



Rapport de stage



Création d'applications mobiles multiplateformes *Vers une optimisation du développement des programmes informatiques*



Stagiaire : José MARTINS
DUT Informatique Année spéciale 2009-2010

Tuteur : Sylvain JUBERTIE
Maître de Conférences - LIFO - Orléans

Table des matières

1	Introduction : développement d'applications sur plateformes mobiles	4
1.1	Le Laboratoire d'Informatique Fondamentale d'Orleans	4
1.2	Développement d'applications mobiles multiplateformes, état des lieux	6
1.2.1	Etude du marché des appareils et des applications mobiles	6
1.2.2	Le développement d'applications informatiques mobiles aujourd'hui	8
1.3	Les objectifs définis pour cette étude : un développement multiplateforme	10
2	Analyse des méthodes de développement pour Android et iPhone	12
2.1	Android : la plateforme Open Source mobile	12
2.1.1	La tablette Archos IT5	12
2.1.2	Installation de l'environnement de programmation	13
2.1.3	Le développement sous environnement Android	13
2.1.4	Mise en oeuvre : <i>Hello World</i> sous Android	15
2.2	Le iPhone	17
2.2.1	Les caractéristiques techniques du iPhone	17
2.2.2	Installation de l'environnement de programmation	17
2.2.3	Le développement sous environnement iPhone	18
2.2.4	Premières lignes de code pour iPhone : <i>Hello world</i>	20
2.3	Définir un mode de développement multiplateforme	24
2.3.1	Intégration du C/C++ sous Android	25
2.3.2	Intégration du C/C++ sous iPhone	26
2.3.3	Eléments d'un projet pouvant être factorisés dans le cadre d'un développement multiplateforme	27
2.3.4	Factorisation du code métier entre l'Android et le iPhone	28
3	Développement d'applications multiplateformes Android/iPhone	30
3.1	Etudes des performances en 2D et 3D de l'Archos IT 5 et de l'iPhone	30
3.1.1	Analyse	30
3.1.2	Réalisation	31
3.1.3	Résultats obtenus	32
3.2	Un projet multiplateforme avec code natif : une calculatrice	33
3.2.1	Analyse et méthodologie appliquée	33
3.2.2	Réalisation du projet	33
3.2.3	Retours d'expériences	37
3.3	Utilisation des instructions NEON	37
3.3.1	L'unité vectorielle NEON	37
3.3.2	Exemple de mise en oeuvre sur les deux plateformes	38
3.3.3	Les problèmes et limitations rencontrés	39
3.4	Méthode de développement multiplateforme : un premier bilan	39

4 Des objectifs aux réalisations : retours d'expériences et bilan	41
4.1 L'apprentissage technique et professionnel : 10 semaines de formation intensive	41
4.2 Apprendre à se connaître : un bilan personnel encourageant	42
4.3 Savoir conclure 10 semaines de stage	43
A Glossaire	44
B Moteur de calcul en code natif pour le projet calculatrice	46
C Présentation des plateformes de téléchargement légal pour Android et iPhone	48
D Aide mémoire pour l'Objective C	49
E Installer une chaine d'outils pour développer sous Android avec Linux	51
F Installer le SDK iPhone pour un ordinateur sous Mac OS	54
G Bibliographie et liens	55

Remerciements

Je tenais à remercier l'ensemble des membres de l'équipe enseignante de l'IUT Informatique d'Orléans qui ont pris le temps du partage. Cette montre là ne marque pas les heures de la même façon que celle de notre quotidien, elle nécessite patience et répétition.

Je tiens à remercier tout particulièrement M. Sylvain Jubertie, membre de l'équipe Parallélisme Réalité virtuelle et Vérification (PRV) du Laboratoire Informatique Fondamentale d'Orléans (LIFO). Son tutorat durant mon stage m'a permis de finaliser l'ensemble des objectifs que nous nous étions fixés.

Un merci tout particulier pour Mme Sylvie Haouy-Maure, le MacBook Pro que vous m'avez prêté m'a permis de réaliser la deuxième partie de mon stage dans de bonnes conditions.

Chapitre 1

Introduction : développement d'applications sur plateformes mobiles

Le DUT (Diplôme Universitaire Technologique) en Informatique est une formation permettant d'acquérir un diplôme de niveau Bac + 2 préparant aux métiers de l'informatique.

L'IUT (Institut Universitaire et Technologique) dispense traditionnellement ce cursus en 2 ans. Elle propose, dans le cadre de l'Année Spéciale, de le suivre en 1 an pour les étudiants titulaires d'un Bac + 2 (d'une autre filière) qui souhaitent se réorienter.

Composé d'une période de cours magistraux et de travaux dirigés sur 10 mois, le DUT se conclut par 10 semaines de mise en pratique des acquis au travers d'un stage en situation professionnelle.

Ce stage est donc un moment privilégié qui va me permettre :

- d'évaluer ma capacité à tenir un poste de travail dans le domaine informatique en terme de connaissances, d'adaptation et de réactivité.
- de valider mon orientation dans ce domaine, en testant une nouvelle situation professionnelle.

Les enseignants de l'IUT sont, pour une grande majorité d'entre eux, chercheurs au sein du laboratoire d'informatique de l'Université d'Orléans, le LIFO (Laboratoire Informatique de Recherche Fondamentale). Souhaitant poursuivre un cursus universitaire j'ai voulu réaliser mon stage dans cette structure. Aussi après discussion avec l'un d'entre-eux, M. Sylvain Jubertie, j'ai pu intégrer le LIFO pour y réaliser une étude sur le développement d'applications informatiques mobiles.

Ce rapport de stage présente cette étude, en 4 points. Dans un premier temps j'expliquerai ce qu'est le LIFO, le développement des applications mobiles et l'intérêt de cette étude au sein du laboratoire de recherche. Dans un deuxième temps je décrirai les méthodes actuelles de développement sur les deux plateformes mobiles phares que sont Android et le iPhone. Je mettrai ensuite en place les moyens techniques pour développer efficacement en multiplateforme avec une présentation des applications réalisées pendant le stage. Enfin je conclurai sur les apports, à titre professionnel et personnel, de cette mise en situation pratique après 10 mois d'enseignement.

1.1 Le Laboratoire d'Informatique Fondamentale d'Orleans

Situé au coeur du campus universitaire d'Orleans le LIFO (figure 1.1 page 5) est un laboratoire de recherche fondamentale en informatique créé en 1987. Il a été reconnu par le Ministère de la Recherche comme Equipe d'Accueil dans le cadre du programme quadriennal de développement de la recherche 2008-2011. Ce contrat a pour objectif de développer un secteur de recherche de haut niveau, et, de construire un pôle scientifique vivant au plan régional et européen. En tant qu'équipe d'accueil il intègre une équipe d'enseignants chercheurs qui travaillent dans le domaine des sciences et de l'ingénierie informatique.



FIGURE 1.1 – Le LIFO

Le LIFO possède un certain nombre de moyens techniques :

- des stations de travail
- une grappe de PC haute performance reliée à un réseau en Gigabit.
- un mur d'image complètement dédié
- de très grandes bases de données (notamment géographiques)

Le LIFO c'est aussi des partenariats nationaux et internationaux :

- en France, avec l'Institut National Recherche Informatique et Automatique (INRIA), le Commissariat à l'Energie Atomique (CEA)
- et dans le monde, avec l'Ecole Polytechnique de Montreal, l'Université Technique de Vienne (Autriche), de Beira Interior (Portugal), de Tokyo (Japon) et de Warwick (Grande Bretagne)

Le laboratoire est constitué de 4 équipes :

- l'équipe Contraintes et apprentissages (CA) qui s'intéresse à la sémantique et l'apprentissage des règles dans un formalisme relationnel comme, par exemple, la mise au point de programmes dans le cadre de la Programmation par Contraintes,
- l'équipe Graphes et Algorithmes (GA) qui travaille sur les graphes, leurs décompositions, leurs particularités, leurs modèles de calculs,
- l'équipe Sécurité et Distribution des Systèmes (SDS) qui étudie les problèmes de sécurité, d'accès, et de tolérances aux pannes lors de la mise en place de grands systèmes d'information,
- l'équipe Parallélisme et Réalité virtuelle et Vérification de système (PRV) qui effectue des travaux liés aux développements de langages, logiciels, bibliothèques pour la programmation parallèle et distribuée. Les domaines d'applications sont la réalité virtuelle, les bases de données, ou encore le calcul scientifique.

C'est au sein de cette dernière équipe que j'ai effectué mon stage pendant une durée de 10 semaines.

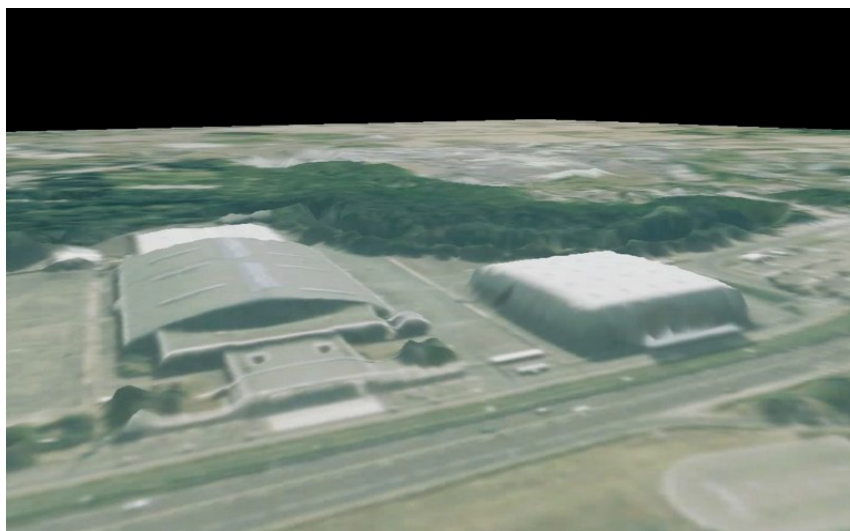


FIGURE 1.2 – Zenith d'Orleans - Modélisation 3D en temps réel - Equipe PRV

Photographie extraite de la page personnelle du LIFO d'Emmanuel Melin

1.2 Développement d'applications mobiles multiplateformes, état des lieux

Imaginons une application informatique, par exemple un jeu vidéo. L'éditeur souhaitant maximiser son marché diffusera sa production sur le plus de machines possibles. Cependant, et à l'image du marché des consoles de jeu de salon, plusieurs machines aux architectures pourtant assez similaires, n'acceptent pas une simple importation du code d'origine. Il faudra souvent reprogrammer une partie du logiciel, voire son intégralité. En termes de temps, de moyens humains, et plus simplement d'intelligence ce n'est pas raisonnable.

Cette étude est également motivée par l'explosion du marché informatique de plus en plus nomade et embarqué.

Enfin du point de vue de la recherche informatique le développement multiplateforme c'est aussi des enjeux en terme de performance de calcul, d'énergie, d'adaptation et de modélisation des projets (en terme humains, matériels etc.).

1.2.1 Etude du marché des appareils et des applications mobiles

Les acteurs économiques sur le marché des appareils mobiles sont nombreux : Apple, Google via Android son système d'exploitation, Blackberry, Samsung, LG ou encore Nokia avec Symbian. Matériellement, nos appareils nomades (téléphone, PDA) intègrent presque tous une architecture basée sur les processeurs développés par la société ARM (Advanced RISC Machines). Ce processeur à plusieurs avantages en termes de puissance, de coût financier, d'économie d'énergie. Pourtant malgré une conception similaire, du processeur jusqu'à l'écran, l'ensemble des constructeurs revendiquent des différences, par exemple, l'iPhone 4 serait plus rapide, le dernier Blackberry plus sûr, les NSeries de Nokia seraient plus proches des besoins des professionnels. Il est vrai que le marché est porteur et que celui qui se démarquera bénéficiera de l'engouement des clients, même si les différences ne semblent que commerciales. Il est important de faire la distinction, en termes de marché, entre les téléphones portables, les smartphones, les tablettes.

Un smartphone, littéralement "téléphone intelligent", est un terme utilisé pour désigner les téléphones évolués, qui possèdent des fonctions similaires à celles des assistants personnels. Certains peuvent lire des vidéos, des MP3 et se voir ajouter des programmes spécifiques.

En 2009 le téléphone mobile en France (téléphone et smartphone confondu) c'est ¹ :

- 58 millions de téléphones portables en France
- 90% de taux de pénétration (9 personnes sur 10 ont un mobile)
- 4,7 milliards d'euros de chiffre d'affaire

Dans le monde, la même année, les ventes se sont élevées à ² :

- 1,16 milliards d'exemplaires
- dont 456 millions pour le continent asiatique (120 millions uniquement au Japon)

Le iPhone, toutes versions confondues représente à lui seul 42 millions d'exemplaires vendus depuis sa mise sur le marché en 2007, le Blackberry près de 28 millions, les téléphones sous Android près de 8,5 millions ³. Sur le marché spécifique du smartphone, Blackberry est historiquement premier avec 36% de parts de marché, Android représente 28% et Apple 21%. Dans des pays comme le Japon, le iPhone représente 72% des ventes de smartphone en 2009.

Les tablettes ont les mêmes capacités hormis le fait de pouvoir téléphoner et possèdent généralement des écrans plus grands. Elles sont plus orientées vers un usage pour la vidéo, les images et la navigation sur Internet. L'arrivée de la tablette iPad de Apple bouleverse encore un peu plus la donne sur ce marché avec 3 millions d'exemplaires vendus en moins de 80 jours ⁴. D'autres fabricants espèrent profiter de ce phénomène, notamment Archos avec ses Internet Tablet IT5/IT7 ou encore l'Inde qui souhaite mettre en production une tablette à un prix réduit, 30 dollars. Ces chiffres semblent ne jamais vouloir s'arrêter et ce n'est que le début d'un vrai phénomène.



FIGURE 1.3 – iPhone 4 et Ipad - Apple.com

-
1. <http://telecom.sia-conseil.com>
 2. http://www.journaldunet.com/cc/05_mobile/mobile_marche_mde.shtml
 3. <http://www.numberof.net/>
 4. <http://www.apple.com/pr/library/2010/06/22ipad.html>



FIGURE 1.4 – Archos IT 5 - Une tablette française

Si ces appareils servaient encore à téléphoner il y a peu, ils ont bien d'autres utilisations aujourd'hui. La montée en puissance des processeurs les équipant, la mise à disposition par les constructeurs de kits de développement, le volume de pièces disponibles, et l'ouverture des plateformes de téléchargement légal ont rendu ce marché plus accessible. Les sociétés de développement de logiciels ont intégré ces machines dans leur stratégie. Pour 2010 le chiffre d'affaires escompté pour la vente d'applications exclusivement mobiles est de 6,7 milliards d'euros. Pour 2013 les prévisions sont de presque 30 milliards d'euros⁵ soit 2 fois le chiffre d'affaire 2009 de Renault⁶.

Le marché de l'application informatique est donc en pleine évolution. Le téléphone devient compagnon du quotidien et tous les acteurs du développement s'engagent sur ces plateformes. Google Maps, Facebook, Ebay, Need for Speed, Les Sims... tous ces succès ont désormais leur version mobile, développée spécifiquement pour iPhone, pour Android, pour iPad.

