Rapport de stage

magic pockets

Maître de stage : Eric ZMIRO

Stagiaire : Damien DOURY
36, chemin de la messe
45390 Puiseaux
France
damien.doury@hotmail.fr

2 Juillet - 26 Octobre 2012

1. Présentation de la société

Magic Pockets est un studio français de développement de jeux vidéo situé en région parisienne à Torcy (RER A, à 10 minutes de Disneyland) et dirigé par Eric ZMIRO en collaboration avec Etienne JACQUEMAIN.

Il développe uniquement des commandes mais va peut-être bientôt se lancer dans un jeu fait maison.

L'entreprise comporte une vingtaine d'employés réalisant tout sur place sauf l'univers sonore.

Ses clients sont principalement de gros éditeurs dont notamment Sega, Activision ou encore Square Enix. On peut encore citer d'autres grands noms comme EA Games ou Ubisoft.

Historiquement, la société ne développait que des jeux pour consoles portables mais vient peu à peu à développer des jeux pour console de salon (dont notamment la Wii ou la Wii U qui sortira prochainement). Depuis sa création, le studio a développé plus de 50 jeux sur une pléiade de consoles, de la Game Boy Color à l'iPad.

1, promenade du Belvédère
77200 Torcy
FRANCE
+33 1 64 62 63 90

http://www.magicpockets.com/

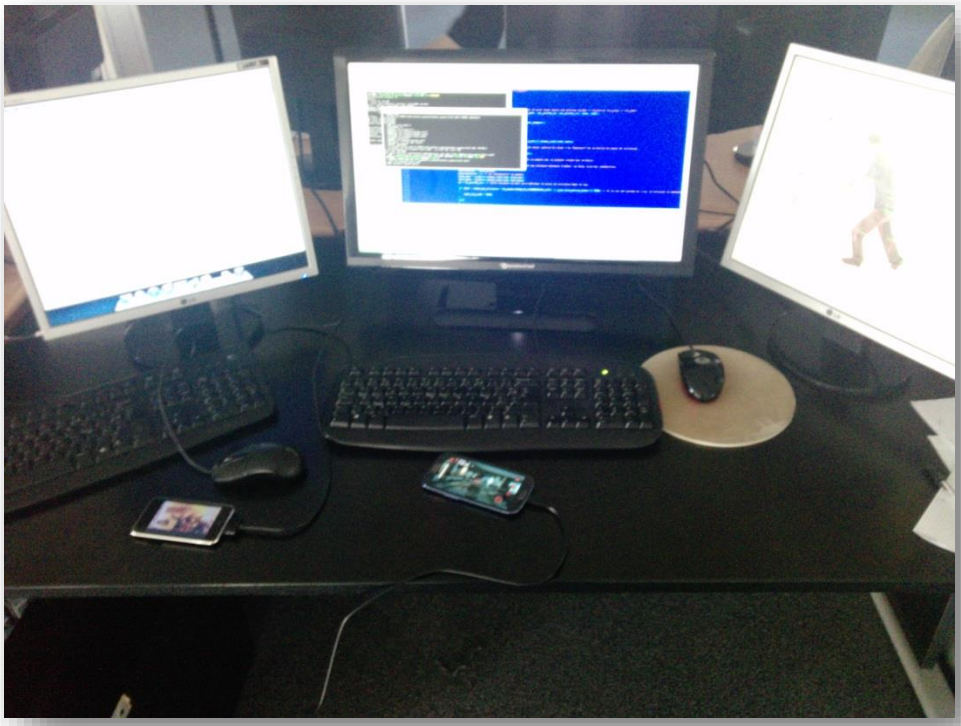
2. Environnement de travail

L'intégralité du matériel est fourni par Magic Pockets.

2.1. Le matériel de développement.

Mon poste est composé de 3 écrans : 2 sont dédiés au Windows XP ou Seven et l'autre à Max OS X Lion.

L'écran principal déjà en full HD combiné à un deuxième écran permet d'obtenir une surface d'affichage maximale pour travailler sans contraintes. Le Mac n'a pas besoin d'un écran géant car il ne sert qu'à compiler pour les iPods, iPhones et autres iPads.



