Si le serpent est mort, pour l'instant on ne fait rien et dans le cas contraire on le fait juste avancer via la méthode **move()**. On le dessine ensuite grâce à la méthode **draw()** de notre classe **Snake**. Remarquez la fonction **drawRect()** de la bibliothèque Arduboy qui nous permet de dessiner un rectangle qui simulera une bordure.
Avant de présenter les méthodes **move()** et **draw()**, voyons comment trouver une taille idéale pour notre serpent afin qu'il soit visible sans pour autant qu'il occupe trop de surface. L'écran de l'Arduboy mesure environ 30 x 15 mm pour une résolution de 128 x 64 pixels, une taille de 4 pixels semble donc être un bon compromis. Nous pouvons donc stocker cette taille dans une constante **SNAKE_PART_SIZE**.
Voyant maintenant comment nous pouvons écrire la méthode **move()** :

```
void Snake::move() {

 if (direction == SNAKEDIR_UP) {

  y -= SNAKE_PART_SIZE;
 }
 }
 else {

  x += SNAKE_PART_SIZE;
 }

 if (checkCollisionWithItself() || checkCollisionWithBorder()) {

  isDead = true;
 }

 if (checkCollisionWithFood()) {

  // Pour l'instant on ne fait rien
 }
 else {

  // Déplacement du serpent
  parts[(partsStart + partsCount) % MAX_SNAKE_PARTS] = SnakePart(x, y);
  partsStart = (partsStart + 1) % MAX_SNAKE_PARTS;
 }
}
```

Cette méthode est simple à comprendre, on déplace le serpent en fonction de sa direction et on vérifie ensuite si ce déplacement engendre une collision avec la nourriture, la bordure de l'écran ou le serpent lui même. Voyons justement à quoi ressemble ces 2 dernières fonctions de détection de collision **checkCollisionWithBorder()** et **checkCollisionWithItself()**. La première est très facile à écrire :

```
bool Snake::checkCollisionWithBorder() {

 if (x == 0 || x == 124 || y == 0 || y == 60) return true;
 else return false;
}
```

Elle retourne simplement **true** si la position du serpent est égale à la position de la bordure du jeu. Rappelez-vous que nous avons placé cette dernière dans la fonction **loop()** à l'aide de la fonction **drawRect()**. La seconde méthode **checkCollisionWithItself()** n'est pas beaucoup plus compliquée, elle se contente de parcourir l'ensemble des blocs du serpent et de retourner true si la position de l'un des blocs est la même que le serpent lui-même :

```
bool Snake::checkCollisionWithItself() {
 ...
 for (uint8_t i=0; i<partsCount; i++) {
  ...
  if (x == parts[currentPart].x && y == parts[currentPart].y) {

   return true;
  }
 }
 ...
}
```

Occupons nous maintenant la méthode **draw()**. A l'instar de la méthode précédente, elle parcourt l'ensemble des blocs et les dessine sous forme d'un rectangle plein via la fonction **fillRect()** de la bibliothèque de l'Arduboy :

```
void Snake::draw() {
 …
 for (uint8_t i=0; i<partsCount; i++) {
  …
  arduboy->fillRect(parts[currentPart].x, parts[currentPart].y, SNAKE_PART_SIZE, SNAKE_
PART_SIZE, 1);
 }
}
```

Si on exécutait notre jeu tel que venons de l'écrire, nous verrions un serpent de 5 blocs se déplacer vers la droite et s'arrêter au moment où il rencontre la bordure (Fig.5A). Pas de quoi faire un jeu pour l'instant mais on a au moins le mérite d'avoir créé notre première animation !

Un peu d'interaction

Voir notre serpent se promener sur l'écran c'est bien mais ce serait encore mieux si on pouvait le contrôler. Pour cela, la bibliothèque de l'Arduboy met à disposition 2 méthodes :

- **boolean pressed(uint8_t buttons)** : retourne **true** si l'utilisateur appuie sur un ou plusieurs boutons.
- **boolean notPressed(uint8_t buttons)** : retourne **true** si un bouton est relâché.

Le paramètre **buttons** peut contenir les valeurs suivantes : **UP_BUTTON, DOWN_BUTTON, LEFT_BUTTON**, **RIGHT_BUTTON**, **A_BUTTON** et **B_BUTTON**. Inutile de vous expliquer à quoi correspondent ces constantes, vous avez déjà deviné par vous-même je pense. Il est par contre important de noter que la méthode **pressed()** prend en charge la pression de 2 boutons simultanés à l'aide d'un opérateur **+**, on peut donc vérifier par exemple que le joueur appuie sur le bouton de gauche et le bouton A en même temps grâce à l'instruction **arduboy.pressed(LEFT_BUTTON + A_BUTTON)**. Maintenant que nous connaissons la théorie, passons à la pratique et mettons à jour notre fonction **loop()** à l'aide du code suivant :

```
void loop() {
 …
 // Si le serpent est en vie, la partie continue
 else {

  if (arduboy.pressed(UP_BUTTON)) {

   snake.direction = SNAKEDIR_UP;
  }
 …
  else if (arduboy.pressed(RIGHT_BUTTON)) {

   snake.direction = SNAKEDIR_RIGHT;
  }

  if (!arduboy.nextFrame()) return;
 …
 }
}
```

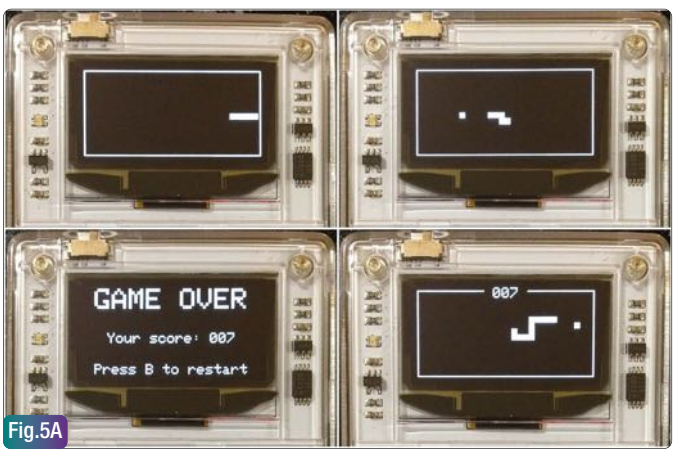


Fig.5A

Si nous exécutons de nouveau notre jeu, nous pouvons maintenant contrôler notre serpent. Génial ! Notre jeu commence à prendre forme, il est temps maintenant de lui donner à manger.

Nourrissons notre reptile !

Puisqu'on parle de nourriture, il est temps de s'occuper de notre classe **Food**. Commençons par écrire la méthode **generate()** :

```
void Food::generate() {

 x = random(1, 31) * SNAKE_PART_SIZE;
 y = random(1, 15) * SNAKE_PART_SIZE;
}
```

Même si elle est très courte, elle mérite quelques explications. La fonction **random(1, 31)** va générer aléatoirement un nombre entre 1 et 30. Oui vous avez bien lu : entre 1 et 30 et non entre 1 et 31, ceci est propre à cette fonction Arduino. Mais pourquoi générer un nombre entre 1 et 30 ou un nombre entre 1 et 14 ? Si on veut que la nourriture soit « mangée » par le serpent, il faut qu'elle se trouve sur sa trajectoire. Notre serpent se déplaçant avec un pas de 4 pixels (constante **SNAKE_PART_SIZE**), il se déplace en réalité sur une grille de (128 / 4) x (64 / 4) = 32 x 16 unités (128 et 64 étant la résolution de l'écran). Vu qu'on dessine une bordure sur l'écran, il faut qu'on ajoute une marge de 1 unité à cette grille de 32 x 16. Il nous reste ainsi une grille de 30 x 14 où notre serpent se déplace horizontalement entre les coordonnées 1 et 30 et verticalement entre les coordonnées 1 et 14. Une fois ces coordonnées obtenues sur cette grille, il suffit de les multiplier par la constante **SNAKE_PART_SIZE** pour obtenir la position de la nourriture sur l'écran. Cette position sera ensuite utilisée par la méthode **draw()** que nous pouvons écrire ainsi :

```
void Food::draw() {

 arduboy->fillRect(x, y, SNAKE_PART_SIZE, SNAKE_PART_SIZE, 1);
}
```

Inutile de s'attarder dessus, vous avez compris son fonctionnement je pense. Revenons plutôt à notre classe **Snake** et plus précisément à la méthode **move()**. Nous avions volontairement ignoré le cas où le serpent entrait en collision avec la nourriture, il est temps remédier à cela et de compléter cette méthode :

```
void Snake::move() {
 …
 if (checkCollisionWithFood()) {

  // Grossissement du serpent
  parts[(partsStart + partsCount) % MAX_SNAKE_PARTS] = SnakePart(x, y);
  partsCount++;

  // Augmentation du nombre de repas mangés par le serpent
```

```
  foodEaten++;

  // Génération d'un nouveau repas
  food.generate();
 }
 else{
 …
 }
}
```

Vous l'avez probablement deviné à la lecture de ce code, pour faire grossir notre serpent il suffit simplement de ne pas supprimer le dernier bloc de sa queue et d'augmenter le nombre de blocs d'une unité. Coté code cela revient donc à ne pas toucher à la variable **partsStart** et à ajouter 1 à la variable **partsCount**. Remarquez également l'augmentation de la variable **foodEaten** d'une unité, ce qui nous permettra de gérer un score par la suite. Nous avons utilisé la méthode **checkCollisionWithFood()** mais elle est vide pour l'instant, il est donc temps de l'écrire :

```
bool Snake::checkCollisionWithFood() {

 if (x == food.x && y == food.y) return true;
 else return false;
}
```

Continuons maintenant avec notre classe **Snake** et revenons juste un instant sur le méthode **init()** pour rappeler qu'elle doit contenir une instruction **food.generate();**. En effet, sans cette instruction au lancement du jeu, aucun repas ne s'afficherait et le processus de génération de la nourriture serait bloqué. Pour finir, n'oublions pas d'afficher la nourriture au niveau de la fonction **loop()** :

```
void loop() {
 …
 snake.draw();
 snake.food.draw();
 …
}
```

Notre jeu nous permet maintenant de diriger un serpent et de manger de la nourriture, on progresse ! Mais il faut reconnaître que le challenge n'est pas très élevé, il est sérieusement temps de compliquer les choses !

Game Over

Afficher un écran « Game Over » lorsque notre serpent meurt ne va pas nous demander beaucoup d'effort. En effet, la structure de notre jeu est déjà prévue pour cela, il suffit de compléter la fonction **loop()** avec le code suivant :

```
void loop() {

 char score[4]; // Score à afficher

 // Si le serpent est mort, un écran « Game Over » est affiché
 if (snake.isDead) {

  if (arduboy.pressed(B_BUTTON)) {

   snake.init(&arduboy);
  }

  if (!arduboy.nextFrame()) return;

  arduboy.clear();

  // Affichage du titre "GAME OVER"
…

  // Affichage du score
  arduboy.setTextSize(1);
  arduboy.setCursor(19, 33);
  arduboy.print("Your score: ");
  arduboy.setCursor(91, 33);
  sprintf(score, "%03u", snake.foodEaten);
  arduboy.print(score);

  // Affichage du message "Press B to restart"
…

  arduboy.display();
 }
 …
}
```

En regardant attentivement, on s'aperçoit que cette portion de code affiche un titre « GAME OVER », le score obtenu et un message « Press B to restart ». Ces éléments seront affichés jusqu'à ce que joueur appuie sur le bouton B. Il n'y a rien de compliqué par rapport à ce que nous avons déjà vu à part un point : l'affichage du score. Vous avez probablement remarqué que nous affichons le score via une variable **score** et non via la variable **foodEaten** directement. En effet, dans un jeu il est généralement plus agréable d'afficher un score sous la forme « 005 » que « 5 ». L'expression **sprintf(score, "%03u", snake.foodEaten);** signifie par conséquent : transformer le nombre **snake.foodEaten** en un nombre sur 3 chiffres, le compléter avec des 0 et sauvegarder le résultat obtenu dans la chaîne **score**. Afficher le score sur l'écran « Game over » est une bonne chose mais il peut être également très utile de l'afficher sur l'écran de jeu. Comme vous pourrez le voir dans le code source, la procédure d'affichage est identique au code précédent.

Pour aller plus loin

Et voilà ! Nous avons écrit notre premier jeu vidéo sur l'Arduboy ! Evidemment, même si notre jeu est parfaitement jouable, il peut largement être amélioré. Nous pouvons par exemple ajouter un menu principal, un écran de pause, du son, des images, une vitesse au serpent, un Highscore, etc...
Implémenter ces fonctionnalités ne vous demanderait pas trop de difficultés j'en suis sûr mais j'ai quand même décidé de vous offrir 2 versions de notre jeu : une version 0.1 qui correspond à cet article et une version 1.0 qui implémente plus de fonctionnalités. C'est ainsi que fonctionne l'esprit de l'Arduboy, partager ses créations avec la communauté. Le site http://community.arduboy.com est d'ailleurs dédié à cela, vous y trouverez beaucoup de sources disponibles, comme par exemple le code source d'un jeu de plateforme ou le code source d'un clone de Flappy Bird. Un autre site intéressant à consulter est celui de l'équipe de développement TEAM a.r.g (http://www.team-arg.com). Cette équipe a créé de beaux jeux sur l'Arduboy (dont celui livré avec) et met à disposition les sources de l'ensemble de ses créations sur son site. Comme vous le voyez, la communauté est déjà très active et c'est ce qui fait d'ailleurs le principal intérêt de l'Arduboy. Pour ma part j'espère vous avoir donné l'envie d'aller un peu plus loin avec cette merveilleuse petite console. Que ce soit pour jouer, apprendre à programmer ou mettre vos idées en valeur, je vous garantis que vous ne le regretterez pas. ⊞

Dis papa, fabrique-moi une console!

Il y a quelques semaines, lors d'un weekend pluvieux, j'étais bloqué à la maison avec mon fils de dix ans. Alors, plutôt que de se mettre devant la télé pour regarder un Disney, j'ai décidé de faire un peu d'électronique avec lui! J'ai sorti mon Arduino, quelques composants électroniques et nous avons cherché ce que nous pourrions construire. Avec un écran à cristaux liquides, des boutons, un buzzer, il ne nous a pas fallu bien longtemps pour trouver: nous allions fabriquer une console de jeu portable old-school!

François Jacob

Les pièces Fig.1

La breadboard

Indispensable pour faire un premier prototype, elle va nous permettre de vérifier nos branchements sans avoir à sortir le fer à souder.

Le microcontrôleur

Le premier réflexe est d'utiliser un Arduino Uno: simple et rapide à mettre en place, cette fameuse carte permet de programmer et d'interfacer un microcontrôleur de manière rapide et efficace. C'est la plateforme idéale pour débuter (et approfondir) la domotique, la robotique et l'informatique embarquée en général. Cependant, dans notre cas précis, cela pose plusieurs problèmes : l'Arduino fonctionne en 5V alors que l'écran lui, demande du 3.3V. Le deuxième problème est que la carte Arduino est déjà relativement volumineuse et que nous voulons faire une console portable et non une borne d'arcade! Un bon compromis est donc de ne garder que la partie indispensable, c'est à dire le microcontrôleur Atmega328P. La datasheet du composant nous apprend qu'on peut l'utiliser en 3.3V à une cadence de 8MHz. Pour régler la fréquence, on peut utiliser le quartz interne du microcontrôleur. Le revers de la médaille, c'est qu'il faut flasher le bootloader correspondant pour lui permettre de fonctionner à cette fréquence. Pour cette opération, nous allons utiliser un Arduino Uno en plus du microcontrôleur lui-même.

==Le flashage==

Préambule: attention, bien respecter les branchements et les instructions, sinon votre microcontrôleur risque d'être mal flashé.

1 - Assurez-vous d'utiliser la version 1.6.x de l'IDE (https://www.arduino.cc/en/Main/Software#) et téléchargez le fichier https://www.arduino.cc/en/uploads/Tutorial/breadboard-1-6-x.zip

2 - Créez un sous-répertoire "hardware" dans votre répertoire de "sketchbook" (on le trouve dans les préférences de l'IDE Arduino).

3 - Décompressez et déplacez le dossier breadboard du 1) vers le répertoire hardware créé en 2).

4 - Redémarrez l'IDE.

5 - Vous devriez voir un "ATmega328 on a breadboard (8 MHz internal clock)" dans le menu Tools/Board.

Effectuer alors les branchements suivants : Fig.2.

6 - Choisissez Tools/Board/Arduino Uno et le port série sur lequel il est branché (dans Tools/Port). Nous allons maintenant l'utiliser comme un in-system programmer(ISP). Téléchargez le programme ISP sur l'arduino Uno (menu Files/Examples/ArduinoISP/ArduinoISP) puis upload.

7 - Choisissez ensuite "ATmega328 on a breadboard (8 MHz internal clock)" dans le menu Tools/Board.

8 - Enfin sélectionnez Tools > Burn Bootloader > w/ Arduino as ISP.

==Uploader un programme sur l'atmega328==

1 - Une fois que le bootloader 8MHz est bien flashé, retirez le microcontrôleur de l'Arduino Uno qui va nous servir de module FTDI (ou mieux, utilisez un module FTDI si vous en avez un), effectuez les branchements suivants : Fig.3.

2 - Choisissez dans Tools/Board menu "ATmega328 on a breadboard (8 MHz internal clock)" et téléchargez les programmes comme avec un Arduino standard.

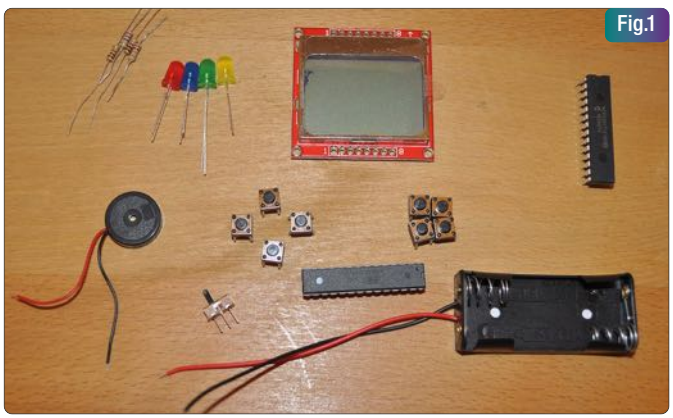


Fig.1

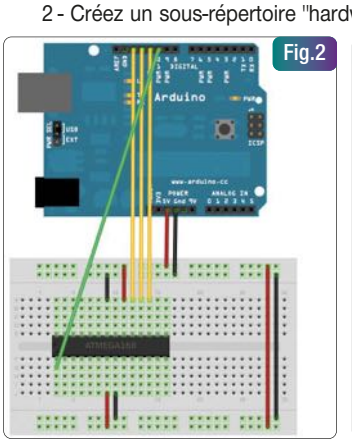


Fig.2

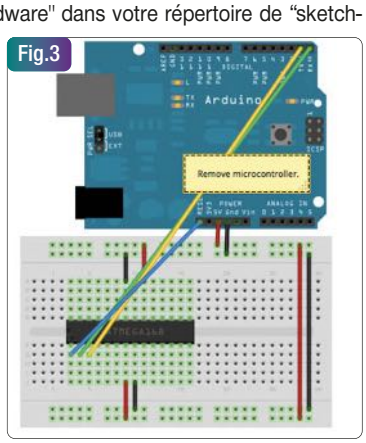


Fig.3

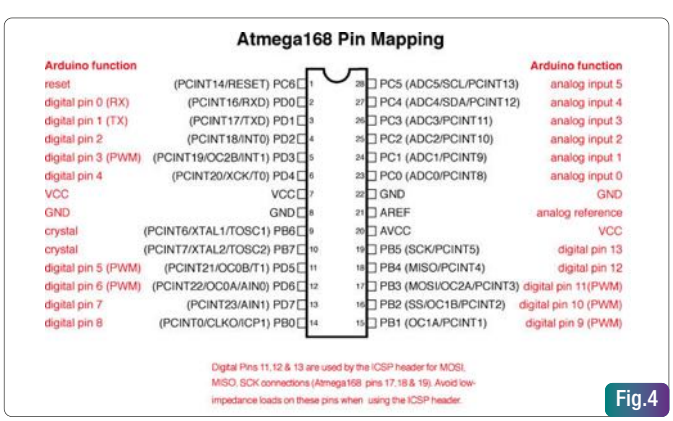


Fig.4

3 - Vous pouvez vérifier que tout s'est bien passé en faisant clignoter une LED sur la broche 19 du microcontrôleur (équivalente à la broche digitale 13 de l'Arduino uno), on trouve le fichier dans le menu File/Examples/01.Basics/blink.

Voilà, votre microcontrôlleur est prêt! Voici le schéma des connexions : Fig.4.

Les boutons

Des petits switchs de 6mm à quatre broches : quatre pour la croix directionnelle et quatre pour les touches d'actions. Petits et robustes: ils sont parfaits pour notre petite console! Branchés sur les broches digitales, le courant est à la masse(GND) quand on appuie dessus, et niveau haut(VCC=3.3V) quand on relâche. Au niveau du programme, GND deviendra false et VCC deviendra true.

Les LEDs

De quoi ajouter un peu de couleur et épauler l'écran. De taille 5mm avec leurs résistances associées: 1,25K pour la bleue et la rouge, 1K pour la verte et la jaune.

L'écran

On utilise un modèle à cristaux liquides monochrome de 1,5 pouce, sa résolution est de 84x48 et il dispose en outre d'un rétro-éclairage par LED. Sa particularité est qu'il a été utilisé dans le passé pour équiper le fameux téléphone Nokia 3310 et a donc été produit à des millions d'exemplaires. La conséquence est qu'il est très bon marché et facile à trouver. De plus, il dispose d'un petit PCB intégré qui va simplifier notre montage. Pour le programmer facilement, il suffit d'utiliser deux bibliothèques open-source de chez Adafruit qui gèrent les bitmaps, les objets géométriques et les textes (https://github.com/adafruit/Adafruit-PCD8544-Nokia-5110-LCD-library et https://github.com/adafruit/Adafruit-GFX-Library). Enfin, on utilise le protocole SPI pour le faire communiquer avec le microcontrôleur. Là encore, une bibliothèque open-source incluse avec l'IDE va s'en charger.

Voilà comment connecter l'écran :

Ecran====ATMEGA328P	DIN=====PD6
RST====PD3	CLK====PD7
CE=====PD4	VCC====VCC
DC=====PD5	LIGHT==PB3
	GND===GND

Le buzzer

En branchant un buzzer sur la broche digital pin 2 de l'Arduino, on peut maintenant accompagner le tout de bips en tous genres.

Voilà comment connecter le buzzer :

+ === PD2 - === GND

Le MCP23017

Si on récapitule, sans tenir compte des broches d'alimentation, on va donc devoir connecter dix-neuf broches sur notre Arduino : six pour l'écran, quatre pour les LEDs, huit pour les boutons et une pour le buzzer. Problème: l'Arduino UNO ne dispose que de dix-huit broches: quatorze digitales, six analogiques (qui peuvent faire office de broches numériques) moins les broches zéro et un qui sont réservées à la communication série. Il a donc fallu utiliser un extenseur de ports. Le MCP23017 effectue cette tâche à merveille: il utilise le protocole I2C qui permet donc à l'Arduino de communiquer avec lui par deux fils. Autre avantage, cela permet de se passer de résistances de tirage au niveau des boutons (elles sont intégrées dans le MCP23017). Pour le programmer, on sélectionne le registre qui nous intéresse, et soit on lui écrit une nouvelle valeur, soit on lit la valeur du registre. Et pour connaître les différents registres disponibles, il suffit de lire la datasheet du composant. Par exemple : Lecture du registre des boutons (0x13) :

```
byte boutons=0;

Wire.beginTransmission(0x20); // init I2C
Wire.write(0x13);             // registre des états des GPIOB (boutons)
Wire.endTransmission();
Wire.requestFrom(0x20, 1);    // on demande à recevoir un octet
boutons = Wire.read();        // on lit la valeur reçue
```

Ecriture du registre :

```
Wire.beginTransmission(0x20);
Wire.write(0x00);       // registre IODIRA (pour nos LEDs)
Wire.write(0x00);       // configurées en sortie
Wire.endTransmission();
```

Voilà comment connecter le MCP23017 avec :

Les LEDs :	Les boutons :	L'Atmega328P:
GPA0===R1===LED1===GND	GPB0===B1===GND	VCC===VCC
GPA1===R2===LED2===GND	GPB1===B2===GND	GND===GND
GPA3===R3===LED3===GND	GPB2===B3===GND	RST===VCC
GPA4===R4===LED4===GND	GPB3===B4===GND	SCL===PC5
	GPB4===B5===GND	SDA===PC4
	GPB5===B6===GND	A0===GND
	GPB6===B7===GND	A1===GND
	GPB7===B8===GND	A2===GND

Sans oublier :

Une alimentation (deux piles 1.5V) à brancher sur VCC/AVCC (+) et GND (-) et un interrupteur. Voilà ce à quoi devrait ressembler votre modèle sur breadboard : Fig.5.

Miniaturisons la console

La console peut maintenant fonctionner, mais si on souhaite un modèle plus durable et vraiment portable, il faudra souder. Vous pouvez utiliser une veroboard, ou mieux, un circuit imprimé. Nous n'allons pas détailler ici tout le processus pour faire un circuit imprimé car c'est un sujet à part entière, mais vous pouvez retrouver sur mon site les fichiers gerber à envoyer à un fournisseur (https://github.com/Papamaker/consolea5euros/tree/master/hardware/Proto0.2/gerber). Avec le circuit imprimé, il suffit de placer les composants à leur place et de les souder. Fig.6 et 7.

Cassons des briques

Maintenant qu'on a une belle petite console, on va passer aux choses sérieuses : l'écriture d'un jeu vidéo ! Classique parmi les classiques, j'ai choisi le casse-brique. Commençons par les données. Il faut savoir avant de commencer qu'un microcontrôleur n'a, en général, pas beaucoup de mémoire intégrée, le nôtre ne dispose que de 2Ko de mémoire vive (ce qui

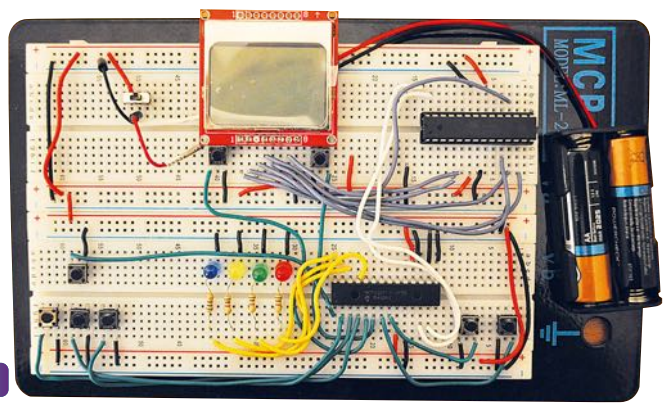


Fig.5

est déjà pas mal pour un microcontrôleur), il va donc falloir structurer ses données avec efficacité. On utilisera principalement le type byte qui est un octet non signé et permet donc de stocker des entiers entre 0 et 255, nous n'aurons pas besoin de plus.
L'ensemble du code source est disponible sur : https://github.com/Papamaker/consolea5euros/blob/master/jeux/breakout/breakout.ino
Au début du programme se trouve une partie déclarative qui concerne les branchements et que vous devrez modifier si vous avez câblé les éléments différemment :

```
/ Les branchements du microcontrôleur
// Rétro-éclairage
#define LCD_LIGHT 11

// Le buzzer
#define BUZZER 2

// Les branchements sur le MCP23017
// Les boutons
#define BOUTONGAUCHE 2
#define BOUTONDROITE 4
#define BOUTONA     32
#define BOUTONB     16
```

Pour le stockage mémoire d'un niveau de jeu, j'ai utilisé le minimum vital: un bit par brique, regroupés par paquets de 8, cela donne donc des niveaux de 16 briques de largeur sur 10 lignes :
// Le niveau en cours: 16 briques par lignes sur 10 lignes = 20 octets de données

```
byte niv[10][2]={ { B11111111,B11111111},
  { B11111111,B11111111},
  { B11111111,B11111111},
  { B11111111,B11111111},
  { B11111111,B11111111},
  { B11111111,B11111111},
  { B11111111,B11111111},
  { B11111111,B11111111},
  { B11111111,B11111111} };
```

Mine de rien, avec une telle optimisation, un niveau n'utilise que 20 octets, on peut donc en stocker une centaine dans nos 2 petits kilo octets ! Une approche non optimisée avec par exemple un int par case aurait occupé 320 octets, soit 16 fois plus ! Un autre avantage d'utiliser des bits est de pouvoir dessiner les niveaux en style pixélisé : effet vintage assuré! **Fig.8, 9 et 10.** On utilisera ensuite quelques variables pour stocker les éléments de jeu (niveau, raquette et balle) :

```
// numéro du niveau
byte niveau;

// la position horizontale de la raquette
byte xRaq;

// la position de la balle
byte xBalle;
byte yBalle;
byte etatBalle;
byte dirBalle;
```

Un programme Arduino comporte deux fonctions principales : setup() qui sera appelée une fois, au démarrage et loop(), la boucle principale du programme. La fonction setup contient principalement les initialisations de l'écran et du MCP23017, qui consiste à configurer les sorties en écrivant dans les bons registres. : on configure les pins des LEDs en sortie, et celles des boutons en entrées avec les résistances de tirage activées.

```
// initialisation matérielle de l'écran
Adafruit_PCD8544 display = Adafruit_PCD8544(7, 6, 5, 4, 3);

// L'initialisation du code
void setup() {

 // initialisation de l'écran
 display.begin();

 // activer le retro-éclairage
 analogWrite(LCD_LIGHT, 0);

 // initialisation du MCP23017
 Serial.begin(9600);
 Wire.begin();              // init bus I2C
 Wire.beginTransmission(0x20);
 Wire.write(0x00);          // registre IODIRA (pour nos LEDs)
 Wire.write(0x00);          // configurées en sortie
 Wire.endTransmission();
 Wire.beginTransmission(0x20);
 Wire.write(0x0D);          // registre GPPUB (les boutons)
```

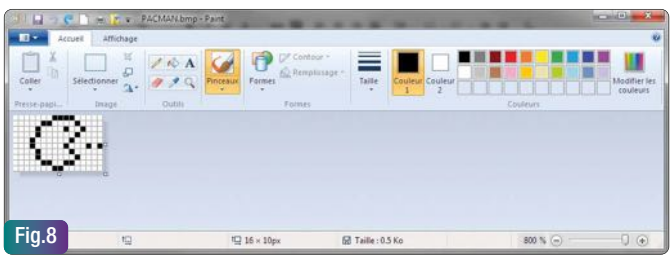


Fig.8

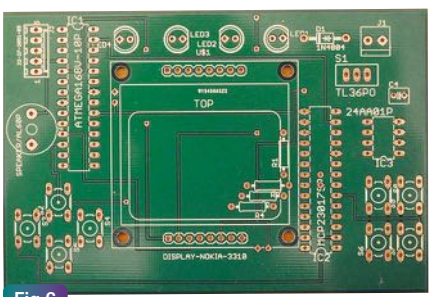


Fig.6

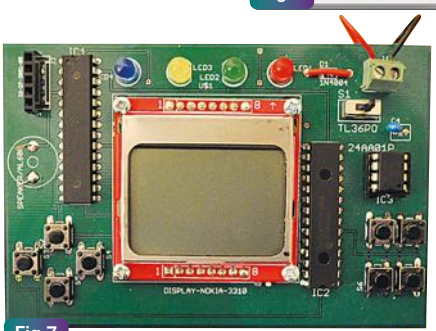


Fig.7

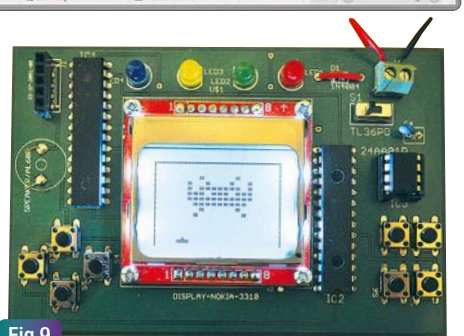


Fig.9

```
 Wire.write(0xFF);          // activer les pullups
 Wire.endTransmission();

 // initialisation du buzzer
 pinMode(BUZZER,OUTPUT);

 // lancement du niveau 1
 niveau = 0;
 prochainNiveau();
}
```

L'algorithme principal est assez simple, on utilise 4 fonctions qu'on appelle en boucle l'une après l'autre :

```
// La boucle principale
void loop() {

 // gestion du graphisme
 afficheEcran();

 // gestion du son
 son();

 // gestion des contrôles
 controles();

 // logique du jeu
 logiqueJeu();
}
```

Pour l'affichage, c'est à dire la fonction afficheEcran(), on trace trois lignes pour les contours de l'écran, puis on affiche les briques du niveau, la balle et la raquette. La fonction son() gère le son et monopolise à ce moment le microcontrôleur à 100%. Il n'y a pas de carte sonore, ni de coprocesseur ni même de multi-coeur comme sur une console moderne. On va donc lui réserver une fenêtre temporelle de 10 millisecondes, on jouera un son quand cela est nécessaire ou alors une pause de 10 millisecondes. Cette durée est suffisante pour être audible et ne pas ralentir le déroulement du jeu. On doit même pouvoir jouer une petite mélodie en utilisant ce principe là... N'hésitez pas à tester.
Pour la gestion des contrôles, on utilise les fonctionnalités du MCP23017 :

```
byte boutons=0;

 Wire.beginTransmission(0x20); // init I2C
 Wire.write(0x13);             // registre des états des GPIOB (boutons)
 Wire.endTransmission();
 Wire.requestFrom(0x20, 1);    // lire un octet
 boutons = Wire.read();        // stocké dans boutons
 boutons = (~boutons)&0xFF;

 // si on appuie sur gauche et qu'on est pas au bord : se déplacer
 if ( (boutons&BOUTONGAUCHE) && (xRaq>1)) {
  xRaq=xRaq-VITESSERAQ;
  if (etatBalle == COLLEE) {
   xBalle = xBalle - VITESSERAQ;
  }
 }
```

Enfin la logique du jeu se charge de :

- Gérer le déplacement de la balle (incrémenter sa position en x et y selon sa direction) ;
- Gérer les rebonds sur les murs, la raquette et les briques (en les détruisant au passage) ;
- Vérifier si on perd ou si on gagne.

La console à 5 euros

L'article que je vous ai proposé est extrait du projet "La console à 5 euros". L'idée est de mettre à disposition de tous, pour le prix le plus bas possible et en open-source, des kits pour fabriquer sa propre console. Des étudiants, adolescents et même pourquoi pas des enfants pourront souder leur propre console de jeu et apprendre l'informatique sur ce support. Le fait de fabriquer sa console soi-même est un aspect très valorisant ! D'autant qu'ils pourront ensuite écrire leurs propres jeux et apprendre la programmation ! De nombreuses structures et associations sont impliquées dans l'apprentissage des nouvelles technologies mais sont souvent limitées par le coût du matériel, j'espère pouvoir les aider en leur fournissant des kits. J'espère pouvoir les mettre à disposition le plus tôt possible. Fig.11. J'ai aussi écrit un programme d'initiation à la programmation : Pac-Rob, il s'utilise directement sur la console sans avoir besoin d'ordinateur et permet à de petits enfants de faire leurs premiers pas en programmation (un peu à la manière de scratch). Le projet dispose maintenant de trois programmes (qui sont tous sur GitHub), cela devrait grandir très prochainement et je vous encourage à écrire les vôtres et à les partager sur le GitHub du projet. J'ai adoré élaborer ces premiers prototypes, j'ai 41 ans et lorsque j'étais enfant, je n'aurais jamais imaginé que pour une poignée d'euros (enfin de francs), on pourrait élaborer et construire une petite console de jeu portable dans sa cuisine (ou son garage) ! Cela est maintenant possible en grande partie grâce au gigantesque partage de connaissance que permet Internet. De formidables projets open-source comme l'Arduino ont également permis de nous ouvrir les portes des microcontrôleurs. Je souhaite partager ma passion avec le plus grand nombre et de manière bénévole. Si vous pensez aussi que c'est un beau projet et que vous voulez l'aider à grandir plus vite, n'hésitez pas à me rejoindre, envoyez-moi un e-mail !

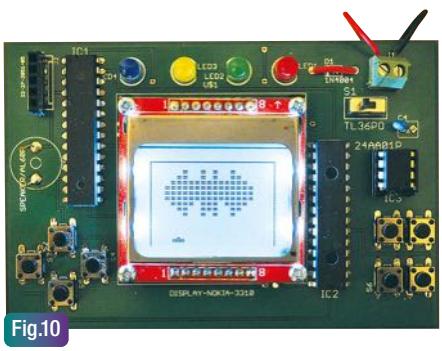


Fig.10

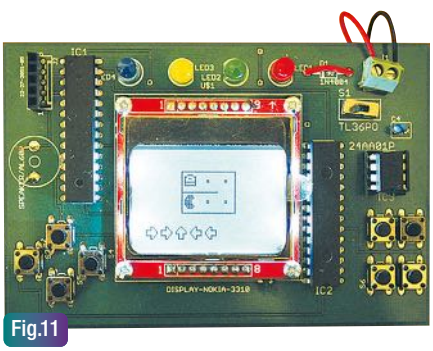
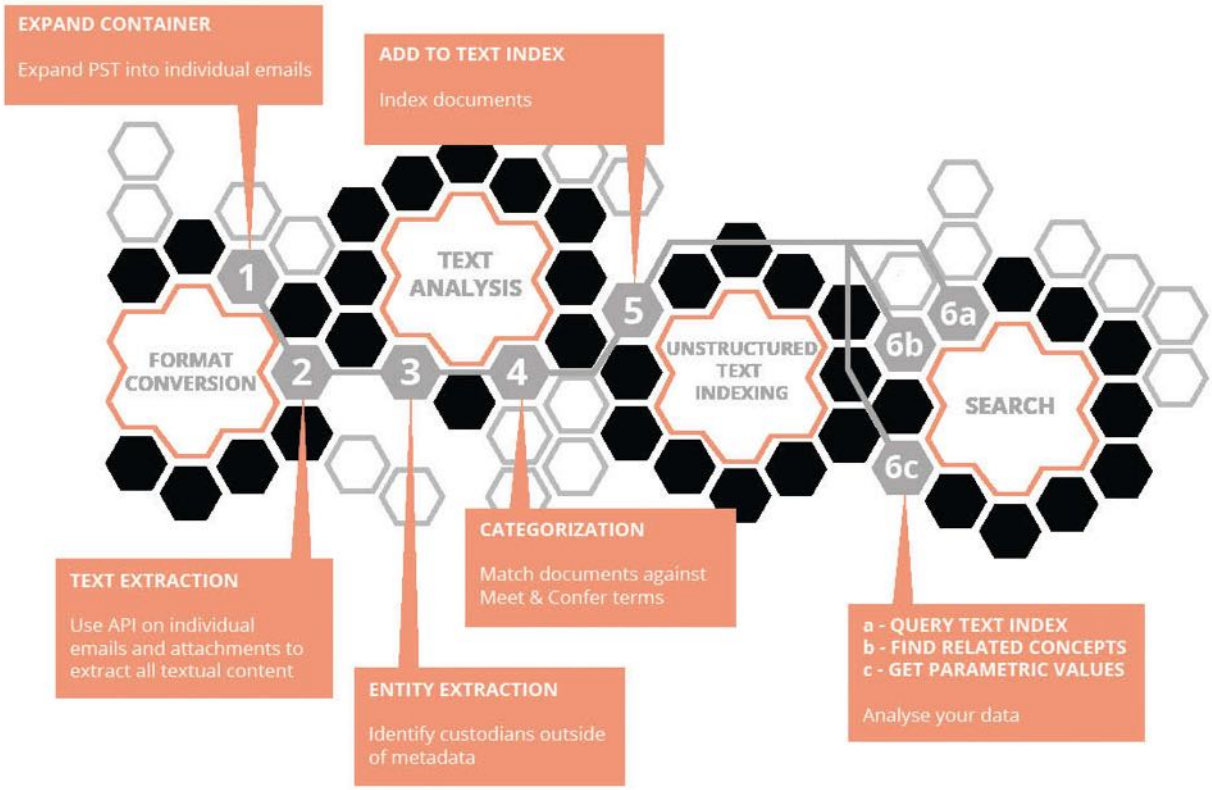


Fig.11

Les liens du projet

La page facebook :
https://www.facebook.com/laconsolea5euros/
Le blog : https://papamaker.fr/category/la-console-a-5-euros/
Instagram : https://www.instagram.com/papamaker.fr/
Twitter : https://twitter.com/Papamaker_Fr
GitHub : https://github.com/Papamaker/consolea5euros
e-mail : papamaker.fr@gmail.com

HPE Haven OnDemand : des API qui vont plaire aux développeurs !