Le développement sur chaque plateforme nécessite des outils, des langages, du matériel spécifique, et par conséquent, sa propre équipe de développement. La multiplication des appareils augmente donc les temps, la complexité, et donc les coûts des projets. Ne peut-on alors rendre le portage d'un jeu d'une plateforme à l'autre moins contraignant, par exemple, en évitant au maximum le code spécifique à chaque appareil ?

1.2.2 Le développement d'applications informatiques mobiles aujourd'hui

Le développement d'une application informatique est découpé en plusieurs étapes :

- l'identification du besoin (par exemple la demande d'un client, une part de marché, une idée)
- l'analyse (les moyens à mettre en place, le choix du langage, la plateforme de développement)
- la programmation proprement dite
- les tests et les corrections.
- la mise sur le marché
- la maintenance et la mise à jour

Pour développer un logiciel sur plusieurs plateformes certaines de ces phases sont mutualisables. L'expression du besoin et l'analyse sont toutes deux réalisées une seule fois par projet. Ce n'est pas le cas de la partie programmation. En effet, une fois les deux premières étapes réalisées les équipes de développement se mettent au travail. Pour chaque plateforme il faut donc une équipe de programmeurs, qui réalise l'intégralité du logiciel.

5. http://www.journaldunet.com/cc/05_mobile/mobile_marche_mde.shtml

6. Wikipédia

Prenons l'exemple du jeu vidéo Need For Speed (figure 1.5) d'Electronic Arts qui est présent sur toutes les consoles et appareils mobiles actuels. Le site de l'éditeur⁷ fait d'ailleurs une place importante à la présence de son jeu sur toutes les consoles et appareils d'aujourd'hui. Il possède en interne des équipes de développement pour les consoles nouvelles générations et PC. Il sous-traite les développements sur mobiles à des studios externes comme la société londonienne Slightly Mad Studios. Les ressources humaines pour couvrir tous ces développements d'un même jeu sont donc importantes. Cependant on peut penser que certaines parties de code, comme la routine de gestion des véhicules, des routes et des scores, n'a pas à être réécrite autant de fois que de plateformes. De plus la multiplication des lignes de programmation augmente d'autant le risque d'erreurs dans le code du logiciel.



FIGURE 1.5 – Le jeu Need for Speed sur PS3



Le même sur plateforme mobile de type Android

FIGURE 1.6 – Le même sur plateforme mobile de type Android

Cette fuite en avant en terme de développement est nécessaire quand de nouvelles architectures font leurs apparitions par exemple les nouvelles cartes graphiques. Mais sur des architectures similaires comme celles présentes dans les smartphones et tablettes, le code redondant pourrait être rendu portable et compatible d'un appareil à un autre.

La notion de portabilité du code n'est pas nouvelle. Dès sa création en 1972 par D. Ritchie et B. W. Kernighan le langage C intègre cette particularité. Un programme écrit en C en respectant la norme ANSI est portable sans modification sur n'importe quel OS ou/et architecture disposant d'un compilateur C : PC sous Windows, station de travail sous UNIX. Cependant la portabilité du C est limitée, elle nécessite de recompiler le code source en C pour chaque plateforme ce qui implique la création d'un programme particulier pour chaque architecture.

7. <http://www.needforspeed.com>

Le Java a été créé en 1992 et corrige le problème de la recompilation. Ce langage s'appuie sur l'idée d'introduire une machine virtuelle (Java Virtual Machine) spécifique à chaque architecture qui exécute un code générique. Le code est compilé une seule fois, cependant cette approche nécessite la présence d'une JVM spécifique à l'architecture. Par exemple un code Java compilé s'exécute indifféremment sur Linux ou Windows, qui disposent chacun de leur JVM, mais pas sur les plateformes mobiles d'Apple, qui n'en disposent pas.

Les constructeurs de plateformes mobiles fournissent des kits de développement appelé SDK (Software Development Kit) qui définissent la ligne directrice pour le développement d'applications. Ainsi Android a choisi de réaliser un SDK basé sur le langage Java. Dans le cadre du iPhone le SDK oblige le développeur à programmer en Objective C. Ces choix s'expliquent par la volonté de rendre les accès à la machine plus simple mais aussi de sécuriser sa plateforme. En effet en terme de concurrence il est indispensable de conserver le plus possible l'exclusivité de sa technologie pour forcer un développement spécifique, voir unique d'une application. C'est une force de vente réelle qui amplifie la place d'un appareil par rapport à un autre.

Dans le cadre de ce stage nous avons voulu démontrer qu'il était possible de repenser le développement d'applications en définissant :

- les facteurs communs à chacune de ces machines (du point de vue matériel et logiciel)
- en choisissant une méthode de programmation la plus portable possible avec l'utilisation d'un langage natif de type C ou C++
- en trouvant des solutions de prises en main des SDK pour intégrer du code natif.

1.3 Les objectifs définis pour cette étude : un développement multi-plateforme

En programmant avec le langage C, le développement informatique devrait ressembler à un code unique et un compilateur qui créerait un exécutable différent pour chacune des plateformes (figure 1.7 page 10).

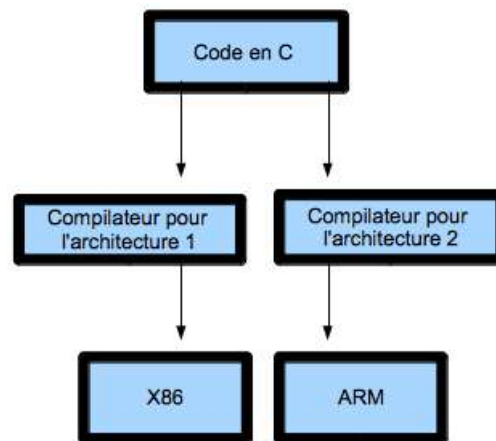


FIGURE 1.7 – Génération d'un programme à partir d'un code C

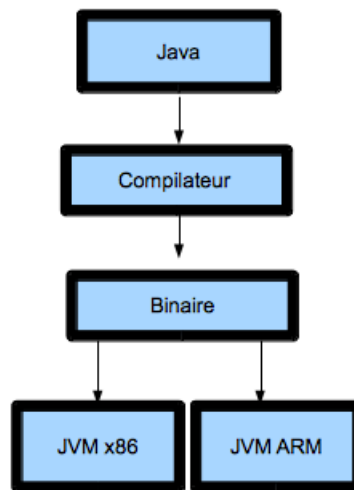


FIGURE 1.8 – Génération d’un programme à partir d’un code Java

Pour un projet en Java le code est réalisé une fois et exécuté sur chaque machine par un programme appelé machine virtuelle (figure 1.8 page 11).

Le Java n’est pas une technologie disponible sur le iPhone pour des raisons historiques, l’Objective C étant le langage de programmation de référence chez Apple, mais également pour des raisons de contrôle et de limitation des applications disponibles sur leur plateforme. En effet, la présence d’une JVM sur leurs plateformes, introduirait la possibilité de lancer des applications tierces.

Le C n’est pas disponible de base sur l’OS Android. Le choix de Google s’est porté sur le Java pour des raisons d’ouvertures aux plus grands nombres de projets et développeurs.

On remarque des restrictions similaires sur l’ensemble des plateformes mobiles. Il y a toujours une raison qu’elle soit technique ou commerciale qui empêche d’utiliser des solutions générales. Il faut donc concevoir une méthode qui soit transversale en s’appuyant sur les techniques déjà existantes, et qui soit flexible pour que des adaptations mineures rendent l’exécution du programme possible sur toutes plateformes.

Le LIFO, au sein de l’équipe PRV, effectue des recherches sur les calculs parallèles et distribués. Dans le cadre de cette étude, les méthodes alternatives de développement pourraient être appliquées aux plateformes mobiles pour mobiliser leurs processeurs, avec d’autres plateformes, sur la base d’un même algorithme mathématique et donc un même code.

Pour réaliser ces objectifs, sur les 10 semaines de l’étude j’ai eu à ma disposition :

- un Mac Book Pro
- un tablette Archos IT 5
- des PC sous environnement Linux
- de la documentation technique
- un accès Internet très rapide.

Le cahier des charges à respecter était le suivant :

- faire des tests de programmation sur la tablette Archos IT5 et l’iPhone et sur PC/Mac
- définir une méthode de travail permettant de développer en dehors des outils existants (SDK)
- faire un projet multiplateforme
- réaliser les documents, dont ce rapport, avec \LaTeX .

Chapitre 2

Analyse des méthodes de développement pour Android et iPhone

Avec une prévision de vente pour 2012 de plus de 150 millions d'exemplaires, les smartphones Android ou iPhone sont les acteurs principaux du marché. Nous avons donc porté notre étude sur ces deux plateformes en particulier.

En prenant le temps de l'analyse de chacune de ces plateformes nous mettrons en évidence les particularités de chaque appareil, les méthodes de développements traditionnels en suivant les SDK avec un exemple. Puis dans un dernier point nous exposerons une méthode plus généraliste de développement rendant le code portable sur les 2 appareils concurrents.

2.1 Android : la plateforme Open Source mobile

Le LIFO a mis à ma disposition tout au long de ces 10 semaines une tablette Archos IT 5. Cette tablette est de conception française et Archos possède, depuis un certain temps déjà, une bonne expérience des appareils mobiles, notamment les lecteurs MP3. Elle a la particularité d'être installée avec un système d'exploitation Android qui a été développé par Google.

2.1.1 La tablette Archos IT5

L'IT5 (figure 1.4) est équipée d'un processeur ARM Cortex A8, en 32 bits avec une fréquence d'horloge à 800 MHz. Elle possède 256 Mo de RAM, un DSP (digital signal processor) additionnel cadencé à 430 MHz. Elle est équipée de base avec un disque dur de 160 à 500 Go.

Elle dispose d'un écran tactile de 4,8 pouces avec une résolution de 800*480 pixels.

Elle est présentée comme un ordinateur personnel miniature, car elle embarque les fonctionnalités que l'on retrouve dans nos PC :

- Wi-Fi intégré
- ports USB disponibles via un adaptateur
- GPS présent
- lecteur de carte mémoire
- tuner télévision par adaptateur.

L'IT5 est capable de lire des vidéos au format Mpeg 4 HD (720p) et décode facilement les fichiers audio MP3. Grâce à son disque dur elle fait également office d'enregistreur audio et vidéo.

L'appareil photo n'est pas disponible, l'Archos IT5 possède pourtant un programme de visualisation d'image.

2.1.2 Installation de l'environnement de programmation

L'IT5 est livrée sous Android Cupcake 1.5. Elle a été mise à jour (1.6 Donuts). Ce nommage est particulier à Google qui donne à chacune de ces versions d'OS un nom de gâteau, Eclair pour la version 2.0 par exemple. Chaque nouvelle version améliore les outils de développement et apporte des fonctionnalités supplémentaires.

Le téléchargement du SDK sur le site officiel est la première étape pour développer sur les plateformes sous Android¹. Il faut ensuite se munir de l'environnement de développement (IDE : Integrated Development Environment) Eclipse², créé par IBM. Les ordinateurs sous Windows, Linux ou Mac OS peuvent accueillir l'ensemble de cette chaîne d'outils, Google et IBM ayant fourni leurs outils SDK sur chacun de ces OS (Operating System, système d'exploitation).

Pour tester nos projets il faut utiliser un simulateur de téléphone Android qui n'est pas intégré à Eclipse. Il faut l'installer et informer Eclipse de son existence pour qu'il traite l'exécution du projet au travers de celui-ci, pour cela il convient de :

- démarrer Eclipse :
- dans le menu Help sélectionner Install New Software
- cliquer sur Add site
- une barre de navigation s'ouvre
- inscrire <https://dl-ssl.google.com/android/eclipse/>
- le téléchargement s'exécute et le plugin s'installe.
- dans la page suivante le plugin apparaîtra avec l'inscription Developer Tools
- après l'avoir coché il faut cliquer sur Next
- après lecture de la licence Android et acceptation, sélectionner Finish
- enfin redémarrer Eclipse.

Nous venons d'installer Android Development Tools (ADT) qui est un plugin pour Eclipse IDE pour l'installation du simulateur et bien plus. Il étend la capacité de l'IDE pour l'OS de Google, permettant de compiler, d'exécuter du code Android, de créer des interfaces graphiques et d'exporter le tout signé pour une mise en vente sur l'Android Market. L'environnement de travail est maintenant prêt à l'emploi.

2.1.3 Le développement sous environnement Android

Le langage de programmation utilisé dans le cadre de projet Android est le Java qui est très répandu dans les développements informatiques actuels..

En ouvrant un premier projet sous Eclipse un certain nombre de fichiers sont déjà présents sans même qu'une seule ligne de code n'ait été écrite. On y trouve (figure 2.2 page 15) :

- un dossier src qui regroupe toutes les classes Java du projet
- un dossier Android Library avec toutes les bibliothèques utilisées
- un dossier assets qui contient des données de type licence qui seront chargées en même temps que l'application lors de l'intégration de celle-ci dans le téléphone
- un dossier res pour toutes les ressources annexes : images, fichiers XML, son etc...
- le fichier AndroidManifest.xml définit le comportement de l'application au système Android. Ce fichier définit par exemple, le nom, l'icône (par défaut drawable/icon.png), le thème, la version minimale du système nécessaire à l'exécution de l'application, les activités, les services.

1. <http://developer.android.com/sdk/index.html>

2. <http://www.eclipse.org/>

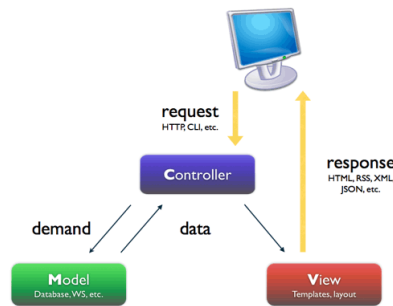


FIGURE 2.1 – MVC

http://www.symfony-project.org/images/jobeeet/1_2/04/mvc.png

Le développement des applications sous Android est associé à la notion de modèle-vue-contrôleur (MVC, figure 2.1 page 14). L'approche MVC vise à décomposer un programme en trois parties :

- la vue constitue la couche graphique, intégrant les boutons, les zones de textes, les images
- le modèle est la représentation codée du comportement d'un bouton par exemple
- le contrôleur agit comme une liaison entre la vue et le modèle.

Ce découpage permet de simplifier le développement en isolant le code métier (le modèle), de l'interface graphique (vue) et de leurs interactions (contrôleur). Ainsi la modification de l'un n'impose pas la réécriture des autres optimisant ainsi la réalisation et la maintenance des projets.

Avec l'arrivée des plateformes mobiles ce type de programmation est devenue un standard destiné à faciliter la conception. Il permet de distinguer la notion d'interface graphique, qui peut être par exemple déléguée à un graphiste, des interactions (appui sur un bouton etc.) et des actions liées après intervention de l'utilisateur.