Vue de mon poste. On remarque le Windows, le Mac, l'Android et l'iPod.

2.2. Logiciels.

Côté logiciel, nous travaillons sous Visual Studio 2005 ou 2010 et Visual Source Safe pour la mise en commun des données.

VisualGDB permet de compiler, d'exécuter et de débuguer du code Java sous Visual Studio. Complété par JNI, il est possible d'appeler du code C depuis le Java et vice-versa.

Quant aux graphistes, ils se servent de Maya, Photoshop, 3DS Max et d'autres logiciels.

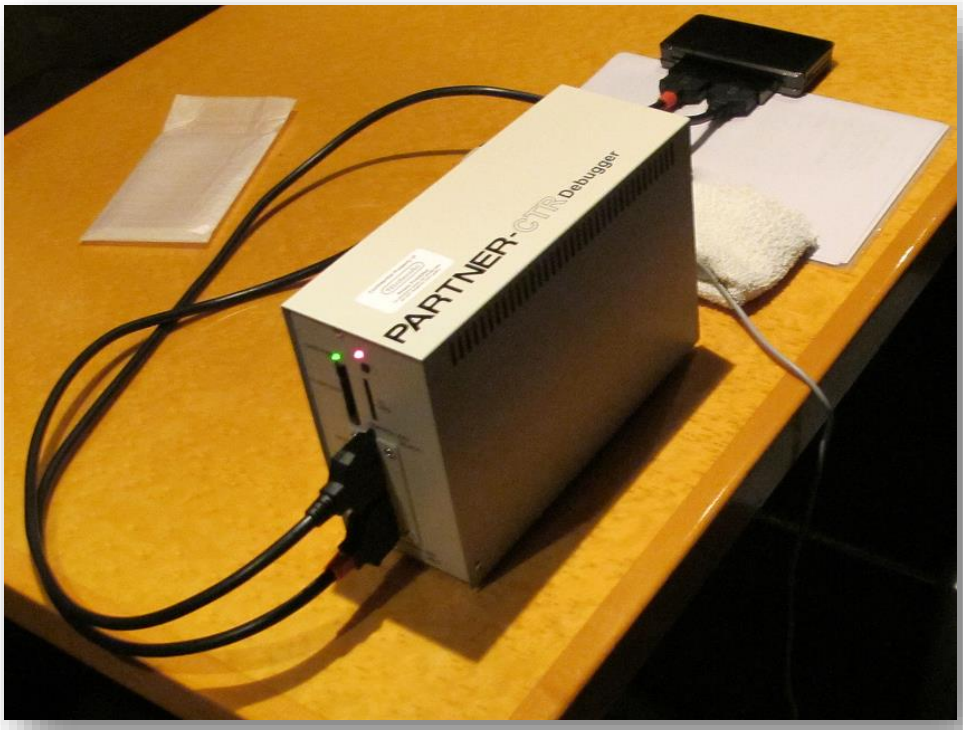
2.3. Les consoles de test.

La société dispose de nombreuses consoles pour faire tourner les jeux et observer le rendu dans les conditions réelles sans avoir à passer par un simulateur.

De nombreux téléphones et tablettes Android de toutes marques et aux configurations variées représentent un échantillon de la population des téléphones Android. Cela permet d'assurer la compatibilité du jeu pour un maximum de modèles. J'ai ainsi pu travailler sur un Samsung Galaxy S3 (qui est à l'époque où j'écris ce rapport le fleuron des téléphones Android mais à l'époque où vous lisez il s'agit probablement d'une antiquité oubliée et complètement dépassée).

De même pour la marque Apple, plusieurs iPod/iPad/iPhone sont à disposition.

Pour le développement sur DS et 3DS, la société possède des kits de développement. Il est peu commun d'en voir car ils sont réservés aux professionnels (environ 2500 USD). Ces kits sont composés d'une console Nintendo reliée par un fil à une mini unité centrale, elle-même reliée au PC.