Hewlett Packard Enterprise (HPE) propose à ses clients des solutions technologiques de pointe en matière d'infrastructures, de services, de logiciels et de services financiers, pour optimiser leur informatique traditionnelle.
Le Big Data constitue l'un des quatre piliers stratégiques et HPE dispose d'une gamme logicielle permettant de traiter et d'analyser 100% des informations pour en tirer de la valeur.
Ce portefeuille vient de s'étoffer d'une solide offre d'API pour les développeurs et les entreprises !

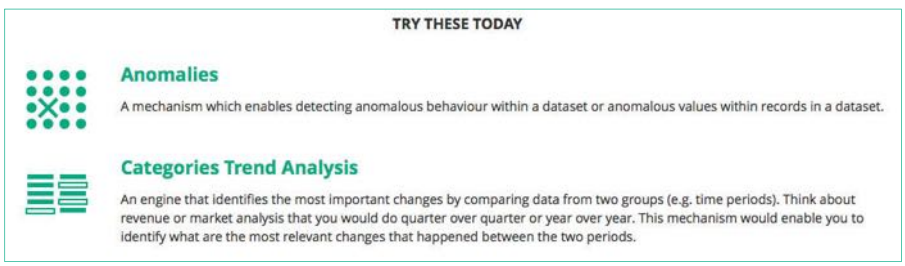
Ces API sont regroupées sous le nom Haven OnDemand. Le slogan est simple à comprendre : une plate-forme, une innovation sans limites ! Certains acteurs proposent quelques API mais Haven OnDemand offre une gamme complète pour répondre à la plupart des demandes et des besoins des développeurs d'applications mobiles, desktop ou web.

Haven OnDemand : c'est quoi ?

Il s'agit avant tout d'un portail web servant de point d'entrée pour accéder aux API, à leur documentation, à des exemples et des cas d'usages. Tout est fait pour faciliter la recherche, et la documentation disponible en anglais est très complète.
Aujourd'hui, ce sont plus de 60 API qui sont disponibles sur le portail. Ces API se regroupent par catégories ou usages :

- Analyses audio et vidéo
- Connecteurs
- Conversion de formats
- Analyses géospatiales
- Représentation en graphes
- Analyses d'images
- Classifications
- Prédiction
- Requêtage / Recherche
- Analyses de textes
- Indexation de données non structurées

Ces API sont en évolution constante et de nouvelles apparaissent régulièrement. Elles sont disponibles en version finale (et prêtes à être déployées et implémentées) ou en pré-version (bêta). Toutes sont accessibles en version libre. Certaines d'entre elles nécessitent d'ouvrir un compte Haven OnDemand (compte gratuit).
Les apps consomment de plus en plus de



services et exigent des analyses complexes. Les API ne s'utilisent plus seules, mais nécessitent d'être connectées entre elles, bref, de s'intégrer et d'interagir. Par exemple, pour créer une application d'analyse de document scanné vous aurez besoin de plusieurs API couvrant différentes fonctions : capture et analyse de l'image, conversion de formats (image vers textes - OCR), indexation des textes extraits, recherche « full text » et affichage du résultat. L'un des avantages d'utiliser une plate-forme comme Haven OnDemand est de disposer d'API d'un seul fournisseur. Vous ne perdrez pas votre temps à chercher ici ou là l'API voulue et vous minimiserez les risques d'incompatibilité et les problèmes d'intégration.

La plate-forme est particulièrement appropriée dans l'analyse des images (fixes ou non) pour détecter les visages, les sentiments ou analyser des comportements. Les fonctions et mécanismes de Machine Learning et de Big Data constituent le cœur de Haven OnDemand. Tout récemment, une nouvelle fonction est arrivée sur le portail : Labs. Vous y trouverez des API sur des fonctionnalités et des thématiques très récentes ou sur des usages d'avenir. Par exemple : comment identifier des tendances et des corrélations entre des jeux de données. La communauté se développe rapidement. Plus de 15 000 développeurs sont inscrits sur la plate-forme et la communauté en ligne permet d'échanger, de poser des questions et de trouver des réponses. De nombreuses ressources sont accessibles : documentations, tutoriels, vidéos, exemples de démarrage. Du Freemium aux applications d'entreprise, Haven OnDemand propose des modèles d'usage et une tarification selon les besoins et les ressources utilisées. Deux critères sont utilisés : Resource Units et API Units.

Resource Units concerne les ressources statiques, par exemple le stockage des données. Les ressources utilisées sont évaluées de deux façons :

- Connector Resource Types : chaque connecteur correspond à une unité de ressources statiques. Cela correspond à la création du connecteur et au stockage de la configuration. Attention : des limitations et des quotas existent. Voir : https://dev.havenondemand.com/docs/RateLimitingBehavior.html
- Text Index Resource Types : cela correspond à la taille d'index et à un coût (par unités). Par exemple, le type Explorer est limité à 200 Mo d'indexation et équivaut à une unité.

La seconde unité utilisée est API Units. Chaque appel aux API est comptabilisé. Pour certaines API nécessitant d'importantes ressources de calculs, telles que les API d'OCR ou l'analyse de la voix, l'unité de mesure sera l'appel à l'API mais aussi la taille des données. Par exemple, pour l'utilisation de l'API OCR, une unité correspond à une page traitée. Pour la plupart des API, 1 appel équivaut à 1 unité API. Pour en savoir plus sur les unités : https://dev.havenondemand.com/docs/Quotas_APIResourceConsumption.html

Pour de petits développements et pour tester les API, l'offre Freemium suffira. Pour un usage plus intensif, l'offre Explorer sera intéressante pour des apps et projets de taille moyenne. Pour les entreprises, il faudra se tourner vers les offres Innovator et Entrepreneur. Les offres payantes démarrent à 10 $ par mois. Au-delà des quotas, des facturations à la demande sont possibles (de 10 à 15 $ selon le plan).

Pour une utilisation en production, un support premium est disponible : 24x7, SLA, ingénieur support identifié et unique. A vous de coder !

Quelques liens utiles

www.havenondemand.com
www.youtube.com/channel/UCj6FJousWryYHadAASRNtRg
community.havenondemand.com
dev.havenondemand.com

Haven OnDemand : Prédiction

Le Machine Learning, ou ML, est une des stars des technologies les plus récentes. L'application, la « machine », apprend de son environnement, son contexte, les usages. Le ML analyse de grandes masses d'informations et par des modèles et des algorithmes, en ressort des données qualifiées, et peut agir en conséquence. Ces algorithmes ne se substituent pas forcément à l'intelligence ou l'expertise humaine. Ils permettent d'accélérer l'analyse des informations, surtout depuis que les données ont considérablement augmenté en volume et en complexité. La puissance de calcul des serveurs, les capacités de stockage et ces nouveaux algorithmes permettent pratiquement de proposer des analyses en temps réel sur des volumes d'information gigantesques. Le Machine Learning et ses algorithmes ont fréquemment pour objectif de prédire les comportements ou les événements, et donc d'anticiper pour prévenir. Les applications sont très variées. Il est par exemple possible d'analyser des compte-rendus de visites (données non structurées) ou des logs de machines (données structurées) pour prédire les potentielles futures pannes ou anomalies pouvant survenir sur ces machines ou sur des machines similaires. Les sites web analysent aujourd'hui les comportements des internautes pour prédire les futurs achats des consommateurs. Le ML joue donc un rôle important dans les applications devant faire appel à la prédiction, par exemple pour l'approvisionnement d'un site marchand selon le jour, la saison, la météo, etc. Actuellement, Haven OnDemand dispose de trois API spécifiques orientées « prédiction » : predict, recommend, train prediction. Ces API font partie des fonctions d'analyses prédictives de la plate-forme.

1 - Son fonctionnement

La première API, Predict, exécute des modèles de prédiction sur des jeux de données. Elle permet de classer, prédire et analyser les données.

L'API Recommend peut conseiller des changements dans des jeux de données pour obtenir les résultats attendus. Ce dernier est par exemple utilisé par l'API Train Prediction. Vous pouvez aussi obtenir des résultats différents de la prédiction initiale en indiquant les modifications à réaliser. Le service retourne alors le jeu de données modifié, conforme au résultat demandé.

La dernière API, Train Prediction, crée un modèle de prédiction, basé sur des données. Différents algorithmes sont utilisés avec différents paramètres pour chaque algorithme. Le modèle des données est au format CSV ou JSON.

2 - Exemple d'utilisation

https://goo.gl/i6jkNg

Pour en savoir plus

https://dev.havenondemand.com/docs/HowTo_Prediction.html

Haven OnDemand : les connecteurs

La force d'Haven OnDemand réside dans la large palette de possibilités qu'offre la plate-forme, et les nombreuses API disponibles illustrent cette infinité d'usage. Afin de s'adresser à n'importe quelle source de données, le développeur dispose de multiples connecteurs. Ils permettent de se connecter à des ensembles de données, des systèmes de fichiers locaux ou distribués (SharePoint, Dropbox...) afin de capturer les documents, les ouvrir et les traiter, par exemple pour les indexer à des fins de recherche. Vous pouvez ainsi automatiser des workflows, des notifications, faciliter la migration de données d'un site vers un autre, etc. Explications.

Le moteur on-premise HPE IDOL est un logiciel orienté recherche et analyse de données structurées et non-structurées, avec ou sans Machine Learning. Sa puissance est due à son caractère universel : il traite tout type de sources, de documents et de langues.
Haven OnDemand permet de mettre en oeuvre tous les connecteurs disponibles avec HPE IDOL.

Fonctionnement

L'objectif d'un connecteur est très simple : récupérer les contenus présents dans un référentiel de données, tel qu'une base de données, un serveur de messagerie, un système de fichiers ou toute autre source.
Haven OnDemand propose deux types de fonctionnement pour les connecteurs : localement (on-premise) ou en mode cloud (sur la plate-forme Haven OnDemand). Le fonctionnement on-premise permet de se connecter aux sources de données situées dans l'intranet des entreprises. Pour utiliser un connecteur on-premise, on l'installera sur les serveurs de l'entreprise.
Les connecteurs supportent tous les formats numériques acceptés par défaut par HPE IDOL, soit plus de 1000 formats différents. Il existe des connecteurs par type de référentiels et tous les connecteurs peuvent traiter tous les types de fichiers.

Sur la plate-forme Haven OnDemand quatre connecteurs sont disponibles :

connecteurs cloud	connecteurs on-premise
web cloud connector : indexe les contenus provenant d'une page web	File System Connector : récupère et indexe le contenu d'un système de fichiers
dropbox cloud connector : indexe le contenu d'un compte DropBox	SharePoint Connector : récupère et indexe les contenus de SharePoint Server ou d'un compte SharePoint Online

Les autres connecteurs du moteur HPE IDOL (ODBC, Documentum, Notes, Exchange, Facebook, Twitter,... soit une petite centaine de connecteurs qui permettent de se connecter à plus de 400 sources de données) sont également compatibles avec Haven OnDemand mais ne sont pas librement accessibles. Vous devez en faire la demande.

Pour faciliter l'usage des connecteurs, plusieurs API vous facilitent la vie : démarrage, mise à jour, récupération, création, statuts, historique, etc.

Pour utiliser les connecteurs, vous devrez élaborer un conteneur via l'API Create Connector qui permet de créer la configuration du connecteur : identification du référentiel de données et connexion à celui-ci, les contenus à récupérer et à indexer. Il est possible de mettre en place un scheduler pour planifier et automatiser le travail du connecteur utilisé. Vous pouvez aussi démarrer manuellement un connecteur (API Start Connector).

Exemple d'utilisation

Voici un exemple d'utilisation avec la création d'un nouveau connecteur :
URL de l'API = https://api.havenondemand.com/1/api/sync/createconnector/v1
Paramètre d'authentification = apikey
Paramètre obligatoire = Flavor (=Mode du connecteur) / Nom du connecteur / Configuration / Destination
Paramètre optionnel = Credentials / politique de credential / description / planification

Ce qui donne pour l'exemple d'un connecteur Filesystem :

```
https://api.havenondemand.com/1/api/sync/createconnector/v1
&flavor=filesystem_onsite
&connector=Connecteur Filesystem
&config={
  "directoryPathCSVs" : "C:\\users,C:\\Documents",
  "directoryRecursive": true,
  "servicePort": 8002,
  "aciPort": 8000,
  "manipulate_reference_prefix": "C:\\Stuff",
  "new_reference_prefix": "\\\\share\\"
}
&destination={
  "action": "addtotextindex",
  "index": "testindex"
}
&apikey=votre_apikey
```

https://dev.havenondemand.com/docs/Connectors_Filesystem.html

Pour en savoir plus

Documentation : https://dev.havenondemand.com/docs/Connectors.html
API pour les connecteurs : https://www.havenondemand.com/docs/api-overview.html#connector

Hewlett Packard Enterprise

Haven OnDemand : index / recherche

Dans un monde où la volumétrie des données explose, il est difficile d'analyser ces données, que les contenus soient structurés ou non-structurés (textes). On utilisera l'indexation de ces données pour procéder à l'analyse ou effectuer des recherches. La plate-forme Haven OnDemand propose une offre complète dans ce domaine.

Les différentes API et leurs fonctions

L'indexation et la recherche sont couvertes par une quinzaine d'API. Nous allons vous présenter rapidement 5 d'entre elles :

- Add to Text Index
- Find Related Conceps
- Find Similar
- Get Parametric Values
- Query Text Index

Add to Text Index est une API permettant d'ajouter du contenu à un index de textes que vous aurez préalablement défini (vous devez obligatoirement créer un Text Index via l'API Create Text Index). Ces contenus seront indexés et visibles par les autres API manipulant les index et la recherche. Il est conseillé d'utiliser cette API en asynchrone pour ne pas bloquer l'indexation. Vous pouvez utiliser des connecteurs pour récupérer des contenus d'un site SharePoint ou d'un site Dropbox. **[Fig.1]**

L'API Find Related Concepts retourne une liste de concepts à partir d'une recherche spécifique. Ce résultat est donné suite à une requête documentaire. Par exemple, pour Egypte, nous pourrions obtenir : pays, pharaon, Louxor, pyramides, etc. Vous pourrez aussi obtenir des résultats identifiant des acceptions différentes. L'exemple "Mercure" est très parlant puisqu'on pourra avoir :

- planète
- composant chimique

Cette fonctionnalité particulière à Haven OnDemand fonctionne dans toutes les langues et ne nécessite pas l'utilisation de thesaurus. Les résultats obtenus sont déterminés par des algorithmes mathématiques exploitant les fréquences, co-occurence et proximité de ces concepts. Cette fonctionnalité est l'une des plus prisées par les utilisateurs du moteur HPE IDOL. **[Fig.2]**

L'API Find Similar a pour objectif d'identifier des contenus similaires à votre document d'origine. Par exemple il s'agira de retrouver tous les CV pouvant correspondre à une offre d'emploi. Tout comme la précédente, cette API repose sur la puissance du moteur HPE IDOL et de ses règles mathématiques. **[Fig.3]**

L'API Get Parametric Values permet de récupérer les métadonnées de tous les documents résultant d'une recherche. Ces métadonnées pourront être les valeurs des champs que l'on peut avoir sur un formulaire, une page web, etc. Par exemple, on pourra obtenir une liste des produits par prix, par catégorie, etc.

Query Text Index est l'API qui permettra de faire des recherches sur les bases de données Haven OnDemand. L'API permet de formuler ces recherches en langage naturel, en utilisant les mots-clés, les expressions booléennes. Les résultats obtenus sont des listes de documents précédemment indexés.

Exemple d'utilisation

Voici un exemple d'utilisation avec l'API « Get Parametric Values» :

URL de l'API = https://api.havenondemand.com/1/api/sync/getparametricvalues/v1

Paramètre d'authentification = apikey

Paramètre obligatoire = Nom du ou des champs paramétriques

Paramètre optionnel = Comptage des documents / Champ recherché / Index à utiliser / Nombre maximal de valeurs / Pertinence / Tri / Texte à rechercher / Profil de recherche à utiliser.

Ce qui donne pour l'exemple de Wikipedia en anglais et le champ « Profession » :

https://api.havenondemand.com/1/api/sync/getparametricvalues/v1
&field_name=person_profession
&apikey=votre_apikey

Pour en savoir plus

Add to Text Index :
https://dev.havenondemand.com/apis/addtotextindex#overview

Find related concepts :
https://dev.havenondemand.com/apis/findrelatedconcepts#overview

Find Similar : https://dev.havenondemand.com/apis/findsimilar#overview

Get Parametric Values :
https://dev.havenondemand.com/apis/getparametricvalues#overview

Query Text Index : https://dev.havenondemand.com/apis/querytextindex#overview

Documentation :
https://dev.havenondemand.com/docs

Haven OnDemand : les API d'analyses d'images

Les technologies d'analyses d'images se multiplient. Il s'agit de pouvoir analyser, interpréter des images fixes ou animées. Ce domaine couvre de nombreux types d'images : textes (documents scannés ou codes-barres), visages (identification ou reconnaissance faciale) ou objets (détection, identification ou classification).

Quand on parle d'analyse d'images, on distingue la détection de l'identification, que ça soit pour le texte, les objets ou les visages. La détection correspond à la reconnaissance de la présence de l'élément (l'image contient-elle des visages ? du texte ? des codes-barres ? etc). Tandis que l'identification d'un élément est le fait de pouvoir le nommer (un élément a été détecté, et je peux le nommer : il s'agit de tel objet, ou de telle personne).
Pour bien comprendre le rôle de ces API, prenons quelques exemples :

Type d'images à analyser	Exemple d'usage
Codes-barres	détecter, et lire, un code à barres 2D
Visages	détection des visages sur des images
Objets	reconnaître des logos, des objets dans une image
Texte	extraire le texte d'un document scanné

Haven OnDemand répond à ces défis par 4 principales API :

- Barcode Recognition
- Face Detection
- Image Recognition
- OCR Document

1 - Les API

Face Recognition

L'API de reconnaissance de visages (Face Recognition) a pour but de détecter sur des images la présence de visages et de les isoler en les encadrant. Par exemple, sur les apps photos des smartphones, vous pouvez détecter des visages à la volée.
L'API supporte les principaux formats d'images fixes (jpeg, png, bmp, tiff,...) et vidéos (mpeg, windows media, mp4,...). Vous pouvez coupler cette API à l'API de reconnaissance vocale. Cette API supporte de nombreux codecs (mpeg audio, wav, AC3,...).

Image Recognition

L'API de reconnaissance d'image se focalise sur une partie spécifique d'une image. Typiquement, vous pouvez identifier et isoler un logo dans un visuel. L'API produira en résultat le nom du logo tel qu'il est indiqué dans la base de données d'Haven OnDemand. La localisation du logo au sein de l'image sera aussi précisée. Bien entendu, l'usage de cette API ne s'arrête pas uniquement aux logos. Vous pouvez l'utiliser pour d'autres types de reconnaissances, tel que l'identification ou la classification d'objets, à condition de disposer des jeux de données adéquats pour procéder à l'apprentissage.

OCR Document

Comme le nom de l'API l'indique, OCR Document (Optical Character Recognition) permet la reconnaissances optique de caractère au sein d'un document, c'est-à-dire l'extraction de texte présent dans une image. L'API supporte différents alphabets (latin, cyrillique, arabe, ...) et langues (anglais, français, grec, italien, russe, etc.) ce qui rend l'API très pertinente dans un contexte Big Data multilingue.
En entrée, l'API nécessite les fichiers et images que vous souhaitez analyser. En sortie, l'API produit le texte extrait et la localisation de celui-ci dans l'image. Vous pouvez alors appliquer une indexation et une recherche full text. L'image que vous envoyez à l'API peut être de différente nature : photo, document scanné ou scène complexe (par exemple un paysage, une scène dans un magasin ou des sous-titres d'une vidéo). Pour optimiser la reconnaissance de caractères, vous pouvez préciser la nature de l'image dans les paramètres de la requête.

Barcode Recognition

Plus « classique », la reconnaissance des codes-barres. L'API permet de les détecter et de les identifier. Il existe plusieurs types de codes-barres : EAN-13, EAN-8, Matrix 2/5, QR Code, etc.

2 - Exemples d'utilisation des API

Vous pouvez appeler les API d'Haven OnDemand pour une utilisation de manière synchrone ou asynchrone :

- Dans l'API synchrone, Haven OnDemand traite la requête immédiatement, et renvoie une réponse unique, contenant vos résultats.
- Dans l'API asynchrone, Haven OnDemand reçoit la requête, et retourne un identifiant de l'opération qui vous permet de suivre le traitement de la requête. Lorsque la requête est terminée, le message d'état renvoie également la réponse et les résultats.

L'API synchrone est idéale pour obtenir des résultats rapides. Vous pouvez l'utiliser pour les petites demandes. Et donc, dans le cas d'une perte de connexion, vous devrez soumettre à nouveau la demande pour obtenir les résultats.
L'API asynchrone, quant à elle, est préférable pour les actions qui pourraient prendre beaucoup de temps, ou si vous n'avez pas besoin du résultat immédiatement. Vous pouvez également l'utiliser pour soumettre plusieurs opérations dans une requête. Pour plus d'information, https://dev.havenondemand.com/docs/AsynchronousAPI.htm

Barcode Recognition

URL de l'API = https://api.havenondemand.com/1/api/sync/recognizebarcodes/v1
Paramètre d'authentification = apikey
Paramètre obligatoire = fichier à traiter / référence d'un document indexé / URL d'une image
Paramètre optionnel = orientation du code-barre / type du code barre
Ce qui donne pour une image de code barre :
https://www.havenondemand.com/sample-content/barcode/bc9.jpg
https://api.havenondemand.com/1/api/sync/recognizebarcodes/v1?url=
https://www.havenondemand.com/sample-content/barcode/bc9.jpg
&apikey=votre_apikey

Face Recognition

https://dev.havenondemand.com/apis/detectfaces#overview

Image Recognition

https://dev.havenondemand.com/apis/recognizeimages#overview

Hewlett Packard Enterprise

EXTRAIRE DU TEXTE D'UNE IMAGE AVEC L'OCR

Appel de l'API "OCR document" depuis Android

Dans cet exemple, nous allons décrire la mise en œuvre de l'API « OCR Document », c'est-à-dire, l'extraction du texte à partir d'une image. Cette API est identifiée par une application nommée "ocrdocument". L'API nécessite une requête POST HTTP au serveur avec l'URL complète qui se compose de 4 parties:

- Le domaine : https://api.idolondemand.com/1/api
- Le mode : /async or /sync
- L'application : /ocrdocument
- La version de l'API : /v1

L'URL complète ressemblerait à ceci :
"https://api.idolondemand.com/1/api/async/ocrdocument/v1"
Les paramètres à passer dans la requête POST sont les suivants :

- L'apikey : Votre clé API
- Le fichier : Un fichier contenant l'image à traiter.
- Le mode (optionnel): scene_photo | document_photo | document_scan | subtitle

Nous allons créer une requête POST avec ces paramètres dans notre code Android :

```
String idol_ocr_service = "https://api.idolondemand.com/1/api/async/ocrdocument/v1";
URI uri = new URI(idol_ocr_service);
HttpPost httpPost = new HttpPost();
httpPost.setURI(uri);
MultipartEntityBuilder reqEntity = MultipartEntityBuilder.create();
        reqEntity.setMode(HttpMultipartMode.BROWSER_COMPATIBLE);
reqEntity.addPart("apikey", new StringBody(apikey, ContentType.TEXT_PLAIN));
reqEntity.addBinaryBody("file", new File(mImageFullPathAndName);
reqEntity.addPart("mode", new StringBody("document_photo", ContentType.TEXT_PLAIN));
httpPost.setEntity(reqEntity.build());
HTTPClient httpClient = new DefaultHttpClient();
HttpResponse response = httpClient.execute(httpPost);
```

Après l'envoi de la demande asynchrone POST ci-dessus, nous nous attendons à recevoir une réponse de texte JSON formatée à partir du serveur. La réponse contient un identifiant nommé "jobID". Nous allons utiliser la valeur du JobID plus tard dans une requête GET pour aller chercher le résultat réel.

Voici une réponse typique avec un jobID du serveur:

```
{"jobID":"usw3p_89f2563e-8742-4def-ae25-9162a6a35c9a"}
```

Nous avons juste besoin d'analyser la chaîne JSON pour obtenir la valeur du jobID puis l'utiliser pour aller chercher le résultat réel :

```
String response = {"jobID":"usw3p_89f2563e-8742-4def-ae25-9162a6a35c9a"}
JSONObject mainObject = new JSONObject(response);
String jobID = mainObject.getString("jobID");
```

L'URL pour récupérer le résultat réel identifié par un jobID est définie comme suit : https://api.idolondemand.com/1/job/result/[jobID]

Nous allons créer une requête GET avec un jobID dans notre code Android :

```
String idol_job_result = "https://api.idolondemand.com/1/job/result/";
String url = idol_job_result + jobID + "?";
url += "apikey=" +  apikey;
URI uri = new URI(url);
HttpGet httpGet = new HttpGet();
httpGet.setURI(uri);
HTTPClient httpClient = new DefaultHttpClient();
HttpResponse response = httpClient.execute(httpGet);
```

Si la requête GET a réussi, la réponse du serveur est une chaîne de texte JSON formatée. Voici une réponse typique :

```
{
  "actions": [
    {
      "result": {
        "text_block": [
          {
            "text": "the text scanned and found in the image...",
            "left": 0,
            "top": 0,
            "width": 1080,
            "height": 1920
          }
        ]
      },
      "status": "finished",
      "action": "ocrdocument",
      "version": "v1"
    }
  ],
  "jobID": "usw3p_89f2563e-8742-4def-ae25-9162a6a35c9a",
  "status": "finished"
}
```

Nous allons utiliser Android JSONObject à nouveau pour analyser la réponse et extraire le texte que nous voulons afficher.

```
String foundText = "";
JSONObject mainObject = new JSONObject(response);
JSONArray textBlockArray = mainObject.getJSONArray("actions");
if (textBlockArray.length() > 0) {
for (int i = 0; i < textBlockArray.length(); i++) {
JSONObject actions = textBlockArray.getJSONObject(i);
        JSONObject result = actions.getJSONObject("result");
        JSONArray textArray = result.getJSONArray("text_block");
        int count = textArray.length();
        if (count > 0) {
          for (int n = 0; n < count; n++) {
JSONObject texts = textArray.getJSONObject(n);
foundText += texts.getString("text");
}
        }
      }
  }
```

Voilà tout ce dont vous avez besoin pour mettre en œuvre l'API OCR dans votre application. La valeur de la variable foundText est le texte reconnu de l'image.

3 - Ressources

Face Recognition :
https://dev.havenondemand.com/apis/detectfaces#overview
Image Recognition :
https://dev.havenondemand.com/apis/recognizeimages#overview
OCR Document :
https://dev.havenondemand.com/apis/ocrdocument#overview
Barcode Recognition :
https://dev.havenondemand.com/apis/recognizebarcodes#overview

Haven OnDemand : reconnaissance vocale

Que les données se présentent sous la forme d'enregistrements dans les centres d'appels, de commandes de recherche vocale comme Siri, ou de discussions (conférences, webinars, etc.), les technologies d'analyse de la parole ont pris de plus en plus d'importance.

Les technologies mises à disposition par Haven OnDemand sont basées sur des algorithmes d'intelligence artificielle, de deep learning et de réseaux de neurones. Ces technologies fournissent une qualité de transcription encore meilleure que les algorithmes statistiques. Elles se fondent sur le traitement de milliers d'heures d'enregistrements pour apprendre les structures présentes dans le langage. Ce processus d'apprentissage automatique produit des modèles linguistiques qui sont ensuite utilisés pour la transcription. L'API Speech Recognition crée une transcription textuelle d'une conversation présente dans une vidéo ou une piste audio. Ce texte peut être ensuite utilisé, par exemple, pour une indexation de ces contenus et fournir des fonctions de recherche et d'analyse sur le contenu des vidéos ou des fichiers audio.

1 - Son fonctionnement

L'entrée est constituée par la source vocale (piste audio, vidéo, etc.). Les formats supportés par l'API sont nombreux :
https://dev.havenondemand.com/docs/ImageFormats.html.
La retranscription fonctionne dans différentes langues et pour des sources audio variées. L'API fonctionne uniquement en mode asynchrone. Vous devrez spécifier la source en entrée (fichier, référence, url), l'intervalle et la langue. Vous aurez plusieurs possibilités comme évoqué plus haut :

- Pack de langue standard : ar-DE, en-AU, etc.
- Pack de langue téléphonique : en-GB-tel, fr-FR-tel

L'API retournera une réponse JSON selon un modèle défini.

2 - Exemple d'utilisation

URL de l'API = https://api.havenondemand.com/1/api/async/recognizespeech/v1
Paramètre d'authentification = apikey
Paramètre obligatoire = fichier à traiter / référence d'un document indexé / URL d'une image
Paramètre optionnel = intervalle de segmentation / pack de langue
Ce qui donne pour une video :
https://www.havenondemand.com/sample-content/videos/hpnext.mp4
https://api.havenondemand.com/1/api/async/recognizespeech/v1?url=
https://www.havenondemand.com/sample-content/videos/hpnext.mp4
&language=en-US&apikey=votre_apikey

Utilisation de Haven OnDemand REST API

```
TranscriptContent = null;
var _checkJobStatus = function (jobID) {
  HTTP.post('https://api.idolondemand.com/1/job/status/' + jobID, {
    params: {
      apikey: "Votre API key"
    }
  }, function (error, result) {
    if (error) {
      console.log('Error when checking job status : ' + error)
    } else if (result.data.actions[0].result) {
      TranscriptContent = result.data.actions[0].result.document[0].content;
      console.log('Transcript content result : ' + TranscriptContent);
    } else {
      Meteor.setTimeout(function () {
        _checkJobStatus(result.data.jobID)
      }, 2200)
    }
  })
};
Meteor.methods({
  uploadFile: function (file) {
    var fd = new FormData;
    fd.append('file', {
      contentType: 'audio/wav',
      filename: 'longer.wav',
      data&colon; file
    });
    var generated = fd.generate();
    HTTP.post('https://api.idolondemand.com/1/api/async/recognizespeech/v1', {
      params: {
        apikey: "Your API key"
      },
      headers: generated.headers,
      content: generated.body
    }, function (error, result) {
      if (error) {
        console.log('Error when posting to Haven OnDemand :' + error);
      } else if (result) {
        console.log('Success when posting to Haven OnDemand :' + result.data.jobID);
        _checkJobStatus(result.data.jobID);
      }
    });
  },
  returnTranscriptContent: function () {
    return TranscriptContent;
  }
});
```

Nous convertissons le Blob dans un fichier multipart / formDate, puis postons à l'API. Nous utilisons la méthode _checkJobStatus pour vérifier si la transcription est prête, en utilisant JobId. Nous avons mis en TranscriptContent avec la valeur du résultat de la tâche. UploadFile est appelé à partir du client, par l'intermédiaire du module _encodeAudio.

```
Template.home.events({
  'click #record': function () {
    Modules.client.recordAudio({action: 'start'});
  },
  'click #stop': function () {
    Modules.client.recordAudio({action: 'stop'});
  }
});
```

Nous obtenons une transcription textuelle d'un flux audio enregistré à partir de votre navigateur.

Pour en savoir plus :

https://dev.havenondemand.com/apis/recognizespeech#overview

Hewlett Packard Enterprise

Haven OnDemand : les Graphes

Un graphe est un ensemble de points qui sont reliés par des traits ou des flèches. L'ensemble des traits entre ces points (ou nœuds) forme une figure qui peut s'apparenter à un réseau. On pourra mentionner que les liens sont orientés ou symétriques et indiquent la nature de la relation existant entre ces points.

Les Graphes sont utilisés pour créer, visualiser et explorer les données et les relations pouvant exister entre différentes entités. Un Graphe pouvant se définir comme un ensemble de nœuds reliés entre eux, chaque nœud pourra faire référence à une entité. Dans le domaine de l'analyse de l'information, ces entités pourront représenter des concepts, mais aussi des personnes, des lieux, voire même toute autre donnée qui sera porteuse de sens par rapport à d'autres entités. Il sera également possible de positionner dans ces graphes des informations élémentaires ou métadonnées qui permettront de représenter les caractéristiques d'autres entités. Par exemple, pour une page web, les noms de personnes citées ou les thématiques abordées pourront être représentés dans un graphe dans lequel les métadonnées de la page seront reliées entre elles. Les nœuds pourront donc être des éléments d'un document (les destinataires ou expéditeurs d'un email par exemple) et sont connectés les uns aux autres. Pour illustrer le concept de Graphe, la plate-forme Haven OnDemand fournit un exemple de graphe basé sur les données de Wikipedia, version anglaise. Dans ce graphe, chaque lien cliquable d'une page a été indexé et est représentable sous la forme d'un nœud. Le Graphe permet alors de visualiser de quelle façon les pages sont reliées entre elles et quel serait le plus court chemin qui relierait deux pages Wikipedia. Il permet ainsi aux développeurs de comprendre les relations entre chaque page et comment les sujets peuvent être reliés les uns aux autres. Ces API montrent des connexions sous la forme d'un graphe que l'on peut utiliser dans une application de plus haut niveau.

1 - Son fonctionnement

Pour réaliser et utiliser (dans les sens de navigation) des graphes, la plate-forme expose plusieurs API. Chacune de ces API peut être testée en ligne sur la structure de graphe provenant des liens Wikipedia.
Le nœud est, dans une approche graphe, un sommet du graphe. On parlera aussi de neighbors et de neighborhood. C'est à dire de voisins et de voisinage. Par exemple, le voisinage d'un sommet est l'ensemble de tous les sommets proches.
Voici l'ensemble des API disponibles autour des Graphes :

API	Description
Get Common Neighbors	trouve les voisins communs entre les nœuds qui ont été spécifiés par le développeur
Get Neighbors	retourne les voisins d'un ou plusieurs sommets. Par défaut, l'API trouve les voisins par similarité sur les nœuds déclarés
Get Nodes	Donne la liste de tous les sommets du graphe. Par défaut, l'API les classe par hiérarchie
Get Shortest Path	Cette API définit le chemin le plus court dans un graphe entre 2 sommets spécifiés
Get Subgraph	retourne le sous-graphe que l'on peut avoir sur un jeu de nœuds.
Suggest Links	suggère des nœuds proches d'un nœud défini par le développeur mais sur lesquels nous ne sommes pas connectés.
Summarize Graph	Cette API fournit un résumé du nombre de nœuds du graphe courant et les détails des attributs des nœuds.

2 - Exemple d'utilisation

Voici un exemple d'utilisation avec l'API « Get Common Neighbors » :
URL de l'API = https://api.havenondemand.com/1/api/sync/getcommonneighbors/v1
Paramètre d'authentification = apikey
Paramètre obligatoire = ID ou Nom de la source / ID ou Nom de la cible
Paramètre optionnel = Nombre maximal de résultat / Nombre minimum de points communs entre la source et la cible
Ce qui donne pour l'exemple de Wikipedia en anglais :
https://api.havenondemand.com/1/api/sync/getcommonneighbors/v1
&graph=wiki_eng
&max_results=5
&source_name=Barack Obama,François Hollande
&apikey=votre_apikey

Démarrage rapide

Les requêtes doivent spécifier un ou plusieurs nœuds pour lesquels vous voulez trouver les voisins communs :

- Pour trouver les « voisins sortants » d'un nœud (c'est-à-dire les « nœuds cibles » vers lesquels le « nœud source » d'entrée fait référence), utiliser les paramètres **source_names** ou **source_ids**.
- Pour trouver les « voisins entrants » d'un noeud (c'est-à-dire les « nœuds sources » qui pointent vers le « nœud cible » d'entrée), utiliser les paramètres **target_names** ou **target_ids**.

Pour illustrer ce fonctionnement, prenons l'exemple suivant qui renvoie **cinq pages Wikipédia** qui sont liées à partir d'au minimum l'une des pages suivantes : **Le Royaume-Uni, la France,** ou **l'Italie**.

POST /1/api/[async|sync]/getcommonneighbors/v1?**source_names**=United Kingdom,France, Italy&**max_results**=5

L'API renvoie tous les voisins des nœuds que vous spécifiez, avec la valeur des points communs (« commonality ») entre les nœuds, qui indique le nombre de lien entre les nœuds d'entrée et leur voisin. Par exemple, une valeur de trois points communs (commonality = 3) montre que le nœud indiqué est lié (ou voisin) à 3 des nœuds d'entrée.

Spécifier « Source » et « nœuds cibles »

Il est possible de spécifier les deux types de nœuds (source et cible) dans la même requête. Par exemple, la requête suivante renvoie les pages qui sont liées à la page «Royaume-Uni» et qui ont également un lien vers la page "France".

POST /1/api/[async|sync]/getcommonneighbors/v1?**source_names**=United Kingdom&**target_names**=France&**min_commonality**=2

Pour en savoir plus

https://dev.havenondemand.com/apis?category=Graph%20Analysis
https://dev.havenondemand.com/docs/HowTo_Graph.html

Dossier réalisé par **Thibaud de Rouzé**,
Sales Engineer HPE Software Big Data
thibaud.de-rouze@hpe.com

Raspberry : intégration d'opencv en java

Voici un article visant à intégrer opencv sur la plateforme Raspberry. Ce projet est issu d'une fonctionnalité développée pour un robot à chenille. L'objectif ? Faire déplacer le robot en suivant son maître !

Bertrand LANNEAU
développeur

Ce robot a été conçu sur une Raspberry Pi B+ et une carte Arduino UNO. Le Raspberry étant la « tête pensante », je l'ai dédié à gérer la vidéo, les traitements multi-thread et la gestion de la communication série. L'Arduino se charge du pilotage des moteurs via des signaux de commande PWM. De plus, il a la tâche de remonter au Raspberry les informations de vitesses, tension batterie,...
L'intégration d'opencv fait suite au fonctionnement initial de mon robot puisque celui-ci est pilotable via une application Java sur Android en WiFi. Voici dans cet article les différentes étapes d'installation et d'utilisation de cette très complète librairie qu'est opencv. Bonne lecture.

La librairie opencv et détection

Opencv est une librairie open source (BSD) de traitement d'images qui offre une grande palette d'outils initialement développés par Intel. Depuis 2008, la société Willow Garage met à jour, enrichit et ouvre le déploiement de cette librairie à de multiples plateformes. Parmi les traitements possibles, nous allons nous intéresser aux détections de formes. La méthode utilisée est celle proposée par Viola et Jones. Elle consiste à reconnaître une cible sur une image en s'appuyant sur un fichier généré à partir de milliers de clichés. Suite à une compilation de fonds ou « background » et d'images à détecter « foreground », un fichier xml déterminera la procédure à suivre pour détecter la forme voulue. On parle d'apprentissage pour la génération de ce fichier, que l'on peut assimiler au Machine Learning. Il est basé sur un fonctionnement en cascade, ce qui permet une plus grande rapidité ce calcul. L'image à traiter est segmentée en plusieurs zones auxquelles s'applique le processus de détection. Fig.1.