Les interfaces graphiques sont réalisées à l'aide de fichiers XML (eXtensible Markup Language). Le XML est un langage informatique qui sert essentiellement à stocker/transférer des données de type texte structurées en champs arborescents. Ce langage est qualifié d'extensible car il permet à l'utilisateur de définir des marqueurs (balises) qui facilitent le parcours au sein du fichier et donc la lecture de l'information. On définira, par exemple un bouton, et ses propriétés dans une balise `Monpremierbouton`. Lors de son utilisation on appellera le fichier XML avec comme marqueur `Monpremierbouton`, l'ensemble des informations nécessaires se trouveront alors à la suite. Ce langage est couramment utilisé aujourd'hui, et son apprentissage est aisé.

```
// Exemple de code XML
<textView
  android:text="@string/hello "
  android:textColor="@color/white "
  android:textSize="@dimen/dim_hello " />
```

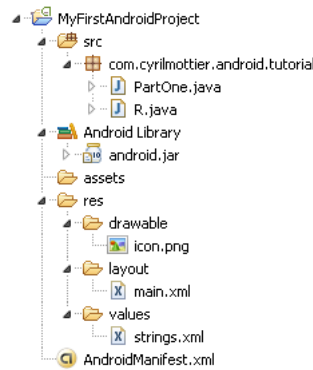


FIGURE 2.2 – Arborescence d'un projet Android sous Eclipse

Pour bien comprendre l'ensemble des étapes de développement d'un programme sous OS Android nous allons détailler un exemple simple et traditionnel : le *Hello World*.

2.1.4 Mise en oeuvre : *Hello World* sous Android

Ce programme consiste à afficher le message *Hello World* à l'écran. Pour ce faire, commençons par créer le projet dans l'environnement Eclipse pour cela il faut :

- cliquer sur File > New > Other > Android Project
- une boîte de dialogue s'ouvre :
- inscrire textitHello World dans le champ project name
- cocher Create new project in workspace
- choisir Android 1.6 comme Target
- donner un nom à l'application, celui-ci sera affiché sous l'icône de l'application
- donner un nom au package Java (ex : com.unnom.android)
- donner un nom à l'Activity (le point d'entrée dans le code)
- définir le SDK minimum à utiliser pour ce programme (dans notre cas 1.6)
- cliquer sur Finish.

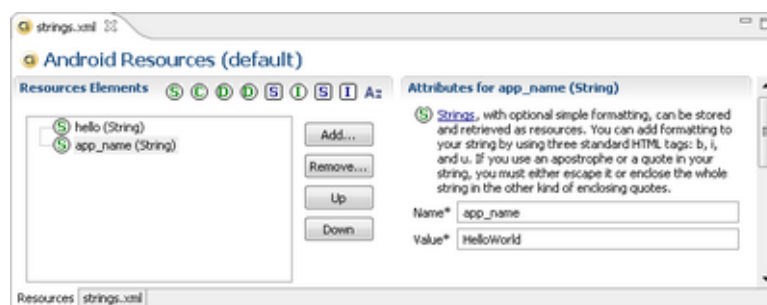


FIGURE 2.3 – Sous Eclipse - création d'une string XML

Pour changer la valeur du texte à afficher, ici « Hello World », il faut, dans l'arborescence du projet (figure 2.2) :

- ouvrir le fichier strings.xml
- une fenêtre (Android Ressource) va s'ouvrir (figure 2.3 page 15)
- changer la valeur de la chaîne hello
- enregistrer la modification.

Afin d'afficher la chaîne il faut définir à l'écran un espace suffisant pour cela, dans notre cas vingt pixels³ suffisent. Pour cela il faut créer un fichier `dimensions.xml` (en sélectionnant l'option Add) dans lequel inscrire le couple :

```
" dim_hello " / " 20px "
```

Pour finir il faut sauvegarder : Save.

Toutes les ressources sont maintenant prêtes. Il faut ajouter celles-ci à l'interface graphique. Après avoir cliqué sur `main.xml` l'éditeur « layout » s'affiche. Celui-ci permet de créer les interfaces qui vont servir de conteneur à notre texte. Nous allons directement taper le code XML suivant et le sauvegarder :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical|center_horizontal"
    android:orientation="vertical">
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="@string/hello"
    android:textColor="@color/white"
    android:textSize="@dimen/dim_hello" />
</LinearLayout>
```

L'interface graphique étant maintenant configurée, il nous reste à créer le programme Java permettant de l'afficher. Il suffit de se rendre dans `HelloWorld.java`. Ce fichier contient le code suivant :

```
package mettez ici votre package;
import android.app.Activity;
import android.os.Bundle;
public class HelloWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

L'objet *HelloWorld* hérite d'*Activity* ce qui permet d'afficher une interface graphique pour l'utilisateur. La mise en place de notre interface graphique s'effectue simplement en redéfinissant la méthode `onCreate`. On applique la vue créée à l'aide de la méthode `setContentView` qui prend en paramètre notre layout. Notre code pour le *Hello World* est prêt, il ne reste plus qu'à générer le paquet. Pour cela il suffit de cliquer sur l'icône Build & Run présent dans la barre.

En utilisant les outils fournis avec le SDK, et en restant dans le cadre habituel d'un projet Android avec l'utilisation de fichiers XML et d'Eclipse, le développement est facilité car on ne code pas les interfaces graphiques qui sont générées automatiquement par le fichier XML. Les outils de compilation et d'exécution, ainsi que le simulateur, sont gérés par Eclipse de manière transparente pour l'utilisateur.

3. Nous utilisons les pixels (px) comme mesure mais d'autres dimensions sont disponibles comme les pouces (inch) ou les points (pt)

2.2 Le iPhone

Le iPhone est un accessoire à la mode mais c'est un téléphone qui, technologiquement, est assez proche de ses concurrents. Le développement s'effectue sur la base du langage Objective C (figure 2.6 page 18) avec l'IDE XCode (figure 2.5 page 18) et avec l'Interface Builder (figure 2.7 page 19) pour gérer le MVC. Nous verrons au travers d'un exemple que la syntaxe de l'Objective C est assez particulière, et qu'un projet sous iPhone se construit autour du MVC.

2.2.1 Les caractéristiques techniques du iPhone

Au coeur l'iPhone 4 on trouve un processeur ARM Cortex-A8 1 GHZ avec technologie Néon (figure 2.4 page 17). Construit sur une architecture éprouvée il dispose de mémoire cache, d'instructions pour le traitement d'image (Néon), d'une gestion des économies d'énergie puissante.

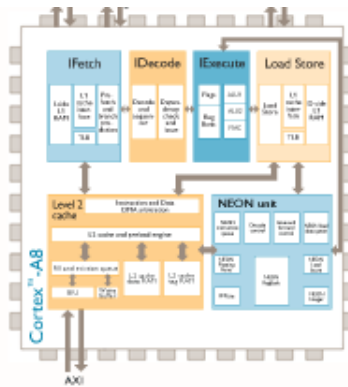


FIGURE 2.4 – ARM Cortex A8

Il intègre :

- un téléphone GSM 3G
- un GPS
- une boussole numérique,
- les réseaux sans-fils Wi-Fi et BlueTooth
- un écran tactile d'une résolution de 320X480 pixels, triplée avec la dernière génération.

2.2.2 Installation de l'environnement de programmation

La première étape pour développer sur cette plateforme est l'achat d'un ordinateur Apple, comme un MacMini ou un IMac, soit environ un investissement de 1000 euros⁴. Avec l'achat d'un iPhone à 400 euros le budget est de minimum 1400 euros pour commencer à développer sur cette plateforme. Les fondateurs d'Apple ont favorisé le développement d'applications dans leurs gammes d'appareils, il n'y a donc pas de kit de développement iPhone disponible sous Windows ou encore sous Linux.

Il faut ensuite, après inscription obligatoire sur le site, télécharger le SDK Apple⁵ et l'installer. En l'associant à l'IDE XCode (figure 2.5 page 18) qui est fourni avec l'OS Snow Leopard on obtient un environnement de développement complet.

4. <http://www.apple.com/fr/imac/design.html>

5. <http://www.apple.com/fr/ipad/sdk>



FIGURE 2.5 – Ouverture de projet Xcode

2.2.3 Le développement sous environnement iPhone

L'objective C est le premier élément de la chaîne d'outils nécessaire à la création d'applications sous iPhone.

```
// on "init" you need to initialize your instance
-(id) init
{
    if( (self=[super init]) ) {
        player[0] = [CCSprite spriteWithFile:@"coeur.png" rect:CGRectMake(0, 0, 82,
88)];
        compteurderadian=0;
        compteurderadian2=3.14*10;
        compteurdesprite=0;
        [self addChild:player[0]];
        [self schedule:@selector(nextFrame:)];

        myLabel = [CCLabel labelWithString:@"Nombre de sprite 1" fontName:@"Marker
Felt" fontSize:20];
        myLabel.position = ccp(350,300);
        [self addChild: myLabel];

        [[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:0
swallowsTouches:YES];
    }
    return self;
}

- (void) nextFrame:(ccTime)dt {
    double ajout=0;
    for (int i=0;i<compteurdesprite;i++){
        player[i].position = ccp(240+200*sin((compteurderadian+ajout)*2)*cos
((compteurderadian+ajout)),160+140*cos((compteurderadian2+ajout)*2)*sin
((compteurderadian2+ajout)));
        ajout = ajout + (((3.14*2)/180)*2);
    }
    compteurderadian=compteurderadian+((3.14*2)/180);
    compteurderadian2=compteurderadian2-((3.14*2)/180);
    if (compteurderadian>=(3.14*10)) {
        compteurderadian=0;
        compteurderadian2=3.14*10;
    }
}
```

FIGURE 2.6 – Objective C - Exemple de code - Sprites

Ce langage est propre à Apple et surtout utilisé pour le développement d'applications sous MAC OS ou encore dans la gamme iPhone et Ipad. Il peut cependant aussi être utilisé sous Linux ou Windows (avec un compilateur approprié sous ces plateformes). Créé par l'américain Brad Cox, au début des années 80, il hérite du C et du SmallTalk, un des premiers langages objet. Il intègre de nouvelles composantes dans le cadre du développement de l'OS NextStep par la société Next. Les typages les plus utilisés en Objective C portent les initiales NS car structurés comme cet OS. Ce langage a une syntaxe différente des langages objets plus répandus. Il nécessite un vrai investissement de la part du développeur et la lecture de nombreux ouvrages pour se familiariser avec sa conception.

En ouvrant un nouveau projet sous Xcode l'option projet pour iPhone est alors disponible. L'utilisation de l'Interface Builder (figure 2.7 page 19) est fortement conseillée, il est toujours présent dans les tutoriels disponibles sur le site du constructeur. IB est un utilitaire qui permet de générer facilement des interfaces graphiques en quelques clics de souris. Il génère le code associé et l'intègre dans le programme.

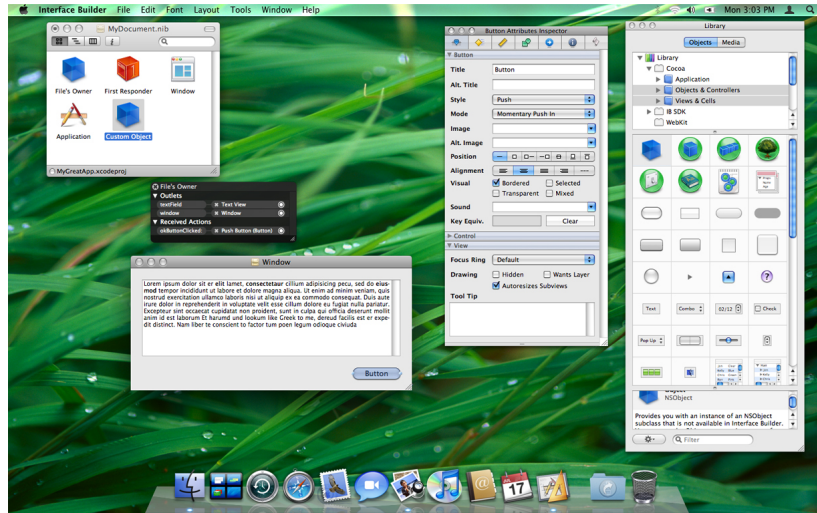


FIGURE 2.7 – Interface Builder sous Imac

Le développement des applications sous iPhone est associé à la notion de MVC avec l'utilisation d'IB à l'instar des plateformes Android.

Lors de la première utilisation sous Xcode on aperçoit la structuration du projet pour une application iPhone(figure 2.8 page 19). Elle est propre aux projets iPhone sous XCode. Elle regroupe l'ensemble des informations et contenus du projet.

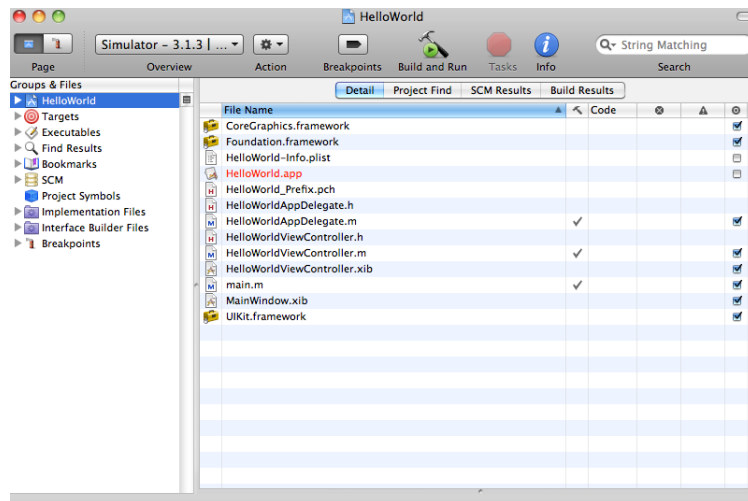


FIGURE 2.8 – Dossier de développement sous Xcode pour un projet iPhone

Comme nous l'avons vu, l'ensemble de la chaîne de développement sous univers Mac est très spécifique et impose l'utilisation de certaines ressources (Objective C, IB).

2.2.4 Premières lignes de code pour iPhone : *Hello world*

Pour mieux comprendre la construction d'un projet sous cette plateforme nous allons analyser le développement d'un *HelloWorld*.

Il faut d'abord ouvrir un nouveau projet sous Xcode, puis choisir la plateforme iPhone et enfin un projet en cliquant sur l'icone View-BasedApplication. Il faut nommer le projet HelloWorld.

Xcode crée alors des fichiers dont HelloWorldViewController.h et .m, ils constituent le contrôleur de la vue. Il faut intégrer ce code à cet endroit.

Pour réaliser un *HelloWorld* il va nous falloir un label où inscrire du texte, y inscrire le message, associer le label à la vue, demander à la vue de s'afficher après rafraîchissement. Mais avant tout définissons la cible pour débbugger : soit le simulateur, soit le iPhone (il faut posséder une licence).

Maintenant il faut créer un nouvel objet :

- faire Files/New files/
- puis Objective C Class et UIView.

Un objet de type vue vient d'être créé dans le MVC :

- nommer le Mavue
- à l'intérieur de celui-ci il faut intégrer les labels et autres boutons nécessaires au projet
- ajouter le code dans MaVue.h :

```
@interface Mavue : UIView {
    IBOutlet UILabel *monlabel;
}
@end
```

Une classe de type view est créée et un label est intégré et géré par IB pour y associer du texte.

Dans le fichier Mavue.m il faut ajouter ce code :

```
-init {
    [monlabel setText:@"Hello world"];
}

- (void)dealloc {
    [super dealloc];
    [monlabel release];
}
```

On gère ici la destruction de l'objet et de ses composantes c'est-à-dire la libération de la mémoire qu'ils occupent.

Il faut ensuite sauver, et double cliquer sur MainWindow.xib. Cela va produire l'ouverture de IB. Dans le menu Tools :

- ouvrir Library(figure 2.9 page 21)
- prendre un objet view dans la librairie
- il faut l'ajouter dans Window qui représente le iPhone, il peut être redimensionné si besoin.



FIGURE 2.9 – Objet disponible dans IB pour ajouter à une vue

Il faut ensuite aller dans l'Inspecteur (figure 2.10 page 21) :

- dans l'onglet View Identity changer le nom de la classe en Mavue.

La vue créée sous IB vient d'être associée à la vue implantée sous Xcode.

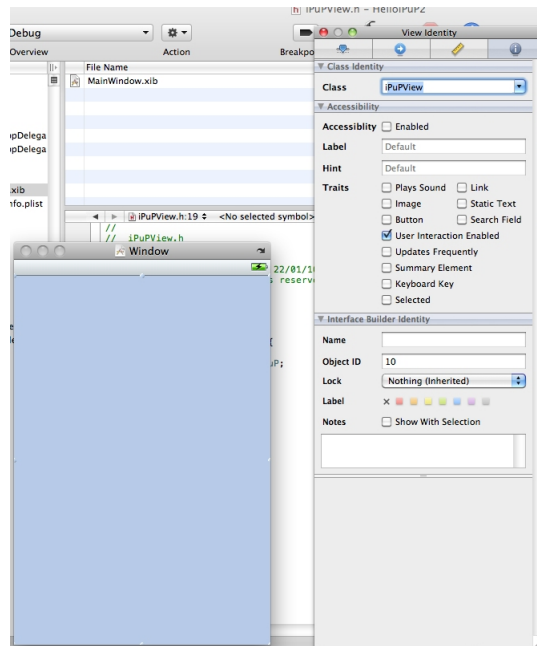


FIGURE 2.10 – L'Inspecteur sous IB

Il faut maintenant retourner dans Library :

- choisir cette fois un objet de type Label
- le déposer dans Window
- relier le label monlabel (dans le code précédemment implanté) à l'élément graphique que nous venons de positionner
- cliquer sur "Mavue" dans la navigation hiérarchique de nos vues (figure 2.11 page 22)
- dans l'inspecteur se rendre dans l'onglet "Mavue connections".
- relier (figure 2.12 page 22) en cliquant dans le rond vide à côté de monlabel
- glisser le tout en maintenant le clic jusqu'au label dans Window.
- une liaison entre le code et l'interface graphique (figure 2.13 page 23) vient d'être créée
- sauver le fichier
- quitter IB

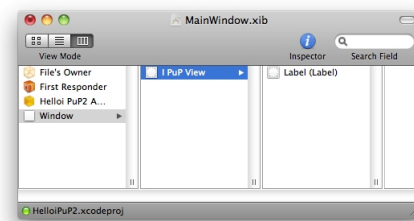


FIGURE 2.11 – Navigation hiérarchique des vues

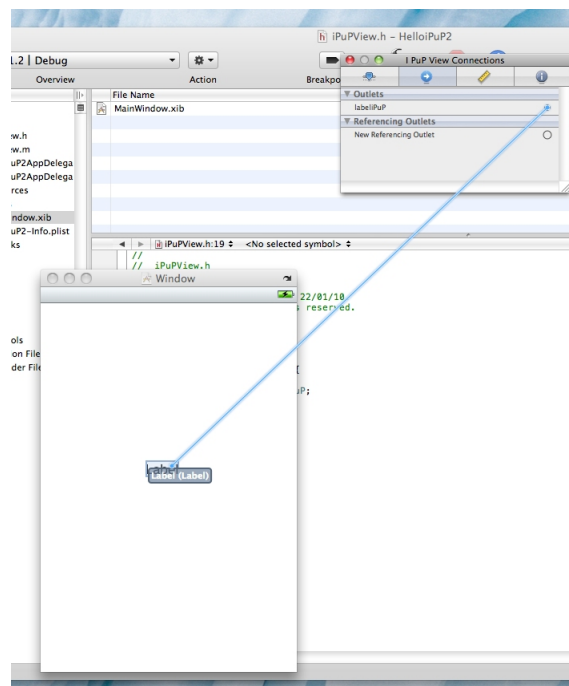


FIGURE 2.12 – Relier une déclaration de label à son interprétation graphique sous IB

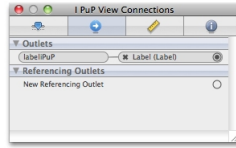


FIGURE 2.13 – Résultat d’une liaison code et élément graphique sous iB

Sous Xcode il faut se rendre dans HelloWorldAppDelegate .h pour importer Mavue.h et déclarer un objet de type Mavue.

```
#import <UIKit/UIKit.h>
#import "Mavue.h"
@interface HelloWorldAppDelegate: NSObject <UIApplicationDelegate> {
    UIWindow *window;
    IBOutlet Mavue *vue;}
@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) Mavue *vue;
@end
```

Dans HelloWorldAppDelegate .m il faut ajouter :

```
#import "HelloWorldAppDelegate.h"
@implementation HelloWorldAppDelegate
@synthesize window;
@synthesize vue;
- (void)applicationDidFinishLaunching:( UIApplication *)application {
    [window addSubview:vue];
    [window makeKeyAndVisible];}
- (void)dealloc {
    [vue release];
    [window release];
    [super dealloc];}
@end
```

Une fois revenu sous IB il faut :

- relier cette vue à Window comme précédemment
- sauver
- cliquer sur Build and Run dans Xcode.
- le texte apparaît à l’écran du simulateur.

L’utilisation d’IB nécessite un temps d’adaptation. Passé ce délai il facilite la création des interfaces graphiques, notamment le placement et le dimensionnement des objets. C’est un logiciel propriétaire Apple, les fichiers Xib ne sont utilisables que sur Mac ou iPhone/Ipad. On ne peut donc pas simplement reprendre tout, ou une partie, du programme et l’intégrer dans un environnement Android si on respecte les méthodes de développement du SDK Apple. Il nous faut donc repenser nos méthodes et trouver les moyens de la factorisation du code et de son implantation sur les deux plateformes.

2.3 Définir un mode de développement multiplateforme

Les avantages d'un mode de développement multiplateforme sont nombreux :

- un code est réalisé une seule fois
- il est facile à maintenir et à adapter sur une nouvelle plateforme
- les équipes spécialisées sur chaque plateforme sont réduites.

Dans notre étude nous avons deux plateformes aux architectures similaires (figure 2.14 24) mais avec des environnements de développement non compatibles (figure 2.15 ??).

Plateforme	Archos IT 5	iPhone 4
Processeur	ARM à 800 Mhz	ARM à 1 Ghz
Ecran	800x480	960x640

FIGURE 2.14 – L'environnement matériel Tablette Archos IT5 et iPhone 4

Plateforme	Archos IT 5	iPhone 4
Langage de programmation	Java	Objective C
IDE	Eclipse	Xcode
Interface graphique	XML	IB
Execute du code natif	Oui via JNI	Oui

FIGURE 2.15 – L'environnement logiciel Tablette Archos IT5 et iPhone 4

Le Java n'est pas disponible sur le iPhone. et réciproquement l'Objective C n'est pas disponible sous Android. Pour permettre un développement multiplateforme il faut donc trouver un autre langage qui soit exploitable sur les deux architectures.

L'Objective C permet d'utiliser des portions de code en C/C++ en intégrant ceux-ci dans un projet iPhone. L'appel du code se fait alors nativement, en respectant la syntaxe du C/C++.

De son côté Google en plus du SDK fourni un NDK (Native Development Kit) qui permet l'utilisation de code en C ou C++ de manière native. Pour l'utiliser il faut installer le NDK après l'avoir téléchargé⁶.

On pourrait donc développer tout, ou une partie, du code en langage natif C et appeler ce code selon les besoins. Il nous faut donc définir les méthodes pour intégrer du code natif sous Android et sous iPhone puis déterminer jusqu'où mutualiser le programme et extraire ainsi les parties propres à chacune des plateformes qui ne pourraient être factorisables.

6. <http://developer.android.com/sdk/ndk/index.html>

2.3.1 Intégration du C/C++ sous Android

L'utilisation du NDK n'étant pas complètement supportée par Eclipse, il nous faut repenser la chaîne de compilation du projet avec le NDK. Pour cela Google fournit des outils en ligne de commande comme alternative à l'IDE Eclipse. Nous allons décrire, sur le principe, l'utilisation des différents outils.

Création du projet Android

Nous allons aussi nous servir du SDK pour générer les fichiers nécessaires à un projet Android :

```
android create project --target <target_ID>
--name <your_project_name>
--path path/to/your/project
--activity <your_activity_name>
--package <your_package_namespace>.
```

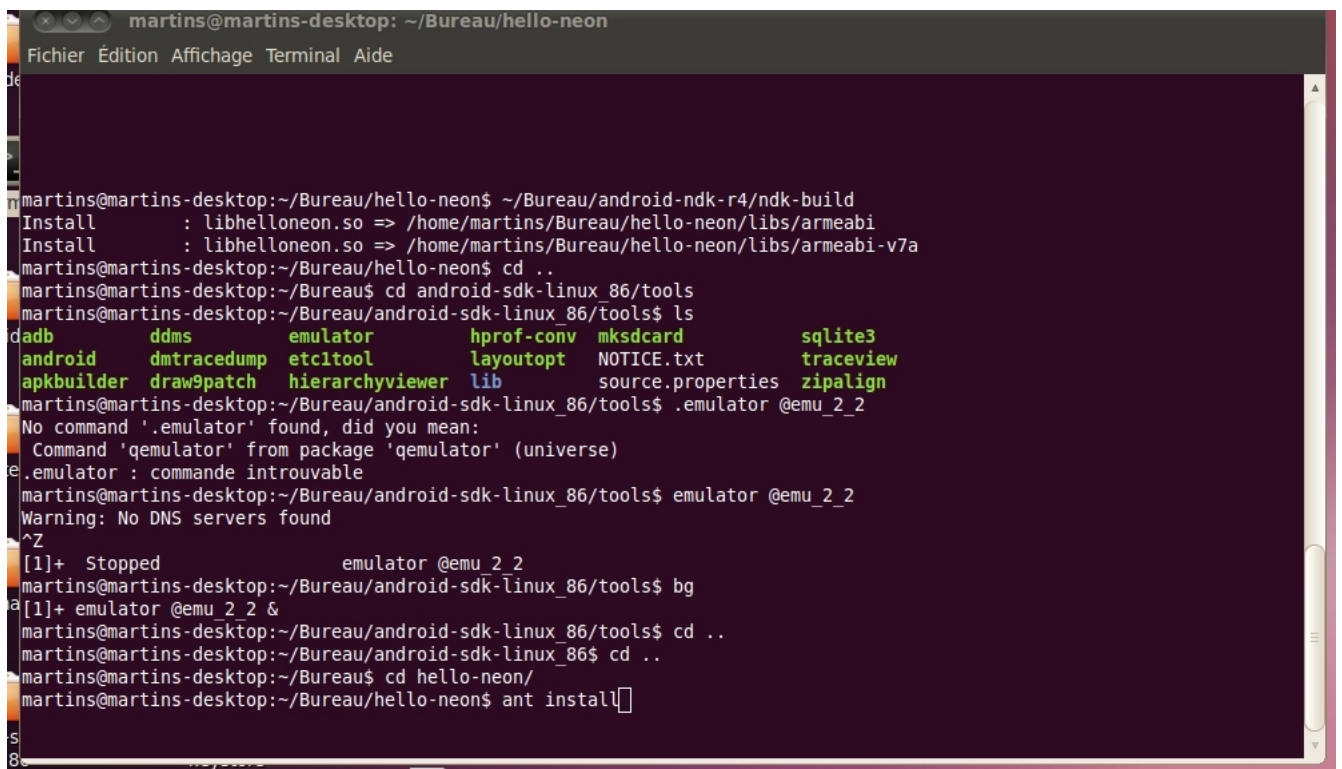
Construction du projet intégrant le code natif à partir de Java

Nous allons utiliser l'interface JNI (Java Native Interface) qui est un canal de communication utilisé par le Java pour s'interfacer avec du code en C. Nous allons utiliser cette méthode sous Android pour permettre à notre code Java de communiquer avec notre code en C natif. C'est le rôle du NDK.

Dans notre projet nous allons créer un dossier jni dans lequel nous allons placer notre code en C.

Avant de lancer une compilation classique du code Java, il nous faut d'abord compiler le code natif, qui sera ensuite intégré au code Java. Pour cela nous utilisons la commande `ndk-build` fourni par le NDK. Le code C sera alors compilé sous forme d'une bibliothèque dynamique en `.so` et l'interface de communication permettant d'appeler le code natif à partir de Java, appelée aussi Wrapper, va être créée. On la retrouve sous la forme d'un fichier `.h` inclu dans le dossier du projet.

Cette première passe réalisée, nous compilons le programme avec Ant installé. Le projet est alors créé après intégration du `.so` issu du code natif.



```
martins@martins-desktop: ~/Bureau/hello-neon
Fichier Édition Affichage Terminal Aide

martins@martins-desktop:~/Bureau/hello-neon$ ~/Bureau/android-ndk-r4/ndk-build
Install      : libhelloneon.so => /home/martins/Bureau/hello-neon/libs/armeabi
Install      : libhelloneon.so => /home/martins/Bureau/hello-neon/libs/armeabi-v7a
martins@martins-desktop:~/Bureau/hello-neon$ cd ..
martins@martins-desktop:~/Bureau$ cd android-sdk-linux_86/tools
martins@martins-desktop:~/Bureau/android-sdk-linux_86/tools$ ls
adb          ddms          emulator      hprof-conv    mksdcard      sqlite3
android      dmtracedump   etcdtool      layoutopt     NOTICE.txt   traceview
apkbuilder   draw9patch   hierarchyviewer lib            source.properties zipalign
martins@martins-desktop:~/Bureau/android-sdk-linux_86/tools$ .emulator @emu_2_2
No command '.emulator' found, did you mean:
  Command 'qemulator' from package 'qemulator' (universe)
.emulator : commande introuvable
martins@martins-desktop:~/Bureau/android-sdk-linux_86/tools$ emulator @emu_2_2
Warning: No DNS servers found
^Z
[1]+  Stopped                  emulator @emu_2_2
martins@martins-desktop:~/Bureau/android-sdk-linux_86/tools$ bg
[1]+  emulator @emu_2_2 &
martins@martins-desktop:~/Bureau/android-sdk-linux_86/tools$ cd ..
martins@martins-desktop:~/Bureau/android-sdk-linux_86$ cd ..
martins@martins-desktop:~/Bureau$ cd hello-neon/
martins@martins-desktop:~/Bureau/hello-neon$ ant install
```

FIGURE 2.16 – Etapes de compilation pour un projet JNI sous simulateur Android

Configuration et installation d'un simulateur

Afin de tester le code réalisé il nous faut créer une instance d'émulateur. Pour cela, on utilise la commande :

```
android create avd -n <name> -t <targetID> avec comme argument le nom et le numéro de l'émulateur.
```

Il faudra avant d'exécuter ce code lancer une session de cet émulateur avec la commande : `emulator @name`.