Installations préalables

Mise à jour de l'OS du Raspberry :
sudo apt-get update
sudo apt-get upgrade
sudo rpi-update

Installez les ressources nécessaires à la compilation et à l'exécution :
sudo apt-get install ant

Utilisons la dernière version de la jdk
sudo apt-get install openjdk-8-jdk

Installez ensuite scrupuleusement ces composants. Répondez "y" lorsqu'il vous le sera demandé.
sudo apt-get install build-essential cmake pkg-config libpng12-0 libpng12-dev libpng++-dev libpng3
sudo apt-get install libpnglite-dev zlib1g-dbg zlib1g zlib1g-dev pngtools libtiff4-dev
sudo apt-get install libtiff4 libtiffxx0c2 libtiff-tools libjpeg8 libjpeg8-dev libjpeg8-dbg libjpeg-progs ffmpeg libavcodec-dev
sudo apt-get install libavcodec53 libavformat53 libavformat-dev libgstreamer0.10-0-dbg libgstreamer0.10-0 libgstreamer0.10-dev

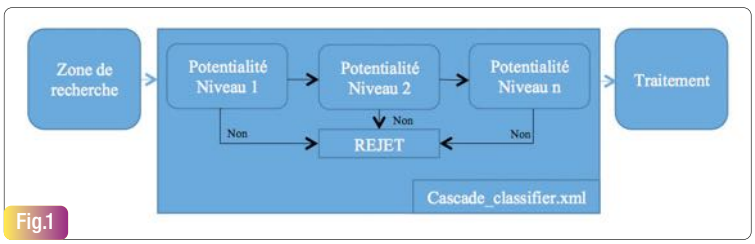


Fig.1

sudo apt-get install libxine1-ffmpeg libxine-dev libxine1-bin libunicap2 libunicap2-dev libdc1394-22-dev libdc1394-22 libdc1394-utils
sudo apt-get install swig libv4l-0 libv4l-dev

Si vous utilisez la caméra du Raspberry, vous aurez besoin au préalable d'installer les drivers nécessaires :
wget http://www.linux-projects.org/listing/uv4l_repo/lrkey.asc && sudo apt-key add ./lrkey.asc
sudo nano /etc/apt/sources.list

Copiez la ligne suivante dans le fichier ouvert puis sauvegardez et fermez.
deb http://www.linux-projects.org/listing/uv4l_repo/raspbian/ wheezy main
sudo apt-get update

Installation des drivers :
sudo apt-get install uv4l uv4l-raspicam
sudo apt-get install uv4l-raspicam-extras

Et pensez à activer la caméra dans le menu principal du Raspberry.
Ajoutez votre variable d'environnement Java :
sudo nano /etc/environment

Puis ajoutez le chemin correct, propre à votre système. Pour ma part :
JAVA_HOME=/usr/lib/jvm/jdk-8-oracle-arm-vfp-hflt

Installation d'opencv sur Raspberry

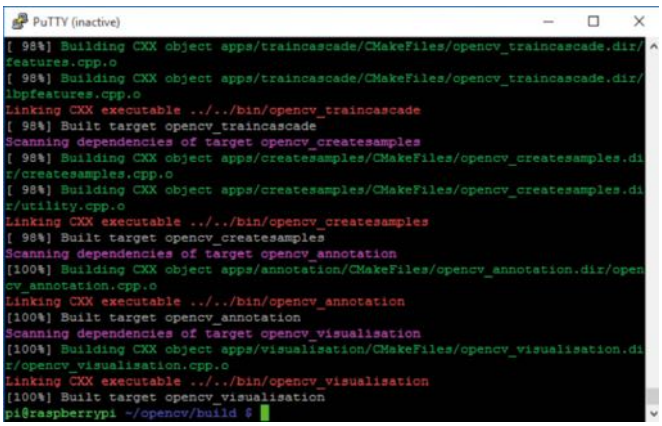
Téléchargez le répertoire d'installation :
git clone git://github.com/ltseez/opencv.git

Créez votre répertoire de compilation :
cd opencv
mkdir build
cd build

Et lancez la commande suivante :
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_EXAMPLES=OFF –D BUILD_PNG=ON ..

Vérifiez dans les logs si vous n'avez pas d'erreur ou de fichiers introuvables concernant l'installation Java ou JNI. Si c'est le cas, faites ceci :
export JAVA_HOME=/usr/lib/jvm/jdk-8-oracle-arm-vfp-hflt

Avant de lancer la commande qui suit, assurez-vous d'avoir alimenté votre Pi sur secteur car il y en a pour quelques heures...
make



Une fois terminé, vous aurez besoin des deux fichiers générés suivants : lib/libopencv_java2xxx.so (xxx étant le numéro de version d'opencv que vous avez téléchargé)

A mettre dans :
/usr/lib/libopencv_java2xxx.so
sudo cp lib/libopencv_java2xxx.so /usr/lib/
Puis le fichier opencv-2xxx.jar.
Récupérez-le et importez-le en tant que librairie dans votre environnement de développement. Pour ma part, j'utilise Eclipse.
Votre Pi est maintenant prêt à recevoir les instructions nécessaires pour faire tourner opencv.

Développement de votre application

En premier lieu nous allons mettre en place une classe qui sera appelée depuis le « main » de votre application.

```
public final class FaceDetect {
    run()
    {
  //Insérez votre code ici
    }
}
```

Intégrez le chargement de la librairie :
System.*loadLibrary*(Core.***NATIVE_LIBRARY_NAME***);

Puis instanciez les classes et variables suivantes:
// Classe fournissant les séquences d'images à traiter depuis votre caméra.
L'argument de type integer permet de pointer vers la caméra souhaitée.
final VideoCapture videoCapture = **new** VideoCapture(0);
//Classe permettant d'effectuer les traitements sur images.
final Mat mat = **new** Mat();
//Définissez une taille à vos images capturées. En ajustant ces valeurs, vous pouvez gagner un temps considérable de calcul.
final Size taille = **new** Size(260, 180);
//Compteur d'images avec visages détectés.
int nbImagesAvecFace = 0;
//Classe invoquant le fichier xml cascade pour la detection. En argument, le chemin de votre fichier.
CascadeClassifier faceDetector = **new** CascadeClassifier("/home/pi/opencv/data/haarcascades/haarcascade_frontalface_alt.xml");
//Cette classe récupère les images avec les objets détectés
MatOfRect faceDetections = **new** MatOfRect();

Maintenant, nous allons utiliser une boucle qui permettra de capturer une nouvelle image à chaque fois qu'elle sera disponible:

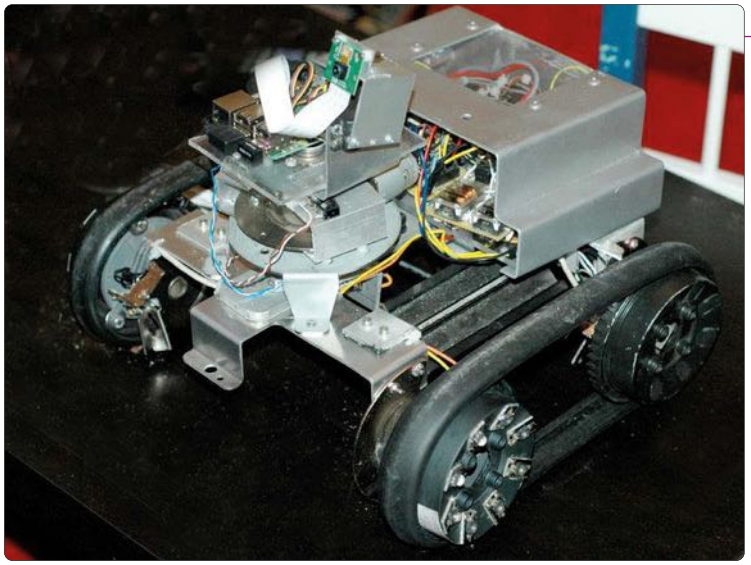
```
while (videoCapture.read(mat))
{
    //Traitement des images
}
```

Dans cette boucle, nous allons dans un premier temps retravailler l'image capturée.
Mat matTraitement = **new** Mat();
matTraitement.convertTo(matTraitement, CvType.***CV_8U***); *// Paramétrage de 8bits/pixel*
Imgproc.*resize*(mat, matTraitement,taille); *//Copie de l'image capturée dans sa coquille avec la dimension voulue.*

Et voilà la fonction qui va pouvoir effectuer les traitements de détections. Je vais décrire plus bas les arguments de cette fonction.
faceDetector.detectMultiScale(matTraitement,faceDetections,1.1,1,1,new Size(10,10),new Size(40,40));
Suite à l'appel de cette fonction, "faceDetection" contiendra les éventuelles images contenant un visage.
On va donc pouvoir récupérer les captures qui ont matché dans une boucle « for ».

```
for (Rect rect : faceDetections.toArray())
 {
      System.out.println("DETECTION!");
      nbImagesAvecFace ++;
 }
```

Si vous souhaitez dessiner un rectangle autour de la cible et récupérer les coordonnées du visage sur l'image ainsi que sa taille, la librairie opencv a mis à notre disposition la fonction suivante :
Core.*rectangle*(videoMatGray, new Point(rect.x, rect.y), new Point(rect.x + rect.width, rect.y + rect.height),new Scalar(0, 255,0)); Enfin, vous souhaitez enregistrer l'image sur votre Raspberry ?
Highgui.*imwrite*("Capture_"+ nbImagesAvecFace +".jpg",mat);
Sans avoir défini un chemin au nom de votre image, elle sera enregistrée par défaut dans le répertoire de votre application.
Pour terminer votre classe, après votre boucle while, libérez votre caméra.
videoCapture.release();



Enfin, lancez votre classe FaceDetect dans votre « main » :

```
public static void main(String[] args) throws Exception
{
    FaceDetect.run();
}
```

Compilation de votre code.

Pour tester l'installation d'opencv sur votre Pi, vous avez deux possibilités.

Méthode 1

Compilez et lancez directement votre code depuis votre Raspberry en utilisant les commandes suivantes :

javac –cp .:* OpencvTest.java

java –cp .:* OpencvTest

Méthode 2

Utilisez votre plateforme de développement et exportez votre projet en un .jar exécutable.

Export de votre projet depuis Eclipse :

Cliquez droit sur votre projet puis sélectionnez **Run/Debug Setting**

Ensuite dans la zone **Main Class** cliquez sur **Search** et sélectionnez votre classe principale et validez.

De nouveau clic droit sur votre projet puis **Exporter...**

Déroulez Java et choisissez **Runnable JAR file** Fig.2

Cliquez sur **Next** et dans la zone « **Launch configuration** », récupérez votre configuration générée à l'étape précédente.

Donnez un nom et un chemin à votre .jar dans la zone **Export Destination.**

Enfin, cliquez sur **finish**.

Récupérez et collez le .jar généré dans le répertoire /home/pi/opencv/ de votre Raspberry.

Lancez votre application avec cette commande :

sudo java -Djava.library.path=/usr/lib -jar opencv/OpencvTest.jar

//OpencvTest étant le nom que vous avez donné à votre jar exécutable.

Si tout s'est bien passé, vous pouvez tester votre programme et l'améliorer en modifiant quelques paramètres de la fonction « detectMultiScale() ».

Descriptif de la fonction « detectMultiScale() ».

Nous allons maintenant nous intéresser aux arguments de la fonction principale de détection de forme **detectMultiScale()**. Vous avez la possibilité de modifier certains paramètres afin d'affiner la détection en fonction de votre environnement.

Cette fonction permet donc de nous retourner un tableau de MatOfRect de taille égale au nombre de détections par image analysée.

detectMultiScale(Mat **image**, MatOfRect **objects**, double **scaleFactor**=1.1, int **minNeighbors**=3, int **flags**=0, Size **minSize**=Size(), Size **maxSize**=Size());

Argument 1 :**image**, type Mat() :

Image d'entrée à analyser par la fonction.

Argument 2 :**object**,type MatOfRect()

Vecteur de rectangles où chaque rectangle contient l'objet détecté.

Argument3 :**scaleFactor**, type Double.

Facteur de redimensionnement de l'image source. Lors du traitement, l'image est redimensionnée à plusieurs reprises (type pyramidale).

Plus vous augmentez ce paramètre, plus le temps de calcul sera long mais plus vous augmentez vos chance de détecter plusieurs sujets sur la même image.

Argument 4 : **minNeighbors**, type integer

Paramètre spécifiant le nombre de rectangle voisins que chaque rectangle candidat devrait avoir à conserver.

Argument 5 : **flags**, type integer.

Non utilisé pour les versions récentes. Mettre 1.

Argument 6 : **minSize**, type Size.

Définition de la taille minimum de la cible. En dessous de cette taille, tous les éléments seront rejetés.

Argument 7 : **maxSize**, type Size.

Définition de la taille maximale de la cible. Au-dessus de cette taille, tous les éléments seront rejetés.

Nous voilà arrivé au terme de cet article. En espérant qu'il vous sera utile pour vos projets de détection sur Raspberry.

Voici quelques liens qui m'ont aidé à mettre en place la librairie opencv :

http://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained/

http://stefanshacks.blogspot.fr/2015/05/build-opencv-with-java-bindings-on_6.html

Et si vous voulez en savoir plus sur mon projet robot, c'est ici :

https://myexperimentalrobot.wordpress.com/

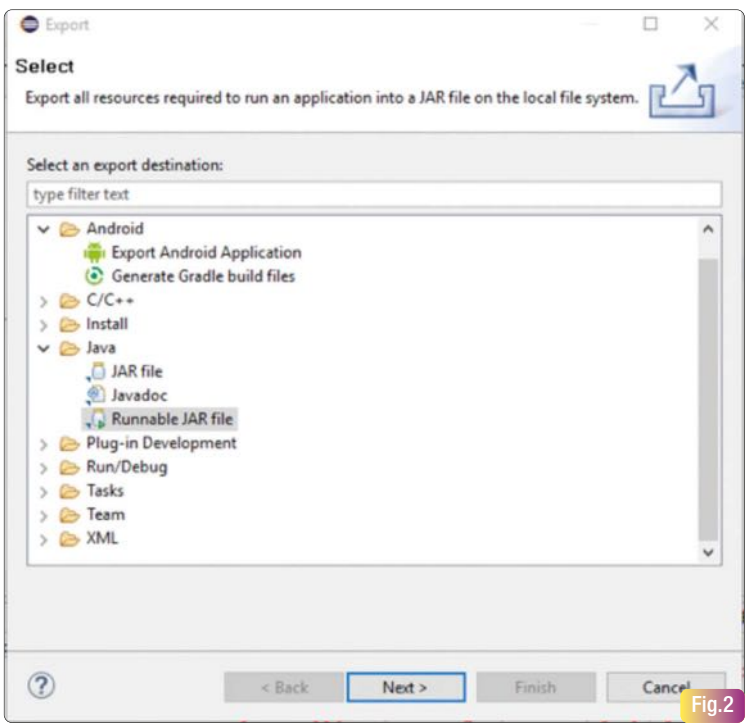


Fig.2

IoT : connecter une usine avec un Arduino 1ère partie

Les objets connectés se répandent de plus en plus dans notre monde. On voit ainsi de nombreux produits grand public se vendre, comme les montres connectées, ou les capteurs d'activités. Les entreprises découvrent avec joie ces nouveaux appareils, qui leur permettent de répondre à des besoins spécifiques.

Vincent Piard,
Expert Technique,
SQLI Nantes

Aurélien Fourmi,
Expert Technique,
SQLI Nantes

Au sein du pôle IOT de l'agence SQLI Nantes, un client nous remonta un besoin lié à la loi sur la pénibilité, adoptée en France. Cette loi demande aux entreprises de prendre en compte certains facteurs de risques liés à des contraintes physiques, à un environnement de travail agressif. Cela suppose en premier lieu de mesurer l'activité des employés. Les objets connectés interviennent ici, car ils permettent de mesurer des données entre le monde physique (ici l'environnement de travail) et le système d'information de l'entreprise. La réflexion a abouti à la construction d'un objet, capable d'identifier un employé et de déterminer ses heures de présence sur un poste de travail dit "pénible". C'est cet objet que nous allons voir plus en détail dans la suite de cet article.

Liste de courses

- Une boîte de type tupperware (2 €) ;
- Des badges RFID (1,45 € le badge) ;
- Une batterie (pour une boîte autonome) (10 €) ;
- Une mini board (4 €) ;
- Un Arduino Yun (75 €) ;
- Un capteur de présence (4 €) ;
- Un lecteur RFID (5 €) ;
- 15 fils conducteurs ;
- 2 résistances de 120 Ohm ;
- Cable USB pour la batterie ;
- Une plaque de bois ;
- 4 vis diamètre 3mm (2,90 € le paquet de 100) ;
- 1 LED Rouge ; 1 LED Verte.

Soit 110 €.

Prototyper son Arduino

Le plus compliqué quand on est développeur et que l'on commence à se frotter au monde de l'IOT, c'est l'apprentissage de l'électronique. En effet, en tant qu'informaticien, on est perpétuellement dans un monde virtuel et là, par définition, l'IOT nous confronte au monde réel. Je ne sais pas pour vous, mais personnellement mes dernières notions d'électronique remontent aux cours d'EMT (Education Manuelle et Technique) de 5ème donc il a fallu se retrousser les manches pour affronter ce nouveau monde. L'idée ici est de pouvoir mesurer le temps de présence d'une personne sur un emplacement géographique précis. Savoir détecter une présence est plutôt trivial, il suffit de voir un couloir d'immeuble s'allumer automatiquement quand il détecte votre présence. La difficulté sera de déterminer le temps de présence.

Choisir son contrôleur

L'une des premières choses à faire sur votre montage est de déterminer quelle carte vous servira de contrôleur. Les deux cartes les plus connues actuellement sont les l'Arduino et le Raspberry Pi. Bien que souvent citées de concert, ces deux cartes sont fondamentalement différentes et ne répondent pas du tout au même usage. L'Arduino est un microcontrôleur tandis que le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM. En d'autres termes, le Raspberry peut être vu comme un petit ordinateur alors que l'Arduino est beaucoup plus bas niveau et doit être utilisé pour des tâches basiques.

Le choix de ces cartes peut être influencé par la topologie de votre montage et notamment le but de votre montage. Par exemple si vous voulez mesurer la température de la pièce vous pouvez le faire avec les deux cartes mais concernant le Raspberry Pi cela reviendrait à sortir la Ferrari pour aller chercher son pain. Vous l'aurez compris, il faut choisir la carte en fonction de son usage. N'hésitez pas à bien vous renseigner car cette étape est cruciale. En ce qui nous concerne, nous avons opté pour un Arduino Yun qui possède le gros avantage d'avoir le Wifi intégré en natif. Avec le recul ce choix n'est peut-être pas le plus pertinent mais cet article est là afin que vous ne fassiez pas les mêmes erreurs que nous. Nous vous conseillons peut-être plus la série LaunchPad chez Texas Instrument qui est beaucoup moins chère. Le seul gros bémol concernant cette carte est le manque de communauté (très performante pour l'Arduino). Autre différence entre l'Arduino et la LaunchPad, le code est légèrement différent mais assez facilement portable, ce n'est pas un simple copier-coller mais il y a peu de choses à modifier.

Comment choisir son sensor

Maintenant que le choix de la carte est fait, il est temps de voir quels capteurs utiliser pour arriver à notre but : mesurer le temps de présence. Commençons par le début : le capteur permettant de savoir si quelqu'un se trouve sur le poste de pénibilité. Pour cela nous avons choisi le SODIAL Module détecteur de Mouvement infrarouge pyroélectrique PIR. Il s'agit d'un détecteur de mouvements et non d'un capteur de présence. La différence entre ces deux sensors est de taille. En effet un capteur de présence indique si une présence est là ou pas. Le résultat est de type binaire. Alors que le détecteur de mouvement ne réagit qu'aux déplacements de l'acteur. La question peut se poser alors de notre choix d'un détecteur de mouvement qui semble contraire à notre but. Les capteurs de présence que nous avons recensés ne remplissaient pas notre cahier des charges. En effet, le capteur/détecteur doit pouvoir être placé loin de l'acteur, or la portée la plus longue d'un capteur que nous avons pu constater est de 5 mètres. Ceci résout la problématique des plafonds souvent élevés dans les locaux du tertiaire. Nous avons donc dû consentir à utiliser un détecteur de mouvement ce qui ne sera pas sans conséquence sur l'algorithme de l'Arduino. A signaler qu'il existe peut-être un capteur de présence qui détecte à longue distance mais l'une des difficultés récurrentes lors de l'approche du monde des objets connectés est de déterminer si tel ou tel composant technique répond à notre besoin. Pour cela il faut lire les documentations électroniques qui ne sont pas toujours faciles d'accès pour le profane.

Comment identifier un utilisateur

Notre cahier des charges indiquait aussi une identification de l'acteur. Pour cela nous avons opté pour le lecteur RFID RDM6300 de chez Asiawill. Il est composé d'une carte électronique et d'une bobine recevant le signal. Le lecteur utilise wiegand comme protocole de communication. Une des difficultés du code présent sur l'Arduino sera de décoder ce protocole. Pour le reste du montage, nous avons eu besoin de matériel plus commun. Nous

avons utilisé la breadboard du starterkit d'Arduino, 15 fils conducteurs, 2 résistances de 120 Ohms chacune, 4 vis de diamètre 3mm, une LED verte et une LED rouge. Voici un schéma du montage provisoire : Fig.1. Le composant bleu est le détecteur de présence, le vert la carte du lecteur RFID et la bobine rouge est justement le lecteur RFID. Le schéma est pas une référence en matière de conception électronique, il est juste là pour illustrer les branchements des différents capteurs sur l'Arduino. Sur l'Arduino, nous avons donc deux LED branchées, une patte du lecteur RFID et le détecteur de présence. Les résistances sont couplées avec les LED. Elles sont là pour indiquer des états du détecteur de présence. Par exemple quand une session commence au badgeage RFID, la LED verte se met à clignoter.

Les pièges à éviter

Les difficultés que nous avons rencontrées lors de la réalisation de ce montage sont nombreuses. A commencer par notre méconnaissance de l'électricité/électronique. Le sens de branchement des LED, la connexions des différents capteurs ainsi que la compréhension du fonctionnement des capteurs et de l'Arduino ne sont que des exemples parmi d'autres. Le gros avantage à prendre un Arduino est la communauté qui se trouve derrière et du coup l'énorme documentation disponible. Par exemple pour le branchement du RFID nous nous sommes basés sur le travail suivant (http://tronixstuff.com/2013/11/19/Arduino-tutorials-chapter-15-rfid/). C'est la somme importante de documentations qui fait que l'apprentissage des branchements sur Arduino ne relève pas de la mission impossible.

En revanche nous avons été étonnés par la plupart des montages Arduino : en général les blogs s'arrêtent à cette étape, à savoir que le montage est fonctionnel et que le code fait bien ce que l'on veut. Ce qui est intéressant si l'on réalise un projet juste pour soi ou pour un petit comité. Mais qu'en est-il si vous voulez proposer votre montage en tant que vrai objet connecté finalisé ? A savoir un objet sans fils qui sortent de partout, avec une finition parfaite. Par exemple, imaginez les Google Glass avec les composants électroniques à nu et des câbles dans tous les sens : c'est moins professionnel. Cela va donc être notre nouvelle étape : rendre ce montage plus pratique à utiliser.

Industrialiser l'Arduino

La première idée quand nous avons voulu rendre ce montage plus ergonomique, a été de mettre l'ensemble dans une boîte. Dans ce contexte, le plus simple a été de modéliser une impression 3D pour créer une boîte sur mesure, collant parfaitement à notre usage. Pour cela nous avons utilisé le logiciel OpenSCAD qui offre le très grand avantage de pouvoir "coder" sa modélisation. L'inconvénient de ce logiciel de modélisation est qu'il est très insuffisant dans le cas de modélisations complexes, par exemple pour modéliser des visages ou des structures fines. Mais concernant notre besoin, à savoir modéliser une boîte, OpenSCAD y répond parfaitement.

Malgré nos maigres connaissances dans ce domaine, la prise en main a été assez rapide grâce à la possibilité de coder la modélisation.

Voici un court extrait de code. Celui-ci permet d'obtenir la modélisation du trou qui accueillera une LED ainsi que la partie RFID :

```
module trou_led(){
  hauteur_cube = epaisseur+3;
  hauteur_cylindre = epaisseur - 1 +2;
  damietre_led = 9;

  union(){
    translate([0,0,0])
    cylinder(r=(damietre_led/2)+1, h=hauteur_cylindre, $fn=100, center=true);
    translate([0,0,-hauteur_cylindre/2-hauteur_cube/2])
    cube(size=[damietre_led, 3, hauteur_cube], center=true);
  }
}

module rfid(){

  difference(){
    cube(size=[largeur_rfid+2, profondeur_rfid+2, hauteur_rfid], center = true);
    translate([0,2,1])
    cube(size=[largeur_rfid, profondeur_rfid, hauteur_rfid], center = true);
  }

}
```

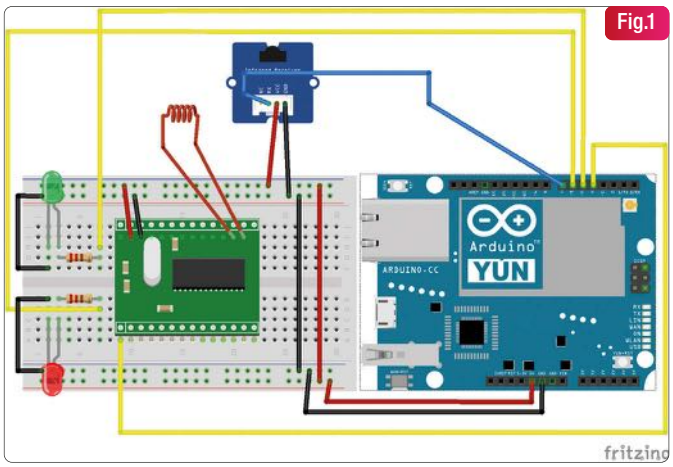


Fig.1

Le cahier des charges pour constituer une boîte efficace en tant que capteur de présence est le suivant :

- Pouvoir ouvrir la boîte ;
- La boîte doit contenir une breadboard ainsi que l'Arduino ;
- Une ouverture doit permettre de passer le capteur IR ;
- Une ou plusieurs ouvertures doivent être présentes à l'arrière pour laisser passer les branchements Arduino (RJ45, USB...) ;
- Une encoche doit être présente sur la boîte pour signifier aux utilisateurs l'endroit où ils doivent badger ;
- Deux encoches sur le couvercle doivent être présentes pour accueillir les deux diodes.

De plus, pour des raisons de coûts, la boîte doit être assez courte. Nous avons ainsi opté pour la solution de superposer la breadboard et l'Arduino. En effet, lors d'une modélisation 3D, il faut essayer de s'arranger pour "consommer" le moins de matière possible et il faut parfois réarranger sa modélisation.

Par exemple, la première version de la modélisation aboutissait à un devis d'environ 400€ pour une impression. Une fois le modèle modifié et retouché, le dernier devis tombe à 130€ soit une baisse de 67,5%. Malgré cela, le coût reste encore trop prohibitif pour un tel objet, 130€ pour une simple boîte cela reste un sacré investissement. La piste modélisation 3D a donc été abandonnée pour des raisons financières.

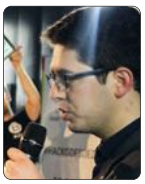
Nous avons donc opté pour une solution plus manuelle et après une courte réflexion, quoi de mieux que la Rolls des boîtes pour arriver à nos fins : à savoir une boîte Tupperware. Elle présente aussi de nombreux avantages :

- Elle n'est pas chère ;
- Elle est pratique ;
- Elle est facile à trafiquer.

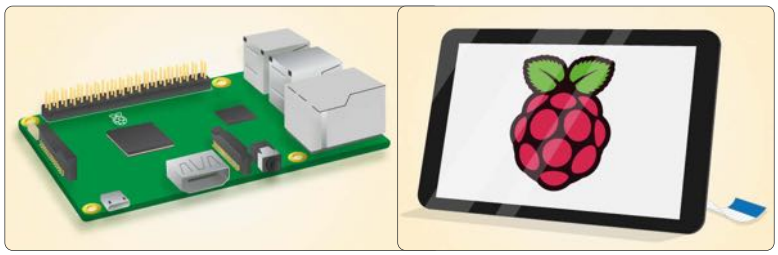
Suite dans le n°199

Créer votre tablette tactile avec Node.js 1ère partie

Avec Javascript tout devient possible, ou presque ! Nous allons donc prendre le contrôle d'une Raspberry Pi et de son écran tactile, afin de lancer au démarrage une application programmée en technologie Web. Le tout en partant d'une version de base de Linux (Raspbian lite).



Matthieu Bouilloux
Développeur web et applicatif, utilisant principalement des technologies issues du monde du web. Couvrant des domaines tel que le scrapping, le datamining au développement de site web sur mesure en passant par le développement d'application de bureau ou mobile. Intéressé par tout type de nouvelle technologie tel que l'Iot ou encore l'analyse des marchés boursier.
Site web: http://www.katlea-edition.com
Email: matthieu@katlea-edition.com



L'objectif est clairement de faire un « proof of concept » sur l'utilisation de Node.js pour faire un WebOS en mode « Quick & Dirty ». Mais aussi de développer des applications tout simplement en HTML/CSS et Node.js. Pour cela nous allons utiliser node-webkit (plus récemment appelé NW.js). Petit rappel, NW.js vous permet d'écrire des applications à l'aide des technologies Web, tout en appelant tous les modules Node.js dans le navigateur, lui-même basé sur Chromium.
Un petit tour du côté de l'assemblage de l'écran officiel 7 pouces du Raspberry Pi. Puis on passe à l'installation et à la configuration des paquets nécessaires de la Raspberry pour héberger notre applicatif. Et le meilleur pour la fin, une petite interface rapide en « material design » avec deux applications, un navigateur Internet et un Paint.

LE MATÉRIEL

Prérequis matériel

- De préférence une Raspberry Pi 3 (avec son alimentation secteur). Pour la Raspberry Pi 2 il vous faudra un adaptateur WiFi compatible (exemple : Adaptateur USB Sans fil N300 Micro, F7D2102 de chez Belkin) ;
- Un écran tactile : j'ai utilisé le modèle officiel 7 pouces dans cet article. Mais nous allons aussi voir comment configurer l'écran « Tontect » de 3,5 pouces ;
- Un câble RJ-45 pour accéder à votre raspberry et une carte microSD.

Prérequis logiciels

- Un client SSH ex : Putty http://www.putty.org/
- Un client SFTP ex : WinSCP https://winscp.net
- Votre éditeur Web favori.

Initialisation

Pour démarrer il faut installer « Raspbian lite » sur la carte micro-SD. Rien de bien compliqué, il suffit de télécharger l'image disque et de l'écrire sur la carte micro-SD à l'aide du logiciel fourni.
Rendez-vous sur : https://www.raspberrypi.org/downloads/raspbian/ pour télécharger l'image disque.
Et pour l'installation : https://www.raspberrypi.org/documentation/installation/installing-images/README.md
Insérer votre carte micro-SD dans la Raspberry et alimenter la via le port micro-usb. Attention la « Raspberry » ne boote que sur la carte micro-SD ; ne pas envisager de booter sur disque externe sans passer au préalable par la carte SD.

Premier démarrage

Connectez la Raspberry en RJ-45 à votre ordinateur et partagez votre connexion Internet. Si vous êtes déjà en RJ-45, connectez-la alors à votre box Internet. Assez rapidement la Raspberry est accessible en SSH à l'adresse : raspberrypi.local (port 22), avec comme login : « pi » et comme mot de passe « raspberry ».
Une fois que vous êtes connecté, on commence par une mise à jour du système et on redémarre la carte.

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

Installation de l'écran officiel 7 pouces en 3 étapes

- Connectez la nappe entre la Raspberry et l'écran cf. image 1 ;
- Puis les deux câbles rouge et noir fournis entre les deux cartes cf. image 2 ;
- Relier la sortie micro-USB de la Raspberry au connecteur USB de l'écran cf. image 3.


Fig.1

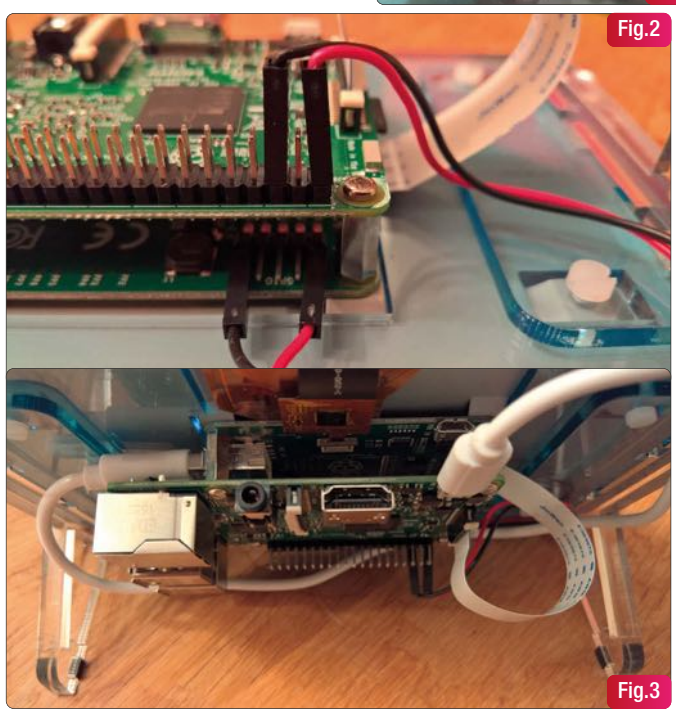


Fig.2 Fig.3

Alimentez désormais la Raspberry avec le cordon secteur via le connecteur micro-USB de l'écran.

Support d'écran

Vous trouverez des supports assez abordables par exemple sur http://www.kubii.fr/ . Fig.4

LA CONFIGURATION

Extension de la mémoire de la carte SD

On relance le SSH et on lance l'outil de configuration de Raspbian :

```
sudo raspi-config
```

Sélectionner : 1 Expand Filesystem, puis redémarrer. Chaque redémarrage implique de se reconnecter en SSH.

```
sudo reboot
```

Connectivité wifi

En vue de contrôler le WiFi en Node.js, on va désactiver la configuration automatique. On ouvre donc le fichier de configuration.

```
sudo nano /etc/network/interfaces
```

On commente la ligne suivante à l'aide de « # »

```
#wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

Et on redémarre

```
sudo reboot
```

Tester la connectivité wifi

Depuis la console pour le debug

```
sudo wpa_passphrase "NOM DU RESEAU" MOT_DE_PASSE > /home/pi/wpa-temp.conf && sudo wpa_supplicant -D nl80211,wext -i wlan0 -c /home/pi/wpa-temp.conf && rm /home/pi/wpa-temp.conf
```

Le nom du réseau est entre guillemets donc pas de souci particulier. Par contre, pour le mot de passe il faut échapper les caractères spéciaux par un « \ » (ce qui donne : « \@Password »). Pour information, la commande ci-dessus utilise le logiciel wpa_passphrase pour écrire la configuration dans un fichier, puis lui fournit le fichier en argument, et, enfin, détruit celui-ci.
Si tout se passe bien, vous verrez quelque chose comme ceci :

```
Successfully initialized wpa_supplicant
wlan0: Trying to associate with SSID 'NOM DU RESEAU'
wlan0: Associated with 7e:8f:b5:ce:f1:48
wlan0: WPA: Key negotiation completed with 7e:8f:b5:ce:f1:48 [PTK=CCMP GTK=CCMP]
wlan0: CTRL-EVENT-CONNECTED - Connection to 7e:8f:b5:ce:f1:48 completed [id=0 id_str=]
```

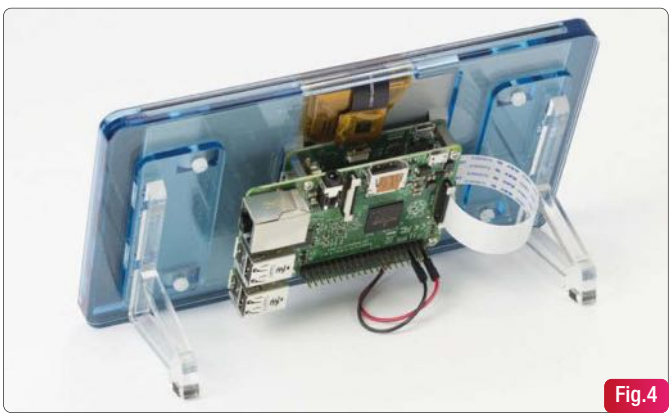


Fig.4

Pour revenir à la ligne de commande, utiliser les touches CTRL + C .
Pour lancer la connexion WiFi en background (daemon), ajouter l'option « -B »

```
sudo wpa_passphrase "NOM DU RESEAU" MOT_DE_PASSE > /home/pi/wpa-temp.conf && sudo wpa_supplicant -B -D nl80211,wext -i wlan0 -c /home/pi/wpa-temp.conf && rm /home/pi/wpa-temp.conf
```

Pour vous déconnecter, on détruit tout simplement la ou les instance(s) du programme via la commande « killall »:

```
sudo killall wpa_supplicant
```

Brancher et configurer un disque vierge (ex. WD PiDrive 314GB)

Identifier le disque

On utilise la commande « fdisk » :

```
sudo fdisk -l
```

Dans la liste des disques attachés, vous verrez un « sdx » (où x représente le numéro du disque en question).
On accède ainsi à ce disque à l'aide de la même fonction, mais cette fois-ci avec l'adresse du disque en argument.

```
sudo fdisk /dev/sdx
```

Puis on crée une nouvelle partition. Pour ce faire, appuyez sur la touche « n » et choisissez une partition primaire grâce à la touche « p ». Ensuite, acceptez les valeurs par défaut : touche « entrer ».
Une fois revenu au menu, utilisez la touche « p » pour voir les détails, puis la touche « w » pour écrire les paramètres sur le disque.

Formater le disque en EXT4

En toute logique le disque sera installé sur /dev/sdx1. Il suffit donc de le formater :

```
sudo mkfs -t ext4 /dev/sdx1
```

En cas de doute sur le chemin d'accès du disque :

```
cd /dev && ls
```

Vous devriez alors voir le disque apparaître dans la liste.

Monter un disque formaté

Tout d'abord, on crée un dossier pour le point de montage, puis on y attache le disque :

```
sudo mkdir /data
sudo mount /dev/sdx1 /data
```

Monter le disque au démarrage

Il faut dans un premier temps récupérer l'ID du disque :

```
sudo -i blkid
```

Ce qui nous donne :

```
/dev/sda1: UUID="1e64a7dd-b1bd-49e2-b21e-04ac3cbe312e" TYPE="ext4" .....
```

Puis l'ajouter comme suit dans le fichier FSTAB :

```
sudo nano /etc/fstab
```

Ajoutez à la fin :

```
UUID=1e64a7dd-b1bd-49e2-b21e-04ac3cbe312e  /data  ext4  defaults  1  2
```

Rotation de l'écran suivant le support

Une rotation de l'écran et du « touch » peut être nécessaire suivant le support de l'écran. Il suffit d'ajouter la ligne suivante dans le fichier config pour faire une rotation de 180° de votre écran :

```
sudo nano /boot/config.txt
```

Ajouter à la fin : lcd_rotate=2
Vous noterez le lcd_rotate, et non un display_rotate qui ne prend pas en compte la partie tactile lors de la rotation.

Installer Node.js

Télécharger et installer node.js tout simplement :

```
cd ~
wget http://node-arm.herokuapp.com/node_latest_armhf.deb
sudo dpkg -i node_latest_armhf.deb
rm node_latest_armhf.deb
```

Installer les paquets relatifs à l'affichage

Il faut installer le serveur d'affichage X et nodm pour le login automatique :

```
sudo apt-get install xorg
sudo apt-get install nodm
```

On active nodm en modifiant son fichier de configuration :

```
sudo nano /etc/default/nodm
```

Il faut ensuite modifier les valeurs suivantes comme suit :

```
NODM_ENABLED=true
NODM_USER=pi
```

Le gestionnaire de fenêtre i3wm

Pour lancer « chromium », et donc node-webkit, il faut un gestionnaire de fenêtres. On va donc choisir et installer un gestionnaire le plus léger possible, tel i3wm (https://i3wm.org/), car une seule fenêtre en plein écran est suffisante pour le développement de notre programme :

```
sudo apt-get install i3
```

On crée par la suite le fichier de configuration dans le répertoire /home/pi :

```
cd && mkdir .i3
sudo nano .i3/config
```

On le remplit à minima comme suit :

```
# i3 config file (v4)
font pango:DejaVu Sans Mono 8
```

Ecran Tontec (Etape supplémentaire)

L'écran Tontec nécessite quelques étapes supplémentaires. Il faut installer le dernier driver et remplacer celui existant :

```
cd /boot/overlays
sudo rm mz61581-overlay.dtb
sudo wget http://www.itontec.com/mz61581-overlay.dtb
sudo reboot
```

Mais aussi activer l'interface SPI et son driver correspondant au démarrage en modifiant le fichier config :

```
sudo nano /boot/config.txt
```

Et en rajoutant à la fin :

```
dtparam=spi=on
dtoverlay=mz61581
```

Il faut de plus créer un fichier de configuration pour X, sinon l'écran s'allumera, mais l'affichage ne fonctionnera pas.

```
sudo nano /usr/share/X11/xorg.conf.d/99-fbdev.conf
```

Et remplir comme suit pour créer le lien vers l'écran :

```
Section "Device"
    Identifier "myfb"
    Driver "fbdev"
    Option "fbdev" "/dev/fb1"
EndSection
```

Un redémarrage est ensuite nécessaire.

Ajouter un fond d'écran comme splashscreen

Il nous faut installer le package « feh » et télécharger une image (celle du site node.js à tout hasard).
Pour cela, on crée un répertoire « wallpaper » dans le dossier de configuration d'i3wm de manière complétement arbitraire :

```
cd
cd .i3
mkdir wallpaper
cd wallpaper
wget https://nodejs.org/static/images/logos/nodejs-2560x1440.png
sudo mv nodejs-2560x1440.png nodejs.png
sudo apt-get install feh
```

Il ne reste plus qu'à tester la commande qui appelle le fond d'écran :

```
DISPLAY=:0 feh --bg-scale /home/pi/.i3/wallpaper/nodejs.png
```

Noter qu'il faut ajouter l'identification de l'écran « DISPLAY=:0 » devant chaque commande impliquant l'écran (xinput, xorg, nw ...). On en profite pour lancer le fond d'écran au lancement de X, tout en désactivant l'économiseur d'écran, ainsi que la coupure d'écran après inactivité, dans le fichier « .xsessionrc » situé dans le dossier de l'utilisateur pi « /home/pi » :

```
sudo nano .xsessionrc
```

Remplir le fichier comme suit :

```
#!/bin/sh
DISPLAY=:0 feh --bg-scale '/home/pi/.i3/wallpaper/nodejs.png' &
DISPLAY=:0 xset s off
DISPLAY=:0 xset -dpms
DISPLAY=:0 xset s noblank
```

Fig.5 et 6

Node Webkit installation

Certains paquets sont prérequis pour le bon fonctionnement de node webkit :

```
sudo apt-get install libnss3 libgconf2-dev  libgtk2.0-0 libnotify4
```

Il faut ensuite télécharger et installer une version « arm » de node webkit :

```
cd
mkdir app && cd app
```

Fig.5

Fig.6

```
wget https://github.com/jtg-gg/node-webkit/releases/download/nw-v0.12.0/nwjs-v0
.12.0-linux-arm.tar.gz
tar -zxvf nwjs-v0.12.0-linux-arm.tar.gz
mv nwjs-v0.12.0-linux-arm nwjs
rm  nwjs-v0.12.0-linux-arm.tar.gz
```

Il faut ensuite ajouter le programme dans les variables d'environnement de l'utilisateur « pi » et les « sudoers » (pour la commande sudo) pour appeler directement le programme par son nom (nw) dans le terminal.

Pour l'utilisateur pi, on modifie le fichier « .bashrc » :

```
sudo echo 'PATH=$PATH:/home/pi/app/nwjs' >> ~/.bashrc
```

Pour les sudoers, on modifie le fichier sudoers :

```
sudo nano  /etc/sudoers
```

Modifier la ligne « secure path » par celle-ci :

```
Defaults  secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
home/pi/app/nwjs"
```

Puis on reboot pour prendre en charge les modifications :

```
sudo reboot
```

Prise en charge du tactile avec xinput

On installe le packet xinput, puis on l'exécute sur l'affiche DISPLAY=:0 pour voir les entrées de données :

```
sudo apt-get install xinput
sudo DISPLAY=:0 xinput
```

Vous devriez voir la sortie suivante :

```
 Virtual core pointer                    id=2    [master pointer  (3)]
    Virtual core XTEST pointer              id=4    [slave  pointer  (2)]
    ADS7846 Touchscreen                     id=6    [slave  pointer  (2)]
    FT5406 memory based driver              id=7    [slave  pointer  (2)]
 Virtual core keyboard                   id=3    [master keyboard (2)]
    Virtual core XTEST keyboard             id=5    [slave  keyboard (3)]
```

Ce qui nous intéresse, c'est l'id du « Touchscreen », ici « 6 ». Noter cet identifiant pour la partie suivante.

Application Node Webkit de base

Créer un dossier dans le répertoire de l'utilisateur « pi » :

```
cd
mkdir nwapp
```

Il faut ensuite créer deux fichiers au minimum dans le dossier « nwapp » nouvellement créé :

- Un fichier index.html (exemple rapide d'un hello world)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
</head>
<body>
  <h1>Hello World</h1>
</body>
```

- Et un fichier de configuration package.json

```
{
 "name": "hello-world",
 "version": "0.0.1",
 "main": "index.html",
 "window": {
  "toolbar": false,
  "frame": false,
  "fullscreen": true
 },
 "dependencies": {}
}
```

Ici on désactive la toolbar du navigateur, ainsi que son cadre et on active le plein écran.

Tester votre application Hello World

```
sudo DISPLAY=:0 nw /home/pi/nwapp
```

Vous devriez voir apparaître sur l'écran un Hello World sur fond blanc !

LE LOGICIEL (WEBOS)

Arborescence Fig.7

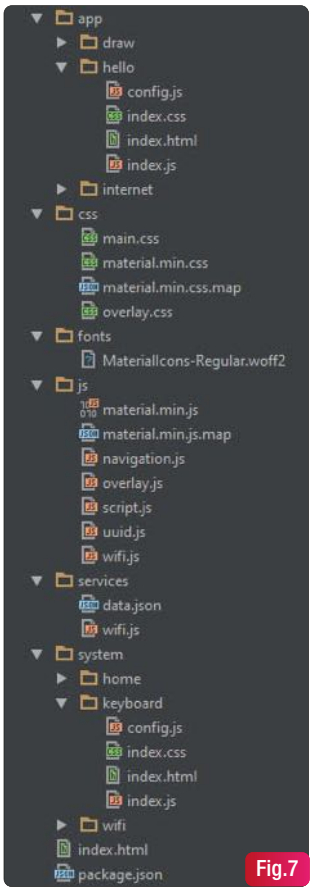


Fig.7

- index.html : Fichier maître de l'application.
- package.json : fichier de configuration de node webkit.
- Dossier css : Dossier contenant les feuilles de styles communes au WebOS et à toutes ses applications.
- Dossier js : Dossier contenant les scripts principaux permettant de gérer l'interface du WebOS
- Dossier fonts : Contient les icônes de Material Design https://design.google.com/icons/ .
- Dossier app : Contient les applications, l'objectif est de pouvoir switcher entre les différentes applications ouvertes.
- Dossier system : Identique au dossier app à la différence que les applications système n'ont pas vocation à rester ouvertes. Elles ne génèrent donc pas de nouvel onglet de navigation.
- Le Dossier services : contient un fichier de stockage de données : data.json, ainsi qu'un script pour gérer la connexion WiFi.

Le fichier package.json

On personnalise un peu plus le fichier en lui rajoutant quelques arguments supplémentaires :
Il faut passer en argument à Chromium l'id du Touchscreen (ici 6), ainsi qu'activer les touch-events. L'option « –enable-pinch » est facultative, car elle permet à l'utilisateur de zoomer dans l'application en pinçant ou inversement de dé-zoomer.

```
"chromium-args" : "--touch-devices=6 --touch-events=enabled --enable-pinch",
```

Pour notre navigateur Internet il faut informer les sites Internet que nous sommes dans un format tablette. La navigation se faisant dans des iframes, la méthode la plus simple reste de modifier le « user-agent » par défaut et de le remplacer par exemple, par celui d'un iPad.

Suite dans le numéro 199

Un miroir connecté

L'école d'études supérieures en informatique, SUPINFO International University Paris, regroupe différents laboratoires, dont le laboratoire Microsoft. Cinq de ses membres ont décidé de réaliser un miroir connecté.

Laura BENTLEY,
étudiante en 4ème année à SUPINFO International University
https://fr.linkedin.com/in/laura-bentley-8238a6101

Etienne BAUDOUX,
étudiant en 4ème année à SUPINFO International University
@VelerSoftware
https://fr.linkedin.com/in/etiennebaudoux
http://www.velersoftware.com

ZIENTEK Lucas,
étudiant en 4ème année à SUPINFO International University
https://fr.linkedin.com/in/lucas-zientek-47201664
http://lucas.zientek.fr/

MAIGA Housseini,
étudiant en 3ème année à SUPINFO International University
https://www.linkedin.com/in/housseinimaiga
https://about.me/housseinimaiga

Mehdi MAAMAR,
étudiant en 4ème année à SUPINFO International University
@Skyboys22
https://www.linkedin.com/in/mehdi-maamar-75023649
http://www.MehdiMAAMAR.com

Un miroir connecté, ou Smart Mirror, n'est pas qu'un simple miroir qui reflète notre image. C'est aussi un appareil rendu intelligent grâce à l'ajout de fonctionnalités.

Le but de ce projet était de créer un outil innovant qui servira aux étudiants, mais aussi de moyen de communication entre les élèves et l'administration.

Dans l'optique d'un projet évolutif dans le temps, des fonctionnalités de base ont été sélectionnées. Ce projet peut donc être complété avec l'ajout d'applications ou de fonctionnalités lors de la prochaine année scolaire. Ce miroir connecté permet actuellement aux étudiants d'être au courant de l'info trafic des transports en commun autour du campus de l'école, d'avoir accès à une application météo et de suivre le fil de l'actualité Twitter de SUPINFO.

Il est possible de reproduire ce projet pour un usage personnel en choisissant des applications et des modules qui correspondent à vos besoins.

Ce projet nécessite la construction physique du miroir et la programmation en WinRT des fonctionnalités connectées choisies.

Voici une liste des éléments qui ont été essentiels à la réalisation du miroir :

- Un écran full HD WideScreen LCD de 21 pouces ;
- Une Raspberry Pi 2 avec Windows 10 IoT core ;
- Un Rouleau de film miroir sans tain ;
- Une carte SD ;
- Un capteur de gestures à distance, SparkFun RGB & Gesture Sensor, APDS -9960 ;
- Une Webcam avec micro intégré ;
- Une enceinte USB.

L'écran a été recouvert d'un film miroir sans tain permettant d'avoir l'apparence et l'utilité d'un miroir classique.

En termes d'interactions entre l'utilisateur et le miroir connecté, il a été décidé de mettre en place deux moyens de communication. Il est en effet possible de naviguer dans les applications par un simple mouvement de la main grâce au capteur de gestion à distance ou bien par commande vocale basique grâce au micro intégré dans la web caméra. Il aurait aussi été possible de choisir une solution tactile, mais afin de correspondre à un usage public cette idée a été écartée.

Un système de modules

Le miroir connecté de SUPINFO a été doté d'un système de modules afin d'harmoniser la façon d'interagir avec. Imaginez un Smartphone ou une Tablette avec un Store qui permet d'ajouter une application dont le design et l'expérience utilisateur respecte les guidelines du système qui l'héberge. Le concept est à peu près le même pour le miroir connecté, à la différence qu'à la place d'avoir une « application », nous avons un « module », et qu'il n'y a pas de Store, du moins pas pour le moment.

Ce système de modules permet ainsi d'avoir un écran d'accueil, avec un résumé des informations du miroir (notre prochain rendez-vous, les notifications Facebook, la température de la maison). À partir de cet écran, l'utilisateur est en mesure de lancer un module (calendrier, Facebook, maison...), qui prendra tout l'écran, et avec lequel il pourra continuer à interagir grâce aux informations complémentaires affichées à l'écran, telles que les détails de son prochain rendez-vous.

L'occasion pour les étudiants de SUPINFO de réaliser ce système de modules est de proposer un SDK réutilisable pour permettre aux futurs étudiants qui travailleront sur le projet de continuer à développer facilement les fonctionnalités du miroir.

Ainsi, un module est au final un ensemble de classes, méthodes et composants graphiques réutilisables. Le modèle ressemble beaucoup à ce que l'on connaît sous Windows 10, Android ou iOS. Un module possède un espace local de stockage dédié, un nom, une icône, une page de démarra-

ge, des paramètres, et bien sûr, un cycle de vie.
On y retrouve également des services pour, par exemple, naviguer d'une page à l'autre dans le miroir, avec la possibilité d'utiliser la reconnaissance vocale pour faire un retour en arrière, vu que le miroir n'a pas d'écran tactile. Un service d'authentification par QR Code, de Text-To-Speech et de gestion du réseau Internet est également disponible en l'état.
Ci-contre le diagramme représentant l'architecture de base du système de modules. Un module sera représenté par la classe « MirrorModule » et une page par « ModulePage ». Tous deux peuvent utiliser les services proposés par « ITools »
Représentation simplifiée de l'architecture d'un module et de ses composants : Fig.1.
Enfin, pour tout orchestrer, une classe de gestion de module permet de découvrir tous les modules de l'Assembly, de les instancier, de les faire interagir entre eux et de démarrer leurs éventuels services périodiques.

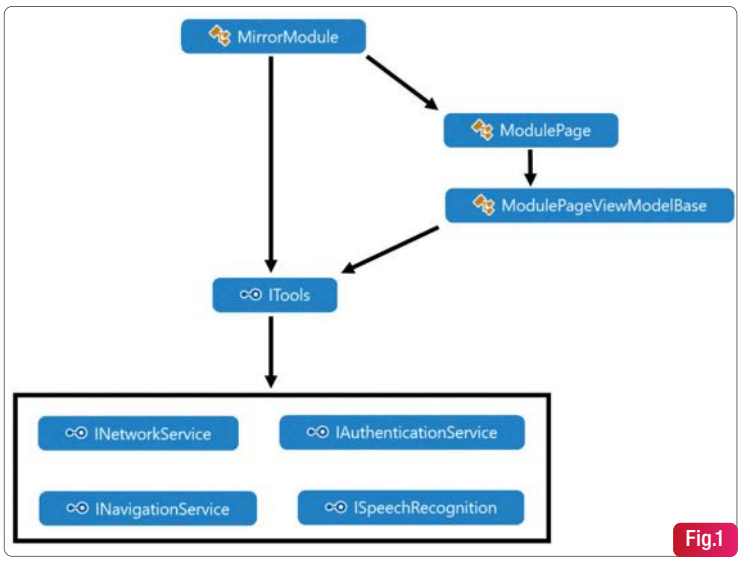


Fig.1

Amélioration de la reconnaissance vocale

WinRT propose un système de reconnaissance vocale facile à utiliser et efficace grâce à un fichier de grammaire Voice Recognition Command (VCD) XML. Les fichiers VCD permettent de déclarer facilement des « contraintes » (mot ignoré ou non, liste de donnée) :

```
<?xml version="1.0" encoding="utf-8"?>
<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.1">
  <CommandSet xml:lang="en-us" Name="AdventureWorksCommandSet_en-us">
    <Command Name="showTripToDestination">
      <Example> show trip to London  </Example>
      <ListenFor> show [trip] to {destination} </ListenFor>
    </Command>
    <PhraseList Label="destination">
      <Item> London </Item>
      <Item> Dallas </Item>
      <Item> New York </Item>
    </PhraseList>
  </CommandSet>
</VoiceCommands>
```

Cette méthode a pourtant un inconvénient, en effet, elle ne permet pas de déclarer un emplacement de texte libre, c'est-à-dire un « show trip to * ». Pourtant c'est une fonctionnalité dont on a besoin pour le miroir, mais on ne souhaitait pas non plus perdre les fonctionnalités liées au fichier VCD.
On a donc créé notre propre interpréteur de fichier VCD qui intègre la possibilité d'accepter du texte libre. Dans un premier temps, on change le système de reconnaissance vocale des grammaires VCD vers une reconnaissance de texte libre, orienté recherche Web ce qui permet d'avoir du texte sur une durée de maximum 10 secondes (ce type de reconnaissance requiert Internet ce qui n'est pas un problème dans notre cas). Ensuite, au lancement du programme, les modules vont ajouter leurs fichiers VCD à notre service de reconnaissance vocale. On va directement traiter ces fichiers pour les convertir en objets. Une fois les fichiers désérialisés, on va les parcourir pour générer nos Regex qui vont nous servir à vérifier à quelle commande correspond la voie de l'utilisateur (on génère une Regex par commande).
Voici comment on procède :

- Notre classe extrait les listes de valeurs pour pouvoir les intégrer par la suite aux Regex ;
- Elle remplace les délimiteurs optionnels VCD par ceux des Regex ;
- Elle remplace les variables de liste par une Regex représentant la liste qu'on a extraite auparavant ;
- Elle remplace notre indicateur de textes libre par son équivalent en Regex ; Dans notre cas « show [trip] to {destination} » va se transformer en : « show(trip)? to ((London)|(Dallas)|(New York)) »

Les Regex sont générées de manière à pouvoir récupérer les informations variables facilement grâce au nommage de groupes. Lors des vérifications des entrées vocales utilisateur, si la Regex passe avec succès, les données de type liste ou texte libre seront stockées dans un dictionnaire renvoyé aux modules qui pourront traiter l'information comme ils le souhaitent.

Authentification par QRCode

Ce miroir connecté sera également amené à s'authentifier, ne serait-ce que pour se connecter au WIFI par exemple. Rappelons-nous que ce miroir n'est pas doté d'écran tactile, mais utilise la reconnaissance vocale et un capteur de gestures à distance.
Quelles options s'offrent à nous pour s'authentifier sur ce miroir, de manière simple et sécurisée ?
La première idée a été d'utiliser la reconnaissance vocale pour s'authentifier. Seulement nous nous sommes vite rendu compte que ce n'était pas la solution idéale :
Primo la sécurité des identifiants est mise en question, ce qui nuit à la confidentialité des données des différents comptes. Secundo ce n'est pas vraiment évident ou même confortable de dicter un mot de passe, qui le plus souvent est assez complexe même à l'écriture.
La solution dans notre cas est de développer une application UWP pour Tablette, PC et Smartphone, qui par la suite va permettre de générer un QR Code contenant nos différents identifiants. Avec la caméra embarquée sur le miroir, on pourra lire le QR Code généré par l'application pour, par exemple, se connecter à un réseau social, au WiFi, etc. Nous avons également évoqué la possibilité de faire de la reconnaissance faciale, mais la Raspberry Pi avec Windows 10 est encore trop jeune et trop gourmande en ressources pour se reposer dessus.
Nous n'allons pas rentrer dans les détails du développement de cette application, mais plutôt nous focaliser sur les deux principales fonctionnalités de l'authentification par QR Code, à savoir la génération (via l'application UWP) et la lecture (à travers la caméra embarquée) du QR Code.
Pour cela, nous utilisons la librairie suivante : XZing.Net. Ci-dessous, un exemple de code de génération et de lecture de QR Code.
Code pour la génération de QR Code :

```
IBarcodeWriter writer = new BarcodeWriter
{
  Format = BarcodeFormat.QR_CODE,
  Options = new EncodingOptions
  {
    Height = 250,
```

```
    Width = 250
  },
  Renderer = new PixelDataRenderer() { Foreground = Colors.Black }
};
var result = writer.Write("Votre identifiant WIFI");
var wbmp = result.ToBitmap() as WriteableBitmap;
Image.Source = wbmp;
```

Code pour la lecture de QR Code

```
BarcodeReader bcReader = new BarcodeReader();
AuthenticationData authenticationData = null;

do
{
  var stream = new InMemoryRandomAccessStream();
  await MediaCapture.CapturePhotoToStreamAsync(ImageEncodingProperties.CreateJpeg(),
 stream);

  stream.Seek(0);
  int captureElementSize = 300;
  var wbm = new WriteableBitmap((int)captureElementSize, (int)CaptureElementSize);
  await wbm.SetSourceAsync(stream);

  var result = bcReader.Decode(wbm);
  if (result != null)
  {
    authenticationData = JsonConvert.DeserializeObject<AuthenticationData>(result.
Text);
  }
} while (authenticationData == null);
```

Récupération de la date depuis Internet

L'une des dernières difficultés rencontrées a été la gestion de la date et de l'heure. Actuellement, la Raspberry Pi sous Windows 10 IoT Core ne conserve pas l'heure correctement entre deux redémarrages. L'affichage de l'heure étant une fonctionnalité indispensable, nous avons dû trouver une solution. Comme l'heure ne se conserve pas localement, nous sommes passés par Internet. Pour cela, la réponse la plus simple est de trouver un service en ligne proposant l'heure universelle. Ainsi, nous passons par un serveur NTP (Network Time Protocol) public.

Nous avons opté pour Internet Time Service (ITS), qui est le service NTP que fournit NIST (National Institute of Standards and Technology) pour avoir la date complète.

Comme le serveur (http://nist.netservicesgroup.com:13/) affiche en temps réel l'heure et la date, nous passons par une requête HTTP et récupérons l'heure à l'aide d'expressions régulières.

Comme l'heure est en UTC, on utilise la fonction « ConvertTime » de la classe « TimeZoneInfo » pour changer cette dernière à GMT+1.

Ci-dessous, un extrait du code permettant la requête au serveur NTP.

```
private async Task<DateTime?> GetNistTime()
{
  DateTime? dateTime = null;
  try
  {
    var httpClient = new HttpClient();
    var request = new HttpRequestMessage(HttpMethod.Get, new Uri("http://nist.netservices
group.com:13/"));
    var httpResponseMessage = await httpClient.SendAsync(request, HttpCompletion
Option.ResponseHeadersRead);

    if (httpResponseMessage.StatusCode == HttpStatusCode.OK)
    {
      var html = await httpResponseMessage.Content.ReadAsStringAsync();
      var time = Regex.Match(html, @"\d+:\d+:\d+").Value; //HH:mm:ss format
      var date = "20" + Regex.Match(html, @"\d+-\d+-\d+").Value; //20XX-MM-DD
      dateTime = DateTime.Parse((date + " " + time));
      var src = TimeZoneInfo.Utc;
      var tz = TimeZoneInfo.FindSystemTimeZoneById("Romance Standard Time");
      dateTime = TimeZoneInfo.ConvertTime((DateTime)dateTime, src, tz);
    }
  }
  catch (Exception e) { }
  return dateTime;
}
```

La question qui se pose maintenant qu'on a l'heure, est comment faire pour l'afficher en temps réel sans faire de requête au serveur fréquemment ? L'idée est de passer par un « DispatcherTimer » qui va incrémenter la date récupérée de 1 seconde à chaque fois, tout simplement.

```
public ClockService()
{
  Internet.InternetConnectionChanged += Internet_InternetConnectionChanged;
  var timer = new DispatcherTimer();
  timer.Interval = TimeSpan.FromMilliseconds(1);
  timer.Tick += Timer_Tick;
  timer.Start();
}

private void Timer_Tick(object sender, object e)
{
  Now = Now.AddSeconds(1);
  if (DateTimeChanged != null)
  {
    var arg = new DateTimeChangedEventArgs();
    arg.CurrentTime = Now;
    DateTimeChanged(this, arg);
  }
}
```

Pour voir l'intégralité du code, nous vous invitons à consulter notre dépôt GitHub à l'adresse suivante : http://bit.ly/1so4bbo

Comment assurer une belle vie à son application mobile ! (Partie 1)

Depuis quelques années, la mobilité a pris un aspect primordial dans la vie de tous. Dans le monde du développement mobile, on a constaté une vraie effervescence des différents acteurs. C'est dans ce cadre que Xamarin s'est imposé en tant que solution viable et efficace auprès des développeurs et entreprises : grâce au parti pris du cross-plateforme tout en conservant une UI native et des performances natives. Comme à l'époque des chevaliers, le couronnement n'était que le commencement du règne. De la même manière, aujourd'hui développer une application ne suffit pas, il faut la faire vivre !

Mathilde Roussel
R&D Software Engineer chez **MEGA International**
https://twitter.com/Math_Roussel

Jason De Oliveira
CTO | MVP chez
MEGA International
http://www.jasondeoliveira.com

Xamarin ayant initié un partenariat très fort avec le géant Microsoft, s'est vu racheté par celui-ci au mois de février 2016, et ce, au grand bonheur de la communauté. En effet, cela a consolidé la crédibilité de cette technologie et a augmenté son adoption à tous les niveaux. Un point clé qui découle de la tendance mobile, est la forte volonté des équipes de Microsoft et Xamarin de fournir des outils toujours plus performants pour permettre la création d'applications mobiles durables dans le temps, et satisfaisantes à la fois pour les développeurs, utilisateurs et entreprises. C'est justement de ces outils et de cette intégration au sein d'une application créée avec Xamarin dont cette série d'articles va parler, car ce sont eux qui assurent en partie la survie de notre application au sein de la jungle qu'est le monde du mobile. Cet article d'introduction rappellera ce qu'est le DevOps et se penchera sur les questions de l'intégration et du déploiement continu.

Tout d'abord, que signifie Mobile DevOps ?

Pour faire simple, DevOps est un ensemble de cultures et pratiques qui consistent à améliorer la collaboration entre les développeurs et les équipes qui assurent le bon fonctionnement du système de production. On parle souvent des Développeurs et des Operations, d'où DevOps. Le terme DevOps, né en 2008 de son père Méthodes Agiles, est devenu très courant. Le concept n'est pas en soi nouveau, il regroupe sous sa bannière un ensemble de bonnes pratiques et met en lumière des principes, méthodes ou outils malheureusement pas toujours utilisés. Parmi les fidèles compagnons que compte DevOps, on retrouve le principe d'intégration continue, les tests unitaires, la livraison continue, le monitoring continu, etc.

Le DevOps à la conquête du mobile

Etant donné la définition de DevOps, on comprend qu'il peut s'appliquer à différents domaines du développement, et particulièrement pour le développement d'applications mobiles. Il est un enjeu primordial pour assurer longue vie et prospérité pour l'application. Le concept « Mobile DevOps » regroupe, entre autres, la rationalisation du développement de l'application. Celle-ci passe par la livraison régulière de versions à valeur ajoutée aux utilisateurs, la maintenance d'une qualité globale, la surveillance de la santé de l'application en temps réel. Une autre composante du concept DevOps est l'automatisation des différents processus intervenant au cours des développements, grâce à l'intégration continue, le déploiement continu, etc.

VSTS et HockeyApp, deux loyaux sujets aux casquettes multiples

Comme l'a explicité l'introduction, Microsoft et Xamarin continuent à s'allier et à proposer des outils DevOps, qui ont l'avantage d'être complets et facilement intégrables à tous types de projets, notamment les projets mobiles, qu'ils utilisent Xamarin ou non.

Visual Studio Team Services (VSTS)

Visual Studio Team Services (VSTS), de son ancien nom Visual Studio Online et de son encore plus ancien nom Team Foundation Services, est un service Cloud jouant un rôle clé sur de multiples aspects. Il fournit tout d'abord un service de contrôle de version, que ce soit avec GIT ou Team Foundation Version Control (TFVC), et c'est un premier point essentiel dans l'univers DevOps. En effet, c'est ce service qui va regrouper le code, permettre de gérer les versions et faciliter les code reviews. Dans le cadre de projets agiles, VSTS offre également des outils de planning comme un tableau kanban, un backlog, ou le suivi des tâches. VSTS ne s'arrête pas là puisque le service propose de plus toute une interface pour gérer l'intégration continue et le déploiement continu, grâce à différents plug-ins disponibles. L'avantage du service est, pour le cas d'une application mobile « multi-plateforme », qu'il gère les trois OS principaux (iOS, Android et Windows Phone), et sous certaines conditions automatise une grande partie des étapes de l'intégration continue. A noter qu'il est tout à fait possible de tester, voire d'utiliser au quotidien Visual Studio Team Services sans frais, puisque le service est gratuit jusqu'à 5 utilisateurs et offre des fonctionnalités largement suffisantes pour un projet basique. La seule condition pour profiter de l'intégration continue pour un projet iOS ou Xamarin.iOS est de posséder un Mac qui se chargera des builds.

Mais plutôt que de longs discours, voici un exemple concret d'association entre un projet Xamarin et Visual Studio Team Services. Cet exemple se concentrera dans un premier temps sur la partie d'intégration continue.

MySuperInventoryManager, un chevalier modèle !

La solution MySuperInventoryManager contient un ensemble de projets pour implémenter une application de gestion de stock. Les projets mobiles sont déjà créés et les sources sont poussés dans le repository Git défini, le code est donc visible dans l'onglet « Code » dans VSTS. Cette première étape terminée, il est temps de passer dans l'onglet « Build » et de créer une nouvelle définition de build (Fig.1). Plusieurs templates sont à disposition : on trouve des templates de build et de déploiement. Dans le cadre de l'exemple, le type choisi sera Xamarin.Android. Il est également possible de créer un template vide. La partie de configuration du template est simplifiée, car VSTS propose automatiquement le repository courant (il est tout de même possible d'en choisir un autre) ainsi que la branche master par défaut. Il suffit dans un cas simple comme celui de l'exemple, de cocher la case « Continuous Integration » pour lancer le pipeline

à chaque nouveau commit. L'interface auparavant vide se retrouve peuplée d'étapes correspondant au type de build choisi (Fig.2).

Décomposons ces étapes : VSTS va d'abord se charger de restaurer les packages NuGet, indispensables pour builder si le projet en possède (il est rare que ce ne soit pas le cas). Un chemin par défaut est proposé pour trouver le fichier de la solution, ainsi que plusieurs options de configuration. Dans le cas de cet exemple, le chemin vers le .sln du projet sera sélectionné. La prochaine étape est aujourd'hui à supprimer, tout comme l'avant dernière, car l'activation et la désactivation de la licence Xamarin sont destinées à disparaître, grâce à l'intégration automatique d'un compte Xamarin dans une licence Visual Studio Community, Enterprise et Professional. Les packages NuGet restaurés, l'étape de compilation du projet Xamarin.Android peut démarrer. Là encore, des valeurs par défauts sont renseignées dans le panneau de droite, et peuvent être modifiées selon le besoin.

Le template propose ensuite le build d'un projet de tests unitaires, ainsi que le lancement de tests UI grâce à Xamarin TestCloud. Evidemment, certaines étapes comme celles-ci sont optionnelles, et l'ordre est au choix de la personne qui configure. Elles seront désactivées dans le cas de l'exemple, mais elles restent néanmoins fortement recommandées pour assurer un livrable de qualité. Une des dernières étapes est la signature de l'application et l'alignement des fichiers, pour ensuite rendre l'application prête à être publiée à l'endroit souhaité via « Publish Artifact ». C'est d'ailleurs là-dessus que s'achève le circuit de build pour l'application Android. Puisque chaque projet client utilise une PCL (qui regroupe la partie métier), il faut ajouter une nouvelle étape au build, afin que la PCL soit buildée avant le projet Xamarin.Android. Cette étape est une étape Visual Studio Build, et si, de base, cette étape cherche à compiler l'ensemble de la solution, il faut modifier le champ « Solution » pour lui fournir le chemin du .csproj du projet Core uniquement. La configuration du pipeline de build est normalement prête maintenant, pour le tester il faut soit passer par un commit de code soit par un lancement manuel, c'est-à-dire en lançant une queue de build. L'historique des queues de build en cours et complétées est visible afin de surveiller l'état des builds (Fig.3). Si le pipeline de build s'est bien déroulé, tout est au vert, l'application publiée est prête pour la suite (Fig.4). Si Visual Studio Team Services fournit de base pour le template Xamarin.Android tout un ensemble d'étapes pour le build, nous avons vu qu'il est possible d'en rajouter, que ce soit parmi celles proposées de base, ou que ce soit en passant par un marketplace. La configuration est donc vraiment libre et customisable à souhait, ce qui est un avantage considérable. C'est dans ce marketplace que l'on retrouve entre autres HockeyApp, le deuxième outil DevOps que nous allons aborder tout de suite.

HockeyApp, un allié fort pour Visual Studio Team Services

HockeyApp est un service qui regroupe la distribution d'applications mobiles en bêta, mais aussi le suivi des crashs, feedbacks utilisateurs et fournit différentes métriques. Disponible pour les plateformes iOS, Android et Windows Phone, HockeyApp a rejoint les équipes de Microsoft il y a bientôt deux ans. Si jusqu'il y a peu le choix d'un service tel qu'HockeyApp pour une application Xamarin était partagé entre Xamarin Insights et HockeyApp, cela n'est plus le cas, car les équipes de Xamarin Insights ont rejoint récemment les équipes d'HockeyApp, dans le but de se regrouper sous une seule et même bannière, qui sera HockeyApp. Les premières répercussions positives de ce changement se font sentir depuis les annonces faites à l'Evolve 2016, avec la mise à disposition d'un nouveau SDK et différentes améliorations du tableau de bord.

Mais si HockeyApp est à lui seul un outil DevOps très intéressant, il devient un allié redoutable s'il est couplé avec Visual Studio Team Services. En effet, VSTS propose le build continu mais égale-

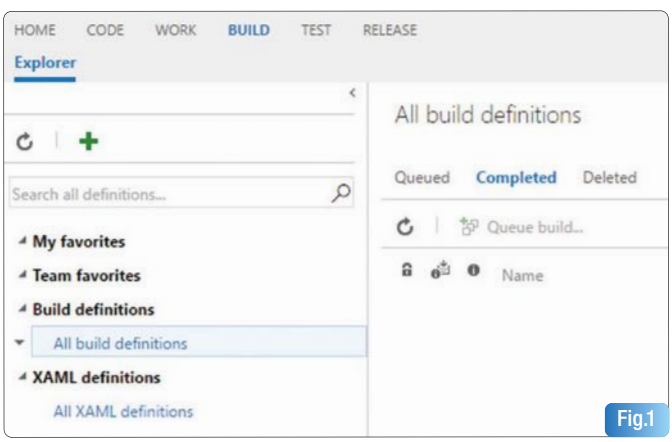


Fig.1

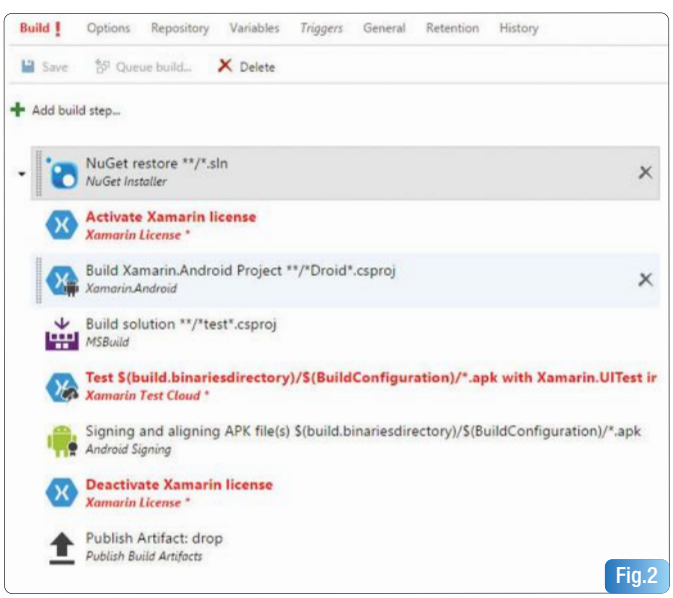


Fig.2

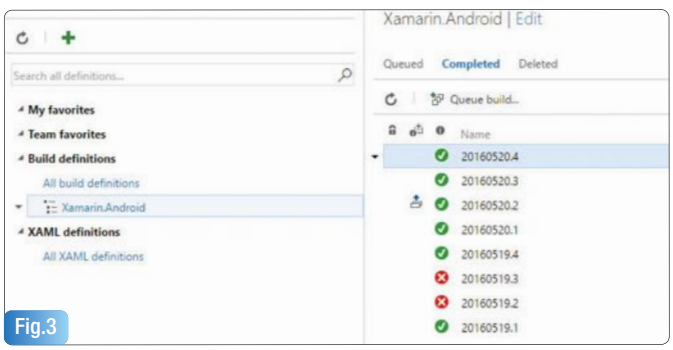


Fig.3

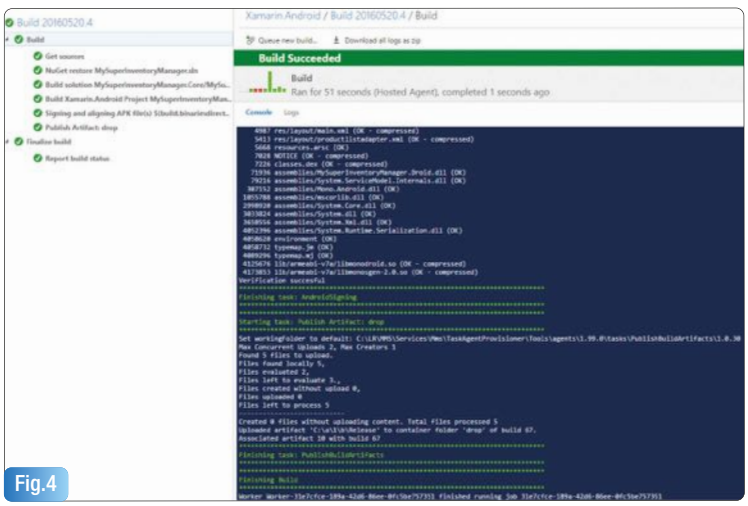


Fig.4

ment une partie d'automatisation des releases pour les déploiements, et c'est là que l'intégration d'HockeyApp dans le processus intervient.
Concrètement, lorsque le pipeline de build vu précédemment s'est bien exécuté, la suite consiste à déployer l'application buildée sur les périphériques pour la bêta, partie gérée par HockeyApp. Cela peut se faire de façon manuelle, comme automatique grâce à VSTS, et nous allons bien sûr nous intéresser à la méthode automatique dans cette partie. Avant de commencer toute configuration, il est nécessaire de s'enregistrer sur le site de HockeyApp pour disposer plus tard d'un token et éventuellement un appID.
De retour dans Visual Studio Team Services, il faut passer de l'onglet « Build », où tout est déjà configuré, à l'onglet « Release ». Tout comme la construction du pipeline de build, il faut créer une nouvelle définition de release, et le choix se portera sur une définition vide. Il faut ensuite choisir la source à partir de laquelle cette définition se basera pour le déploiement. Dans le cas de l'exemple ce sera le pipeline créé précédemment, qui a été nommé Xamarin.Android. Il reste à cocher la case pour automatiser le déploiement et à sélectionner la queue (qui sera la même que précédemment). L'interface pour la partie release est sensiblement la même que pour la partie build, à la différence qu'il est possible de définir plusieurs environnements de release, ce qui peut être intéressant si on souhaite avoir une pré-production en plus de la production par exemple. Il est maintenant temps d'ajouter la tâche de déploiement. Pour cela, il faut installer l'extension HockeyApp, qui se trouve sur le marketplace (attention cette étape requiert les droits administrateurs sur le domaine VSTS, dans le cas contraire une notification est envoyée à celui-ci). L'extension installée, il est maintenant possible de la sélectionner dans la catégorie « Deploy » (Fig.5).
Il faut ici configurer deux champs obligatoires : la connexion HockeyApp et le chemin vers le binaire de l'application. La connexion HockeyApp peut déjà exister et dans ce cas il suffit de la sélectionner dans la liste déroulante, sinon le lien « Manage » permet d'accéder à l'interface de gestion des connexions aux services (Fig.6).
Comme indiqué, l'icône « plus » permet de choisir le type de service, qui sera bien sûr HockeyApp. La popup qui apparaît montre deux champs à renseigner, le nom qui est au choix, et l'API Token. Celui-ci n'existe pas encore forcément, comme c'est le cas dans cet exemple, mais la procédure pour le créer est très simple, et VSTS indique le lien de création en cliquant sur l'icône information à droite du champ texte (Fig.7). Si tout s'est bien passé lors de la création de ce token, il reste à le renseigner dans le champ texte de départ, et bien sûr valider. De retour dans VSTS, un refresh de la liste des connexions permettra de voir celle qui vient d'être créée. En dessous il faut également renseigner le chemin où est construit le fichier .apk, qui correspond au fichier généré lors du build. Ici ce sera $(System.DefaultWorkingDirectory)/Xamarin.Android/drop/MySuperInventoryManager.Droid.MySuperInventoryManager.Droid.apk. Comme pour chaque étape de build, un grand nombre d'options de configuration sont à disposition, mais les deux champs cités sont les seuls obligatoires pour faire fonctionner de façon basique le déploiement de l'application sur HockeyApp.
Afin de tester la configuration finale, il faut comme précédemment procéder via un commit du code (pour lancer la chaîne de build puis le déploiement, comme configuré au départ), soit via un lancement manuel de la queue de build depuis VSTS. Encore une fois, si l'ensemble a bien fonctionné tout est vert dans l'interface (Fig.8) et l'application se retrouve directement disponible sur le portail HockeyApp (Fig.9). Mission réussie !

Conclusion

Vous avez maintenant vu comment appliquer les pratiques de build continu et déploiement continu, inscrites dans les pratiques DevOps, au développement des applications mobiles XAMARIN. L'utilisation des services Cloud (Visual Studio Team Services, HockeyApp, etc.) est très efficace et nous facilite grandement la vie pour bâtir des applications fortement résistantes contre tous type d'attaques (invasions de bugs, problèmes de performances, etc.). A vous de conquérir le nouveau monde des applications mobiles !