Nous venons de décrire l'intégration d'un code natif sous Android qui nécessite de prendre la main sur la procédure de compilation. Sous iPhone le code natif est plus simple à mettre en oeuvre.



FIGURE 2.17 – Une session d'émulateur Android sous Linux

2.3.2 Intégration du C/C++ sous iPhone

Pour intégrer du code natif dans un projet iPhone il suffit de prendre le fichier `.c` (ou `cpp` qu'il alors renommer en `.mm`) et le fichier `.h` et l'insérer dans le projet sous Xcode. Dans le fichier source Objective C (un fichier avec l'extension `.m`) il faut importer le `.h` du code natif :

```
// en objective C:  
#import "ma_fonction_nativeC.h"
```

Puis la fonction (ou objet) peut être alors appelée avec la syntaxe traditionnelle du C ou du C++.

```
// en objective C:  
ma_fonction_nativeC(a,b);
```

On constate que l'intégration du code natif est transparente pour l'utilisateur sous iPhone.

2.3.3 Éléments d'un projet pouvant être factorisés dans le cadre d'un développement multiplateforme

Dans le cadre du MVC le modèle est, comme nous l'avons vu précédemment, transférable. Mais la vue et son contrôleur restent typiques à la plateforme où ils s'exécutent. En effet la vue est constituée par un fichier XML sous Android et Xib sous iPhone. Il n'y a pas de concordance entre les deux, ni de conversion possible.

Le fichier XML ne comporte que les éléments constituant la vue et leurs placements dans celle-ci. Il attribue un index à chaque composante pour qu'elle soit ensuite utilisable dans le code proprement dit. Le fichier Xib, issu de l'utilisation d'IB, contient lui aussi les objets graphiques, leurs styles et leurs emplacements. Mais il intègre en plus le lien avec la vue dans lequel ils doivent être utilisés. Ce fichier doit être inclus dans le code via une déclaration de chacune de ses variables (boutons, champ texte etc.) dans un header avec un format particulier (IBOutlet). Ce n'est pas le cas pour Android. Un élément graphique doit être simplement déclaré dans la classe où il va être utilisé.

La gestion du comportement de ces objets est aussi différente d'une plateforme à l'autre. Pour le iPhone chaque élément graphique est autonome dans sa gestion d'événement. Il déclenche lui-même les appels aux fonctions grâce à un selector (son écouteur d'événement).

```
// Gestion d'un bouton sous iPhone
[monbouton addTarget:self action:@selector(buttonPressedmonbouton)

-(void)buttonPressedmonbouton{
[listeChiffre appendString:@"%3"];
[listeTouche appendString:@"%3"];
[monLabel setText: listeTouche];
}
```

Sous Android cette gestion se fait au travers d'un Listener (écouteur d'événement) qui lance une fonction void onClick(View v) qui prend comme attribut la vue actuelle. Dans cette fonction nous allons ensuite faire un travail d'extraction pour connaître l'élément graphique déclencheur de l'événement pour lui attribuer enfin une action.

```
// Déclaration du bouton
Button casequit = (Button)findViewById(R.id.tquit);
casequit.setOnClickListener(this);
//Fonction gérant les événements
public void onClick(View v) {
if (v==(Button)findViewById(R.id.tquit)) finish();
}
```

L'interface graphique est un élément propre à chacune des plateformes. Elle semble donc peu ou pas transférable. Le reste du code, calcul, manipulation de données, semble pouvoir être factorisé. Nous venons donc d'identifier, dans le développement d'une application informatique pour plateforme mobile, les éléments factorisables et donc réutilisables, et ceux qui seront spécifiquement développés pour la machine. La vue et son contrôleur représente 20% du code, le modèle les 80% restant, nous pouvons donc optimiser les 4/5 du développement. Pour valider l'ensemble de nos choix nous avons développé un projet complet en utilisant le code natif.

Nous allons maintenant appliquer la factorisation du code au code métier d'une application.

2.3.4 Factorisation du code métier entre l'Android et le iPhone

Lors de notre étude nous avons constaté que le travail de factorisation du code est différent selon les données manipulées.

Travailler en code natif avec des types primitifs

Lorsque l'on appelle une fonction en C à partir de l'Objective C on peut passer des arguments de types primitifs sans ajout de code :

```
// en objective C:
int i;
int a=2;
i=fonctionenC(a) // Appel de la fonction C
```

```
// du coté natif C:
int fonctionenC(a){
    return a++;
}
```

Ce code déclare une variable de type entier a, on lui affecte la valeur 2. On exécute la fonction C qui retourne au programme en Objective C la valeur de a incrémentée de 1.

Si l'on souhaite intégrer cette même fonction sous Android la transmission de données du Java vers le C est plus complexe :

```
// en Java sous Android dans la classe
// déclaration de la fonction en C de type natif
public native int fonctionenC(int a);
// chargement de la fonction C en mémoire pour appel
static{ System.LoadLibrary("fonctionenC");}
// appel de la fonction
int a=2;
int i=fonctionenC(a) // Appel de la fonction C
```

```
// du coté natif C:
jint nompacagejava_fonctionenC(JNIEnv* env,jobject this ,jint a){
    return a++;
}
```

Au sein du code Java nous mettons en place une méthode native sous le protocole JNI. Elle est déclarée comme native et chargée en mémoire pour être appelée. Cette méthode est déclarée *static* car elle dépend de la classe (et non pas de l'instance de l'objet). Une fois déclarée on peut l'appeler comme toute méthode de type Java.

Dans le cadre du code natif C nous devons interfacer le passage des données. Le nom de la méthode change, car elle va être appelée en mode natif par le code Java et doit donc appartenir à son package. Ensuite elle reçoit un environnement de type JNI et un objet de type Java. On se servira plus tard de ce type JNIEnv pour des données plus complexes. Ce pointeur nous renseigne sur l'adresse où va être stocké la série de variables type string, array. Le type int étant primitif il est passé directement par la fonction. L'objet java va assurer la capacité de la fonction native à retourner des données.

Comme nous le constatons le passage de type primitif est plutôt simple. Le passage d'une donnée de type string (chaîne de caractères) ou tableau est bien plus complexe à réaliser.

Problématiques liées aux manipulations de chaîne et tableau de données

Lors de la réception des deux strings dans la méthode native on constate un traitement particulier à cause du JNI mais aussi des limites du NDK Android. Celui-ci intègre la librairie string du C avec des modifications. Le type string n'existe pas, mais l'ensemble des méthodes de la classe string sont disponibles. Ces méthodes fonctionnent à partir d'un `const char *` et le manipulent comme une string. Il nous faut donc à la réception des données issues du code Java les convertir en `const char *`. On s'appuie alors sur le pointeur env pour connaître la zone mémoire où se trouve le contenu des strings. Il nous faut faire la même manipulation pour un return de type string. Voici un exemple d'utilisation :

```
// Sous Android méthode native avec 2 strings
public native float affiche
    (String listedeschiffres,String listedesoperandes);
static { System.loadLibrary("Jnicalc");}
```

```
// du côté natif C:
jfloat packagejava_affiche( JNIEnv* env, jobject this, jstring listedeschiffres_java ,
    jstring listedesoperandes_java){
// On retrouve les 2 Strings mais on doit les traiter
// contrairement au int précédent
const char * chif = (*env)->GetStringUTFChars(env, listedeschiffres_java ,NULL);
const char * oper = (*env)->GetStringUTFChars(env, listedesoperandes_java ,NULL);
}
```

Du côté de l'Objective C nous sommes confrontés au même type de problèmes pour gérer un objet de type string. L'objet NSString remplace le type string natif. Il possède une terminaison différente et bénéficie de très nombreuses méthodes de manipulation et traitement. Nous utiliserons la méthode retournant un string de type UTF8 pour le convertir et le rendre utilisable dans la méthode native.

```
// Appel en Objective C
NSString * listeOperande = [NSString Alloc] ;
[listeOperande initWithString:@"Hello world"];
fonctionenC([listeOperande UTF8String]);
```

```
// Dans la méthode native
void moteur(const char * listeOperande);
```

Comme nous venons de le constater plus les éléments à interfacier sont complexes et plus le traitement est difficile. La méthode étant définie, nous allons maintenant la mettre en pratique avec le développement de projets multiplateformes.

Chapitre 3

Développement d'applications multiplateformes Android/iPhone

Nous avons développé plusieurs applications avec des objectifs différents. Nous avons réalisé des applications permettant de tester les capacités de chacune des machines en terme d'affichage en 2D et en 3D. Nous avons ensuite programmé une calculatrice sur les 2 supports en intégrant un moteur de calcul en C. Nous avons également tenté d'utiliser un jeu d'instructions spécifiques au processeur ARM équipant nos deux appareils. Pour chacun de ces programmes nous avons pu valider des niveaux de performances et des méthodes de développement.

3.1 Etudes des performances en 2D et 3D de l'Archos IT 5 et de l'iPhone

Les deux plateformes disposant du même processeur (ARM) à des cadences légèrement différentes, nous avons donc souhaité comparer leurs performances en terme d'affichage graphique et de calcul 3D en temps réel. De plus, nous développerons ces applications en utilisant une méthode classique, c'est-à-dire pour chaque plateforme son propre code, pour évaluer les temps de travail.

3.1.1 Analyse

Nous souhaitons tester la capacité de chacun d'entre eux à afficher des sprites (un élément graphique qui peut se déplacer à l'écran) en 2D, en animation constante, avec un rafraichissement du fond à chaque tour d'animation. Cette méthode allait nous permettre de déterminer la vitesse d'affichage et les temps de réaction.

Les deux plateformes intègrent la bibliothèque de programmation graphique OpenGL ES spécifiquement créée pour les machines portables. Similaires en terme de matériel, les deux appareils devaient nous présenter des résultats proches en terme de rendu 3D. Nous souhaitons ainsi vérifier l'apport des différents OS sur les performances.

Pour réaliser ces tests nous avons utilisé la méthode traditionnelle de développement. L'ensemble du code a été systématiquement réécrit pour chaque plateforme, sans utilisation de code natif. Deux raisons à cela :

- ces applications sont graphiques et utilisent essentiellement des vues et un contrôleur de vue
- nous voulions obtenir des résultats dans un cadre standard d'utilisation.

3.1.2 Réalisation

Nous avons mis au point un programme qui affiche des sprites de 88X82 pixels selon une courbe pour tester les capacités des appareils mobiles à notre disposition. Lorsque l'utilisateur appuie sur l'écran un sprite supplémentaire est ajouté à la courbe, le compteur est incrémenté et le nombre de frames (temps d'exécution) nécessaires à l'exécution de l'ensemble de l'animation est recalculé et affiché (figure 3.1 page 31).

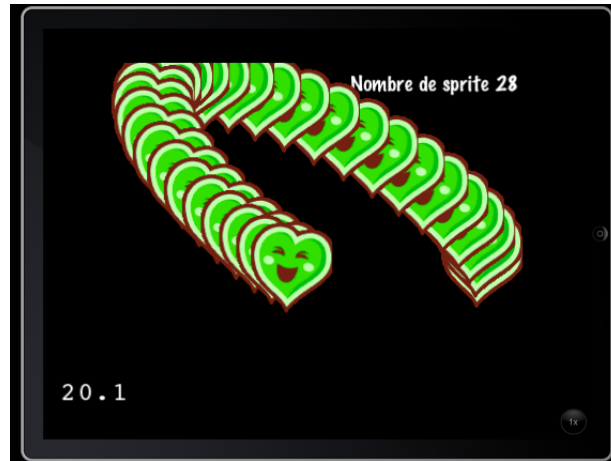


FIGURE 3.1 – Test de performance d’affichage de sprite 88X82 sous simulateur iPad

Nous avons aussi réalisé une application utilisant l’OpenGL ES pour tester les capacités de rendu de ces plateformes portables. Nous avons testé le calcul et l’affichage de 3200 faces 3D en temps réel avec une rotation sur les axes et un total de 4800 vertex modifiés à chaque nouvel affichage (figure 3.2 page 31).

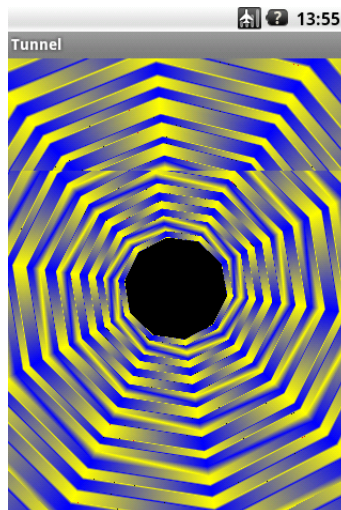


FIGURE 3.2 – Test de performance tunnel en 3D

3.1.3 Résultats obtenus

Nous n'avons pas pu tester l'application de rendu de sprites en 2D sur une plateforme Apple car nous ne disposons pas de licence permettant d'installer l'application sur iPhone. Nous avons donc comparé l'Archos avec une autre plateforme Android ayant le même processeur que le téléphone d'Apple. Dans le cadre d'un rendu en 2D les capacités des machines sont les suivantes.

Plateforme	Archos IT 5	HTC Desire
Processeur	ARM à 800 Mhz	ARM à 1 Ghz
Ecran	800x480	800x480
Nombre de sprites	72	130

FIGURE 3.3 – Performances réalisées par plateforme - Sprites 88X82 - 25 FPS

Dans le cadre d'un rendu en 3D nous obtenons ces résultats. Nous n'avons pas développé cette application sous iPhone (le même problème d'installation que le précédent test), aussi nous avons fait le choix de comparer deux plateformes Android ayant une différence en terme de puissance de processeur assez proche de celle de l'iPhone.

Plateforme	Archos IT 5	HTC Desire
Processeur	ARM à 800 Mhz	ARM à 1 Ghz
Puce graphique	SGX535	
Ecran	800x480	800x480
Framerate/seconde	25	60

FIGURE 3.4 – Performances réalisées par plateforme - Tunnel 3D - 3200 faces

Les différences sont très importantes entre les deux machines. Cela s'explique par les faits suivants :

- le HTC Desire bénéficie d'un processeur fonctionnant à une fréquence plus élevée
- les deux plateformes n'ont pas la même version d'OS : l'Archos IT 5 possède un Android 1.6 et le HTC Desire un Android 2.1

En effet les 200 Mhz de différence entre les deux processeurs ne peuvent expliquer à eux seuls le doublement du framerate ou du nombre de sprites affichables. En faisant évoluer son OS, Google le rend plus rapide dans l'exécution des tâches.

En terme de temps de développement les chiffres sont les suivants :

Plateforme	Android	Simulateur iPhone
Temps passé	16 heures	14 heures

FIGURE 3.5 – Répartition du temps de développement sur le projet Sprites

Nous avons ensuite développé un projet entier en suivant une méthode de développement multi-plateforme incluant du code natif.

3.2 Un projet multiplateforme avec code natif : une calculatrice

Le choix de ce projet s'explique par :

- une interface graphique utilisant plusieurs objets différents
- des interactions multiples
- un moteur de calcul bien identifié.