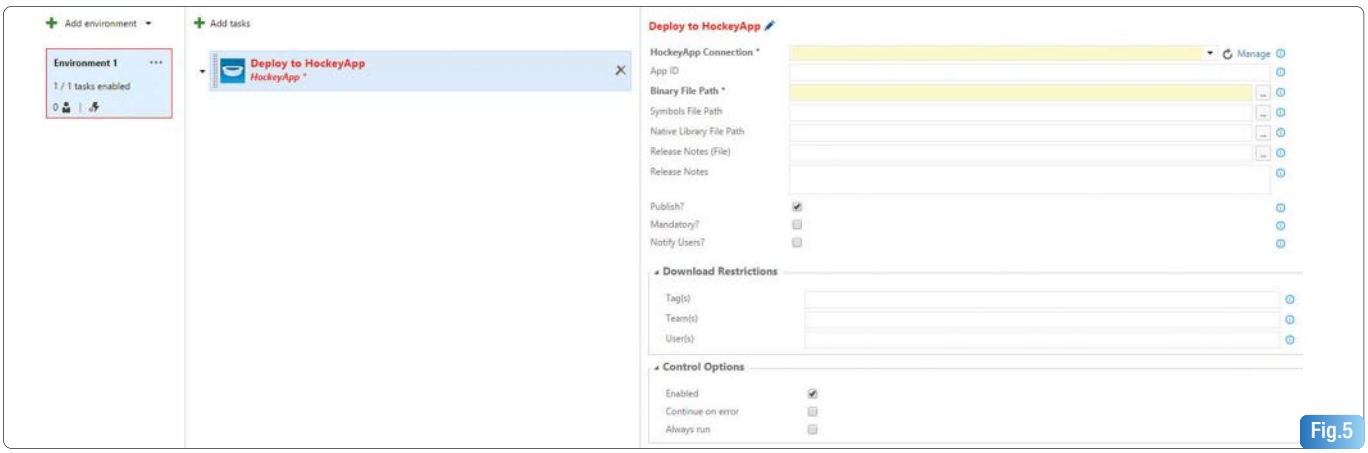


Fig.5

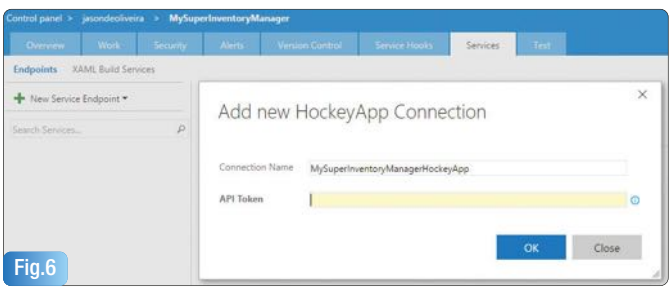


Fig.6

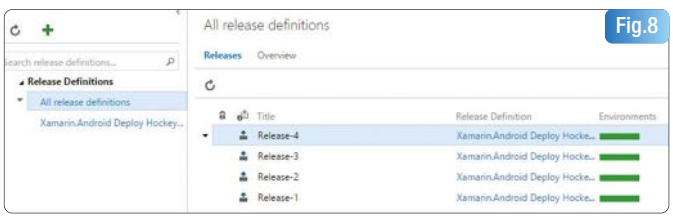


Fig.8

Fig.7

Fig.9

Créer son propre langage de programmation

La création d'un langage de programmation est quelque chose qui parait insurmontable et abstrait pour les développeurs. Mais on peut arriver à faire un mini langage qui gère certains concepts. Dans cet article nous allons voir comment faire un compilateur en utilisant F#.

Aurélien Galtier
TechLead Software Craftsman chez **Cellenza**
MVP : Visual Studio and Development Technologies

Pourquoi créer un langage ?

En tant que développeur, l'utilisation des langages de programmation est courante tous les jours. Le choix est vaste et nous utilisons des langages créés et éprouvés en production. Mais il arrive que de nouveaux langages de programmation apparaissent. L'envie de créer un langage de programmation existe pour différentes raisons. Que cela soit pour créer un DSL (Domain Specific Language) qui permettra de faire évoluer les règles de l'application dans le langage du métier ou que l'on ait envie d'augmenter un autre langage (exemple TypeScript qui permet de rajouter des concepts de types et interfaces dans Javascript), ou tout simplement de créer un langage fun tel que le Emojicode. Dans tous les cas, la création d'un langage utilise les mêmes concepts, il faudra créer un compilateur.
En Javascript l'utilisation des « transpiler » permet de convertir des versions de Javascript dans d'autres versions de Javascript. C'est le même principe qu'un compilateur. On convertit un langage de programmation dans un autre langage de programmation. Il y a différents types de langages de programmation. La première étape est de savoir quel type de langage créer. Par exemple la création d'un langage impératif. Cela veut dire que le langage est basé sur l'idée d'une exécution étape par étape. Ou encore un langage fonctionnel où l'idée est basée sur la déclaration de fonctions mathématiques. Une fois le type de langage choisi, l'idée est de s'inspirer d'un ou plusieurs langages qui existent pour en prendre les concepts qui nous intéressent et les inclure dans notre nouveau langage.
J'ai eu envie de créer un langage pour expliquer comment fonctionne un compilateur. Mon compilateur est loin d'être parfait, mais il intègre beaucoup de briques importantes. Un compilateur c'est une suite d'étapes. Chacune des étapes est intéressante, il m'est arrivé d'utiliser certaines étapes plus que d'autres pour créer des DSL, parser des fichiers avec un format spécial, ou encore faire des transformations.

Le langage

Avant de créer un compilateur, il nous faut un langage. Nous allons réfléchir au langage de programmation que l'on souhaite créer. J'ai décidé de créer un langage de programmation impératif et scriptable. C'est-à-dire que je peux commencer à écrire du code sans avoir de concepts de méthodes et objets. Un exemple simple de ce que je souhaitais écrire :

```
var x = 1 + 2 ;
x = 5 ;
```

Dans mon langage, je ne souhaite pas non plus spécifier le type comme on peut le faire en C# :

```
int x = 1 + 2 ;
x = 5 ;
```

Cela veut dire que mon compilateur retournera une erreur lorsque j'utiliserai le mot cléf « int ». Par contre je souhaite que mon langage soit fortement typé. Ce qui veut dire que je ne pourrai pas écrire :

```
var x = 1 + 2 ;
x = true ;
```

En effet, mon compilateur va déduire de l'utilisation que x est de type int donc il ne peut pas recevoir la valeur booléenne « true ».
Maintenant dans ce langage je souhaitais une notion de méthodes. J'ai repris une syntaxe similaire au fonctionnel :

```
var maMethod = fun(x) -> { return x + 1 ; } ;
```

Il me reste à donner une notion d'objet. Pour cela j'ai décidé que mes objets allaient se définir par leur création :

```
var monObjet = { MaPropriete=1, MaPropriete2=true} ;
```

Ainsi je crée un objet avec deux propriétés, la propriété « MaPropriete » de valeur entière et l'autre « MaPropriete2 » de valeur booléenne.
J'ai nommé mon langage KISS pour Keep It Simple Stupid. Maintenant que l'on a quelques syntaxes de notre langage, il nous reste à créer le compilateur qui sera capable d'interpréter ce langage pour l'exécuter.

Qu'est-ce qu'un compilateur ?

Un compilateur est une application qui permet de lire un ou plusieurs fichiers texte et d'en faire un fichier exécutable. Ce fichier exécutable sera l'exécution de ce qui est décrit dans les fichiers textes.
Pour faire un fichier exécutable nous allons produire du code machine. Notre compilateur va donc lire un fichier texte et produire un code machine. Le code machine est un langage de programmation. Ainsi notre compilateur doit transformer un langage de haut niveau en langage de bas niveau. Un langage de haut niveau est un langage facilement compréhensible par un humain. Du C# par exemple, peut être lu et compris. Il dispose de plein de mots clefs qui lui permettent d'avoir une grande expressivité. Ce langage de programmation peut être compris de manière simple par un humain. Plus le langage de programmation est de haut niveau, plus votre langage est facilement compréhensible.
Un langage de bas niveau, c'est celui qui sera le plus proche de la machine. Ainsi les instructions sont très proches du processeur ; nous aurons moins de facilité à le lire, mais le processeur, lui, n'aura aucune difficulté à l'exécuter. Par contre le programme devra gérer des problématiques liées au processeur. Ainsi les stacks, emplacement mémoire, et registres devront être correctement utilisés. Le langage de plus bas niveau que nous pouvons générer est de l'assembleur. Pour simplifier nous allons générer du MSIL (Microsoft Intermediate Language)
Par exemple, le compilateur C# généré du MSIL qui sera lui compilé, à l'exécution par le CLR de .Net. Le MSIL est proche d'un langage assembleur mais enlève beaucoup de contraintes.
MSIL simplifie pas mal de concepts bas niveaux tels que les gestions des jumps et de la pile. Ce sont des concepts à prendre en compte lorsque nous voulons faire des méthodes dans notre code. La pile permet de stocker le contexte d'appel tandis que le jump passe à une suite d'instructions. En assembleur nous n'avons pas de notions de méthode, mais seulement une suite d'instructions. Avec la CLR nous allons aussi profiter d'un garbage collector, de check de division par zéro...

MSIL introduit une notion de classe et de méthode, mais le contenu de ses méthodes reste un langage à base d'instructions.
Nous allons maintenant voir quelles sont les étapes pour passer d'un langage de haut niveau à un langage de bas niveau.

Structure d'un compilateur

Un compilateur contient trois grandes phases :

- Analyse Syntaxique ;
- Analyse Sémantique ;
- Générateur.

Pendant la première phase nous allons lire un fichier texte et le convertir en arbre syntaxique abstrait. Cet arbre syntaxique est une représentation abstraite du code écrit dans le fichier. Nous allons donc avoir un objet manipulable par code pour pouvoir faire des analyses. Pendant cette phase, si notre fichier texte ne contient pas les bons mots clefs, ou si l'ordre des mots clefs n'est pas correct nous ne pourrons pas déduire l'arbre syntaxique abstrait. La deuxième phase est l'analyse de cet arbre syntaxique. A ce moment, nous allons vérifier que notre langage est sémantiquement correct. Pendant cette phase nous allons faire beaucoup de tests pour valider que notre programme peut être converti en code machine et exécuté par le processeur d'une manière logique. Exemple :

```
var maVariable = 1 ;
var maVariable = true ;
if(maVariable){ return 1 ; } else { return 0 ; }
```

Nous pouvons vérifier plusieurs choses :

- maVariable est créée deux fois. Mais mon langage accepte de redéfinir le scope de la variable à partir du moment où l'on refait un « var ». Beaucoup de langages ne l'acceptent pas, donc dans ce cas-là on génère une erreur.
- maVariable existe bien avant d'être utilisée. C'est-à-dire que l'on a bien un « var maVariable » avant l'utilisation dans le « if ». Certains langages sont plus permissifs et acceptent que le var soit après, et certains langages n'ont même pas besoin d'une initialisation de variable.
- maVariable est bien du type bool. Car elle est utilisée dans un if.
- Mon programme renvoie bien un entier. Un programme peut renvoyer soit un entier, soit rien.

Dans un code plus complexe, nous allons vérifier la portée des variables et des méthodes. A savoir est-ce que je peux accéder à ces variables dans la méthode où je me trouve ? Est-ce que j'ai accès à la méthode là où je me trouve ? Après la phase d'analyse sémantique, notre compilateur ne génèrera plus aucune erreur sur le code écrit. Il restera la troisième phase du compilateur qui est la génération de code. Ainsi nous allons générer le code machine qui va exécuter notre code. C'est dans cette phase que le fichier exécutable va être écrit. A partir des deux analyses précédentes, nous allons convertir notre arbre syntaxique en code machine interprétable par le compilateur. C'est pendant cette phase aussi que nous allons pouvoir ajouter des optimisations sur les instructions.

Analyse Syntaxique

L'arbre de syntaxe abstraite

L'analyse syntaxique va transformer notre fichier texte en arbre de syntaxe abstraite. C'est cet arbre que nous allons manipuler après. Nous l'appellerons dans la suite de l'article un arbre syntaxique.
Mais qu'est-ce qu'un arbre syntaxique ? L'arbre syntaxique représente de manière hiérarchique notre code pour pouvoir le parcourir et le manipuler plus facilement. Si on prend l'exemple suivant :

```
1 + 1
```

On peut le représenter sous la forme d'un arbre :

```
  +
 / \
1  1
```

Ainsi chaque nœud représente une opération. Les feuilles représentent un entier. Si je prends quelque chose de plus complexe :

```
mavariable = 1 + 2 * 3
           =
          / \
mavariable  +
           / \
          1   *
             / \
            2   3
```

Lorsque l'on va vouloir évaluer notre programme, il suffira de parcourir notre arbre. Pour faire son analyse sémantique, nous allons pouvoir parcourir chaque nœud et vérifier la cohérence du programme. Par exemple pour le typage, nous pourrons déduire que « mavariable » est de type entier car toutes les opérations sont entières. Nous avons vu comment faire une représentation visuelle de notre arbre syntaxique. En F#, nous pouvons créer des types récursifs. Ainsi la représentation précédente en F# peut s'écrire en utilisant des parenthèses.

```
Affect("Mavariable",  Add(Int(1), Mult(Int(2), Int(3))))
```

Nous avons :

- Int() : qui nous indique une constante entière ;
- Mult(,) : qui nous indique une opération de multiplication entre deux sous-arbres ;
- Add(,) : qui nous indique une opération d'addition entre deux sous arbres ;
- Affect(,) : qui indique une affectation à une variable de la valeur d'un sous-arbre.

L'écriture en F# nous permet d'écrire un arbre de manière beaucoup plus simple que dans certains langages. Ainsi notre type F# ressemblera à :

```
type Prog = Program of Statement list

and Statement =
| Create of string * Expression
| Assign of Variable * Expression
| Return of Expression

and Expression =
| Int of int
| Bool of bool
| Float of float
| Add of Expression * Expression
| New of Property list
| Use of string
| Call of Variable
| Fun of string list * Statement list
| Get of Variable
| Greater of Expression * Expression
| GreaterOrEqual of Expression * Expression
| Less of Expression * Expression
| LessOrEqual of Expression * Expression

and Property =
| PropertySetter of string * Expression

and Variable =
| Variable of string
| Property of Variable * string
```

Ce type est récursif. Ainsi nous pouvons fournir des combinaisons infinies de notre syntaxe. C'est grâce à cette récursivité que nous pouvons écrire des syntaxes avec différentes combinaisons.

Lexer

Pour effectuer la première phase « analyse syntaxique » nous avons besoin de deux choses. La première est le lexer, et la deuxième le parser. L'un ne va pas sans l'autre. J'ai utilisé FsLexYacc qui est une implémentation d'un lexer/parser pour F#. Il utilise une syntaxe simple pour définir nos règles et génère un fichier F# associé plus complexe. L'installation et l'utilisation se trouvent à l'adresse suivante : http://fsprojects.github.io/FsLexYacc/. Une fois que l'on a défini notre langage, il faut identifier tous les mots clefs qui composent ce langage. C'est dans cette phase que tous les mots clefs vont être spécifiés. Si un mot clé n'est pas défini, alors notre langage ne comprendra pas le fonctionnement. Le lexer va découper notre fichier texte en une suite de tokens. Un token est une brique de construction de votre langage. Et le parser va récupérer ces tokens pour les convertir en arbre syntaxique. Les tokens représentent :

- Des mots clés de notre langage if, else, fun, ... ;
- Des opérations +-/*... ;
- Des constantes 1, 2.0, "mastring" ;
- Des noms de variable, méthode, propriétés, ... ;
- La fin du fichier.

Notre lexer va transformer :

```
var maVariable = 1 + 2 ;
```

En une suite de token :

```
VAR NAME EQUAL INT PLUS INT SEMI EOF
```

Nous retrouvons le token EOF pour indiquer que c'est la fin du fichier. Cela est important pour vérifier que notre fichier est bien formaté.
Pour que mon programme fonctionne je vais avoir besoin de la définition de tokens suivante :

```
%token <string> NAME
%token <int> INT
%token TRUE FALSE
%token LBRACE RBRACE
%token LPAREN RPAREN
%token PLUS MINUS
%token VAR IF ELSE INCR USE
%token FUN ARROW RETURN
%token GREATER LESS GREATEREQUAL LESSEQUAL
%token SEMI COMMA
%token DESINCR AND DOT EQUAL STRING
%token EOF
```

La liste de tokens est à ajouter dans le fichier « fsy » qui contient les définitions des règles de grammaire que nous allons voir plus tard. Mais comme le lexer et le parser sont très liés, la liste des tokens est définie avec les règles de grammaire. Dans le fichier d'extension « fsl » nous allons définir par exemple le token VAR :

```
| "var"      { VAR }
```

Les tokens avec valeurs

Un token c'est une expression régulière. Ainsi vous pouvez récupérer ce qui a été associé à chaque token. Dans le cas de NAME et INT, nous pourrons récupérer plus tard pendant la phase du parser, leurs valeurs associées. Pour les tokens qui doivent récupérer les valeurs associées, nous allons faire :

```
| ['0'-'9']+  { INT(int(lexeme lexbuf)) }
```

La fonction lexeme est la substitution de LexBuffer<_>.LexemeString

```
let lexeme = LexBuffer<_>.LexemeString
```

L'instruction « lexeme lexbuf » permet de récupérer la valeur courante dans le buffer. Dans le cas d'un entier, nous avons fait une expression régulière seulement sur des nombres sans virgules. Donc il peut être converti en entier facilement par la méthode int()

Cas du token STRING

Le cas du token STRING est un peu plus complexe. Le token commence à l'ouverture d'un guillemet et va jusqu'à la fermeture du guillemet. Le token STRING peut aussi utiliser un caractère d'échappement pour indiquer que le guillemet fait partie de la valeur et non pas de la fin de la chaîne de texte. Pour commencer nous allons dire que le guillemet va commencer le token STRING.

```
| [""]  { stringToken "" false lexbuf }
```

Avec ce code nous lui indiquons d'aller chercher le contenu en utilisant « stringToken » ci-dessous et non plus notre lexer de départ :

```
and stringToken str ignorequote = parse
| ""          { if ignorequote  then (stringToken (str+"\"") false lexbuf) else STRING (str) }
| '\\'        { stringToken str true lexbuf }
| [^ '"' '\\']+ { stringToken (str+(lexeme lexbuf)) false lexbuf }
| eof         { failwith ("String is not terminated") }
```

Cette méthode va :

- Si c'est un guillemet renvoyer le token STRING avec la valeur, sauf si le guillemet a été précédemment échappé, dans ce cas-là nous allons continuer la lecture du buffer ;
- SI c'est un caractère d'échappement nous l'indiquons pour échapper le prochain guillemet ;
- Si c'est un autre caractère, nous allons le rajouter à notre valeur ;
- Si c'est la fin de fichier alors il y a une erreur il manque le guillemet fermant la chaîne de texte. Dans ce cas-là, il faut générer une exception pour indiquer l'erreur. Idéalement il faudrait retrouver le numéro de ligne et l'afficher dans le texte.

Cas des espaces et du retour à la ligne

Du point de vue d'un Lexer, un espace est un caractère comme un autre. Dans certains langages cela peut devenir un token important pour respecter le langage. Mais dans notre cas nous n'avons pas envie d'avoir une liste de tokens du style :

```
VAR ESPACE NAME ESPACE EQUAL ESPACE INT ESPACE SEMI ESPACE EOF
```

Cela va rendre difficile le parsing de notre suite de token. Donc pour cela nous allons consommer le buffer et réappliquer le lexer sur la suite :

```
| [' ' '\t' ]  { token lexbuf }
```

En revanche, le cas des retours à la ligne n'est pas géré. Pour les retours à la ligne nous allons faire le même traitement. En plus, nous allons faire avancer le compteur de lignes :

```
| ('\n' | '\r' '\n')  { newline lexbuf; token lexbuf }
```

« newLine » permet d'incrémenter la ligne dans « lexbuf » sa définition est :

```
let newline (lexbuf: LexBuffer<_>) =
  lexbuf.EndPos <- lexbuf.EndPos.NextLine
```

L'intérêt d'incrémenter la ligne de EndPos, c'est de pouvoir afficher sur quelle ligne se trouve une erreur de parsing.

Parser

Une fois notre lexer créé il nous reste le parser. Le parser va lui prendre ce flux de tokens et le traduire en arbre syntaxique. Pour transformer notre suite de lexem en arbre, nous allons décrire nos règles de grammaire. Les règles de grammaire permettent de définir la forme du langage. Ce sont ces règles qui permettent de définir l'ordre logique de nos tokens. Ce sont elles qui vont transformer :

```
VAR NAME(Mavariable) EQUAL INT(1) PLUS INT(2) STAR INT(3)
```

En

```
Create("Mavariable",  Add(Int(1), Mult(Int(2), Int(3))))
```

Tous les tokens précédemment décrits vont être réutilisés pour définir notre syntaxe. La grammaire d'un langage permet de définir les règles de forme de notre langage. Ainsi chaque règle va nous dire comment transformer notre langage en arbre syntaxique. Pour notre exemple, il nous faudra les règles :

```
statement = VAR NAME EQUAL expression
expression = INT
expression = expression PLUS expression
```

Avec ces règles de grammaire, nous indiquons que notre ligne se nomme un « statement » ; il est formé d'un Var, d'un nom, suivi d'un égal et d'une « expression ». Puis nous spécifions des règles qui indiquent à quoi ressemble une « expression ». Ainsi une expression peut-être un entier, une addition, une multiplication etc. Dans le cas d'une addition nous allons indiquer qu'une addition est une composition d'une « expression », un plus et une deuxième « expression ». Ainsi nous bouclons de manière récursive. Nous aurons des compositions d'addition, soustraction, ... En F# grâce à FsLexYacc nous allons ajouter des règles de grammaire dans le fichier « fsy » :

```
statement: VAR NAME EQUAL expression        { Create($2, $4) }
expression: INT                             { Int($1) }
      | expression PLUS expression          { Add($1, $3) }
...
```

Le dollar permet de récupérer la valeur :

- Associé au token qui a été renvoyé par le lexer, quand c'est un token comme INT, on aura la valeur constante.
- L'arbre syntaxique est déjà déduit des autres règles de grammaire. Dans le cas de l'addition nous allons récupérer tout le sous-arbre déjà précédemment déduit des règles de grammaire.

Dans notre fichier fsy nous allons lui spécifier à quel type de base correspond notre arbre syntaxique :

```
%type < AbstractSyntax.Prog > start
```

Avec cette ligne nous avons indiqué que « Prog » est l'élément de base. Et que le parsing va commencer par les règles « start »

```
start: File EOF { $1 }
```

« start » est la première règle de grammaire de notre langage. Nous indiquons que le fichier contient « File » et se termine par EOF.

```
File: StatementList { Program(List.rev($1)) }
```

La règle « File » va déterminer le premier élément de notre arbre syntaxique « Program ». Program contient une StatementList qui va être la liste d'instruction de notre programme.

```
StatementList: Statement SEMI              { [$1] }
      | StatementList Statement SEMI { $2::$1 }
```

« StatementList » nous indique qu'une instruction est séparée par un point-virgule. La première ligne décrit comment la première ligne est interprétée. Ainsi, un « statement » suivi d'un point-virgule indique que c'est une liste de seulement un « statement ». Sinon, nous avons une liste de « statements » suivie d'un autre « statement ». Dans ce cas-là il faut ajouter dans la liste notre nouveau « statement ».

Analyse Sémantique

Une fois que notre arbre syntaxique est créé, nous allons analyser ce langage pour vérifier que notre programme est cohérent. Nous allons vérifier si les variables sont accessibles, si les types sont correctement utilisés etc. Un langage peut être syntaxiquement correct mais ne peut pas être exécuté. Exemple :

```
var i = x ;
```

Dans ce code nous utilisons une variable « x » qui n'a jamais été définie avant.

Closure

L'un des premiers principes est la portée des variables. Le principe de closure est de définir le scope d'une variable. Par exemple, dans le cadre d'un langage tel que le C# nous pouvons écrire des lambda, et les lambda utilisent des variables de la méthode parente. Et même après l'exécution de la méthode, les variables sont toujours disponibles. C'est cette portée qui demande à être définie. Pour simplifier, mes variables son scopées sur la méthode courante et pas la méthode enfant. Ainsi le code suivant émet une erreur :

```
var x = 0 ;
var m = fun() -> x ;
```

D'un point de vue du compilateur, « x » n'est pas défini dans le scope de la sous-méthode. Par contre, dans mon langage, j'ai souhaité gérer la redéfinition d'une variable. À tous moments je peux recréer une variable avec un autre type, et dans la suite du programme on utilisera cette variable. Exemple :

```
var i = 1;
var i = true;
```

Pour cela je recrée un scope pour la suite du programme. Ce que je veux c'est avoir un équivalent à :

```
var i = 1;
var i1 = true;
```

Je parcours mon arbre syntaxique et je le réécris en enlevant cette ambigüité. Ce qui va me permettre de faire des checks de syntaxe plus intelligentes dans la suite du langage.

L'analyse du typage.

Une étape importante dans un compilateur est de vérifier le typage. Un langage de programmation peut avoir deux typages différents :

- Typage fort : le langage ne permet pas d'utiliser une variable entière pour stocker un booléen, un float, ou autre type. (C# par exemple) ;
- Typage faible : le langage permet de réutiliser une variable pour stocker différents types de valeurs (Javascript par exemple).

Certains langages permettent un typage faible grâce à un mot clé. Exemple C# utilise le mot clé dynamic. Ainsi la variable définie par dynamic devient une variable avec un typage faible.

Qui veut dire typage fort, veut dire que le compilateur va vérifier les types. Il va donc vérifier que l'on ne met pas un entier dans une variable qui contient un booléen. C'est-à-dire que nous allons vérifier si les types utilisés à chaque opération sont compatibles entre eux. Exemple :

```
var i = 0 ;
var b = i + true ;
```

Il est sémantiquement impossible dans ce langage d'additionner un entier avec un booléen. Certains langages ne poseraient pas de problème car ils considèrent un booléen comme un entier. C'est à vous de définir la sémantique que vous souhaitez mettre derrière le langage. L'autre chose c'est le mot clé « var ». J'ai fait le choix dans ce langage de ne pas spécifier le type. Mais je suis quand même en typage fort. Ainsi comme en F# mon compilateur doit déduire le type par rapport à l'utilisation que l'on en fait. C'est ce que l'on appelle de l'inférence de type. Le principe est de déduire le type de l'utilisation. Nous allons parcourir notre arbre syntaxique pour en déduire le type des différents éléments. Pour faire de l'inférence de type, nous allons créer un nouvel arbre syntaxique. Cet arbre syntaxique va quant à lui être identique au précédent, mais avec un typage. Nous allons commencer par créer tous les types possibles :

```
type TypeName =
| Type of string * (string * TypeName) list
| TypeInterface of string * (string * TypeName) list
| TypeInt
| TypeFloat
| TypeBool
| TypeVoid
| TypeString
| TypeGeneric of string
| TypeFunc of TypeName List * TypeName
```

Nous avons défini que nos éléments pouvaient être de type :

- Objet avec des propriétés ;
- Interface avec des propriétés ;
- Entier ;
- Float ;
- Booléen ;
- Void : pour les méthodes qui ne renvoient pas de valeur ;
- String : pour les chaînes de textes ;
- Générique : un type qui nous permet dans certains cas de dire que nous ne connaissons pas le type d'une variable. Mais ce type sera déduit plus tard. Il prend un nom unique au moment de sa création pour pouvoir l'identifier ;
- Fonction : pour le typage des méthodes.

Puis nous allons reprendre notre premier arbre en ajoutant des informations de typage. Ces informations de type seront importantes pour la génération de code IL plus tard.

Exemple pour la création de variables

Notre type Create("maVariable", _) va ce transformer en :

```
| TypedCreate of TypeName * string * TypedExpression
```

Nous retrouvons le nom de notre variable, l'expression associée à la création, et un objet qui indique de quel type est la variable. Pour cela nous parcourons notre arbre syntaxique avec le pattern matching de F# :

```
| Create(name, e) ->
  let (typeAccu, typeExpression, exp) = (checkTypeExpression e typeAccu)
  let typeAccu = (name, typeExpression)::typeAccu
  in (typeAccu, TypedCreate(typeExpression, name, exp))
```

Première ligne, nous allons évaluer notre arbre d'expression pour connaître son type. Ainsi checkTypeExpression va nous renvoyer trois choses :

- typeAccu : un tableau contenant une liste de variables avec leur type associé ;
- typeExpression : le type déduit de l'évaluation de notre sous-arbre d'expression ;
- exp : le nouvel arbre d'expression typé.

Sur la deuxième ligne, je rajoute dans ma liste de variables la nouvelle variable que je suis en train de créer. Ce qui me permet d'agrandir ma liste de mapping entre mes variables et leurs types. Sur la troisième ligne je renvoie ma liste de variables et mon nouveau « TypedCreate ». Nous avons maintenant un nœud « TypedCreate » qui contient le type de notre sous-arbre « exp ».

Autre exemple l'addition

Notre type Add(ex1, ex2) va se transformer en :

```
| TypedAdd of TypeName * TypedExpression * TypedExpression
```

Nous retrouvons nos deux arbres d'expressions, et un autre objet qui détermine que c'est une addition d'un certain type.
De la même manière que pour le « Create », nous allons évaluer par pattern matching notre arbre et lorsque nous tombons sur un Add :

```
| Add(ex1, ex2) ->
  let (typeAccu, type1, ex1) = (checkTypeExpression ex1 typeAccu)
  let (typeAccu, type2, ex2) = (checkTypeExpression ex2 typeAccu)
  in compareType typeAccu type1 ex1 type2 ex2
```

Première ligne nous évaluons le premier sous-arbre qui va nous renvoyer un type et le sous-arbre de manière typé. La deuxième ligne fait la même chose avec le deuxième sous-arbre. Puis nous allons comparer les types des deux sous-arbres. Si les deux types sont identiques alors cela veut dire que le typage est cohérent. Dans ce cas-là nous allons renvoyer notre nœud « TypedAdd ». Par contre si les deux types sont différents alors nous générons une erreur car le typage est incorrect. Dans la liste des types, nous avons introduit la notion de type générique. Ce type permet de définir une variable sans connaître son type au début. Cela nous permet de déduire le type des paramètres des méthodes. Exemple :

```
var f = fun(x) -> x + 1 ;
```

De son utilisation, nous pouvons déduire que f est de type « TypeFun([TypeInt], TypeInt) ». Mais au moment où nous parcourons l'addition de « x + 1 », le type de x sera « TypeGeneric » car nous ne connaissons pas le type. C'est pour cela que nous allons ajouter dans « typeAccu » la variable x avec un nouveau type générique. Puis quand nous allons comparer les deux types, nous allons remplacer le type générique de la variable par le vrai type. Dans la méthode compare nous avons le code :

```
let compareType typeAccu type1 t1 type2 t2=
  match (type1, type2) with
  | (TypeGeneric(tg1), TypeGeneric(tg2)) ->
    (replaceType typeAccu tg2 (TypeGeneric(tg1)), TypeGeneric(tg1), TypedAdd(TypeGeneric
(tg1), t1, t2))
  | (TypeGeneric(tg1), type2) ->
    (replaceType typeAccu tg1 type2, type2, TypedAdd(type2, t1, t2))
  | (type1, TypeGeneric(tg2)) ->
    (replaceType typeAccu tg2 type1, type1, TypedAdd(type1, t1, t2))
  | (type1, type2) ->
    if type1 = type2 then
      (typeAccu, type1, TypedAdd(type1, t1, t2))
    else
      raise(TypeError("Expression at the left is " + (typeToString type1) + " and the right
is " + (typeToString type2)))
```

Il y a quatre cas :

- Les types des deux sous-arbres sont génériques. Dans ce cas-là nous remplaçons le type générique 2 des variables par le type générique 1 ;
- Le type 1 est générique et pas l'autre. Dans ce cas-là nous remplaçons le type générique 1 des variables par le type 2 ;
- Le type 2 est générique et pas l'autre. Dans ce cas-là nous remplaçons le type générique 1 des variables par le type 2 ;
- Les deux types ne sont pas génériques. Nous vérifions qu'ils sont identiques. Si différents, on renvoie une erreur.

Le principe de cela est de faire des remplacements de nos types génériques par les types déduits pendant l'évaluation de notre addition. Nous savons que si d'un côté nous avons un type générique et de l'autre un type, nous pouvons remplacer le type générique par l'autre type.

Optimisation d'arbre

Notre arbre syntaxique a été analysé et il est correct. Nous avons maintenant un typage de nos variables. Nous pouvons à ce moment évaluer notre arbre pour exécuter le programme. Si nous continuons à faire notre compilateur, nous pouvons ajouter une phase d'optimisation de notre arbre syntaxique. Tous les compilateurs modernes sont capables de faire ce genre d'optimisation. Dans mon compilateur je n'ai pas encore implémenté d'optimisation, mais voilà quelques idées d'optimisation simple.

Le remplacement des variables

Une variable constante qui n'est lue qu'une seule fois peut être remplacée directement par sa valeur. Exemple :

```
var i = 1 ;
var x = i + 2 ;
```

Nous pouvons remplacer ce code par un code similaire à :

```
var x = 1 + 2 ;
```

Evaluation

L'idée est d'évaluer un maximum notre code pour éviter qu'il soit fait à l'exécution. Ainsi notre code n'évaluera pas l'addition à l'exécution.
Si on reprend le code :

```
var x = 1 + 2 ;
```

Alors il sera remplacé par :

```
var x = 3 ;
```

Suppression des variables

Les variables qui ne sont jamais lues, ne sont pas utiles dans l'exécution. Il n'est pas intéressant de stocker la valeur d'un calcul si celui-ci n'est pas utilisé. Il faut éviter d'allouer de la mémoire pour rien, et de faire des opérations d'affectations. Si je reprends le code :

```
var x= 3 ;
```

Nous pouvons supprimer la variable, et dans le même cas, supprimer la constante 3. Attention si ce n'est pas une constante, il ne faudra pas supprimer seulement l'affectation, mais aussi l'exécution de l'arbre d'expression.

Génération de code

Génération en bas niveau

Le but du compilateur est de générer un langage de bas niveau, du MSIL dans notre cas. La stratégie est de transformer votre arbre syntaxique en suite d'instruction MSIL. MSIL intègre une notion de type et méthode, donc nous allons générer un arbre avec des classes et des méthodes. Une fois que l'on à notre arbre MSIL, il restera à générer l'assembly grâce à la création d'assembly dynamique. Pour générer du MSIL il faut comprendre que l'on est à un niveau proche du processeur ; ainsi pour faire une addition entre deux entiers, il faut charger un entier dans le premier registre, le deuxième entier dans le deuxième registre. Et puis faire l'instruction d'addition. Ainsi le résultat est dans le premier registre. Pour cela nous allons faire des Ldloc. Cela permet de charger la valeur d'une variable en haut de la pile. Ce qui donne pour une addition :

```
Ldloc.1
Ldloc.2
Add
Stloc 0
```

Pour l'addition, nous prenons la valeur de la variable 1 et la valeur de la variable 2. Puis le résultat est stocké dans la variable 0. Il reste à parcourir les deux sous-arbres pour générer le code MSIL. Pour le premier sous-arbre, nous allons demander de mettre la valeur de retour dans la variable 1. Et pour le deuxième sous-arbre, nous allons mettre la valeur de retour dans la variable 2. Alors nous allons évaluer la première branche dans une variable générée automatiquement. Puis faire la même chose avec la deuxième branche. Ce qui donnera pour un arbre TypedInt(5):

```
Ldc.i4 5
Stloc 1
```

Première instruction, nous chargeons la valeur 5 dans la pile. Puis la deuxième instruction charge la valeur en haut de la pile dans la variable 1. Si je prends l'exemple de l'addition de deux entiers, j'ai le code suivant :

```
Ldc.i4 3
Stloc 1
Ldc.i4 5
Stloc 2
Ldloc 1
Ldloc 2
Add
Stloc 0
```

De ce que nous pouvons voir, c'est que nous faisons appel à beaucoup de variables intermédiaires pour faire notre addition. A chaque nœud nous allons créer des variables intermédiaires pour stocker le résultat de notre nœud. L'idée est de générer des instructions sans se préoccuper de l'espace de stockage ou du nombre d'instructions appelées.
L'idée pour générer le MSIL est de convertir notre arbre syntaxique typé en arbre syntaxique MSIL. Voici l'arbre implémenté aujourd'hui :

```
type IlAssembly =
   Assembly of string * IlType list
and IlType =
   Class of string * IlElement list
and IlParameter = string * int
and IlVariable = string * TypeName * int
and IlElement =
   | Method of IlParameter list * IlVariable list * string * IlInstruction list
   | EntryPoint of IlParameter list * IlVariable list * string * IlInstruction list
   | Field of string * TypeName
and IlInstruction =
   | Ldc_I4 of int
   | Ldc_R4 of float
   | Stloc of int
   | Ldloc of int
   | Stfld  of string * string
```

```
| Newobj of string
| Add
| Ret
| Nop
```

Dans cet arbre, nous retrouvons la notion d'assembly, classes, méthodes... Pour le contenu des méthodes par contre nous allons retrouver une liste d'instructions. L'implémentation en F# est similaire à ce que nous avons déjà fait. Nous allons parcourir notre arbre avec le pattern matching. Par exemple quand nous allons tomber sur l'addition :

```
| TypedAdd(t, e1, e2) ->
  let var1 = returnVar + 1
  let var2 = returnVar + 2
  let variables = List.append variables [("", t, var1);("", t, var2)]
  let (variables, instr1) = (tollExpression e1 variables var1)
  let (variables, instr2) = (tollExpression e2 variables var2)
  let instr = List.append instr1 instr2
  in (variables, (List.append instr [Ldloc(var1); Ldloc(var2); Add ; Stloc(returnVar)]))
```

Les trois premières lignes permettent de générer deux variables qui vont stocker les valeurs de retour des deux sous-arbres d'expression.
Ensuite nous allons générer les instructions des sous-arbres d'expression de gauche et de celui de droite. Nous allons retrouver dans « instr » la liste des instructions pour les deux sous-arbres d'expression. A cette liste d'instructions nous allons ajouter les quatre instructions vues précédemment pour faire une addition. Nous allons renvoyer deux choses :

- Une liste variable qui nous permet d'exécuter le code MSIL généré. La liste est un tuplet de trois éléments :
  - Un nom (Vide si c'est une variable intermédiaire) ;
  - Le type de la variable à créer ;
  - Un numéro unique incrémenté.
- La liste des instructions MSIL générées.

Optimisation MSIL

Maintenant que nous avons généré une suite d'instructions MSIL, en regardant les instructions, nous voyons qu'il est possible de faire des optimisations pour améliorer les performances du langage.
Exemple si nous avons :

```
Stloc 3
Ldloc 3
```

Il faut aussi vérifier que dans la suite du code nous n'avons plus de « Ldloc 3 » ; dans ce cas-là nous pouvons facilement enlever les deux instructions.
Dans l'étape précédente nous avons généré beaucoup d'instructions sans regarder l'utilisation des variables. L'idée était de ne pas se préoccuper des problématiques pour transmettre les valeurs entre les différentes instructions. Avec cette étape nous réduisons le nombre d'instructions et le nombre de variables pour stocker les valeurs intermédiaires.

Builder

Le builder est la dernière partie de notre compilateur. Nous allons la faire en deux étapes :

- Créer des builders pour toutes les classes, méthodes, champs, variables, ...
- Utiliser ces builders pour appeler les instructions.

Quand nous faisons un appel à un champ de classe, nous avons besoin de passer le FieldInfo en paramètre à la méthode « Emit ». Pour cela il faut créer la définition de la classe et des champs associés. C'est pour cela que la première étape va créer le FieldBuilder. Ce qui permet d'avoir une référence vers la définition du champ au moment où nous générons les instructions MSIL.

```
and buildEmitIl e ilGenerator locals classes members =
  match e with
  | Nop -> ilGenerator.Emit(OpCodes.Nop)
  | Ldc_I4(i) -> ilGenerator.Emit(OpCodes.Ldc_I4, i)
  | Ldc_R4(i) -> ilGenerator.Emit(OpCodes.Ldc_R4, i)
  | Stloc(index) -> ilGenerator.Emit(OpCodes.Stloc, ((findLocal locals index):LocalBuilder))
  | Ldloc(index) -> ilGenerator.Emit(OpCodes.Ldloc, ((findLocal locals index):LocalBuilder))
  | Stfld(className, name) -> ilGenerator.Emit(OpCodes.Stfld, ((findField classes class
Name name):FieldBuilder))
  | Newobj(name) -> ilGenerator.Emit(OpCodes.Newobj, ((findTypeConstructor classes name)
.GetConstructor([||])))
  | Add -> ilGenerator.Emit(OpCodes.Add)
  | Ret -> ilGenerator.Emit(OpCodes.Ret)
```

Ainsi quand nous tombons sur une instruction, la méthode « Emit » est appelée pour générer l'opération associée. Dans le cas de LdLoc par exemple, le « LocalBuilder » est indispensable pour générer l'instruction. Du coup il faut appeler la méthode « findLocal » pour récupérer le « FieldBuilder » précédemment créé et associé au numéro de la variable (« index »).
De même pour les opérations :

- Stfld qui permet de stocker dans un champ de classe. Nous cherchons le FieldBuilder associé ;
- Newobj qui permet de créer une instance d'objet. Nous cherchons le constructeur par défaut du TypeBuilder associé.