3.2.1 Analyse et méthodologie appliquée

Voici le schéma de développement de projet que nous avons mis en place (figure 3.6 page 33).

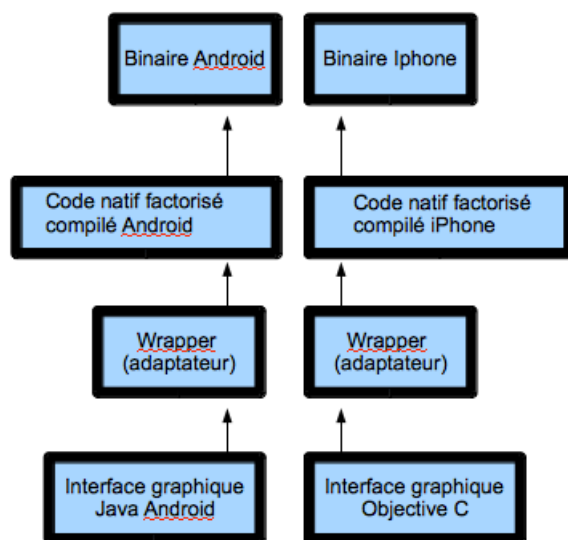


FIGURE 3.6 – Génération d'un programme avec code natif et wrapper

Nous avons développé les interfaces graphiques et la gestion des choix de l'utilisateur sur chacune des plateformes, dans leur langage, en XML et en Java pour Android, en Objective C sous iPhone. Nous avons ensuite réalisé 2 wrappers, du Java vers le C et de l'Objective C vers le C. Enfin le moteur de calcul a été développé en C, sur PC, et réutilisé sur les deux plateformes.

3.2.2 Réalisation du projet

Le moteur de calcul de la calculatrice est une fonction qui prend en paramètre deux strings. La première contient la liste des nombres insérés par l'utilisateur séparés par un caractère « / », la seconde contient les opérandes à associer à chaque étape du calcul. Volontairement le moteur ne gère pas les priorités des opérations. Il réalise des calculs sur les 4 opérateurs de base, le plus, le moins, le diviser et le multiplier.

A la réception des deux strings, il extrait de la chaîne de caractères des nombres les premiers éléments jusqu'à rencontrer le caractère séparateur. Il prend alors la première opérande dans la chaîne des opérandes et effectue l'opération. Le résultat est stocké dans une variable de type float. On passe ensuite au nombre suivant jusqu'au prochain séparateur, puis la deuxième opérande, et ainsi de suite jusqu'à ce que l'on atteigne la fin de la chaîne des nombres.

```

\\ Code source du moteur en C
#include <string.h>
#include <stdio.h>

int main(){
\\ Deux strings pour simuler l'appel
    const char * chif = "24/2.5";
    const char * oper = "++";
\\ Calcul des longueurs des strings
    int longueurchiffre;
    int longeuropere;
    longueurchiffre=strlen(chif);
    longeuropere=strlen(oper);
\\ Création des tableaux de char de la taille
\\ des strings
    char chiffre[longueurchiffre];
    char operande[longeuropere];
\\ Copie des strings dans le tableau de char
\\ car on prévoit le traitement des strings
\\ venant de format différents vers un
\\ format plus accessible au C
    strcpy(chiffre ,chif);
    strcpy(operande ,oper);

\\
\\ Le reste du code traite le calcul et le stocke le résultat
    .....
}

```

Ce code (version complète en annexe B page 46) fonctionne sous Linux et Windows. Nous allons maintenant le rendre utilisable par un appel en Java sous Android.

```

//
// Code java de la calculatrice sous Android
//
public class Jnicalc extends Activity implements OnClickListener{
    String frappe="";
    String resultat="";
    TextView ecran;
    String listedeschiffres="";
    String listedesoperandes="+";
// Déclaration de la fonction C en natif
// Chargement de la fonction native
    public native float affiche(String listedeschiffres ,String listedesoperandes);
    static {
        System.loadLibrary("Jnicalc");
    }
    ...
    ...
}

```

```
//
// En tête de la fonction en C
// Utilisation du JNI pour le transfert des données et l'appel
//
jfloat Java_com_chuck_android_Jnicalc_Jnicalc_affiche( JNIEnv* env, jobject
    thiz, jstring listedeschiffres_java, jstring listedesoperandes_java){
    const char * chif = (*env)->GetStringUTFChars(env, listedeschiffres_java, NULL);
    const char * oper = (*env)->GetStringUTFChars(env, listedesoperandes_java, NULL);
    int longueurchiffre;
    int longueuroperande;
    longueurchiffre=strlen(chif);
    longueuroperande=strlen(oper);
    char chiffre[longueurchiffre];
    char operande[longueuroperande];
// Après traitement des string au format Java
// elles deviennent des const char * le reste
// de la fonction ne change plus ...
    strcpy(chiffre, chif);
    strcpy(operande, oper);
    ...
    ...
// Variable de type float
// les primitives en C se gèrent sans modification
// en JNI
    return resfinal;
}
```

Dans le projet java il faut compiler, à la racine du répertoire :

- avec la commande ndk-build (pour le moteur en C)
- puis avec la commande ant release
- un fichier app est ainsi créé puis installé dans la tablette Archos.



FIGURE 3.7 – Calculette avec moteur natif sous Archos IT 5

Maintenant présentons la version pour l'iPhone.

```
//  
// Code en Objective C  
//  
// Importation du header du moteur natif  
#import "moteur.h"  
...  
...  
// Appel de la fonction native  
// Avec récupération du résultat  
[monLabel setText: [NSString stringWithFormat:@"%f", moteur([listeChiffre  
    UTF8String],[listeOperande UTF8String])]];
```

Quelques explications sont nécessaires. L'appel à la fonction native se fait en lui transférant deux strings. Ces strings sont à l'origine des NSString. Elles sont converties par le biais d'une fonction ([nomstring UTF8String]) au format natif C. On obtient ainsi une nouvelle string que l'on passe à la fonction native.

```
// Dans la fonction C  
double moteur(const char * chif,const char * oper){  
    int longueurchiffre;  
    int longueuroperande;  
    longueurchiffre=strlen(chif);  
    longueuroperande=strlen(oper);  
    char chiffre[longueurchiffre];  
    char operande[longueuroperande];  
    strcpy(chiffre,chif);  
    strcpy(operande,oper);  
    ...  
    ..  
    return resfinal  
}
```

L'entête est modifié pour récupérer les strings maintenant au format const char *. On les intègre dans les tableaux de char et le reste de la fonction native n'est pas modifié.



FIGURE 3.8 – Calculatrice avec moteur natif sous simulateur IPAD

3.2.3 Retours d'expériences

Les avantages de cette méthode de développement sont multiples. En terme de temps de travail, la décomposition du projet calculette est de :

- 12 heures de code pour l'interface graphique sous Android et 10 heures sous Objectif C
- 4 heures pour le moteur en C
- 4 heures pour les 2 wrappers

Soit un total de 30 heures.

En comparaison, pour un développement spécifique à Android et tout en Java, le temps de travail nécessaire à la réalisation de la calculette intégrant l'interface graphique et le moteur est de 18 heures.

De plus ce moteur natif fonctionne sur 5 plateformes : iPhone, Android, PC Linux, PC Windows, Mac. Une adaptation du code sous PC Linux avec du Java prendrait moins de 4 heures. Le facteur temps passé pour une adaptation sur une autre plateforme est de 30% au lieu de 60% pour une refonte complète du code. Ces chiffres ne tiennent pas compte de la gestion des erreurs qu'il faut corriger sur autant de programmes que de plateformes, alors que le moteur n'est corrigé, testé qu'une seule fois.

En terme de ressources humaines cette méthode de développement nécessite :

- une équipe spécialisée dans le code natif pour faire les moteurs
- une personne spécialisée dans chaque plateforme pour les interfaces graphiques.

Cependant nous avons encore des contraintes en terme d'utilisation du code natif. En effet celui disponible grâce au NDK sous Android est pour le moment limité. Des projets open-source travaillent actuellement à la mise en place de la STL et autres bibliothèques usuelles du C et du C++. En attendant il faut se limiter aux possibilités offertes par le NDK qui, comme nous avons pu le voir, fait des choix quand aux primitives disponibles et à leurs utilisations (les strings par exemple). Avant de se lancer dans le développement tout azimuth il convient de lire les .h disponibles. Nous avons passé, par exemple, un après-midi à essayer de comprendre une erreur en utilisant les strings dans la fonction native. En relisant le header de la bibliothèque string disponible sous NDK nous nous sommes aperçu que le type string n'était pas disponible mais que les méthodes de manipulations étaient définies au travers de const char *. L'utilisateur de code natif pourra toujours rajouter ses propres bibliothèques et définir un type string grâce à un #typedef.

3.3 Utilisation des instructions NEON

3.3.1 L'unité vectorielle NEON

Le NEON est une unité de calcul présente dans les dernières versions des processeurs ARM actuellement disponibles dans les smartphones et tablettes. Cette unité, dite parallèle, vise à accélérer les calculs arithmétiques et logiques en les effectuant non pas sur une donnée à la fois comme dans les unités de calculs standards des processeurs, dites scalaires, mais sur quatre données simultanément. En effet les calculs entrant en jeu dans le traitement multimédia ou dans les calculs scientifiques consistent majoritairement à effectuer les mêmes opérations sur des données différentes. Des unités parallèles similaires nommées SSE (Streaming SIMD Extensions) sont également présentes dans les processeurs Intel et AMD du commerce. Des tests montrent que l'utilisation de telles unités permet d'au moins doubler les performances et dans certains cas de décupler les performances. Cependant la programmation de l'unité NEON nécessite des connaissances approfondies en architecture des processeurs ainsi qu'une programmation de bas niveau afin d'obtenir de tels gains de performance. Nous allons maintenant montrer comment exploiter l'unité NEON à partir de la plateforme Android et iPhone.

3.3.2 Exemple de mise en oeuvre sur les deux plateformes

Nous avons utilisé l'exemple fourni dans le SDK Android : Helloneon. Celui-ci utilise du code natif, via un wrapper. Ce code en C utilise le fichier `arm_neon.h`. Il réalise dans un premier temps un calcul en C sans utilisation du Néon, puis avec. Il calcule le temps passé et affiche le résultat. L'ensemble du code est disponible dans le répertoire `sample` dans le dossier `android-sdk-linux`.

Sur iPhone nous avons utilisé un calcul de matrice entièrement réalisé en assembleur pour le Néon. Il utilise le même fichier `arm_neon.h`. La fonction est appelée en code natif et renvoie le résultat sous forme d'un float.

```
// Code utilisé sous iPhone
// Extrait du blog de Jeff Wolfgang's
// https://www.blogger.com/comment.g?blogID=398682525365778708
void NEON_Matrix4Mul(const float* a, const float* b, float* output )
{
    __asm volatile
    (
        // Store A & B leaving room at top of registers for result (q0-q3)
        "vldmia %1, { q4-q7 } \n\t"
        "vldmia %2, { q8-q11 } \n\t"

        // result = first column of B x first row of A
        "vmul.f32 q0, q8, d8[0]\n\t"
        "vmul.f32 q1, q8, d10[0]\n\t"
        "vmul.f32 q2, q8, d12[0]\n\t"
        "vmul.f32 q3, q8, d14[0]\n\t"

        // result += second column of B x second row of A
        "vmla.f32 q0, q9, d8[1]\n\t"
        "vmla.f32 q1, q9, d10[1]\n\t"
        "vmla.f32 q2, q9, d12[1]\n\t"
        "vmla.f32 q3, q9, d14[1]\n\t"

        // result += third column of B x third row of A
        "vmla.f32 q0, q10, d9[0]\n\t"
        "vmla.f32 q1, q10, d11[0]\n\t"
        "vmla.f32 q2, q10, d13[0]\n\t"
        "vmla.f32 q3, q10, d15[0]\n\t"

        // result += last column of B x last row of A
        "vmla.f32 q0, q11, d9[1]\n\t"
        "vmla.f32 q1, q11, d11[1]\n\t"
        "vmla.f32 q2, q11, d13[1]\n\t"
        "vmla.f32 q3, q11, d15[1]\n\t"

        // output = result registers
        "vstmia %0, { q0-q3 }"
        : // no output
        : "r" (output), "r" (a), "r" (b) // input - note *value* of pointer
        : "memory", "q0", "q1", "q2", "q3", "q4", "q5", "q6", "q7", "q8", "q9", "q11"
        : //clobber
    );
}
```

3.3.3 Les problèmes et limitations rencontrés

Les résultats sont inexistantes sur cette partie du projet. En effet nous n'avons pas pu faire fonctionner le moindre code sous Néon que ce soit sur Android ou iPhone. Cette unité de calcul est pourtant bien présente sur les deux plateformes en notre possession mais malgré plusieurs essais nous n'avons pu utiliser les exemples fournis par les 2 constructeurs. Nous ne pouvons donc pas afficher de performances.

Cependant cet exemple est significatif de la difficulté de faire un code multiplateforme qui soit facilement abordable. Malgré de nombreuses recherches nous n'avons pas pu régler les problèmes de compilation que nous affiche l'Archos IT5 (program not compiled with ARMv7 support) ou encore le HTC. Nous ne pouvons donc que nous fier aux résultats trouvés sur Internet ou donnés par le constructeur. L'utilisation du Néon permettrait une multiplication par sept des performances.

Plateforme	Archos IT5	Simulateur iPhone
Temps passé	16 heures	24 heures
Résultat	Compilation et exécution	Pas de compilation possible en mode simulateur
Bilan	Pas de reconnaissance du Néon	Obligation de signer l'application

FIGURE 3.9 – Le Néon sous Android et iPhone

3.4 Méthode de développement multiplateforme : un premier bilan

Au cours de la construction du code de test pour l'affichage 2D et 3D nous avons pour le calcul des courbes dupliqué le code d'une plateforme à l'autre en l'adaptant. En effet les résultats de calculs ne pouvaient être stockés dans le même type d'emplacement sous Android ou en Objective C, dans un canvas pour la première plateforme, dans une structure pour la seconde. Nous avons alors pensé qu'il devait être possible d'exporter la partie calcul d'une application vers du code natif.

En réalisant la calculatrice, nous avons donc codé le moteur de calcul en premier et en code natif. Une fois que celui-ci fonctionnait nous avons programmé les interfaces graphiques en adaptant le passage des informations d'un code vers l'autre. C'est donc un autre mode de développement qui nécessite un autre type d'analyse dans un projet. Qu'est ce qui est factorisable ? Les plateformes peuvent-elles utiliser du code natif ? Ces questions doivent être posées, et les réponses techniques apportées avant de se lancer dans ce type de programmation. Une période d'analyse est donc indispensable.

En effet, comme nous l'avons vu, tout ne peut pas être pensé en multiplateforme pour des multiples raisons :

- pas les mêmes interactions d'une plateforme à l'autre
- pas les mêmes méthodes pour l'affichage.