Pour créer notre assembly exécutable, il reste à le sauvegarder :

```
(buildAssembly a:AssemblyBuilder).Save(assemblyOutput, PortableExecutableKinds.
Preferred32Bit, ImageFileMachine.I386)
```

Le buildAssembly renvoie notre AssemblyBuilder avec tous nos types et méthodes créés. Il suffit de faire un Save en préférence 32bit pour une machine de type i386.

Conclusion

En décortiquant un compilateur, nous voyons que chaque phase est indépendante. Dans beaucoup de cas, ces phases peuvent être utilisées pour répondre à un besoin particulier. Par exemple, le principe de construire un arbre syntaxique pour parcourir un fichier, permet de bien différencier la partie lecture du fichier, de la partie analyse de ce fichier.
Nous avons vu toutes les parties importantes d'un compilateur. Un compilateur est quelque chose de compliqué. Mais nous avons vu que les concepts de base sont simples. La difficulté est de définir notre langage. Plus nous allons chercher à faire un langage complexe, plus notre compilateur devra être complexe. Par exemple l'inférence de type demande au compilateur de faire tout le travail pour vérifier si notre langage est sémantiquement correct. La seule chose que nous n'avons pas regardée est la gestion des erreurs. Ainsi il faudrait stocker pour chaque nœud de notre arbre la ligne et la colonne, ce qui nous permettrait de renvoyer l'emplacement des erreurs. De plus les compilateurs modernes ne s'arrêtent pas sur la première erreur, ils continuent à vérifier les erreurs.
Le code complet du compilateur est disponible en open source sur https://github.com/swoog/Kiss. Ce compilateur est loin d'être terminé, il reste beaucoup de cas à générer. Grâce à F#, les cas manquants sont les warnings indiquant les matching manquants dans les parcours des arbres syntaxiques. De plus le compilateur est développé en TDD ce qui me permet de valider chaque phase du compilateur sans régression.

Etendre VSTS avec les extensions 1ère partie

Jusqu'à présent, Microsoft avait mis à disposition des APIs qui permettaient de visualiser et de manipuler les données de Visual Studio Team Services – le nouveau nom de Visual Studio Online – à partir d'applications tierces.

Mikaël Krief
Microsoft MVP
Visual Studio ALM Rangers
Consultant ALM chez **Cellenza**
@MikaelKrief

Avec ses API, de nombreuses plateformes utilisent ses services afin de s'intégrer avec VSTS. C'est le cas par exemple de Trello, UserVoice, Jenkins, etc. Annoncé lors de l'évènement Connect() qui a eu lieu en Novembre, Visual Studio Team Services (VSTS), n'est plus un outil figé mais peut être désormais étendu grâce à des extensions. Qu'est-ce que vont nous apporter ces extensions ? Comment développe-t-on une extension pour VSTS ? Et peut-on installer ces extensions dans TFS 2015 on-premise ?

Qu'est-ce qu'une extension ?

L'un des problèmes rencontrés avec les API existantes est l'obligation de créer une application tierce pour appeler les API. Une extension va permettre d'ajouter de nouvelles fonctionnalités au portail VSTS en appelant les API de VSTS tout en conservant une ergonomie homogène, car nous allons utiliser les mêmes composants graphiques que ceux de VSTS.
Une extension peut étendre VSTS de plusieurs façons : une nouvelle page dans le portail ; une action faite par un nouvel élément dans un menu ; un widget pour le dashboard ; une section pour étendre le formulaire d'un work item ; une tâche de build personnalisée.

Le Visual Studio Marketplace

Le Visual Studio Marketplace est le portail qui référence pour toutes les extensions publiques mises à disposition. A partir de celui-ci, on peut voir la liste des extensions classées par catégorie (code, plan and track, build et release, collaboration, intégration et tests), visualiser leurs fiches détaillées et installer une extension sur son compte VSTS. L'installation d'une extension se fait depuis sa fiche détaillée en cliquant sur le bouton Installer.

Créer une extension

Les prérequis

Les langages utilisés pour créer une extension sont le Javascript (ou TypeScript), le HTML et CSS donc pas besoin d'avoir des connaissances en C# ou tout autre langage serveur.
Concernant l'éditeur de code, Visual Studio Code suffit, le développement d'une extension a été conçu pour être multiplateforme. Cependant avec Visual Studio il sera possible d'utiliser un Template de projet. Pour tester l'extension, un compte VSTS est nécessaire, avec un projet d'équipe qui contient les données servant au test de l'extension.
Enfin, un publisher est requis, celui-ci correspond à l'espace d'upload de l'extension. Pour créer un publisher se rendre sur http://aka.ms/vsmarketplace-manage, les informations à renseigner sont un identifiant et un nom Fig.1.

Le Template de projet pour Visual Studio

Pour créer une extension, il existe un Template de projet pour Visual Studio, disponible dans la galerie Visual Studio https://visualstudiogallery.msdn.microsoft.com/a00f6cfc-4dbb-4b9a-a1b8-4d24bf46770b celui-ci contient tous les fichiers nécessaires, avec un exemple d'extension prête à être déployée. L'utilisation de ce template est facultative, l'ensemble des fichiers pouvant être créé manuellement si l'on utilise Visual Studio Code par exemple.

Fig.1

Architecture du projet

Une fois le projet généré depuis le template, on obtient l'ensemble des éléments nécessaires à l'extension.
Si on observe bien, il s'agit d'une petite application Web avec :

- Un fichier HTML qui affichera le contenu de l'extension ;
- Un fichier CSS pour la mise en forme ;
- Un fichier TypeScript qui fera le traitement désiré par l'extension ;
- Quelques fichiers JavaScript : les librairies nécessaires ;
- Un fichier Json de configuration (vss-extension.json) ;
- Un fichier markdown qui servira pour la fiche détaillée dans le Visual Studio Marketplace.

D'autres fichiers peuvent se rajouter comme le fichier *gruntfile.js* qui comporte des tâches de packaging et de déploiement de l'extension, ou encore des fichiers pour les tests JavaScript. L'avantage d'utiliser le template pour Visual Studio c'est qu'il crée un projet avec tous ses fichiers et quelques lignes de code pour en donner un exemple d'utilisation.

Le fichier de configuration

Le point d'entrée de l'extension est le fichier de configuration *vss-extension.json* (aussi appelé manifest). Celui-ci contient toute la configuration de l'extension et est composé de 3 parties :

- Les propriétés de l'extension telles que son Nom, l'id, la version, le scope, la visibilité (publique ou privée) ainsi que l'identifiant de votre publisher.

```
"manifestVersion": 1,
"id": "sample-extension",
"version": "0.1.0",
"name": "Sample Extension",
"scopes": [ "vso.work", "vso.work_write" ],
"description": "Description de l'extension",
"publisher": "fabrikam",
"public": false,
```

- Les informations qui seront affichées sur la page du Marketplace comme les liens, le nom du fichier markdown qui contient les infos de la fiche détaillée, les tags ainsi que la couleur et le thème de la page.
- Les contributions qui représentent l'emplacement de l'extension dans VSTS. C'est dans cette partie que l'on indique s'il s'agit d'une nouvelle page, une nouvelle entrée dans un menu contextuel, si c'est une nouvelle tâche de build, ou bien même un widget pour le dashboard.

```
"contributions": [
  {
    "id": "home",
    "targets": [
      "ms.vss-work-web.work-hub-group"
    ],
    "type": "ms.vss-web.hub",
    "properties": {
      "name": "Sample Contribution",
      "order": 100,
      "uri": "index.html"
    }
  }
]
```

Tous les détails sur ce fichier manifest se trouvent sur la documentation officielle https://www.visualstudio.com/en-us/integrate/extensions/develop/manifest

Utiliser les données de VSTS

Une extension pour VSTS a pour but d'ajouter ses propres fonctionnalités au sein même du portail VSTS en utilisant et manipulant les données de VSTS. Microsoft a donc exposé une API REST qui va permettre d'interagir en lecture comme en écriture avec les données stockées dans VSTS. Ces données retournées peuvent appartenir à tous les niveaux de VSTS, que ce soit le contrôleur de source, les work items, les builds, les projets ou l'administration. Ces API REST sont appelées depuis les fichiers Typescript (ou JavaScript). Elles ont pour avantage d'implémenter nativement toute la partie d'authentification. La documentation sur l'API client se trouve dans la documentation officielle dans la section API.

Rester intégré avec les composants d'interfaces

En plus des API fournies, Microsoft met à disposition des composants graphiques qui sont identiques à ceux présent dans VSTS. Ces composants sont à l'heure actuelle : la grid, une combobox, un splitter, une fenêtre modale, un treeview et un menu. L'utilisation de ces contrôles est assez simple et permet donc à l'extension d'avoir la même expérience utilisateur que les interfaces natives de VSTS. Voici un exemple d'extension qui récupère le résultat d'une requête VSTS et qui l'affiche dans une grid par utilisateur et par état des work items : Fig.2. Par les client API on récupère les données de la requête, puis on les affiche dans la grid.

Mode d'hébergement du contenu

Une extension pour VSTS se compose de 2 parties : la 1ère est son fichier de configuration, la 2eme est composée de ses fichiers de contenus (html, css, js, images...). Concernant ces fichiers de contenus il y a 2 possibilités d'hébergement :

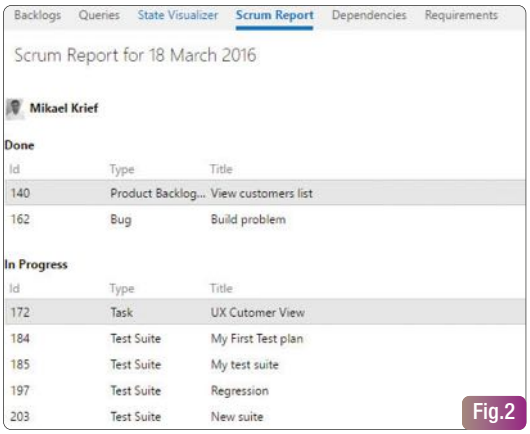


Fig.2

- Sur un hébergeur tiers, comme par exemple dans une Web App Azure
- Les fichiers sont contenus dans le package et seront donc hébergés dans la plateforme VSTS.

Le choix d'un hébergeur tiers est utilisé si l'extension a besoin d'une logique métier externe. Pour indiquer à VSTS ce site, il faut rajouter dans le fichier de configuration l'url (obligatoirement en https) de celui-ci dans la propriété baseUri.

```
"baseUri": "https://localhost",
```

Dans l'autre mode où tous les fichiers sont intégrés au package, c'est l'utilisation de la propriété *Files* qui indique la liste des répertoires et fichiers à packager.

```
"files": [
  {
    "path": "scripts",
    "addressable": true
  },
  {
    "path": "images",
    "addressable": true
  },
  {
    "path": "index.html",
    "addressable": true
  }
]
```

Debugger une extension

Pour débugger une extension VSTS, on peut :

Utiliser les outils développeurs intégrés au navigateurs Web comme les outils développeur de IE ou Chrome et Firebug pour Firefox. Ces outils permettent de debugger en direct du code JavaScript ou aussi inspecter le code html. Cette solution est pratique pour debugger une extension qui est déjà publiée.

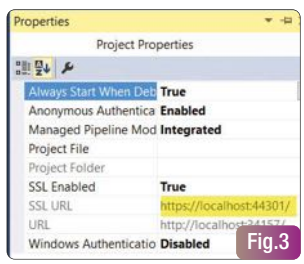


Fig.3

- Utiliser le débugger local en indiquant la propriété baseUri avec l'url https du website local intégré à Visual Studio, et en indiquant la page de VSTS à lancer dans la configuration du projet. Ainsi vous pouvez directement débugger en local directement Fig.3 et 4.

Configurer la page du Visual Studio Marketplace

C'est dans le fichier de configuration vss-extension.js que se configure la page de détails de l'extension sur le Marketplace. Cette page est constituée :

- d'un logo

```
"icons": {"default": "images/squareicon.png"}
```

- d'une catégorie et de tags

```
"categories": [ "Plan and track"],
"tags": [
  "scrum",
  "report"
],
```

- d'une couleur et d'un thème pour l'en tête

```
"branding": {
  "color": "rgb(220, 235, 252)",
  "theme": "light"
},
```

- de captures d'écran :

```
"screenshots": [ { "path": "screenshots/screen1.png" }, { "path": "screenshots/screen2.png" } ],
```

- de liens externes par types

```
"links": {
  "home": {
    "uri": "https://bit.ly"
  },
  "getstarted": {
    "uri": "https://bit.ly"
  },
  "learn": {
    "uri": "https://bit.ly"
  },
  "support": {
    "uri": "https://bit.ly"
  },
  "repository": {
    "uri": "https://bit.ly"
  },
  "issues": {
    "uri": "https://bit.ly"
  }
},
```

- Le chemin vers le fichier de licence et le chemin vers le fichier markdown qui contient tout le texte de la page

```
"content": {
  "details": {
    "path": "overview.md"
  },
  "license": {
    "path": "Licence/MIT-1.txt"
  }
},
```

Visibilité de l'extension

La visibilité de l'extension dépend de la visibilité du publisher dans lequel elle est inscrite. Par défaut un publisher n'est pas public, toutes ses extensions sont alors privées, la propriété *public* du fichier de configuration vaut *false* obligatoirement. Dans ce cas il est possible de partager une extension avec un autre compte VSTS en utilisant la fonction Share Fig.5.

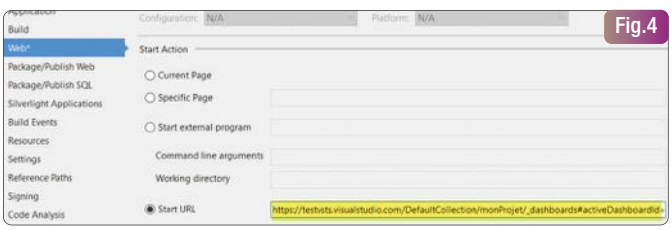


Fig.4

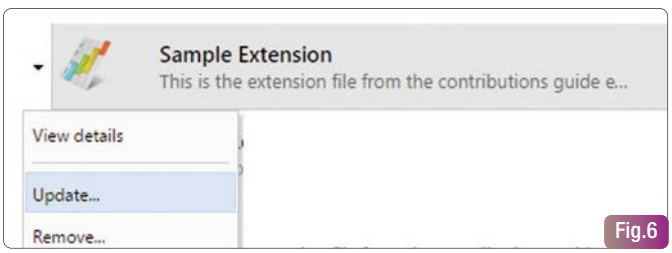


Fig.6

Pour rendre public un publisher il faut en faire la demande comme l'indique la procédure dans la documentation officielle. Puis pour rendre l'extension publique, mettre la propriété *public* du fichier de configuration à true.
Il est aussi possible d'indiquer qu'une extension est en Preview avec la propriété "galleryFlags": ["Public", "Preview"]

Création du package

Dès que l'extension est prête à être testée ou à être publiée, elle doit être packagée sous un format VSIX. Pour créer le package il faut utiliser l'utilitaire *tfx-cli* disponible via *NPM*, puis lancer en ligne de commande dans le répertoire de l'extension, la commande :
tfx extension create –manifest-globs vss-extension.json
Cet outil possède d'autres paramètres qui permettent par exemple de changer les valeurs du fichier de configuration lors du packaging.

Publier l'extension dans le Marketplace

Le package généré doit maintenant être publié dans le Visual Studio Marketplace

- Manuellement : en uploadant la package via l'interface du Visual Studio Marketplace Fig.6.
- Avec une commande du tfx-cli

Au prélable, il est nécessaire de créer un token dans VSTS, qui autorisera l'upload via une ligne de commande. Puis exécuter la commande :

```
tfx extension publish --manifest-globs vss-extension.json --token <votre-token>
```

L'extension est maintenant publiée dans le Visual Studio Marketplace.

Gérer vos extensions

VSTS permet de gérer les extensions installées sur son compte, à travers cette interface il est possible de désactiver ou de supprimer une extension.

Monitorer l'activité d'une extension

Depuis Février, Microsoft a intégré la possibilité de monitorer des extensions VSTS grâce au service Azure Application Insight.
Celui-ci va permettre d'avoir des indicateurs sur :

- Le nombre d'utilisateurs qui utilisent l'extension ;
- Les fonctionnalités les plus (ou moins) utilisées ;
- L'endroit où l'extension est-elle utilisée géographiquement.

Il suffit dans Azure de créer une nouvelle ressource Application Insight, et de rajouter le code JavaScript fourni dans les pages de l'extension.

Conclusion

Bien que depuis longtemps les APIs de VSTS permettaient de communiquer avec VSTS, aujourd'hui les extensions permettent en plus de s'intégrer dans VSTS et ainsi de fournir une meilleur expérience utilisateur.
La disponibilité des extensions dans TFS 2015, va offrir aux entreprises la possibilité d'intégrer leur besoins métiers autour de l'ALM au sein même de Team Foundation Server.

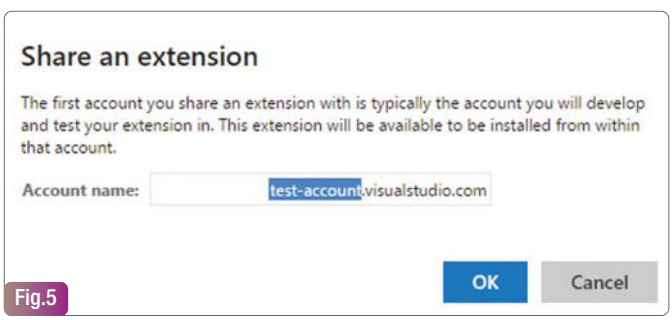


Fig.5

A la découverte du .NET Core

L'écosystème .NET a connu de nombreux changements en 2015 notamment avec l'apparition des Universal Windows Platform apps ou encore de la dernière version d'ASP.NET intitulée pour l'occasion ASP.NET Core 1. Les problématiques modernes de développement (cross-platform, Cloud first, mobile first ...) nous demandent à nous, développeurs, d'être toujours plus agiles et performants dans nos développements d'aujourd'hui. De ce fait, nous demandons également à nos outils d'en faire de même, et de nous permettre de mieux nous adapter aux techniques de développement d'aujourd'hui.

Christophe GIGAX

La plateforme Microsoft ne déroge pas à la règle et c'est ainsi que .NET Core est né. Le framework .NET Core est initialement un simple fork du .NET Framework classique que l'on connaît, mais en se *focalisant sur les problématiques modernes de développement citées ci-dessus.*

Principes de bases

Le .NET Core a été conçu en parallèle avec ASP.NET Core 1 afin de mieux servir le développement Web dans l'écosystème Microsoft et permet également de rassembler les principes suivants :

- **Pay-as-you-go** signifie que le framework Core permet d'inclure dans vos déploiements uniquement ce qui est nécessaire pour faire fonctionner votre application ; inutile donc d'installer des composants / librairies non utilisées ;
- **Cross-platform** intervient dans le domaine de la portabilité de l'application et permet ainsi de lancer une application Web sur un serveur Linux ou OS X ;
- **Cloud-optimized** indique que le .NET Core est plus léger et permet un déploiement plus facile dans le Cloud tout en garantissant uniquement les packages qui sont nécessaires à l'application et ainsi éviter de transférer des briques logicielles inutiles.

De plus, .Net Core est open-source et s'ouvre à la communauté pour le plus grand bonheur des contributeurs. Avec ceci, on se retrouve avec un Framework modulaire à souhait qui permet facilement de s'exporter vers d'autres plateformes et d'embarquer de lui-même les packages dont il a besoin (via NuGet). Nous allons voir un peu plus en détail ce qui compose .NET Core.

CoreCLR

Depuis les débuts du framework .NET, la plateforme de Microsoft embarque année après année un moteur d'exécution très puissant appelé le Common Language Runtime. Ce dernier offre un environnement managé d'exécution des logiciels développés et permet une compilation à la volée pour ainsi transformer un code intermédiaire en code natif spécifique au système d'exploitation Windows. Avec l'expansion de l'OpenSource dans le monde de l'informatique et l'impact du Cloud dans nos développements d'aujourd'hui, Microsoft a du réécrire son célèbre moteur pour ainsi le rendre plus performant, cross-platform et plus léger. En effet, faisant partie intégrante de l'écosystème de .NET Core, CoreCRL se doit de fonctionner sur Windows, Linux et Mac. Une phrase résume assez bien la philosophie derrière CoreCLR :

> *Le but du CLR est de rendre la programmation plus facile*

L'un des composants les plus connus de CoreCLR est le Garbage Collector (GC), qui permet aux développeurs de travailler dans un environnement managé et donc d'éviter de se soucier des problématiques de libération de mémoire ou autre. Ce dernier a également été réécrit pour intégrer les nouvelles contraintes de .NET Core.

Deux éléments composent le GC aujourd'hui : l'allocateur et le collecteur. Chacun dans leurs rôles respectifs permet de gérer la mémoire soit en allouant sur demande de la mémoire pour les programmes, soit en collectant les objets en fin de vie ou inutilisés. Cependant, les principes de base du GC sont restés les mêmes :

- Le GC doit s'exécuter assez souvent pour éviter que la pile ne se remplisse d'objets inutilisés ;
- Le GC ne doit pas s'exécuter trop de fois non plus pour éviter d'utiliser trop de temps CPU ;
- Le GC doit être éphémère et ne doit pas conserver de la mémoire ou du temps de CPU quand il ne s'exécute pas ;
- Chaque collecte par le GC doit être rapide et efficace ;
- Le code managé des développeurs ne doit pas connaître le GC.

Ecrit à la fois en C# et en C++, le CoreCLR contient approximativement 2,6 millions de ligne de code, et le GC à lui seul rassemble 55k lignes tout en utilisant un nouveau compilateur multiplateforme qui est CMake. De ce fait, Microsoft réaffirme sa volonté de s'ouvrir à l'Open Source.

CoreFX

Anciennement connu sous le nom de BCL (Base Class Library), CoreFX est le 2ème composant essentiel de .NET Core pour ainsi faire fonctionner les applications dans le nouvel écosystème de Microsoft. Composé de plusieurs librairies connues telles que **System.Collections** ou encore **System.XML**, ces briques logicielles possèdent maintenant le moins de dépendances entre elles, permettant ainsi de mieux moduler les importations faites dans les applications. Tout comme CoreCLR, CoreFX est distribué via des paquets Nuget et chaque librairie a son propre package.

L'exemple le plus probant que nous pouvons citer pour montrer toute l'ouverture de .Net Core est que la communauté Mono peut maintenant bénéficier de la librairie CoreFX et permet ainsi d'éviter de dupliquer des librairies communes. De plus, .NET ne se veut pas forcément plus « petit » que le framework .NET complet, mais puisque maintenant il suffit d'importer uniquement ce dont on a besoin, les applications d'aujourd'hui auront un impact au niveau du déploiement moins significatif qu'avant. CoreFX participe également à sa manière à la révolution du Cloud.

Au niveau du code, CoreFX est écrit entièrement en C# et représente 25% du code du CoreCLR, soit à peu près 500 000 lignes de code. L'avantage ici est que l'on détient alors une librairie consistante et managée quelle que soit la plateforme utilisée. Avec ceci, la librairie CoreFX inclut quelques nouveautés telles que :

- Les **FormattableString** proposant de

nouvelles méthodes pour mieux formaliser les chaînes de caractères selon des conventions de culture invariantes ;
- Les **ImmutableArray** permettent de mieux gérer les collections fixes pour ainsi améliorer les performances et éviter tout problème au niveau de l'intégrité de la collection (notamment si ce sont des collections qui sont exposées via des API). Le pattern **Builder** est quant à lui mis ici en pratique pour le côté amélioration des performances ;
- Les **Sockets** et **HttpClient** sont maintenant disponibles via CoreFX ;
- Inspiré du C++, on retrouve des classes telles que **Vector2** ou **Vector3** pour faciliter les calculs vectoriels. Ces dernières sont notamment utilisées dans **l'API Composition** disponible dans les *Universal Windows Platform* apps afin de mieux gérer les animations ;
- Certaines méthodes de la librairie **Reflection** sont maintenant devenues des méthodes d'extensions (comme GetMembers()).

Lors de la compilation avec CoreCLR, le résultat génère plusieurs éléments bien distincts. On retrouve l'exécutable d'un côté, les DLLs de CoreFX d'un autre côté et les références à des librairies externes à part également. En revanche, lors de la compilation en mode Release (avec .NET Native), le résultat génère une seule DLL avec tout le code managé et toutes les dépendances et librairies dont l'application a besoin pour fonctionner.
Les particularités de tous ces changements se retrouvent également dans la nouvelle version de NuGet. Dans les précédentes versions de .NET, les dépendances relatives à NuGet était intégrées au .csproj du projet, ce qui pouvait rendre difficile la maintenance via un contrôleur de code source. Les dépendances ont maintenant été séparées dans un projet à part intitulé **project.json**, permettant de mieux gérer les packages importés dans le projet. Chaque package peut potentiellement importer d'autres dépendances, qui nous sont inconnues, mais qui sont indispensables au bon fonctionnement de la librairie. NuGet va toutefois nous montrer uniquement les dépendances que nous avons importées, et non les autres, permettant ainsi de mieux se retrouver dans l'arborescence des dépendances. Il est quand même possible de voir toute la liste de dépendances importées dans le projet via le fichier **project.lock.json**.
Via Nuget, la diffusion des librairies et du Framework s'en retrouve ainsi facilité, tout comme la diffusion des nouvelles fonctionnalités et des bugs fixes.

Roselyn

Grâce au CoreCLR et au CoreFX, nous avons ainsi de quoi faire fonctionner du code au runtime avec des librairies modulables à souhait, tout ceci optimisé pour le cross-platform et le Cloud. Cependant, il est clair qu'un développeur souhaitant développer dans l'écosystème de Microsoft à partir de Linux ne pourra pas le faire puisqu'il ne possède pas de compilateur.
A ce sujet, Microsoft a publié sur GitHub il y a plusieurs mois son compilateur : Roselyn.
Avant le développement de l'Open Source, les compilateurs étaient des boîtes noires et très peu de personnes ne pouvaient accéder au fonctionnement interne. Aujourd'hui, le monde a bien changé, et les compilateurs s'ouvrent et deviennent des APIs pouvant être utilisables dans d'autres contextes que des IDE. Les compilateurs deviennent des plateformes, permettant ainsi à la créativité de chacun d'opérer pour améliorer l'efficacité de ces outils.
Roselyn ne déroge pas à la règle, et a subi un profond remaniement afin de rendre le compilateur tel une plateforme utilisable de l'extérieur. Le pipeline d'exécution de Roselyn est le suivant : Fig.1.
Chaque phase est maintenant séparée en différents composants. La première phase est un simple parsing du code source pour mieux analyser la syntaxe suivant une certaine grammaire du langage (C#, VB...). On retrouve ensuite la phase de déclaration, permettant d'associer les métadonnées et les déclarations du code source pour former des symboles nommés. Ensuite, la phase d'association, ou de « bind », permettant d'associer les identifiants du code avec les symboles. Enfin, la dernière phase permet de reprendre toutes ces informations et de produire un assembly.
Chaque phase est alors exposée via un modèle objet permettant d'y accéder depuis l'extérieur. La phase de parsing est exposée via un arbre de syntaxe, la phase de déclaration via une table des symboles hiérarchique, la phase de

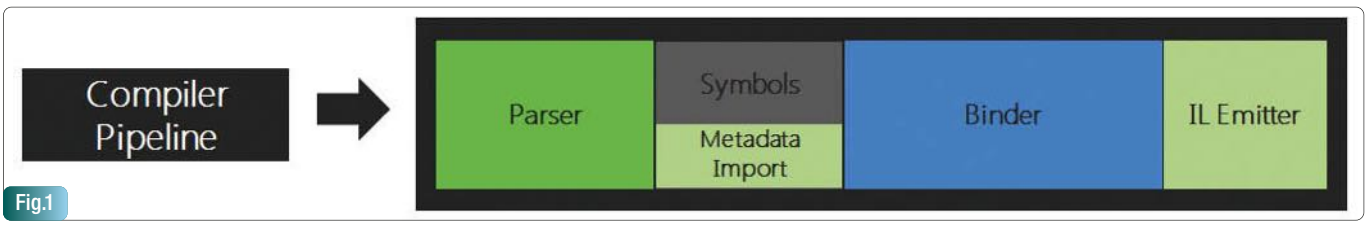


Fig.1

Processus d'exécution de Roselyn

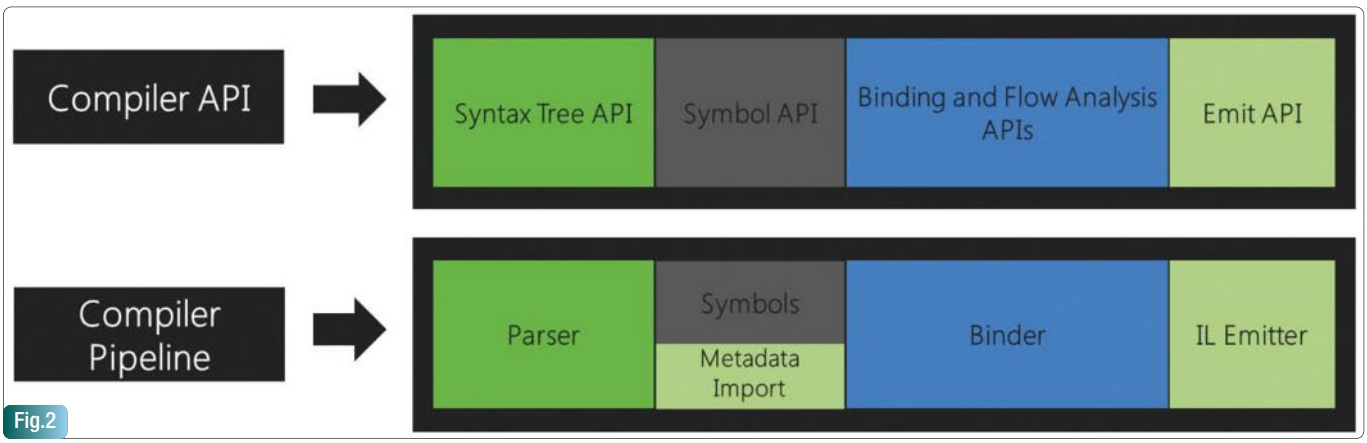


Fig.2

Surcouche d'API afin d'accéder de l'extérieur au compilateur

« binding » via un modèle exposant le résultat de l'analyse et la dernière phase est représenté par une API qui produit le code intermédiaire Fig.2.

Pour prouver la bonne faisabilité de ce mécanisme, Microsoft a intégré ces APIs dans Visual Studio 2015 via des menus comme **Go to Definition**, **Object Browser** ou encore **Find all references**. Devenu Open Source, cross-platform et ouvert via des APIS, Roselyn est devenu un excellent outil de compilation qui permet maintenant bien plus d'usage qu'auparavant.

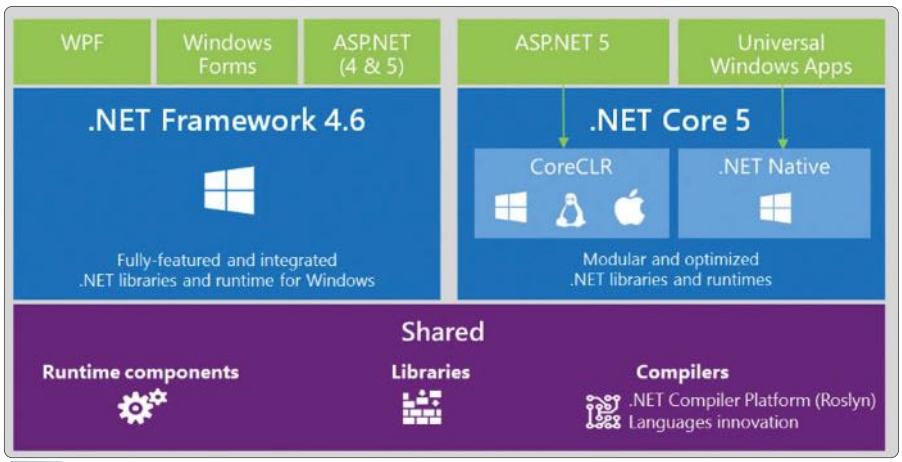


Fig.3 *Schéma résumant bien la disposition de .NET Core dans la plateforme de Microsoft*

RyuJIT

RyuJIT est la nouvelle génération de compilateur développé par Microsoft ; il arrive avec la nouvelle boîte à outils proposée par .NET Core. Spécialement conçu pour les systèmes 64 bits, il sera à l'avenir la base pour tous les compilateurs de Microsoft : x86, ARM, MDIL... Les principales caractéristiques de RyuJIT sont :

- Amélioration des performances de compilations (jusqu'à 30% plus rapide au démarrage des applications) ;
- Diminution de l'empreinte mémoire ;
- Homogénéité du code entre plateformes 32 bits et 64 bits ;
- Fin à la fragmentation du code des compilateurs.

Grâce au travail de fond réalisé par Microsoft, RyuJIT est supporté sur Linux et OS X. Intégré maintenant dans le .NET Core, RyuJIT est utilisé dans les nouveaux projets ASP.NET 5 pour assurer la portabilité de ces projets sur les autres plateformes.

.NET Native

Depuis les débuts de .Net, les applications conçues sur la plateforme de Microsoft subissaient d'abord une pré-compilation en code intermédiaire puis une 2ème compilation en code natif. Avec l'arrivée de .NET Native, il n'est plus nécessaire de passer par la pré-compilation pour faire fonctionner du code C# ou VB. Les applications sont ainsi plus rapides avec un faible coût de démarrage et une utilisation optimisée de la mémoire.

.NET Native est étroitement lié à .NET Core dans le sens où .NET Native est un ensemble d'outils, intégré à .NET Core, permettant de compiler des applications Universelles via une compilation statique AOT (ahead-of-time) avant son exécution, à l'inverse de CoreCLR qui compile à la volée. On retrouve alors 2 compilations bien distinctes en fonction du type d'application, mais utilisant les mêmes librairies sous-jacentes :

- Compilation à la volée avec CoreCLR pour les applications ASP.NET Core 1 ;
- Compilation AOT statique avec .NET Native pour les applications Universelles Windows 10. Fig.3.

Embarquez vers la Constellation 1ère partie

Pour les lecteurs réguliers de Programmez! vous entendez parler de la plateforme Constellation depuis maintenant plus d'un an. Pour rappel, Constellation est une plateforme d'interconnexion des objets, des services et des applications multi-langages et multi-technologies.

Sébastien Warin
Owner / CEO @ myConstellation.io
http://www.myconstellation.io
http://sebastien.warin.fr

Concrètement, un script Python sur un Raspberry peut par exemple invoquer des fonctions d'une application .NET/WPF, d'un Arduino ou bien d'une page Web Javascript. En clair, n'importe quel système/application connecté dans Constellation peut échanger des données et invoquer des méthodes avec les autres clients connectés.

Il devient alors possible d'imaginer des scénarios très complets, car toute la tuyauterie technique est cachée par Constellation que ce soit dans un contexte Smart Home/IoT ou autre.

Principes et concepts proposés par Constellation

S'il fallait résumer la plateforme Constellation, on pourrait dire qu'elle repose sur trois grands principes :

- Orchestration : déployez vos packages sur un réseau de sentinelles et gérez la configuration, le versionning, les logs depuis la console Constellation ou vos propres applications.
- Messaging : tout système connecté à Constellation peut envoyer et recevoir des messages en temps réel dans un langage universel.
- StateObjects : l'état d'une lampe, d'une batterie, d'une porte, le volume de la radio ou la température d'une chambre, ... sont tous des StateObjects.

Constellation simplifie le développement, le déploiement et facilite l'administration des services et des programmes qui la composent tout en maximisant la facilité d'interconnecter ces composants. En effet, dans la Constellation tout est « package », on y retrouve donc ces éléments que l'on peut, d'une certaine manière, assimiler à des services, des applications ou des objets. Chacun de ces packages est un processus qui s'exécute sur une machine qui communique à travers un bus commun.

L'exécution de ces packages s'opère sur des sentinelles qui peuvent correspondre à n'importe quel type de machine, Windows ou Linux. Toutes ces sentinelles communiquent avec un serveur centralisant les configurations et les informations partagées, le serveur Constellation. L'ensemble des données produites par les packages peuvent être remontées au serveur Constellation en temps réel.

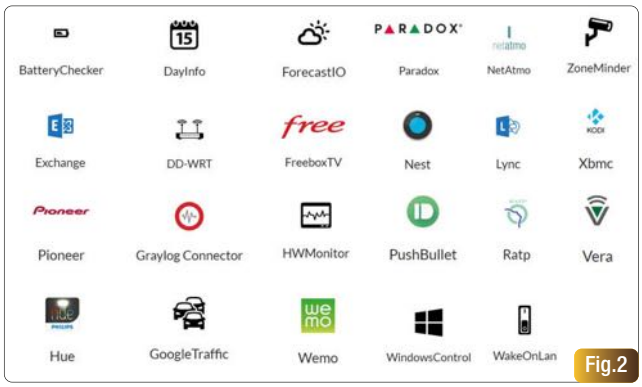


Fig.2

Catalogue de package

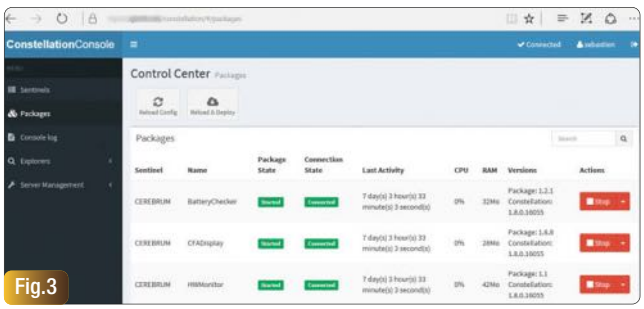


Fig.3

Constellation Console 1.8

Fig.4

S-Panel

De plus, toutes les configurations des packages sont centralisées sur le serveur et la mise à jour de ces packages devient aisée puisqu'aucune action sur l'équipement n'est alors directement nécessaire.

Une fois les packages déployés et orchestrés par la Constellation, ils sont tous connectés dans cette même Constellation et peuvent alors échanger des messages entre eux.

Chaque package peut envoyer et recevoir des messages avec les autres packages mais également avec des pages HTML/Javascript connectées sur le hub, ou encore avec n'importe quel système connecté sur l'interface HTTP/REST, comme un Arduino ou même un script Powershell (on parle alors de package « virtuel »).

Constellation apporte aussi de nombreuses fonctionnalités autour du hub de messaging telles l'auto-description des contrats, la notion de scope, de saga, etc.

Aussi, et il s'agit là d'une force de cette plateforme, l'utilisateur de Constellation a la possibilité d'utiliser des packages partagés par la communauté mais il peut également créer ses propres packages en utilisant notamment le SDK Constellation directement intégré à Visual Studio. Il peut par ailleurs utiliser l'ensemble des APIs Constellation mises à disposition et réaliser ses projets en .NET, Microframework/Gadgeteer, Javascript et AngularJS, Arduino/ESP8266, NodeMCU, Python, Powershell et bien plus encore, au besoin, en utilisant l'API REST de Constellation.

Ces packages téléchargés ou développés par l'utilisateur peuvent ensuite être ajoutés au catalogue des packages disponibles sur la Constellation et déployés sur la sentinelle de son choix en quelques clics.

Ces packages déployés peuvent alors produire des StateObjects qui correspondent à une variable issue d'un objet, d'une application ou d'un service partagé qui est transmise dans la Constellation. En pratique, le volume de mon ampli, la chaîne de la TV, l'état d'une lampe, la T° du jardin, mon agenda Exchange ou le statut de mon alarme (...) sont tous des StateObjects qui sont mis à disposition de l'ensemble des packages. Ces StateObjects peuvent alors être directement utilisés à travers des pages Web ou n'importe quel équipement ou applicatif connecté à la Constellation ; les packages peuvent éga-

lement s'abonner aux modifications de ces StateObjects afin de faciliter le traitement ou l'affichage en temps réel. Vous retrouvez un catalogue de packages Constellation, dès à présent utilisables dans votre Constellation, mis à disposition par la communauté et permettant de connecter différents objets, services ou applications.

Usage et licence

Depuis Mars 2015, la plateforme Constellation est en beta privée avec plus d'une centaine de membres. Cette version était jusque-là restée privée car, bien que la plateforme technique soit prête, plusieurs éléments étaient restés à améliorer sur la documentation afin d'en faciliter l'accès au plus grand nombre.

Depuis Avril 2016, la nouvelle version 1.8 a été publiée intégrant sa nouvelle console et le nouveau modèle de licence.

Et justement, en termes de licence, la plateforme Constellation est et restera gratuite pour un usage personnel ou éducation (école et université). Elle est soumise à une licence payante pour les professionnels et entreprises développant des projets ou prototypes avec Constellation.

La plateforme s'ouvrira progressivement dès cet été, et le lancement public se fera à la rentrée 2016 ; vous pourrez alors utiliser ou créer les packages de votre choix pour connecter les éléments de votre quotidien.

Constellation dans Programmez

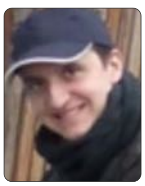
A travers les précédents numéros du magazine Programmez! j'ai pu vous présenter plusieurs solutions comme par exemple S-Panel, une application Cordova connectée dans Constellation pour le pilotage des différents objets connectés, ou encore différents projets basés sur des puces ESP8266 permettant de créer tout type de capteurs pour l'intelligence de la maison ou encore de permettre le contrôle de la maison avec la télécommande de la TV. Dans ce dossier retrouvez trois nouveaux projets basés sur Constellation.

Vous y découvrirez comment rendre intelligente une simple sonnette pour ne pas réveiller votre enfant quand il dort, comment connecter votre voiture Tesla à Constellation afin d'ajouter de l'intelligence ambiante, et comment garder le contrôle de vos objets et services à votre poignet en créant une application Tizen pour Samsung Gear S2 connectée à Constellation.

Connectez votre voiture Tesla dans votre Constellation

La plateforme Constellation permet l'intégration d'une multitude de périphériques ou de services de façon aisée, et ce, sans qu'importe la technologie et les langages utilisés. Dans cet article, nous allons nous intéresser à l'ajout d'un périphérique ultime puisqu'il s'agit de l'interaction avec la voiture électrique de la marque Tesla, le Model S, et d'un ensemble de différents capteurs déjà présents dans la constellation.

Fig.1 Vue d'ensemble de la solution



Nicolas Boonaert
MVP - Bing Maps for Enterprise
CEO @ Ketto
www.ketto.fr

Créer un package Tesla pour Constellation Fig.1

Découverte de l'API non-officielle de Tesla

La Tesla Model S est en permanence connectée via une connexion 3G intégrée qui communique avec les serveurs de Tesla leur permettant de récupérer des informations de diagnostic sur le véhicule mais aussi de délivrer des mises à jour « over the air » et d'apporter des fonctionnalités d'interaction avec le véhicule depuis l'application officielle disponible pour iOS et Android. Certains développeurs ont donc cherché à comprendre comment ces applications interagissent avec l'infrastructure de Tesla pour piloter un ensemble de fonctionnalités de la voiture ou pour récupérer les informations sur la voiture et ils sont arrivés à fournir une spécification relativement exhaustive qui a permis le développement d'applications tierces (mobiles ou statistiques). Il est à noter que cette interface peut tout à fait changer en fonction des choix faits par Tesla néanmoins nous allons explorer quelques fonctionnalités et préciser comment il est possible d'interagir avec le véhicule et le reste des équipements connectés à la Constellation. Pour la sécurité, elle repose sur la mise en œuvre d'un protocole OAuth 2.0 classique dont il faut ensuite transporter le jeton en entête des requêtes ensuite transmises. L'API gère plusieurs véhicules associés au même utilisateur, il est donc tout à fait possible de gérer plusieurs Tesla et pour les lister nous utiliserons la méthode dédiée (GET) :

```
https://owner-api.teslamotors.com/api/1/vehicles
```

L'API propose des fonctionnalités pour récupérer des informations sur le véhicule mais également pour exécuter des commandes qui peuvent agir sur la voiture. Voici un schéma présentant les fonctionnalités proposées : Fig.2.

Fig.2 API Tesla

Intégration dans un package Constellation

En utilisant le SDK de Constellation directement intégré à Visual Studio, la création d'un package est simplifié et la réalisation du package dédié à Tesla se fait directement en utilisant le Framework .Net et en utilisant le langage C#. On commence par créer une application console en prenant soin d'intégrer l'API Tesla, nous créons

une tache en background pour venir régulièrement interroger l'état de la voiture :

```
Task.Factory.StartNew(
  async () =>
  {
    while (PackageHost.IsRunning)
    {
      // Get every information about the car
      this.GetDrivingInformation();
      this.GetChargeInformation();
      this.GetClimateInformation();
      this.GetVehicleInformation();

      await Task.Delay(PackageHost.GetSettingValue<int>("RefreshInterval"));
    }
  });
```

Par exemple, la méthode « GetClimateInformation » invoque la méthode REST suivante :

```
https://owner-api.teslamotors.com/api/1/vehicles/{vehicle_id}/data_request/climate_state
```

Les données remontées nous permettent de récupérer l'état de la climatisation ainsi que les températures à l'intérieur du véhicule :

```
{
 "response": {
  "inside_temp": 17.0,              // Température en degrés Celsius (°C) intérieur
  "outside_temp": 9.5,              // Peut être nulle quand pas initialisé
  "driver_temp_setting": 22.6,      // Consigne côté chauffeur (°C)
  "passenger_temp_setting": 22.6,   // Consigne côté passager (°C)
  "is_auto_conditioning_on": false, // Fonctionnalité en Beta pour chauffage auto
  "is_front_defroster_on": null,    // Dégivrage avant
  "is_rear_defroster_on": false,    // Dégivrage arrière
  "fan_status": 0                   // Vitesse de la ventilation
 }
}
```

Une fois les données désérialisées dans un objet .NET, il suffit simplement de le publier dans la Constellation en tant que StateObject grâce à la méthode « PushStateObject ». Par exemple, lorsque nous récupérons l'état de la climatisation (classe ClimateInformation), nous remontons cet objet au niveau Constellation avec la ligne :

```
PackageHost.PushStateObject<ClimateInformation>("Tesla.Climate", informations,
    new Dictionary<string, object>
      {
        { "TimeStamp", DateTime.UtcNow.ToUnixTimestamp() },
        { "VehicleId", vehicleId }
      });
```

Ainsi notre package remonte actuellement quatre StateObjects dans Constellation à intervalle régulier :

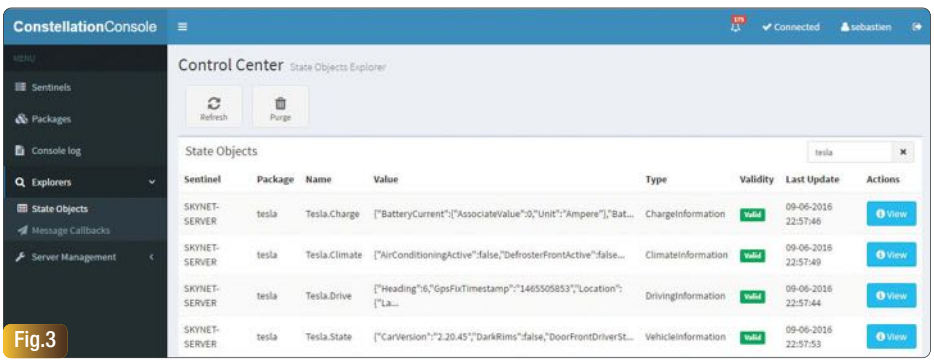
- Tesla.Charge : informations quant à la batterie (méthode REST « charge_state ») ;
- Tesla.Climate : informations quant à la climatisation (méthode REST « climate_state») ;
- Tesla.Drive : informations quant à la position et le déplacement de la voiture (méthode REST « drive_state») ;
- Tesla.State : informations quant à l'état de la voiture (méthode REST « vehicle_state»). **Fig.3.**

Vous pouvez explorer et visualiser vos StateObjects depuis la console Console. Par exemple, si nous ouvrons le StateObject «Tesla.Climate », nous retrouvons toutes les informations fournies par l'API Tesla : **Fig.4.**

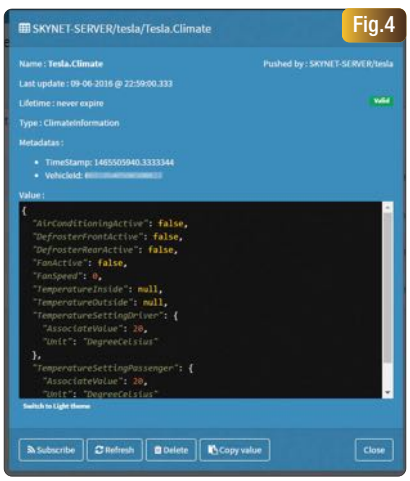
Dès lors, n'importe quel système connecté à Constellation, que ce soit le type d'application, peut connaitre en temps réel l'état de ma voiture, de la climatisation, la batterie, etc... Le package réalisé a donc la responsabilité d'interroger régulièrement l'API de Tesla pour mettre à jour les StateObjects dans la Constellation et donc agir tel un « connecteur » faisant le lien entre l'API Tesla et le monde Constellation.

Au-delà des StateObjects, un package Constellation peut également exposer des actions, c'est ce qu'on appelle des « MessageCallbacks ». En .NET, il suffit d'ajouter l'attribut « MessageCallback » pour exposer une méthode dans Constellation comme suit :

```
[MessageCallback]
public bool CommandFlashLight(string vehicleId)
{
  bool result = false;

  if (this.VehicleConfigurations.FirstOrDefault(c => c.Id == vehicleId) != null)
  {
    result = new VehicleCommandConnector(this.AuthenticationInformation, vehicleId)
.Flash();

    PackageHost.WriteInfo("Tesla Package - Command - Flash lights invoked");
  }

  return result;
}
```



StateObject Explorer



StateObject sur la climatisation

Concrètement, la méthode « Flash() » invoquera la méthode http/REST « flash_lights » avec l'URL suivante :

https://owner-api.teslamotors.com/api/1/vehicles/**{vehicle_id}**/command/flash_lights

Tout comme les StateObjects, il est possible d'explorer les MessageCallbacks de chaque package de la Constellation depuis la Console. Ici notre package Tesla expose plusieurs méthodes invocables depuis tous les autres packages selon les paramètres de sécurité établis. Ces méthodes permettent de récupérer des informations sur le véhicules (charge, position, climatisation) ou de contrôler la voiture comme faire des appels de phare, klaxonner, ouvrir le toit ouvrant, régler la climatisation...

Scénario 1 – Pilotage en fonction de la météo

De manière logique et simple, lorsque les conditions météos ne sont pas optimales et que le véhicule est resté ouvert, un module de supervision et d'intelligence ambiante provoque la fermeture du toit ouvrant ou signale à son utilisateur qu'une porte est restée ouverte. Pour cela nous utilisons la station météo Netatmo installée chez moi et qui est composée de plusieurs modules ayant des rôles distincts utilisés en intérieur ou en extérieur (module central, module d'intérieur, module d'extérieur, pluviomètre et anémomètre) et fournissant ensuite des données accessibles à travers une API dédiée. Au sein du catalogue de package mis à disposition sur le site de Constellation et produits par la communauté, on retrouve un package NetAtmo permettant de remonter l'état de toutes les sondes sous forme de StateObjects et il nous suffit alors de créer un package .NET pour gérer cette « intelligence » souhaitée. Dans ce scénario, notre package a besoin de connaitre l'état de la voiture et du pluviomètre, pour cela, nous pouvons utiliser l'attribut « StateObjectLink » sur les propriétés souhaitées afin de lier notre package d'intelligence aux données des packages NetAtmo et Tesla :

```
[StateObjectLink(Package = "NetAtmo", Name = "Pluviomètre.Rain")]
public StateObjectNotifier Pluviometre { get; set; }

[StateObjectLink(Package = "Tesla", Name = "Tesla.State")]
public StateObjectNotifier Tesla { get; set; }
```

Puis au démarrage du package, il suffit de s'attacher au changement d'état du pluviomètre pour être notifier en cas de pluie afin de vérifier si le toit ouvrant est bien fermé sinon, le fermer de manière automatique en invoquant le MessageCallback « CommandRoof » du package Tesla :

```
this.Pluviometre.ValueChanged += (s, e) => // lorsque l'état du pluviomètre change
{
  if ((int)e.NewState.DynamicValue > 0) // si de la pluie est mesurée
  {
    PackageHost.WriteInfo("Il pleut !");
    if (this.Tesla.DynamicValue.RoofOpeningPercent.AssociateValue != 0) // si le toit
 ouvrant n'est pas complément fermé
    {
      PackageHost.WriteInfo("Toit ouvrant ouvert ! Fermeture automatique");
      PackageHost.CreateMessageProxy("Tesla").CommandRoof(vehiculeId, 0);
    }
  }
};
```

Il est également possible de connaitre via le StateObject « Tesla.Drive » si le véhicule est en mouvement et sa position géographique pour vérifier qu'il est bien à proximité de la station météo utilisée ou sinon et au besoin, utiliser un service tiers de météo comme source d'information (ex : le package Forecast.io du catalogue disponible). Contrairement au Model X, le Model S ne propose pas des portes qui soient pilotables à distance, cependant cet exemple, utilisant le toit ouvrant, illustre de manière simple la mise en œuvre d'un traitement externe ayant une possibilité d'interaction avec le véhicule via la Constellation.

Scénario 2 – Préchauffage/Pré-climatisation le matin

En se basant sur les déplacements au sein de la maison ou du bureau, le module d'intelligence ambiante historise les déplacements dans les pièces, et donc le parcours réalisé avant de quitter le lieu ; il anticipe ainsi le départ afin de préparer le véhicule en préchauffant ou pré-climatisant le véhicule en fonction des conditions météos.

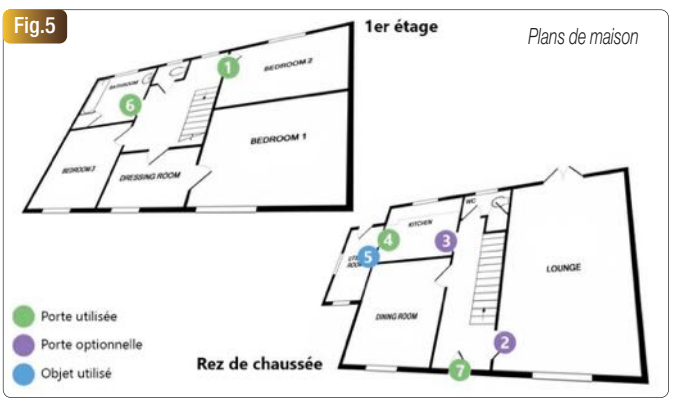
Présentation des capteurs Estimote

Les capteurs Estimote utilisés ici sont des stickers résistants à l'eau et permettant simplement de rendre « connecté » n'importe quel objet sur lequel cet élément est collé. Ces stickers intègrent des capteurs de déplacement, de température et d'orientation et peuvent renvoyer ces informations aux périphériques à proximité en plus de leur état logique (batterie, firmware...).
Ils peuvent communiquer selon deux protocoles de communication : iBeacon ou Nearable qui reposent tout deux sur une utilisation de Bluetooth Low Energy (BLE 4.0) ce qui permet de communiquer avec de multiples périphériques.

Analyse des parcours usuels et scénario retenu

Dans une application mobile déployée sur le téléphone, il est possible de remonter les informations broadcastées de manière régulière au server Constellation. Il est possible, pour cela, d'utiliser le SDK fourni par Estimote ou pour les plus courageux, implémenter les communications Bluetooth de manière à gérer les trames reçues, dans le cas présent une application PhoneGap permet de gérer ce cas avec notamment l'emploi d'un plugin spécifique. En pratique, le suivi permet de tracer les modifications des observations des stickers et des signaux liés qui pourront être historisées à travers la Constellation en communiquant les StateObjects utiles via l'API REST disponible. Si l'on traduit cela à des emplacements de stickers spécifiques et en filtrant quelque peu les données observées, on récupère des statistiques qui, une fois retranscrits sur un plan et dans une séquence organisée dans le temps, permettent d'identifier le scénario à retenir dans un créneau horaire visée : Fig.5.
Il est également possible d'analyser dynamiquement les déplacements et en extraire un « pattern » habituel pour une journée type : jour de semaine, weekend, jour férié... pour déclencher des scénarios sur des habitudes de déplacements identifiés dans la maison. Plusieurs limites apparaissent dans cette première implémentation mais ces dernières ne limitent pas pour autant la possibilité d'appliquer le traitement souhaité. Parmi celles-ci on peut indiquer qu'il n'y a pas de différenciation des utilisateurs activant les capteurs ou que l'on n'a pas d'état direct d'une porte par exemple notamment si elle est déjà ouverte. Néanmoins, à travers l'implémentation

Fig.5 *Plans de maison*



actuelle, les captures permettent de faire fonctionner le scénario et devront faire l'objet d'amélioration dans le traitement des messages émis par l'application ou reçus dans la Constellation.

Vérification des paramètres météorologiques depuis le package Netatmo

A travers le package NetAtmo, on récupère les informations météo de la station et on récupère également la température observée.
Aussi, la climatisation est un instrument à utiliser avec parcimonie, et il est recommandé de ne pas excéder 5°C entre l'habitacle et l'extérieur. C'est la raison pour laquelle la température extérieure est récupérée et utilisée au sein de la règle mise en œuvre que voici :

```
SI la [Température Extérieure] > 25°C ALORS
  Mettre la température à [Température Extérieure] – 5
SINON
  Mettre la température à 20°C
```

Dès lors, ce paramètre évidemment modifiable dans la configuration du module garantit un confort optimal d'usage du véhicule.

Commandes et contrôle du véhicule

La récupération de l'état du système de climatisation permet d'éviter de modifier les paramètres qui pourraient avoir déjà été spécifiés par l'utilisateur dans le véhicule ou depuis l'application, ceci afin de ne pas écraser les valeurs spécifiées. Les données remontées nous permettent de récupérer l'état de la climatisation ainsi que les températures à l'intérieur du véhicule.
Pour récupérer l'état de la climatisation de la voiture et du capteur de température extérieure, on créée des propriétés « StateObjectLink » :

```
[StateObjectLink(Package = "NetAtmo", Name = "Jardin.Temperature")]
public StateObjectNotifier TemperatureExt { get; set; }

[StateObjectLink(Package = "Tesla", Name = "Tesla.Climate")]
public StateObjectNotifier ClimTesla { get; set; }
```

Lorsqu'il est l'heure de partir, on vient définir la température de consigne pour le conducteur et le passager grâce au MessageCallback « CommandTemperature » du package Tesla en fonction de la température extérieure mesurée par notre sonde NetAtmo :

```
if ((int) this.TemperatureExt.DynamicValue > 25)
{
  int targetTemperature = (int) this.TemperatureExt.DynamicValue – 5;
  PackageHost.WriteInfo("Il fait trop chaud, réglage de la climatisation de la Tesla à
{0}°C !" , targetTemperature);
  PackageHost.CreateMessageProxy("Tesla").CommandTemperature(vehiculeId, target
Temperature, targetTemperature);
}
else
{
   PackageHost.WriteInfo("Réglage de la climatisation de la Tesla à 20°C !");
   PackageHost.CreateMessageProxy("Tesla").CommandTemperature(vehiculeId, 20, 20);
}
```

Scénario 3 – Gestion de la charge liée au planning

Précisions sur la charge et la recharge

Même si la Model S propose une autonomie suffisante pour effectuer plusieurs centaines de kilomètres (420 km réels pour le modèle concerné 85D), il arrive parfois que le véhicule ne soit pas assez chargé pour le parcours du lendemain. Là où avec un modèle thermique, il suffit de se rendre dans une station-service et repartir en quelques minutes, la recharge à domicile prend davantage de temps (Pour information, chez moi de 0 à 90% en environ 13h, en pratique quelques heures suffisent car la charge n'est jamais si peu élevée). Avec l'ensemble des modèles électriques du marché on a l'obligation d'attendre plus ou moins selon le véhicule, l'équipement à domicile ou les bornes sur lesquelles l'on est connecté. Dans l'optique de parcourir plusieurs centaines de kilomètres sans problèmes, Tesla est arrivé avec une approche disruptive et novatrice et propose un réseau, toujours en expansion, de bornes de recharge ultra-rapides appelées « superchargeurs » qui facilitent le déplacement lors des longs parcours sans effectuer des arrêts prolongés.

Synopsis

Dans le scénario retenu, on souhaite inspecter l'agenda de l'utilisateur et le calendrier professionnel pour vérifier les déplacements prévus, leur emplacement et donc vérifier que la charge du véhicule sera suffisante pour se rendre au rendez-vous ou s'il faut charger le véhicule en le connectant à domicile la veille ou en partant légèrement plus tôt en passant via un superchargeur.

Vérification du planning de la journée et récupération des détails des événements

La plateforme Office 365 est accessible à travers l'API Microsoft Graph qui permet également de se connecter à de multiples autres plateformes, dans le cas présent nous souhaitons récupérer le calendrier et les événements contenus. Afin de récupérer les événements à venir du calendrier, il est possible d'utiliser l'API dédiée et d'exécuter la méthode suivante :

```
https://graph.microsoft.com/beta/me/events?$top=5
```

A noter que cette méthode supporte les compléments de requêtes en supportant le protocole OData, ce qui permet de filtrer davantage sur des propriétés des événements. Ces derniers ainsi retournés permettent de récupérer entre autres l'emplacement du rendez-vous, la date et l'heure précise qui seront utiles pour les traitements qui suivent.
Encore une fois, il existe un package Constellation permettant d'injecter un événement issu du calendrier dans un StateObject sur la Constellation.

Calcul de l'itinéraire et récupération des informations de parcours

Pour réaliser le calcul de l'itinéraire, on retrouve plusieurs plateformes dédiées qui proposent des fonctionnalités géographiques. Afin d'anticiper sur d'autres scénarios qui seront implémentés par la suite, nous choisissons d'utiliser la plateforme proposée par Here. Voici la requête (GET) permettant de calculer l'itinéraire avec les alternatives utiles et le profil d'élévation :

```
https://route.cit.api.here.com/routing/7.2/calculateroute.json
?app_id={APP_ID}&app_code={APP_CODE}
&waypoint0=geo!51.325947,0.704923
&waypoint1=geo!51.093373,1.119516
&mode=fastest;car;traffic:disabled;
&alternatives=3
&routeAttributes=shape
&returnelevation=true
```

Il est également possible ensuite de régénérer un itinéraire incluant l'arrêt au superchargeur éventuellement disponible, et en partant à l'heure souhaitée par l'utilisateur tout en prenant en compte le trafic routier prévisionnel et en ajoutant au final le temps de charge estimé à l'arrêt :

```
https://route.cit.api.here.com/routing/7.2/calculateroute.json
? app_id={APP_ID}&app_code={APP_CODE}
```

```
&waypoint0=geo!51.325947,0.704923
&waypoint1=geo!51.286429,0.545411
&waypoint2=geo!51.093373,1.119516
&mode=fastest;car;traffic:enabled;
&departure=2016-06-08T07:00:00Z
&routeAttributes=shape
&returnelevation=true
```

Il est ensuite possible de régénérer un itinéraire incluant l'arrêt au supercharger en tenant compte de nombreux paramètres (trafic prédictif, heure de départ/arrivée, temps de charge...) ce qui peut être fait en utilisant le package Here disponible dans le catalogue et exposant les méthodes sous forme de MessageCallbacks permettant à tout autre package de tirer profit des fonctionnalités (géocodage, calcul d'itinéraires avancés, génération d'isochrone...).

Récupération de l'état de charge et stratégie de recharge

La récupération du niveau de charge du véhicule s'effectue par la récupération du dernier StateObject concerné dans la Constellation et les données renvoyées permettent de récupérer l'état de la batterie ainsi que des informations sur l'autonomie estimée du véhicule.
Voici par exemple la valeur de mon StateObject « Tesla.Charge » actuel :

```
{
  "BatteryCurrent": { "AssociateValue": 0, "Unit": "Ampere" },
  "BatteryLevel": { "AssociateValue": 69, "Unit": "Percentage" },
  "ChargeActive": false,
  "ChargePortOpen": false,
  "ChargerActualCurrent": { "AssociateValue": 0, "Unit": "Ampere" },
  "ChargeRate": { "AssociateValue": 0, "Unit": "KilometerPerHour" },
  "ChargerMaxCurrent": { "AssociateValue": 0, "Unit": "Ampere" },
  "ChargerPower": { "AssociateValue": 0, "Unit": "KilowattHour" },
  "ChargerVoltage": { "AssociateValue": 0, "Unit": "Volt" },
  "RangeBatteryEstimated": { "AssociateValue": 192.21, "Unit": "Mile" },
  "RangeBatteryNominal": { "AssociateValue": 180.09, "Unit": "Mile" },
  "RangeBatteryRated": { "AssociateValue": 225.11, "Unit": "Mile" },
  "RangeStart": null,
  "Supercharging": false,
  "TimeToCharge": { "AssociateValue": 0, "Unit": "Minute" }
}
```

Un phénomène lié au caractéristique physique et chimique des batteries occasionne une légère déperdition d'énergie même au cours d'une période d'inactivité du véhicule, on parle alors de vampirisation de la batterie à l'arrêt, pouvant être négligeable dans la majorité des cas, on peut estimer ce phénomène à 1 km de perte / heure d'inactivité.
La charge s'effectue à une vitesse dépendante de l'équipement de recharge, mesurable via l'API et donc le package. A domicile, je charge à une vitesse de 32 kilomètres par heure de recharge et on peut alors calculer le temps nécessaire pour la recharge et on pourrait même envisager de piloter la recharge automatiquement (début et arrêt de charge).
Pour calculer le temps, l'énergie consommée et le coût estimé :

```
{Charge nécessaire} = ({distance du parcours} - {distance restant} + [marge confort/arrivée])
* {consommation estimée sur le parcours} / [efficacité de recharge]
{Temps} = {Charge nécessaire} / {Taux de chargement}
{Coût} = {Charge nécessaire} * [taux du kwh]
```

En pratique, dans un cas appliqué, la charge nécessaire :

```
{Charge nécessaire} = (150 kilomètres – 60 kilomètres restants + 40 kilomètres de marge)
* (220 kwh/km moyen sur le parcours)
/ (0.82 efficacité de recharge moyenne sur équipement 220V/32A)
{Charge nécessaire} = Environ 35 kwh (34.9 kwh)
```

Le temps nécessaire :

```
{Temps} = 35 / 7.4 kwh en recharge
{Temps} = 280 min environ soit 4h40
```

Le coût associé :

```
{Coût} = 35 kwh * [0.17£]     // 0.22 €, oui l'électricité est chère en UK
{Coût} = 5,95 £               // 7,70 €
```

A noter, l'API de Tesla et donc de notre package Tesla dans la Constellation retourne également le temps estimé de charge nécessaire pour atteindre la limite qui serait éventuellement configurée sur le véhicule. Il est alors aisé de prendre la décision et de ne pas « oublier » de charger en prévision du rendez-vous et donc d'arriver à l'heure pour ce dernier.

Alternative via superchargeurs sur le parcours

Une autre option consiste alors à étudier la possibilité d'utiliser un supercharger sur la route et de calculer le temps nécessaire pour s'y rendre au cours du parcours, charger et continuer le chemin. Pour cela, nous avons créé une Web API dédiée, consommée depuis le package de gestion de la recharge et permettant de trouver un supercharger à proximité d'une route possible pour se rendre à l'emplacement prévu, calculée par l'API Here décrite plus tôt, et donc ensuite de calculer les temps nécessaires de parcours et de charge. Le traitement réalisé peut être illustré comme suit : **Fig.6.**
Il est alors possible d'ajouter enfin un déplacement dans l'agenda piloté via le package vers le Microsoft Graph pour intégrer cet arrêt de recharge au planning de la journée qui sera présenté en arrivant dans le véhicule.

Pour aller plus loin

La plateforme Constellation permet d'apporter le squelette facilitant les échanges entre les plateformes externes, les APIs personnalisées ou les objets connectés, et ses usages dépassent dans le cas présent, la domotique habituellement présentée. Les objets connectés sont en effet de plus en plus présents dans notre quotidien et le véhicule n'est pas en reste avec une nouvelle fois davantage d'échanges d'informations directement rendus possibles ; nous avons pu le voir ici, dans la Constellation, l'ensemble des objets connectés peuvent interagir avec le véhicule ou fournir une information utile pour faire un traitement encore plus riche.
La Tesla, en véhicule avant-gardiste, propose des fonctionnalités très originales, souvent utiles au quotidien, parfois plus triviales mais toujours dans le sens de l'innovation et des avancées permettant de faciliter l'intégration de ce véhicule dans un contexte d'intelligence ambiante.

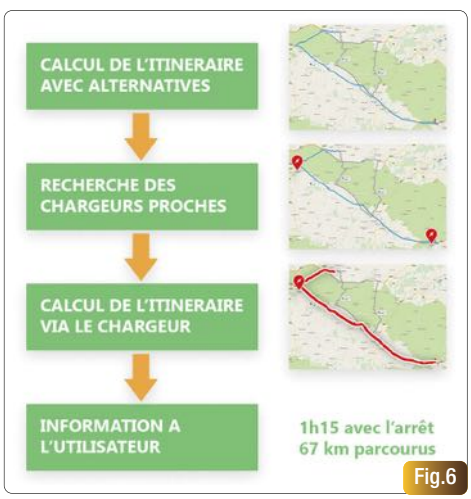


Fig.6 *Processus*

Consul, solution de service discovery

Le début des années 2000 a connu le règne sans partage des applications monolithiques déployées sur des serveurs d'applications. Aujourd'hui, la tendance commence à s'inverser : nos architectures se découpent en plusieurs composants (proxy, serveur Web, base de données, broker de messages, ...) et des applications qui commencent à suivre le paradigme des micro-services (http://martinfowler.com/articles/microservices.html).

Jean-Eudes Couignoux
Développeur Freelance
(A la frontière entre dev et ops)

En réponse à cette multiplication de services et de composants, un paradigme a commencé à émerger pour référencer à un endroit unique l'ensemble des informations nous permettant d'accéder aux services (adresse IP, port, ...) : les annuaires. Dans cet article, je vais présenter Consul, la solution de service discovery proposé par la société Hashicorp (éditrice entre autres de Vagrant, Terraform, ...)

La problématique

L'accès à un service se fait grâce à deux informations principales : son adresse IP et son port. Pour chaque service que nous exposons, il est donc nécessaire de communiquer à leurs consommateurs ces informations. Lorsque leur nombre est restreint, il est possible de gérer cela de manière manuelle. Mais dès lors que ce nombre augmente, il convient de centraliser ces informations. C'est là que consul entre en scène.
Consul est un outil pour implémenter le "service discovery". Il permet de centraliser la configuration de nos services en un point unique. En démarrant, ceux-ci vont s'enregistrer auprès de consul pour y référencer leur IP et port. Celui-ci va ensuite exposer ces informations, soit à l'aide d'une API HTTP, soit à l'aide du protocole DNS.

Installation de consul

Consul est un outil écrit en go (https://golang.org/). Son installation est relativement aisée, puisqu'il est livré sous la forme d'un simple binaire. Il existe des versions compilées pour la plupart des architectures courantes (Windows, linux, bsd, 32/64 bits, ...).
Dans la suite de cet article, les exemples seront donnés pour un système linux, mais ils pourront très facilement être adaptés à un autre système.

```
wget https://releases.hashicorp.com/consul/0.6.4/consul_0.6.4_linux_amd64.zip
```

Il suffit ensuite de dézipper le fichier téléchargé, et de déplacer le binaire dans un répertoire contenu dans le PATH de notre serveur. Le binaire Consul permet de le lancer en deux modes : agent simple ou serveur. Un agent consul est un démon capable d'enregistrer des services au sein du cluster Consul, et de répondre aux différentes requêtes des consommateurs. En mode serveur, consul a comme responsabilité supplémentaire de gérer l'état du cluster et de participer à la gestion du consensus (qui se base sur raft). Pour initialiser notre cluster consul, il suffit de lancer la commande :

```
consul agent -server -bootstrap-expect 1 -data-dir /tmp/consul
```

L'option -server signifie, comme son nom l'indique, que l'on démarre l'agent Consul en mode serveur. L'option -bootstrap-expect, quant à elle indique que le cluster Consul s'attend à avoir au moins 1 serveur pour commencer à fonctionner. Enfin l'option -data-dir correspond au répertoire de travail de Consul. Il est possible de consulter à tout moment l'état du cluster avec la commande :

```
consul members
```

ou en utilisant l'api http exposé par Consul

```
curl localhost:8500/v1/catalog/nodes
```

Pour ajouter un agent à notre cluster consul, il faut simplement lancer un nouvel agent, puis le rajouter à un nœud déjà existant de notre cluster.

```
consul agent -data-dir /tmp/consul -config-dir /etc/consul.d
consul join <ip-consul>
```

Avec cette commande, nous voyons une nouvelle option : -config-dir qui fait référence à un répertoire pouvant contenir la définition de nos futurs services. L'agent Consul devra être déployé sur chacun de nos serveurs.

Enregistrons notre premier service

Maintenant que notre cluster Consul est configuré, nous allons pouvoir enregistrer notre premier service. Sous Consul, cela signifie que nous allons associer à un nom de service (par exemple Web), au minimum son adresse IP et son port. Pour définir un nouveau service, il suffit de créer un fichier au format JSON dans le répertoire "/etc/consul.d" :

```
{"service":
 { "Name": "web",
  "Port": 8000,
  "Address": "10.0.0.1"}
}
```

Puis, pour notifier l'agent qu'un service a été créé ou mis à jour, on va lancer la commande :

```
consul reload
```

Il est également possible d'enregistrer notre service auprès de Consul via un appel HTTP à l'API REST de Consul : "/v1/agent/service/register" avec le format suivant :

```
{ "Name": "web",
  "Port": 8000,
  "Address": "10.0.0.1"
}

curl -H "Content-Type: application/json" --data @web.json  localhost:8500/v1/agent/service/register
```

Ceci est surtout utile pour enregistrer notre service auprès de Consul au moment où notre application va démarrer, par exemple, via un listener de Servlet, si notre application est écrite en Java.

À la recherche de notre service

Maintenant que nous avons enregistré nos premiers services au sein de

Consul, il est temps de commencer à interroger notre annuaire via une API HTTP ou le protocole DNS.

Requêtage en http

Il est possible d'interroger Consul pour obtenir la liste des services déployés via le chemin : "/v1/catalog/services" (Les API sont exposées sur le port 8500).

```
curl localhost:8500/v1/catalog/services
```

Il est aussi possible de récupérer les informations d'un service en particulier. Dans ce cas, le chemin est : /v1/catalog/service/<service-name>.

```
curl localhost:8500/v1/catalog/service/<service-name>
```

La réponse est renvoyée au format JSON.

```
{
  "web":{
    "ID":"web",
    "Service":"web",
    "Tags":null,
    "Address":"10.0.0.1",
    "Port":8000,
    "EnableTagOverride":false,
    "CreateIndex":0,
    "ModifyIndex":0
  }
}
```

Requêtage en DNS

Il est certes aisé de requêter Consul en HTTP, mais dans la plupart de nos applications, nous avons pris l'habitude d'utiliser un nom de domaine pour s'abstraire de l'adresse IP de notre service. Consul peut être interrogé comme serveur DNS (sur le port 8600 par défaut) avec le domaine "service.consul" :

```
dig @127.0.0.1 -p 8600 <service-name>.service.consul
```

Il est possible de modifier le domaine par défaut de Consul avec l'option "-domain" pour remplacer le domaine par défaut "consul." par celui de votre choix.

Il est possible d'intégrer le DNS Consul avec le DNS interne de votre entreprise. Pour cela, il suffit de le configurer pour lui déléguer les requêtes du domaine géré par Consul ("consul.", ou ce que vous aurez configuré) à Consul (exemple : https://www.consul.io/docs/guides/forwarding.html)

Mise en place d'un health check

Consul offre la possibilité de rajouter une fonctionnalité de health check sur l'ensemble des services que nous avons enregistrés. Un health check permet à Consul de savoir si un service est bien disponible ou non. Si un service devient indisponible, alors Consul ne renverra plus d'informations le concernant.

De manière analogue à la création d'un service, l'ajout d'un health check se fait en ajoutant une section dans le fichier JSON. Par exemple :

```
{"service":
 { "Name": "web",
 "Port": 8000,
 "Check": {
   "HTTP": "http://localhost:8000/health",
   "Interval": "10s",
   "Timeout": "1s"}
 }
}
```

Le health check sera effectué par l'agent auprès duquel le service aura été enregistré. C'est pour cela que l'on retrouve "localhost" dans l'url du check. Consul permet de définir plusieurs types de check :

- HTTP : le test est considéré comme un succès si le code de retour de la requête est du type 2xx ;
- Script : on définit un script qui sera exécuté à intervalle régulier. Si le code retour est 0, alors le test est considéré comme un succès ;
- TCP : on essaie d'ouvrir un socket TCP à intervalle régulier. Si l'ouverture du socket est accepté, alors le test est considéré comme OK.

Il est possible d'accéder à l'état d'un service via une API HTTP "/v1/health/checks/<service-name>" :

```
curl localhost:8500/v1/health/checks/<service-name>
[
  {
    "Node":"n2",
    "CheckID":"service:web",
    "Name":"Service 'web' check",
    "Status":"passing",
    "Notes":"",
    "Output":"HTTP GET http://localhost:8000/: 200 OK",
    "ServiceID":"web",
    "ServiceName":"web",
    "CreateIndex":165,
    "ModifyIndex":189
  }
]
```

Consul UI : une interface pour gérer nos services

Consul est livré avec une interface d'administration : consul-ui. Pour l'activer, il faut ajouter l'option "-ui" au démarrage de l'un de vos agents.

```
$ consul agent -data-dir /tmp/consul -ui -config-dir /etc/consul.d -client 0.0.0.0
```

On note également que l'on a dû ajouter l'option "-client" pour indiquer à l'agent d'écouter sur toutes les adresses, et pas uniquement sur 127.0.0.1 (la valeur pas défaut). Consul-UI permet de consulter la liste des services déployés sous Consul : Fig.1.

ainsi que la liste des noeuds de notre cluster Consul : Fig.2.

Haute disponibilité de nos services

Nous savons maintenant enregistrer nos services dans Consul, et nous avons également un moyen de tester leur disponibilité. Imaginons maintenant que nous souhaitons faire gérer plus de charge à notre service. Pour cela, une des approches les plus simples est d'en déployer plusieurs instances, et de positionner en frontal un répartisseur de charge (load balancer). Fig.3.

Du fait que le nombre d'instances d'un service peut varier selon la charge, la configuration du load balancer a besoin d'être modifiée en conséquence. Comme les services s'enregistrent déjà auprès de Consul, ce dernier dispose des informations nécessaires pour mettre à jour lui-même la configuration du load balancer. L'outil "consul-template" peut générer des fichiers dynamiques à partir des informations disponibles dans Consul, ou encore de lancer une commande en cas de mise à jour d'un service.