Donc dans le cahier des charges, les développeurs doivent extraire les codes factorisables, les développer en intégrant les contraintes de passage des données. C'est pour cela que, dans le cadre du MVC, le modèle semble tout à fait natif, de même on peut aussi facilement imaginer appeler une base de données dans les mêmes conditions avec le passage des champs via JNI sous Android, ou encore pour le LIFO réaliser une répartition des calculs sur des plateformes complètement différentes, via du code natif, pour profiter de leurs spécificités, et retourner les résultats.

Néanmoins comme ce type de développement n'est pas habituel, il semble complexe à mettre en oeuvre, notamment au niveau du transfert de données. Il nécessite une très bonne pratique du C (ou du C++) et une bonne connaissance des machines en terme de structuration des variables.

Au final, cette méthode de réalisation de projet informatique fonctionne. Elle permet, et sans contestation possible, d'optimiser les temps de développement en évitant la redondance des codes, la gestion de correction des bugs. Elle permet aussi de faire cohabiter des technologies différentes, on peut sans peine imaginer un iPhone qui envoie, au travers d'un code natif des signaux à une plateforme sous Android.

La prochaine étape pourrait alors être de créer une chaîne d'outils avec un makefile générique qui selon une option de compilation créerait du code pour une plateforme, avec son wrapper, et son code natif.

Chapitre 4

Des objectifs aux réalisations : retours d'expériences et bilan

Après 10 semaines de travail au sein du LIFO dans un nouveau domaine professionnel le bilan en terme d'acquis professionnels et personnels est positif.

4.1 L'apprentissage technique et professionnel : 10 semaines de formation intensive

La formation acquise à l'IUT visait à me rendre capable de programmer avec tous types de langage ou outils de développement. Les fondamentaux qui m'ont été enseignés devaient me permettre d'appréhender un nouveau langage, de comprendre sa structuration, sa syntaxe. En fin d'année j'avais suivi des cours en Java sur une période, malheureusement, assez courte. C'est ce langage que j'ai retrouvé sur la plateforme Android. J'ai ainsi pu acquérir davantage de pratique sur des projets longs écrits dans ce langage. En terme d'adaptation l'usage de la documentation a été déterminant et m'a permis de (re)trouver les fonctions nécessaires à la réalisation des différents projets.

Le développement sur les plateformes mobiles s'appuie sur le MVC. Ce concept m'était parfaitement inconnu. Les tutoriels présents dans les différents SDK en plus d'être faciles d'approche, car peu techniques, sont aussi très documentés. Après de nombreuses recherches (notamment sur Internet) et la lecture de nombreux documents en anglais comme en français, j'ai aujourd'hui une bonne compréhension du rôle de chacune de ses composantes. Mes connaissances actuelles dans ce domaine me permettent de créer des scénarios dans lesquels chaque vue est gérée comme un objet à un moment donné, de passer de l'une à l'autre etc.

Dans le cadre de mes recherches sur Android j'ai appris à construire une chaîne d'outils complète, de l'écriture du code jusqu'à sa compilation et son installation sur un simulateur ou un appareil mobile. J'ai rencontré de nombreuses difficultés pour trouver des solutions à chacune des étapes. Les recherches en ligne, une connaissance des systèmes (acquise en cours) et des processus de compilation ont permis de finaliser l'ensemble de la procédure de création de l'application. Le tout étant réalisé sous Linux j'ai dû compléter ma pratique de cet OS que je n'utilisais pas en dehors des travaux dirigés à l'IUT.

L'utilisation du JNI a été une période assez complexe en terme technique. Ce concept de passage des informations, au travers d'un wrapper, est peu documentée, toujours en anglais, et les exemples fonctionnants sont rares. Du point de vue théorique on retrouve une explication dans la documentation Java pour une utilisation simple. Le site de Roger Astier¹ de l'IUT d'Orsay, est la seule ressource française que j'ai pu trouver dans laquelle est décrit le passage de données du Java vers du code natif (C ou C++). C'est sur la base de ses exemples que j'ai pu mettre en place le projet de la calculatrice sous Android.

J'ai aussi acquis de nouvelles connaissances en Open GL ES. Cette version d'Open Gl est spécifique au portable. J'ai ainsi pu mettre en oeuvre un tunnel 3d qui est un des codes que j'utilise pour appréhender une machinerie 3D d'une plateforme à une autre.

En travaillant sur iPhone j'ai appris à utiliser MAC OS et surtout l'Objective C. Ce langage est très particulier en terme de syntaxe, et sa compréhension aux premiers abords est vraiment complexe. Après immersion dans le code du projet Sprite et ensuite de la calculette je suis maintenant assez à l'aise avec les concepts particuliers comme la gestion de la mémoire (retain et release) ou encore le passage de messages au receveur. La documentation que j'ai utilisée sur ce point est uniquement en anglais. Pour programmer dans ce langage il m'a fallu aussi me servir de l'IDE fourni par Apple nommé Xcode. Comme il est assez proche d'Eclipse, que nous avons vu à l'IUT, sa prise en main fut aisée.

J'ai aussi découvert tout l'univers du iPhone, ses frameworks, le projet Cocos 2D (une bibliothèque pour faire des jeux), la gestion des interactions avec l'utilisateur etc. J'ai depuis peu commencé à développer des projets personnels sur cette plateforme.

Enfin, j'ai appris à utiliser tout l'environnement de Latex et Beamer. Ce rapport de stage a d'ailleurs entièrement été conçu avec Latex et sa distribution sous Mac TexShop. Très différent à l'usage, par rapport à Word ou Open Office, il m'a permis d'aborder la rédaction avec une autre méthode. L'écriture du texte s'est faite sans aucune présentation. Celle-ci a été appliquée ensuite, en respectant des standards de parution comme dans les rapports ou les thèses. L'intégration des images a été unifiée (un seul style pour toutes les images) comme pour les extraits de code, ou des tableaux. Son apprentissage n'est pas simple, il est même frustrant car il faut un temps certain avant de réussir à faire un document présentable. Mais avec une certaine pratique il s'avère très rationnel et efficace.

Du point de vue technique ces 10 semaines ont été riches en terme d'apprentissage. Mais l'enjeu du stage c'est aussi d'apprendre sur soi-même.

4.2 Apprendre à se connaître : un bilan personnel encourageant

Lors de ma réorientation professionnelle il y a deux ans je me suis appuyé sur l'une de mes passions pour la transformer en métier. L'informatique que je pratiquais était celle que j'avais appris en autodidacte. La formation à l'IUT m'ayant permis de me mettre à niveau j'étais pressé de mettre en application ce que j'avais appris pour valider ma capacité à être productif, à apporter un plus à un projet. J'ai pu finaliser les projets demandés par mon tuteur dans les délais que nous avons définis. Tous sauf un, je n'ai pas réussi à tester les capacités de l'unité de calcul appelé NEON présente sur Android et iPhone. Malgré de nombreux essais, la lecture de plusieurs documentations, la prise de contact avec des programmeurs ayant réalisé des programmes sur ce type de matériel, je n'ai jamais réussi à faire fonctionner me moindre code : problème de compilation, non-exécution ou refus d'installation. Je me pose alors la question de la réaction d'un employeur face à ce genre de situation et de ma capacité à réagir. Je n'ai pas de réponse actuellement.

1. http://www.iut-orsay.fr/dptinfo/Pedagogie/Roger.Astier/app_java/javaJni.html

J'avais une appréhension quand au fait de m'adapter à des nouvelles plateformes, de nouveaux langages. J'avais besoin de savoir si je pouvais apprendre vite, dans des délais déterminés. En 10 semaines j'ai pu travailler sur deux plateformes différentes que je ne connaissais pas, programmer en Objective C que j'ai complètement appris sur place avec de la documentation, des tests et les méthodes qui m'avaient été enseigné à l'IUT. Je programme maintenant en Objective C, sous iPhone, un jeu que j'espère mettre en vente sur l'Apple Store en décembre.

Je souhaitais aussi valider ma capacité à travailler toute la journée devant un clavier, à réfléchir en terme d'algorithme, de code. J'avais besoin de vivre des journées de travail au sein d'une équipe, dans un espace ouvert, pour tester ma concentration et mon aptitude à réfléchir et trouver des solutions. Sur ces points il m'a fallu rapidement trouver un moyen de m'enfermer pour fixer ma réflexion. J'ai donc apporté un casque et un lecteur mp3 que j'ai pu utiliser quand je ne me sentais plus capable de rester indifférent à ce qui se passait autour de moi. Je dois encore travailler ce point.

J'ai retrouvé, en travaillant sur des plateformes mobiles, le même goût du challenge qui me fait aimer la programmation. En effet du point de vue matériel ces appareils sont figés, il n'y a pas de moyens techniques pour rendre un Archos IT 5 plus rapide. Le programmeur doit donc optimiser son code pour qu'il puisse faire faire à la tablette le plus de choses possibles en un minimum de temps afin de ne pas user l'utilisateur. Le choix de continuer à développer sur ces architectures particulières est donc quasi obligatoire à mes yeux.

En travaillant au LIFO j'ai pu côtoyer des chercheurs, voir leur travail. J'ai validé le choix de mon Master IRAD à cette occasion. Je pense aussi, si je valide l'ensemble de mon diplôme, postuler à une thèse dans le domaine de la recherche informatique en continuant le travail que j'ai commencé dans les nouveaux modes de développement informatique.

4.3 Savoir conclure 10 semaines de stage

Le stage est une période intense en terme d'apprentissage et de travail. Malgré un investissement important il semble difficile de profiter de l'ensemble des connaissances à notre portée. Au LIFO les stagiaires sont un peu plus privilégiés car les chercheurs, qui sont aussi des enseignants, sont pédagogues et prennent le temps de nous répondre pour que l'on puisse continuer à avancer. Aussi après cette période nous sortons renforcés en terme d'acquis et de capacité à mettre en oeuvre.

De plus en faisant notre stage au LIFO nous avons appris à mieux connaître le secteur de la recherche informatique. Si celui-ci est moins médiatisé que d'autres, il n'en reste pas moins que les évolutions rendues possibles par le travail des chercheurs s'appliquent ensuite au quotidien comme nous avons pu le constater (travail sur le traitement de gros fichiers, calculs parallèles distribués, sécurisation etc.). Nous avons donc pu transposer nos travaux en terme d'applications futures pour les rendre plus concrets.

Ainsi en travaillant sur les appareils mobiles et la recherche d'une méthode permettant d'optimiser le développement nous avons démontré qu'il était possible d'optimiser le temps de travail et les lignes de code. En rendant l'application fonctionnelle sur les deux supports, Android et iPhone, nous avons réalisé les objectifs définis.

Enfin à titre plus personnel, j'ai pu valider mon projet professionnel et continuer à le construire. Je souhaite maintenant profiter des connaissances que j'ai acquies pour faire des jeux vidéos sur un mode de développement mixte (plateforme dédiée et code natif) pour le iPhone tout en continuant mon cursus universitaire.

Annexe A

Glossaire

Algorithme : Un jeu de règles ou de procédures bien défini qu'il faut suivre pour obtenir la solution d'un problème dans un nombre fini d'étapes. Un algorithme peut comprendre des procédures et instructions algébriques, arithmétiques, et logiques, et autres. Un algorithme peut être simple ou compliqué. Cependant un algorithme doit obtenir une solution en un nombre fini d'étapes. Les algorithmes sont fondamentaux dans la recherche d'une solution par voie d'ordinateur, parce que l'on doit donner à un ordinateur une série d'instructions claires pour conduire à une solution dans un temps raisonnable.

ANSI (American National Standard Institute) : Commission américaine chargée de valider et de normaliser des applications techniques, entre autres dans les domaines de l'informatique. Ainsi, les fabricants de disques durs, par exemple, font des propositions visant à intégrer de nouvelles technologies, que l'Ansi se charge de valider ou non. Si cela est le cas, on dit qu'une caractéristique est à la norme Ansi. En revanche, le fait qu'une technologie ne soit pas normalisée par cette commission n'empêche nullement un constructeur de l'intégrer à ses nouveaux produits.

ASCII (American Standard Code for Information Interchange) : Développé en 1968 afin de normaliser la transmission de données entre logiciels et équipements disparates, le code ASCII est incorporé dans la plupart des mini-ordinateurs et ordinateurs personnels. ASCII est l'acronyme de American Standard Code for Information Interchange (code américain normalisé pour l'échange d'informations). Il s'agit d'un système de codage sur 7 ou 8 bits qui affecte des valeurs numériques à 256 caractères : lettres, chiffres, signes de ponctuation, caractères de contrôle et autres symboles.

Base de données : Ensemble de données organisé en vue de son utilisation par des programmes correspondant à des applications distinctes et de manière à faciliter l'évolution indépendante des données et des programmes. Anglais : data base.

Bibliothèque : Les bibliothèques regroupent un ensemble de fonctions de base utilisées par de nombreux programmes. De nombreuses bibliothèques sont désormais intégrées aux systèmes d'exploitations (DirectX est une bibliothèque de Microsoft spécialisée dans le multimédia). Les programmes doivent être optimisés pour pouvoir tirer parti des bibliothèques additionnelles.

Browser : En anglais, désigne les navigateurs Internet qui vous permettent de vous connecter. Netscape est le browser Internet le plus répandu dans le monde. Français : Navigateur Internet

C : Langage général de programmation orientée objets, très populaire, inventé en 1960 par Dennis Ritchie pour AT&T

DLL : Bibliothèque de liens dynamiques, autrement dit programme qui seconde une application Windows pour des tâches précises.

DSP : Composant qui traite un signal audio et le numérise. Anglais : Digital Sound Processor.

Ecran tactile : Écran muni d'un dispositif qui permet de sélectionner certaines de ses zones par contact.

FPU (Floating Point Unit) : Coprocesseur arithmétique. Ce module est intégré à tous les processeurs depuis le 486 d'Intel. Il s'occupe de tous les calculs à virgule flottante déchargeant ainsi le CPU de cette tâche.

Grappe : Ensemble d'appareils de même type (terminaux, ordinateurs, etc.) rattaché à une même unité de contrôle. Anglais : cluster.

HTML (HyperText Markup Language) : Langage de description de pages adopté par la communauté Internet.

Java : Langage de développement créé par Sun. Dérivé du C++ dont il n'en possède pas la complexité, Java est un langage orienté objet. Les programmes créés à partir de Java ont la propriété de fonctionner sur n'importe quelle plateforme matérielle grâce à un système nommé "Machine virtuelle" (voir JVM). Pour cette raison, Java est très employé dans la communauté Internet.

JPEG (Joint Picture Expert Group) : Procédé de compression d'images dont de nombreuses pages Web usent dans leurs illustrations. Groupe d'experts communs au CCITT et à l'ISO responsable de la normalisation dans le domaine de la compression d'images fixes. Par extension, désigne la méthode de compression normalisée par ce groupe.

JVM (Java Virtual Machine) : La JVM est à la base de la "portabilité" de Java. Elle est la partie de Java qui permet aux applications programmées dans ce langage de fonctionner sur n'importe quelle plateforme. Le code Java est en fait interprété et exécuté par la JVM qui agit alors comme une machine virtuelle au sein de la machine réelle (un PC, par exemple).

Machine virtuelle : Couche logicielle au sein d'un système d'exploitation qui permet d'émuler le fonctionnement d'une machine au sein de la machine physique.