En combinant ces deux fonctionnalités, consul-template peut être utilisé pour régénérer la configuration de notre load balancer dès qu'un service est créé ou modifié. Voici un exemple avec HAProxy :

Installation de consul-template

A l'instar de Consul, Consul-template est également livré sous la forme d'un binaire. La procédure d'installation est donc relativement similaire : télécharger le fichier Zip et copier le binaire dans un répertoire contenu dans le PATH de votre serveur.

```
wget https://releases.hashicorp.com/consul-template/0.14.0/consul-template_0.14.0_linux_amd64.zip
```

Création d'un template

Les fichiers template utilisés par consul-template sont basés sur la syntaxe HCL (https://github.com/hashicorp/hcl). Elle permet de référencer des informations (par exemple des adresses IP ou des ports) stockées dans Consul. Voici un exemple de configuration pour HAProxy :

```
global
    daemon
    maxconn 256

  defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000m

  listen http-in
    bind *:8000
{{ range service 'web' }}
    server {{ .Node }} {{.Address }}:{{ .Port }}
{{ end }}
```

Dans cet exemple, nous itérons sur l'ensemble des services nommés web référencés dans notre annuaire, et pour chacun d'entre eux, nous générons une ligne avec son adresse IP et son port. Pour lancer le démon consul-template, il suffit d'exécuter la commande suivante :

```
consul-template -consul localhost:8500 -template haproxy.tmpl:/etc/haproxy/haproxy.cfg
```

L'option "-consul" fait référence à une instance de notre cluster consul, tandis que l'option "-template" comporte deux arguments : le premier contient le chemin de notre fichier template, tandis que le second indique où le fichier final sera généré. Dès qu'un nouveau service "web" sera ajouté ou supprimé de notre annuaire, consul-template va régénérer le template.
Il est possible d'ajouter un troisième paramètre dans l'option "-template" pour lui signifier une commande à exécuter, par exemple un reload de notre load balancer.

```
consul-template -consul localhost:8500 -template "haproxy.tmpl:/etc/haproxy/haproxy.cfg systemctl reload haproxy"
```

Quelques bonnes pratiques

Avant de déployer Consul sur les environnements, il est bon d'observer quelques bonnes pratiques :

- Utiliser un utilisateur spécifique pour lancer l'agent consul ;
- Prévoir 3 ou 5 "serveurs" Consul pour assurer la haute disponibilité ;
- Ne pas hésiter à spécialiser les agents Consul, par exemple avoir un agent spécifique pour gérer les requêtes DNS ;
- Activer l'encryption pour sécuriser la communication entre les agents ;
- Utiliser syslog pour récupérer les logs. Par défaut, c'est la sortie standard qui est utilisée.

Pour aller plus loin

Nous avons vu qu'une petite part des fonctionnalités de Consul. Celui-ci peut s'intégrer avec un orchestrateur (Mesos, Swarm, Nomad, ...), gérer votre cluster grâce à son implémentation de raft, gérer des services au sein de plusieurs data centers, ... N'hésitez pas à parcourir sa documentation pour en voir les usages avancés (https://www.consul.io/).

Bibliographie :

https://www.consul.io/
http://martinfowler.com/articles/microservices.html
https://github.com/hashicorp/consul-template
https://github.com/hashicorp/hcl

Fig.1

Fig.2

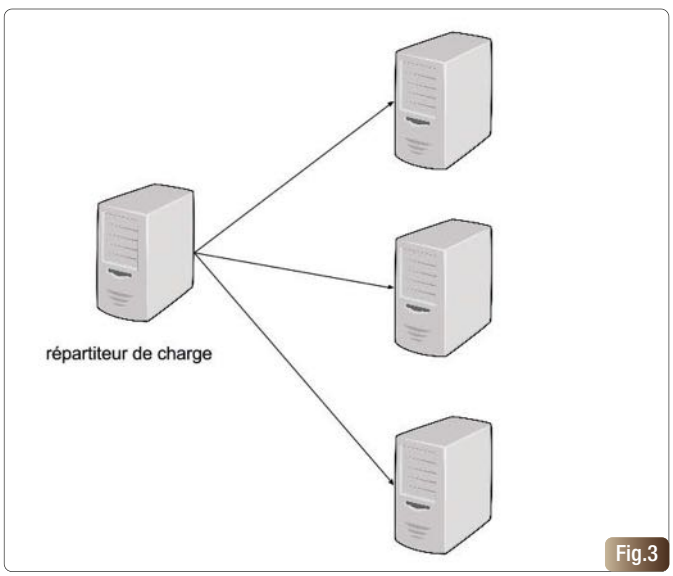
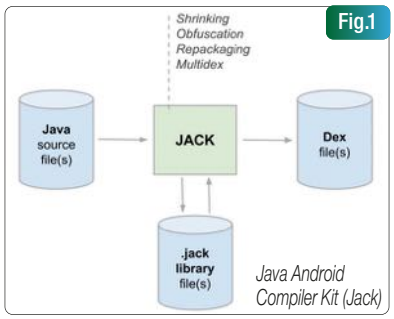


Fig.3

Découvrez Jack, le nouveau kit de compilation Java pour Android

Basé sur OpenJDK, Android N introduit le support de plusieurs fonctionnalités de Java 8. Afin d'être compilé correctement, le code utilisant ces fonctionnalités nécessite d'activer Jack le nouveau kit de compilation Java pour Android. Ce nouveau kit s'accompagne également d'un nouveau format remplaçant le .class, à savoir le .jack. Dans cet article, nous vous proposons une présentation du fonctionnement de Jack.

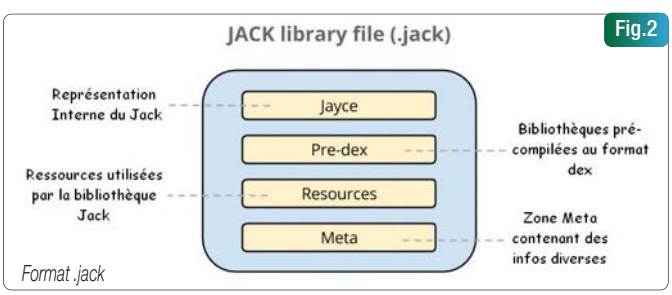


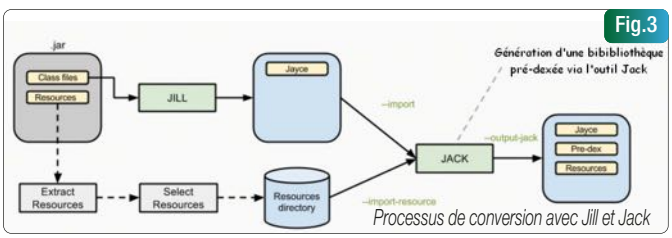
Sylvain SAUREL
Ingénieur d'Etudes Java / JEE
sylvain.saurel@gmail.com

Jusqu'à présent, la chaîne de compilation Java pour Android était basée sur le compilateur javac ce qui induisait le cheminement suivant :

```
javac (.java => .class) => dx (.class => .dex)
```

Ainsi, le fichier source .java était tout d'abord compilé en fichier .class avant que l'utilitaire dx du SDK Android ne transforme le fichier .class en fichier .dex utilisable par la machine virtuelle Android. Android M avait déjà introduit Jack mais son utilisation demeurait plutôt expérimentale. Android N recommande désormais l'utilisation de ce nouveau kit de compilation Java pour Android qui s'amène avec son propre format intermédiaire le .jack remplaçant le traditionnel .class. Jack réalise ainsi directement la compilation des fichiers sources Java au format bytecode dex. Le premier avantage par rapport à la chaîne de compilation existante jusqu'alors est le fait de ne recourir qu'à un seul outil puisqu'avant la compilation s'appuyait sur javac, ProGuard, jarjar ou encore dx. En outre, Jack est complètement open source et donc intégré à l'Android Open Source Project (AOSP). Il réduit drastiquement le temps de compilation grâce à un certain nombre de techniques telles que le pre-dexing, la compilation incrémentale ou encore la compilation Jack en mode serveur. Enfin, il intègre directement en son sein les fonctionnalités proposées par ProGuard à savoir le shrinking, l'obfuscation, le repackaging et le support multidex. Son fonctionnement est résumé dans le schéma de la Fig.1. Afin d'optimiser au maximum la vitesse de compilation, le format .jack a été créé. Il remplace le format .class et a été spécialement conçu pour contenir du code de bibliothèques pré-compilé au format dex. Tout ceci constituant la fameuse phase de pre-dexing réduisant les temps de compilation. Globalement, le format Jack se décompose en 4 zones principales (Fig.2) : la partie





Jayce décrivant le contenu du fichier .jack, la partie Pre-dex contenant les bibliothèques Java pré-compilées au format dex, la partie Resources où sont stockées toutes les ressources utilisées par le fichier .jack et enfin la partie Meta qui contient diverses informations telles que les dépendances et leur type par exemple. Pour les projets utilisant des bibliothèques au format .jar, un outil nommé Jill, pour Jack Intermediate Library Linker, est prévu. Il se charge de convertir ces bibliothèques dans le nouveau format en ayant in fine recours à l'outil Jack pour créer la bibliothèque pré-dexée dans le fichier .jack. Le processus de conversion impliquant Jill et Jack est détaillé dans la Fig.3. Côté développeur, il n'y a pas de grand changement si ce n'est qu'il est nécessaire d'activer l'utilisation de Jack au sein de son build.gradle une fois récupéré Android N sur son ordinateur. Cela se fait en ajoutant la propriété jackOptions et en positionnant à true le champ enabled. Votre fichier de build Gradle aura alors l'allure suivante :

```
android {
  compileSdkVersion 'android-N'
  buildToolsVersion '24.0.0 rc1'
  ...
  defaultConfig {
    minSdkVersion 'N'
    targetSdkVersion 'N'
    ...
    jackOptions {
      enabled true
    }
  }
  compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
  }
}
```

Ici, on positionne également les options de compilation de manière à supporter les nouveautés autour de Java 8 apportées par Android N puisque c'est aussi une des raisons pour lesquelles on souhaite utiliser Jack. Au lancement du premier build d'un projet utilisant Jack, un serveur de compilation Jack local est lancé sur l'ordinateur afin de réaliser la compilation de manière plus rapide. Tout ceci étant bien sûr transparent pour le développeur.

Conclusion

Le nouveau kit de compilation Java pour Android est désormais pleinement utilisable avec Android N et va connaître un grand succès puisqu'il est un prérequis nécessaire à l'utilisation des fonctionnalités de Java 8 au sein des projets Android. Il dote la plateforme Android d'un outil de compilation complet, plus performant, capable de réaliser de la compilation incrémentale offrant en sus les fonctionnalités de shrinking, obfuscation ou de repackaging devant être réalisées de manière externe jusqu'alors avec ProGuard.

Préparez vos applications pour supporter le mode Multi-Fenêtres d'Android N

Réclamé depuis longtemps par les utilisateurs, le mode Multi-Fenêtres fait son apparition avec Android N. Cette nouvelle fonctionnalité va connaître un grand succès à n'en pas douter, et il est donc primordial de préparer dès maintenant son application Android pour supporter ce mode. Dans cet article, nous vous proposons les 5 conseils essentiels pour être prêt à supporter le mode Multi-Fenêtres au sein de votre application dès la mise à disposition grand public d'Android N.

Sylvain SAUREL
Ingénieur d'Etudes Java / JEE
sylvain.saurel@gmail.com

Le mode Multi-Fenêtres est une des nouveautés majeures d'Android N. Ce mode, permettant de séparer l'écran d'un smartphone ou d'une tablette en deux afin d'utiliser deux applications simultanément, ouvre la voie à du vrai multi-tâches sous Android (Fig.1).

Du point de vue du développeur Android, il n'y a en réalité pas vraiment de nouveautés côté code puisque aucune API spécifique n'a été ajoutée par exemple. En revanche, ce nouveau mode nécessite l'introduction d'un certain nombre de nouveaux attributs XML qui vont permettre de définir le comportement que l'application devra adopter lorsque le mode Multi-Fenêtres sera activé par l'utilisateur.

Pour le reste, tout l'outillage permettant le bon fonctionnement de ce mode est déjà présent au sein du SDK depuis bien longtemps. En effet, c'est bien le système de gestion des ressources d'Android qui rend le mode Multi-Fenêtres quasiment transparent pour le développeur du point de vue du code Java. Ce système que l'on va utiliser pour définir des ressources alternatives pour les dimensions, les layouts, les drawables ou autres menus en fonction de qualifiers spécifiques.

Le Multi-Fenêtres bénéficie de ce système en ajustant la configuration de l'application par rapport à la taille de la fenêtre courante.

Le qualifier utilisé en priorité étant bien évidemment la taille de l'écran mais la taille minimale en largeur ou hauteur sera également utile de même que l'orientation de l'écran.

Utiliser le Context adapté

Toutes les instances de Context ne sont pas créées de la même façon sous Android. Ainsi, l'instance de la classe Context que vous allez récupérer peut varier selon que vous soyez au sein d'une classe Application, d'une Activity ou d'un Service, d'un BroadcastReceiver ou encore d'un ContentProvider. La bonne pratique à adopter est d'utiliser uniquement le contexte lié à une activité pour le chargement de ressources impactant l'interface utilisateur sous Android. Cela concerne donc bien évidemment le chargement de layouts au sein d'une activité. Procéder de la sorte garantit un fonctionnement correct lors du chargement des ressources alternatives sous Android N avec un mode Multi-Fenêtres activé. De fait, les développeurs utilisant le contexte lié à l'objet Application pour des opérations liées à l'interface utilisateur devront revoir leur copie et effectuer les modifications adéquates dès à présent au sein de leurs applications.

Gérer correctement les changements de configuration

En récupérant tout ce qui est lié à l'interface utilisateur via la bonne instance de Context, vous êtes déjà sûr que vos ressources seront chargées correctement en fonction de la taille d'écran disponible pour votre application. Néanmoins, cela ne suffit pas. Il faut également gérer correctement les

Fig.1

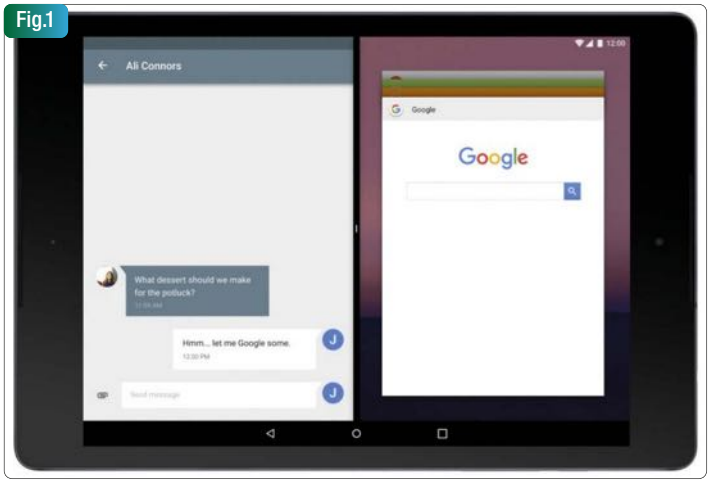


Figure 1 : Support Multi-Fenêtres sous Android N

changements de configuration pour répercuter les éventuelles modifications sur le contenu de votre application. Le cas nominal consiste à ne rien faire et à laisser votre activité être détruite puis recréée en sauvegardant l'état en cours au sein du callback onSaveInstanceState() puis en utilisant cette sauvegarde à la recréation avec les éventuels nouveaux layouts chargés. Pour aller plus loin et gérer plus finement ces changements de configuration, vous pouvez le faire vous-même en surchargeant le callback onConfigurationChanged() au sein de vos activités ou fragments. Ainsi, au lieu d'être détruits puis recréés, ces derniers verront cette méthode être appelée. Vous devrez ensuite réaliser manuellement la mise à jour des vues et les rechargements éventuels de ressources. Pour être notifié des changements de configuration liés à l'activation ou la désactivation du mode Multi-Fenêtres, vous aurez juste à ajouter l'attribut android:configChanges pour votre activité au sein du Manifest Android avec au moins ces valeurs :

```
<activity
  android:name=".MyActivity"
  android:configChanges="screenSize|smallestScreenSize
    |screenLayout|orientation"
/>
```

Attention à bien prendre en compte toutes les ressources ayant vocation à être changées puisque cela est à votre charge avec ce paramétrage. Prenons l'exemple de dimensions que vous auriez définies au sein de deux répertoires de ressources :

- values pour le cas général
- values-sw600dp pour les écrans ayant une largeur d'écran minimale de 600 dp

Avant l'introduction du Multi-Fenêtres, vous n'auriez rien eu à faire au runti-

me puisque la sélection aurait déjà été effectuée et il n'aurait pas été possible de changer de taille minimum d'écran. Désormais, avec le mode Multi-Fenêtres, il est possible et nécessaire de basculer entre ces fichiers de ressources lorsque votre application est redimensionnée.

Supporter toutes les orientations

Jusqu'à présent, par souci de gain de temps probablement, les développeurs Android choisissaient bien souvent de négliger le support du mode paysage bloquant leur application en mode portrait. Ce choix va malheureusement leur porter préjudice avec Android N. En effet, même si un périphérique Android est en mode paysage, il se peut que l'application, elle, soit en mode portrait ! Cela vient du fait que pour déterminer le mode d'affichage d'une application, le système Android va simplement comparer la largeur et la hauteur d'affichage. Ainsi, "portrait" signifie que la hauteur est plus grande que la largeur et "paysage" signifie que la largeur est plus grande que la hauteur. Lors d'un redimensionnement de la fenêtre associée à votre application, celle-ci va donc pouvoir basculer dans un mode différent de celui du périphérique hôte. Il est donc grand temps de supporter le mode paysage au sein de vos applications pour préparer au mieux l'arrivée d'Android N.

Pour les développeurs ayant fait le choix de bloquer l'orientation via l'attribut android:screenOrientation affecté à une activité, et n'ayant pas ciblé Android N pour leur application à la compilation, cela signifiera que le mode Multi-Fenêtres ne sera pas supporté. L'utilisateur ne pourra donc en bénéficier avec leur application lancée. En revanche, si votre application cible Android N, alors la valeur de cet attribut est ignorée. Enfin, l'appel à la méthode setRequestedOrientation() au sein d'une Activity n'aura aucun impact en mode Multi-Fenêtres quelle que soit la version d'Android ciblée.

Construire des UI responsives

Le support de toutes les orientations n'est bien entendu pas suffisant pour assurer un fonctionnement optimal en mode Multi-Fenêtres. En effet, la possibilité de redimensionnement offerte par ce mode va ainsi confronter votre application et son interface utilisateur à une taille d'affichage probablement jamais rencontrée jusqu'alors. Comment réagira votre application ? C'est là que l'intérêt de concevoir des UI responsives apparaît. En effet, en suivant à la lettre ce mode de conception des UI, votre application doit d'ores et déjà être prête pour Android N. Pour les retardataires, pas de soucis : Google fournit tout un tas de ressources sur le sujet. Son guide des patterns UI responsives est un point de départ essentiel : https://www.google.com/design/spec/layout/responsive-ui.html .

Les Activités lancées par d'autres Apps doivent supporter le mode Multi-Fenêtres

Dans le monde du Multi-Fenêtres, une tâche entière est représentée par une simple fenêtre. C'est pourquoi en choisissant de lancer une activité adjacente, il faudra créer une nouvelle tâche. Qui dit nouvelle tâche dit donc nouvelle fenêtre. La réciproque est vraie également. La documentation d'Android N précise que si une activité est lancée à l'intérieur d'une stack de tâches, l'activité remplace l'activité à l'écran tout en héritant de ses propriétés quant au support du mode Multi-Fenêtres.

Ainsi, si vous créez une activité ayant vocation à être lancée par d'autres applications, celle-ci va hériter des mêmes propriétés de paramétrage du mode Multi-Fenêtres que l'activité appelante. Cela inclut notamment les attributs liés à la taille minimale. Vous l'aurez donc compris il faut dès à présent concevoir vos interfaces utilisateurs en intégrant la problématique du support Multi-Fenêtres qui deviendra dans les années à venir essentielle.

Conclusion

Attendu et réclamé de longue date, le mode Multi-Fenêtres va obliger les développeurs Android à quelques ajustements au niveau des IHM de leurs applications. Ces derniers seront plus ou moins importants selon que les développeurs en question aient suivi ou non les recommandations et bonnes pratiques prônées par les équipes de Google en charge d'Android. Dans tous les cas, une seule démarche à adopter : récupérer la developer preview d'Android N et tester vos applications dès maintenant. ⊞

Windows App Studio

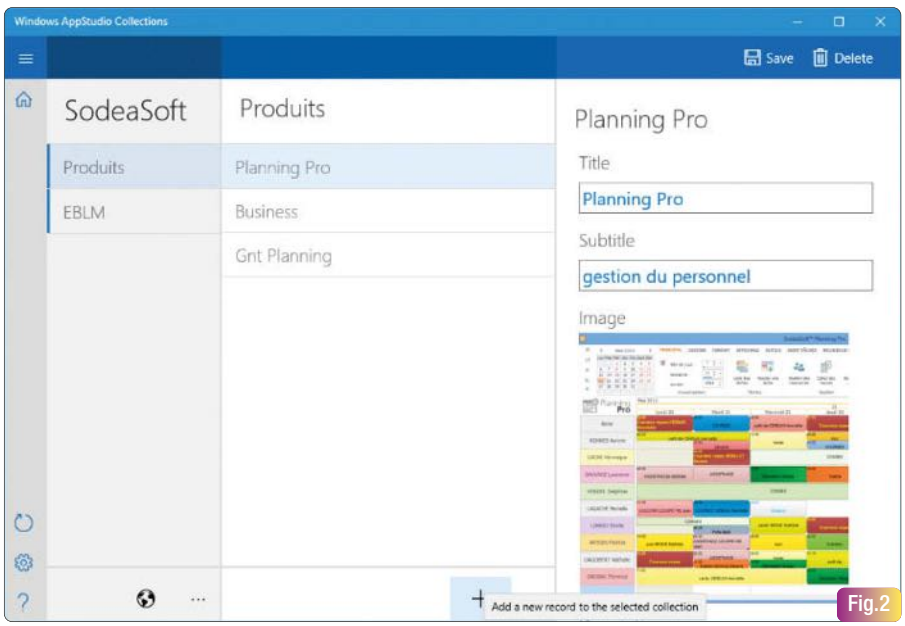
Windows App Studio est un outil en ligne qui va vous permettre de créer sans une seule ligne de code une application Windows 10. Vous allez être plongé dans un environnement où l'on n'aura absolument rien à installer, où aucune notion de développement ne vous sera requise, et en étant capable d'aller proposer directement une application Windows 10 dans le store ; voire même de la déployer sur n'importe quel appareil PC, tablette, téléphone, IoT, bientôt Xbox et SurfaceHub, sans qu'elle passe par le store !

Christophe Peugnet
MVP Windows Development chez **SodeaSoft / EBLM**
http://www.sodeasoft.com
Blog : http://www.peug.net
Twitter : @tossnet1

Cependant le développeur n'est pas mis de côté, Windows App Studio (WAS) peut être un bon outil pour démarrer une application rapidement, et vous découvrirez plus loin des *samples* qui satisferont bon nombre de développeurs. Au final, WAS vous permettra de créer 3 choses : un package pour le *sideloading* (procédé de déploiement sur les postes clients sans passer par le Windows Store), un package pour Visual Studio, et un package pour le store.

Aujourd'hui lorsqu'on génère une application, on pourrait dire que la seule limite c'est l'imagination. Mais attention, ce n'est pas ce que Microsoft a voulu faire avec Windows App Studio. Windows App Studio supporte un nombre limité de scénarios comme - par exemple - prendre des informations sur Facebook, YouTube, Instagram, WordPress etc., et Microsoft va aller jusqu'à créer le package et le signer pour vous. Avec Windows App Studio, on est dans un environnement possédant des scénarios limités, connus et maitrisés. Comprenons bien que ce n'est pas un éditeur de code qui pourra faire tout ce que l'on veut. WAS va nous donner votre solution (le code de votre application) et un package prêt à l'emploi ou à la publication.

Pour utiliser WAS, il ne faut pas obligatoirement de compte développeur, il vous suffit d'un compte Microsoft (@outlook.com etc.) et vous pourrez l'utiliser gratuitement. Vous n'êtes pas obligé d'avoir Visual Studio, c'est totalement optionnel. Vous pourrez faire votre application dans le site Internet. Cependant si vous souhaitez étendre votre application en dehors des scénarios actuellement disponibles, vous pourrez récupérer la solution et continuer le développement avec votre outil Visual Studio.
Pour créer votre application, rendez-vous sur le site https://appstudio.windows.com . Une fois connecté vous serez invité à créer votre projet. Vous aurez ensuite un choix de quelques modèles pour démarrer comme « Web App Templates » qui respectera toutes les spécifications du W3C et qui créera un packaging standard d'une application Web, et d'autres Templates comme « Mon Catalogue d'articles ». Vous pourrez aussi tout simplement partir du modèle « Vide ». Mais il faut souligner que ces derniers templates sont simplement partis du template vide où des éléments seront déjà prépositionnés pour encore accélérer votre création. Juste après vous choisirez le nom de votre application. Notez que dès le départ vous apercevrez un émulateur extrêmement bien fait sous forme d'une tablette, d'un mobile ou d'un ordinateur où vous visualiserez instantanément l'aperçu de votre application. Ce n'est pas juste un rendu ; j'ai bien noté « émulateur », c'est-à-dire que vous pouvez interagir avec les boutons, les slides dans ce dernier. Fig.1.



Techniquement ils ont développé un parser XAML / HTML. Cet émulateur est aussi disponible en version plein écran et l'intérêt est que le lien de ce rendu Full Screen peut être partagé avec les autres acteurs avec qui vous travaillez qui sont liés à cette application !
L'étape suivante est vraiment le cœur de notre application où vous choisirez différentes sections avec des sources associées. C'est aussi à ce stade que vous définirez le thème général (couleurs, titres...). Lorsque vous ajoutez une section comme Flickr, votre blog WordPress, un compte Twitter et que vous la reliez à votre compte, l'émulateur situé à droite de l'écran s'actualisera directement ; vous aurez donc ainsi instantanément un aperçu de l'application avec vos données !
A noter qu'une *Update* importante de juin apporte, entre autre, la possibilité de se connecter à une source avec des API REST ! (Get pour l'instant)

Dans le choix des sections, il existe aussi un élément appelé « Collections » où Microsoft va stocker pour vous vos données, qu'elles soient statiques, dynamiques ou externes. Vous pourrez effectuer des recherches, des tris ou encore des groupes datés. Jusque-là rien d'extraordinaire vous me direz sauf qu'une fois que votre application est publiée sur le store par exemple, vous pourrez enrichir vos données sans revenir sur votre application !

En effet il existe une application nommée **Windows App Studio Collection** avec laquelle vous aurez la liste de vos applications générées via ce site et qui vous permettra à partir de votre mobile ou PC de gérer vos collections en live ! La team Windows App Studio travaille aussi sur une future version qui permettra d'envoyer des notifications à vos application sans devoir les modifier. Qu'elles soient en *sideloading* ou publiées dans le store ! Fig.2.

Windows App Studio Collections permet donc d'ajouter du contenu « à distance » sans la republier ! Imaginez le restaurateur qui a son application avec sa Carte et qui peut ainsi modifier son menu ou ajouter des plats !
Vous pouvez aussi choisir et configurer votre tuile dynamique. Il existe maintenant un grand nombre de formats de tuiles et si vous choisissez une tuile statique avec votre logo, Windows App Studio va générer pour vous l'ensemble des images de vos tuiles. Et pour les développeurs, vous aurez le code généré XAML de votre tuile !

Lorsque votre application est terminée, vous pourrez la partager, via un email, Facebook ou encore via un QR code et autre : le *sideloading* a été simplifié à l'extrême. Par exemple, sur PC il faut lancer un script PowerShell avec le certificat etc. Avec une application spéciale, toutes ces étapes disparaissent ! Et ceci grâce à **Windows App Studio Installer** disponible sur le store. Fig.3.

Il existe aussi un composant Open Source qui se trouve sur GitHub sous le nom de *waslib* pour Windows App Studio Libraries (https://github.com/wasteam/waslibs/) et qui permet à un développeur de créer de nouvelles sources de données (Data Providers).
D'ailleurs une application d'exemple existe sur le store et se nomme **Windows App Studio UWP Samples** ; elle vous permettra d'évaluer l'ensemble des contrôles avec le code source correspondant ! Vous pouvez ainsi tester les composants, modifier les dimensions, l'apparence sans toucher à une ligne de code. Puis, si vous le désirez, vous pouvez cliquer sur un bouton [C#] pour visualiser le code C# correspondant à ce rendu ! Diabolique n'est-ce pas ! Fig.4.
Windows App Studio évolue tous les mois. On sait par exemple que des scenarios plus orientés Entreprises sont à l'étude, des connexions avec des bases Azure aussi... Pour les développeurs, il sera aussi bientôt possible d'ouvrir le projet directement dans Visual Studio Online ; ça sera un quatrième type de package téléchargeable qui s'appellera « Composant ». L'équipe WAS fait là un excellent travail et on ne peut que les féliciter. A noter que le responsable de WAS, est français, il est Senior Program Manager à Microsoft Corp. et s'appelle Michel Lopez. Avec Denis Voituron nous l'avons interviewé dans un podcast accessible via l'adresse :
http://devapps.be/podcast/windowsappstudio
N'hésitez pas à écouter cette émission !

Fig.3

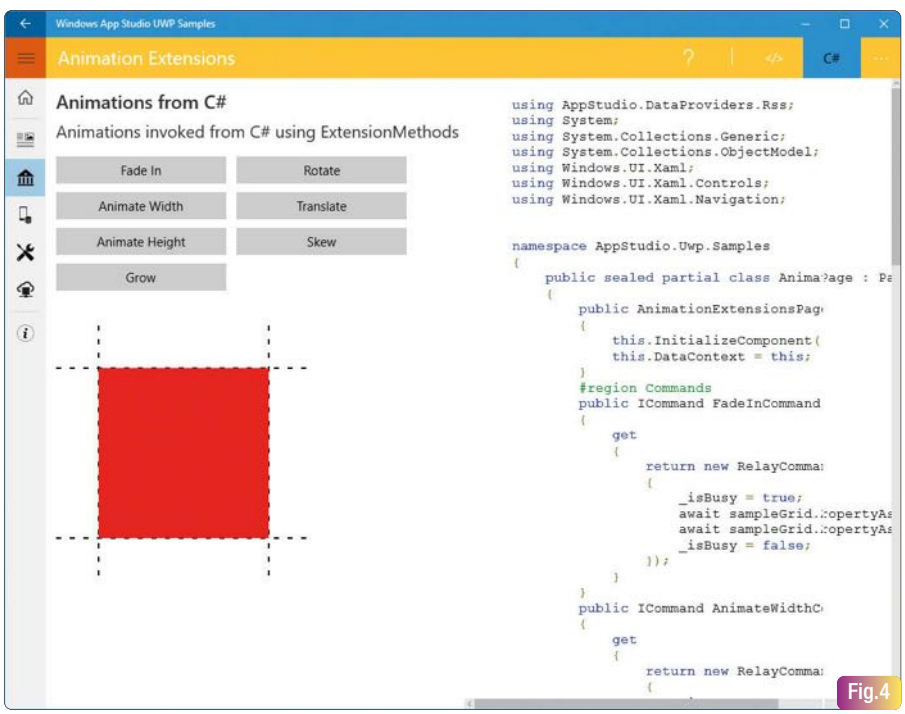


Fig.4

De Stargate aux Comics : l'Egypte des pharaons dans la culture geek

Le musée royal de Mariemont, non loin de Bruxelles, propose jusqu'au 20 novembre, une exposition atypique et surprenante rappelant un peu l'exposition « les nouveaux mythes » du Louvre : « de Stargate aux Comics, les dieux égyptiens dans la culture geek (1975-2015) ». Cette exposition rassemble une soixantaine d'objets (costumes, objets, dessins, planches...). De quoi assouvir sa passion geek !

François Tonic
Historien
Rédacteur en chef
de Pharaon Magazine

Depuis l'Antiquité, une forme d'égyptomanie a toujours existé. Au Proche-Orient ancien, on ne compte plus les décors s'inspirant de civilisation pharaonique (les formes, les divinités). Dans l'Empire romain, de nombreux sanctuaires égyptiens existaient pour célébrer les cultes d'Isis, d'Osiris, de Sérapis, ce qui voulut parfois la répression féroce de l'empereur qui y voyait un pouvoir d'influence trop important. Un véritable style égyptien existait et certaines villas de Pompéi et d'Herculanum le montrent encore aujourd'hui. Cette égyptomanie n'a jamais cessé. Trois événements vont déclencher une nouvelle égyptomanie, et, avec elle, une certaine idée de l'Egypte : la campagne d'Egypte de Bonaparte (et qui créa par la suite le style « retour d'Egypte »), le déchiffrement des hiéroglyphes par Champollion (1832), la découverte de la tombe de Toutankhamon (1922).

Une culture nourrie par une certaine vision de l'Egypte

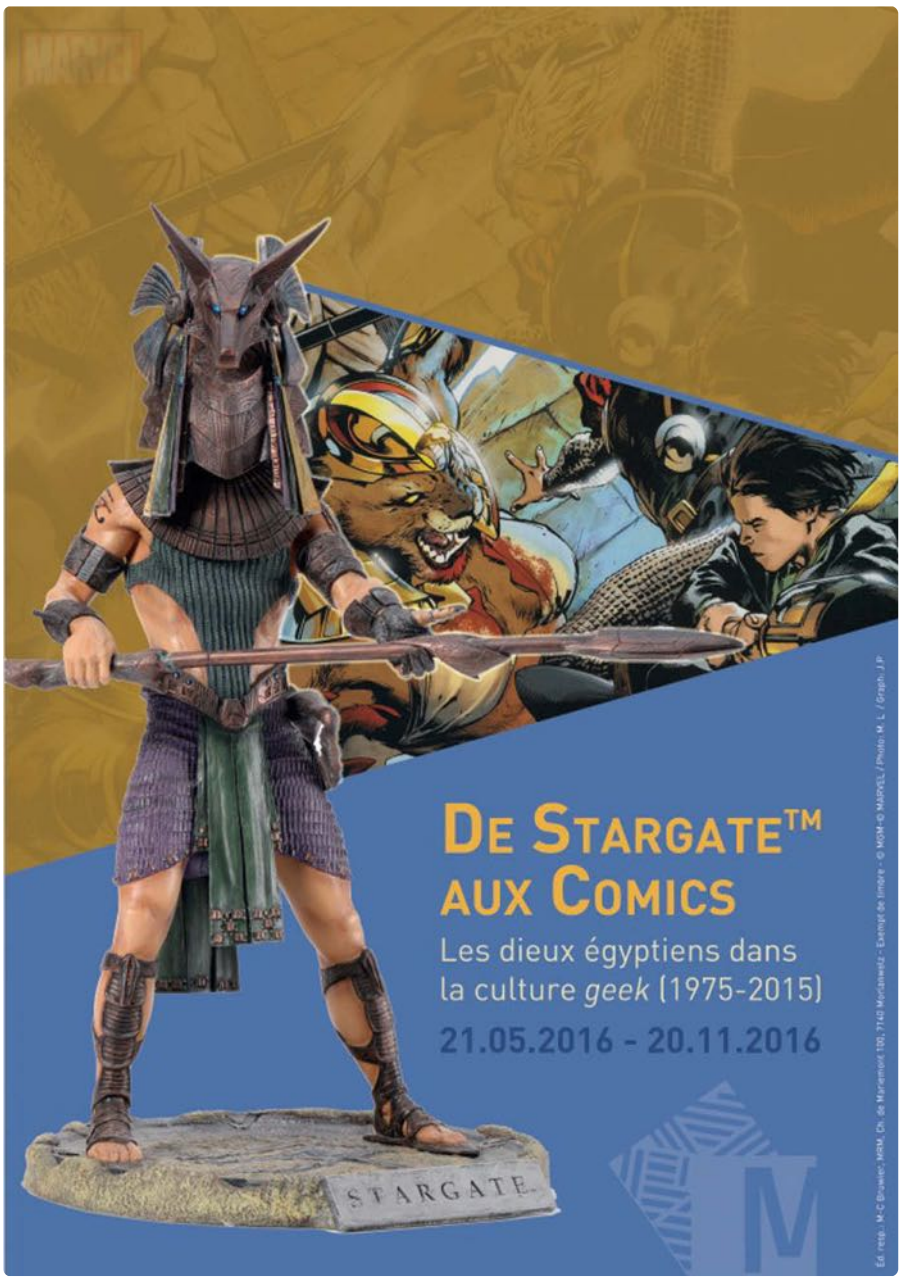
La malédiction de Toutankhamon va susciter des réactions dans le monde entier même si cette malédiction fut totalement inventée. Malédiction + momie, et nous avons un cocktail classique des films de momies. Un film incontournable est là pour nous le rappeler : La Momie (1999), remake des films des années 1920-1930. Toujours dans la catégorie « méchante momie qui nous pourrit bien la vie » : Mum-Ra (qui se régénère grâce à son sarcophage, tiens comme dans Stargate). Mais dans notre culture geek, Indiana Jones a une place à part et notamment « les aventuriers de l'Arche perdue » qui prend racine justement en Egypte.

Que dire de l'autre film culte, pour beaucoup

d'entre nous : Stargate. Le méchant Râ qui asservit une planète entière, les pyramides comme stations d'accueil pour des vaisseaux spatiaux, des armures reprenant les têtes des dieux égyptiens, des objets mystérieux... bref, tout pour plaire. Plusieurs séries ont suivi même si nous sommes moins fans, surtout après les saisons 3 et 4 de SG1.
Plus récemment, nous retrouvons les dieux égyptiens dans des références incontournables : X-Men Apocalypse. Ce mutant, surpuissant, est né en Egypte et est vénéré comme un dieu mais pour empêcher le transfert de son esprit et ses pouvoirs dans un corps plus jeune, il est enfermé dans sa pyramide... La représentation de l'Egypte, des monuments et des dieux, reprend les codes d'une Egypte fantasmée, stéréotypée (= ok, on fait tout dans la démesure) que l'on retrouve parfaitement dans les grands péplums comme Cléopâtre (1963) et dans les films récents. Cette ambiance égyptienne, principalement les pyramides et les dieux, se retrouve dans plusieurs films plus ou moins réussis tels que Immortel (d'Inki Bilal) ou Pyramide (2014) et le très récent Gods of Egypt. Ce dernier, pas si mauvais que cela, reprend les codes déjà vus dans Le choc des Titans ou encore John Carter.
Les séries TV ne sont pas absentes : Dr Who (avec Sutekh), Stargate, Rome.

BD, jeux vidéo, Comics

E.P. JACOBS (Blake & Mortimer), Hergé posent cette influence de l'Egypte dans notre culture. Par contre, plus spécifique à notre culture geek, nous retrouvons moins cette égyptomanie dans les Comics et autres Marvel, même si quelques exemples existent : Thor 240 (les dieux d'Héliopolis, 1975), ou encore des tentatives de rapprochements entre Loki et le dieu Seth. Nous trouvons quelques références dans les séries Batman.
Impossible ne pas évoquer le jeu vidéo. Plusieurs classiques ont fait la part belle à l'Egypte (monuments, rois, dieux) : Age Of Mythology, Civilization, Lara Croft and the temple of Osiris. Mais l'une des meilleures représentations de ces jeux de stratégie reste le superbe Pharaoh et son extension Cleopatra ! Ces jeux datent de 1998 – 99. Ils reprennent les bases d'un autre jeu bien connu, Caesar III. Nous sommes dans la grande période des Age of Empire et Civilization. Dans les cartes à jouer, nous trouvons des créations égyptiennes, chez Deus et Magic.
Site officiel du musée : www.musee-mariemont.be



Abonnement : Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com* - Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30. **Tarifs** abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € Autres pays : nous consulter.
PDF : 30 € (Monde Entier) souscription sur www.programmez.com

Une publication **Nefer-IT**
7 avenue Roger Chambonnet
91220 Brétigny sur Orge
redaction@programmez.com
Tél. : 01 60 85 39 96
Directeur de la publication : François Tonic
Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : J. Chenard, S. Saurel
Nos experts : T. Lebrun, E. Muraton, A. Coton, R. Degret, A. Caillaud, S. Lopes Neves, A. Vache, A. Lakhal, J. Trevier, W. Ben Rabat, J. Antoine, J. Moreau-Mathis, J. Perrot, F. Jacob, B. Lanneau, M. Bouilloux, L. Bentley, E. Baudoux, Lucas Zientek, H. Maiga, M. Maamar, M. Roussel, J. De Oliveira, A. Galtier, M. Krief, C. Gigax, S. Warin, N. Borneert, J-E Couignoux, C.Peugnet

Couverture : © Orion
Maquette : Pierre Sandré

Publicité : PC Presse,
Tél.: 01 74 70 16 30, Fax : 01 40 90 70 81
pub@programmez.com

Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.

Marketing et promotion des ventes :
Agence BOCONSEIL - Analyse Media Etude

Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Contacts
Rédacteur en chef : ftonic@programmez.com
Rédaction : redaction@programmez.com
Webmaster : webmaster@programmez.com
Publicité : pub@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

sales@ikoula.com
01 84 01 02 66
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 58
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

Openska

Formations

Web & Open Source

www.openska.com