Objet : Unité structurée et limitée. On le définit toujours par la tâche ou la fonction qu'il accomplit. Il doit contenir en lui-même tous les éléments dont il a besoin. En informatique, on peut créer un objet par exemple à l'aide d'un enregistrement composé de champs de données (voir champ de données) définissant ses propriétés. Pour le programmeur, un objet est un ensemble fermé, composé de données et d'un code. On ne peut manipuler les données de l'objet qu'au moyen de ce code. Sous Windows, le terme "objet" désigne aussi un ensemble de données défini dans une Application source et transféré dans un document d'une Application cible.

Portabilité : Aptitude d'un programme à être utilisé sur des systèmes informatiques de types différents.

Programmation par objet : Mode de programmation dans lequel les données et les procédures qui les manipulent sont regroupées en entités appelées objets.

Script : Programme simple consistant en un ensemble d'instructions destinées à exécuter ou automatiser des tâches ou fonctions spécifiques.

Système exploitation (Operating System.) : Logiciel qui contrôle l'affectation et l'utilisation de ressources matérielles telles que la mémoire, le temps processeur, l'espace disque et les périphériques. Un système d'exploitation est la base sur laquelle s'exécutent les logiciels (applications).

XML (eXtended Markup Language.) : Dérivé du HTML, ce langage destiné à Internet est beaucoup plus puissant notamment pour la réalisation de mises en page complexes.

Annexe B

Moteur de calcul en code natif pour le projet calculatrice

```
\\ Code source du moteur en C
#include <string.h>
#include <stdio.h>

int main(){
\\ Deux strings pour simuler l'appel
const char * chif = "24/2.5";
const char * oper = "++";
\\ Calcul des longueurs des strings
int longueurchiffre;
int longueuroperande;
longueurchiffre=strlen(chif);
longueuroperande=strlen(oper);
\\ Création des tableaux de char de la taille
\\ des strings
char chiffre[longueurchiffre];
char operande[longueuroperande];
\\ Copie des strings dans le tableau de char
\\ car on prévoit le traitement des strings
\\ venant de format différents vers un
\\ format plus accessible au C
strcpy(chiffre ,chif);
strcpy(operande ,oper);

int cptchiffre;
int ctoperande;
char ligne[longueurchiffre];

double res;
double resfinal;
double virg;
cptchiffre=0;
ctoperande=0;
resfinal=0;

while (ctoperande<longueuroperande){
    res=0;
    virg=0;
    while (chiffre[cptchiffre]!='/' && chiffre[cptchiffre]!='.' &&
        cptchiffre<longueurchiffre){
        res=res*10;
        res=res+(chiffre[cptchiffre]-48);
        cptchiffre++;
    }
    if (chiffre[cptchiffre]=='.'){
        cptchiffre++;
        while (chiffre[cptchiffre]!='/' && cptchiffre<longueurchiffre){
```

```

        virg=virg+(chiffre[cptchiffre]-48);
        virg=virg/10;
        cptchiffre++;
    }

    res=res+virg;

    if (chiffre[cptchiffre]=='/') cptchiffre++;

    if (operande[cptoperande]=='/' && res==0) printf("Error \n");;
    if (operande[cptoperande]=='+') resfinal=resfinal+res;
    if (operande[cptoperande]=='-') resfinal=resfinal-res;
    if (operande[cptoperande]=='/') resfinal=resfinal/res;
    if (operande[cptoperande]=='*') resfinal=resfinal*res;
    cptoperande++;
}
printf("Resultat final %lf \n",resfinal);
}

```


Annexe C

Présentation des plateformes de téléchargement légal pour Android et iPhone

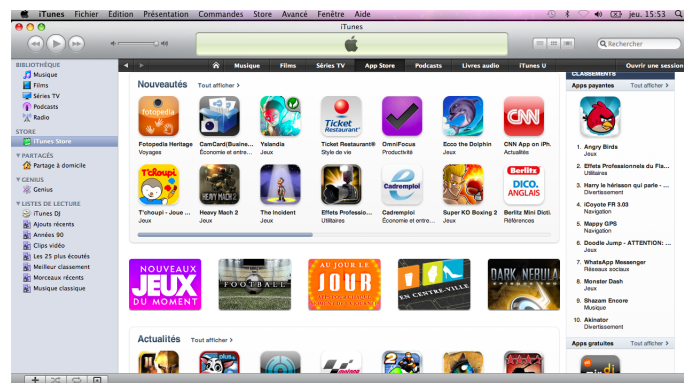


FIGURE C.1 – iTunes - L'Installation de ce logiciel est obligatoire pour pouvoir télécharger sur l'Apple Store

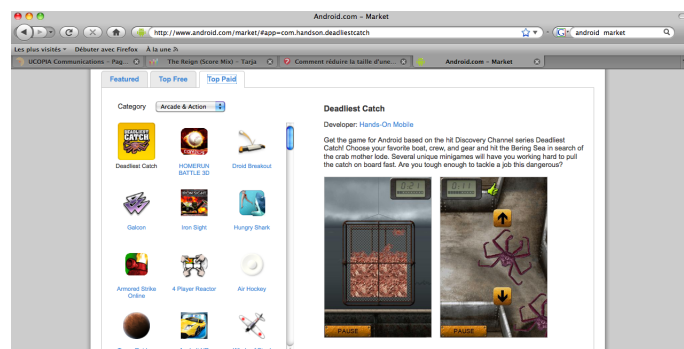


FIGURE C.2 – L'Android Market accessible avec un simple navigateur internet
<http://http://www.android.com/market/#app=com.handson.deadliestcatch>

Annexe D

Aide mémoire pour l'Objective C

```
// Appel d'une méthode à partir d'une instance de la classe  
[object method];
```

```
// Appel d'une méthode à partir d'une instance de la classe avec passage de variable  
[object methodWithInput:input];
```

```
// Appel d'une méthode de classe et affectation à un objet  
// dont le type n'est pas connu  
id myObject = [NSString string];
```

```
// Déclaration d'un objet de type string constant  
NSString* myString = [NSString string];
```

```
\\ Prototype d'une méthode renvoyant un booléen  
\\ avec passage de paramètre  
-(BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile;
```

```
\\ Appel de la même méthode  
BOOL result = [myData writeToFile:@"tmp/log.txt" atomically:NO];
```

```
// Affectation d'une string à la propriété caption (légende) de l'objet photo  
// Accéder à la propriété caption de l'objet photo  
[photo setCaption:@"Day at the Beach"];  
output = [photo caption];  
// ou  
output = photo.caption;
```

```
// Création d'un objet de type string  
NSString* myString = [NSString string];
```

```
// Définition du header d'une classe fichier .h  
#import <Cocoa/Cocoa.h>  
@interface Photo : NSObject {  
// Attributs de l'objet  
    NSString* caption;  
    NSString* photographer;  
}  
// Méthode de l'objet  
- (NSString*)caption;  
- (NSString*)photographer;  
@end
```

```
// Code source de la classe
#import "Photo.h"
@implementation Photo
- (NSString*) caption { return caption; }
- (NSString*) photographer { return photographer; }
@end
```

```
\\ Constructeur de l'objet
- (id) init {
    if ( self = [super init] ) {
        [self setCaption:@"Default Caption"];
        [self setPhotographer:@"Default Photographer"]; }
    return self;
}
```

```
// Destructeur de l'objet
- (void) dealloc {
    [caption release]; //
    [photographer release]; //
    [super dealloc];
}
```

```
// Afficher un message dans la console
int nombre = 5;
NSLog(@"Message %d",nombre);
```

Annexe E

Installer une chaine d'outils pour développer sous Android avec Linux

Télécharger et installer le SDK et le NDK pour Android

Pour télécharger le SDK Android : <http://developer.android.com/sdk/index.html>

Pour télécharger le NDK Android : <http://developer.android.com/sdk/ndk/index.html>

Décompresser les deux dossiers.

Mettre à jour les chemins sous Linux

Créer son projet et générer les dossiers automatiquement

```
android create project \
--target <target_ID> \
--name <your_project_name> \
--path path/to/your/project \
--activity <your_activity_name> \
--package <your_package_namespace>
```

Explications :

- target : de 1 à 7, il correspond à la plateforme Android choisie pour tester l'application¹
- name : nom du projet
- path : chemin où créer le dossier
- activity : nom de la classe qui est le point d'entrée d'un programme sous Android
- package : nom du package comme pour Java

1. Pour voir la liste des plateformes disponibles : `android list targets` dans le fichier Tools du répertoire du SDK Android

Liste des fichiers créés :

- AndroidManifest.xml - fichier de configuration de l'Activity (nom de l'application)
- build.xml - fichier de configuration pour la compilation sous ANT
- default.properties - propriétés par défaut pour Android, il ne doit pas être modifié
- build.properties - Fichier pour ajout de propriétés pour la compilation (signatures par exemple)
- src/your/package/namespace/ActivityName.java - Le main
- bin/ - Regroupe les fichiers binaires créés par la compilation
- gen/ - Regroupe les fichiers générés automatiquement comme R.java.
- libs/ - Les bibliothèques
- res/ - Les fichiers annexes comme les images, les sons, les XML
- src/ - Les sources du programme
- tests/ - Conserve une copie de tout le projet pour test

Mettre à jour un projet

Il arrive parfois qu'il faille changer le nom d'un projet, ou encore la plateforme de test. La commande update du SDK permet de mettre à jour les dossiers.

```
android update project --name <project_name> --target <target_ID>
--path <path_to_your_project>
```

Compiler un projet

A la racine du répertoire du projet pour une version debug exécutée sous simulateur :

```
ant install
```

A la racine du répertoire du projet pour une version release à installer sous appareil mobile :

```
ant release
```

Dans le cas d'une version release, il faut signer l'application.

Signer une application

Il faut d'abord créer une clé publique :

```
$ keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA
-keysize 2048 -validity 10000
```

Explications :

- my-release-key.keystore : le nom du fichier contenant la clé créée
- alias_name : le nom de la clé
- validity : nombre de jours de vie de la clé

Dans le fichier build.properties ajouter :

```
key.store=/pathtofilekey/my-release-key.keystore
key.alias=aliasyouchoose
```

Créer et démarrer un émulateur pour tester en mode debug

Dans un terminal aller dans le répertoire du SDK puis Tools et lancer la commande :

```
android create avd -n <name> -t <targetID> avec comme argument le nom et le numéro de l'émulateur.
```

Il faudra avant d'exécuter du code lancer une session de cet émulateur : `emulator @name`. Il restera ainsi en arrière plan et lors de la compilation d'une application la commande `install` trouvera un simulateur où placer le fichier `.apk`.

Annexe F

Installer le SDK iPhone pour un ordinateur sous Mac OS

Télécharger et installer le SDK sous Mac OS

Il faudra s'enregistrer pour obtenir un IDApple sur le site d'Apple (<http://www.apple.com>) puis télécharger le SDK (<http://www.apple.com/fr/ipad/sdk>).

Créer un projet pour Iphone sous Xcode

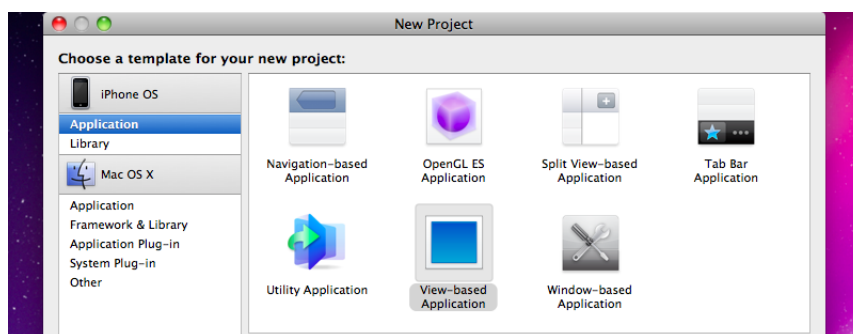


FIGURE F.1 – Créer un projet Iphone sous Xcode

Compiler sous simulateur

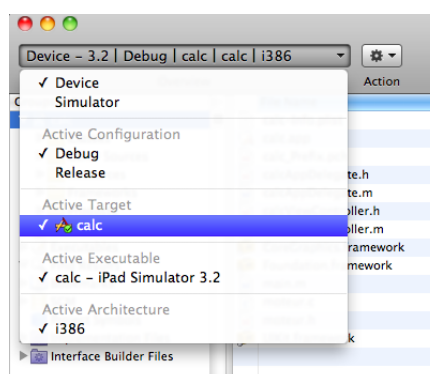


FIGURE F.2 – Choisir le device où installer et lancer l'application

Annexe G

Bibliographie et liens

Liens pour les plateformes sous Android

Installer le SDK Android sous Linux :

<http://wiki.smartphonefrance.info/%28X%281%29S%28rd5peqnhbisdhv45vkyhil55%29%29/Default.aspx?Page=sdk-androidlinux&AspxAutoDetectCookieSupport=1>

Tutoriel pour un Hello world sous Android

<http://developer.android.com/guide/tutorials/hello-world.html>

Installation du SDK sous Android

<http://android.cyrilmottier.com/?p=3>

Programmer pour Android sans Eclipse

<http://developer.android.com/guide/developing/other-ide.html>

Faire de l'Open GL sous Android

http://www.anddev.org/colored_3d_cube-t4.html

JNI sous Java

http://www.iut-orsay.fr/dptinfo/Pedagogie/Roger.Astier/app_java/javaJni.html

Tutoriel pour utiliser Ant sous Linux

http://forum.korben.info/topic/1656-developpement-android-sans-eclipse/page__p__26060?s=57ab17a1d1723baefcc19b5b954624c5#entry26060

Tutoriel pour apprendre l'OpenGL ES sous Android

<http://blog.jayway.com/2009/12/03/opengl-es-tutorial-for-android-part-i/>

Les écouteurs d'événements sous Android

http://wiki.frandroid.com/wiki/Ev%C3%A8nements_de_l%27interface_utilisateur

Sprites et gestion 2D

<http://www.helloandroid.com/tutorials/how-use-canvas-your-android-apps-part-2>

Liens pour les plateformes sous iPhone

Utiliser du code C ou C++ avec Objective C

<http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/ObjectiveC/Articles/ocCPlusPlus.html>

Bloc note pour l'Objective C

http://cocoadevcentral.com/d/learn_objectivec/

Tutoriel Hello world

<http://iphonesoft.fr/tutoriel-developpement-01-hello-world>

Autres documents

Un article sur les parts de marché des deux concurrents

http://www.mac4ever.com/news/54257/plus_de_telephones_android_vendus_que_d_iphone_aux_usa_ce_trimestre/

Tutoriel Latex

<http://www.tuteurs.ens.fr/logiciels/latex/>

Classement des applications sous iPhone

http://www.unsimpleclic.com/classement-des-applications-pour-iphone_934.html

FAQ pour Latex

<http://www.grappa.univ-lille3.fr/FAQ-Latex/>

Liens Néon

Néon sous iPhone

<http://wanderingcoder.net/2010/06/02/intro-neon/>

<http://monkeystylegames.com/?p=82>

Les livres utilisés

Programmation Cocoa sous Mac OS X, 3ème édition, Edition Pearson, Aaron Hillegass

Beginning iPhone 3 Développement, exploring the iPhone SDK, Edition Apress, Dave Mark et Jeff LaMarche