Kit de développement 3DS. Image provenant de planet3ds.net

Ce système permet de compiler sans avoir à graver une cartouche de jeu à chaque fois mais possède les inconvénients d'une console filaire. Pour pallier à ce problème, la société dispose aussi d'un modèle de Nintendo 3DS de dev sans fil dite « Panda » du fait de sa coloration noire et blanche (coloris introuvable dans le commerce pour la distinguer). Toutefois ce modèle accueille des cartouches spéciales et n'est pas compatible avec celles du marché.

3DS Panda. Image provenant de thebitbag.com

Dans des vitrines, on peut voir les kits de développement des anciennes consoles, généralement assez imposants.

Récemment, la société a fait l'acquisition d'un kit de développement Wii U, reçu avant la sortie de la console (ce qui est assez plaisant du point de vue du consommateur) et le moteur est en cours de portage pour cette console.

3. Rôle au sein de l'entreprise.

Mon stage s'est principalement découpé en 2 missions : tout d'abord, travail sur le moteur et découverte des technologies de la société, puis développement du gameplay pour une commande. Chaque partie ayant duré environ 2 mois de chacune.

3.1. Travail sur les technologies (Juillet-Août).

Durant la première partie de mon stage, j'ai été amené à effectuer diverses missions pour découvrir puis enrichir la technologie de la société.

Magic Pockets dispose de son propre moteur de jeu. Celui-ci, développé au fil des années et toujours en développement, permet de créer un jeu sur différentes plateformes en ne codant qu'une seule fois le gameplay. D'autres part, il contient tout un tas de fonctions de calculs mathématiques, d'affichage 2D, 3D, de lecture de bases de données et plein d'autres qui facilitent la tâche du développeur en charge du gameplay.

L'avantage d'un moteur maison est qu'il permet d'avoir la main sur le développement de A à Z. Ainsi, si un élément ne fonctionne pas, il est plus aisé d'aller le corriger qu'avec un moteur acheté. D'autre part, il est aussi possible de l'adapter à la demande du client (comme pour intégrer de nouvelles bibliothèques par exemple). Et il est toujours gratifiant d'avoir développé son propre moteur, ce qui peut être un argument commercial pour montrer le professionnalisme et le savoir-faire de l'entreprise.

En amont du moteur se trouvent différents outils permettant principalement de compiler les ressources dans un format compris par le moteur. Ces outils sont tous regroupés dans un fichier BAT nommé simplement « m ». Cette simple commande, permet de récupérer de manière intelligente les données sur un serveur, de les compiler et de les stocker en local. Tout le processus est automatisé : il n'y a pas de place à l'erreur humaine, et un gain de temps par rapport à un transfert à la main.

3.1.1. Mission 1.

Ma première mission aura été l'amélioration d'un de ces outils : l'outil de capture de sprites.

Découpée en plusieurs jalons, ma première tâche aura été la création d'un fichier image pouvant afficher le contenu d'un fichier image compilé par cet outil.

Le défi de cette première tâche aura été la compréhension du système à bas niveau. En effet, dans le but d'optimiser l'espace mémoire, il arrivait fréquemment que plusieurs informations soient stockées dans un seul et même octet. On ne pouvait donc pas directement lire l'information voulue de manière « conventionnelle » ou plutôt « scolaire » (car on descend rarement à un niveau si bas en cours).

Solution employée : utilisation de masques (&), de décalages de bits (<< / >>), l'opérateur OU (|), etc.

Après quelques explications utiles de la part de mon maître de stage, le concept fût assimilé et la tâche réalisée en moins d'une semaine. Cela m'aura permis de me familiariser avec le binaire et l'hexadécimal. J'aurais aussi découvert la structure de différents fichiers image.

Exemple de rendu. Les rectangles représentent les différents morceaux d'images recollées.

Pour info, le format original est un fichier Photoshop (PSD) qui est lu et compilé par le moteur en un fichier GR (format interne). Un fichier GR est créé pour chaque PSD. Une fois la récupération terminée, la liste de ces fichiers GR est regroupée au sein d'un seul et unique fichier DAT (format interne, contenant la liste des images). En parallèle, d'autres fichiers sont créés comme un fichier contenant la liste des en-têtes, un fichier contenant la liste des define des noms des sprites, etc.

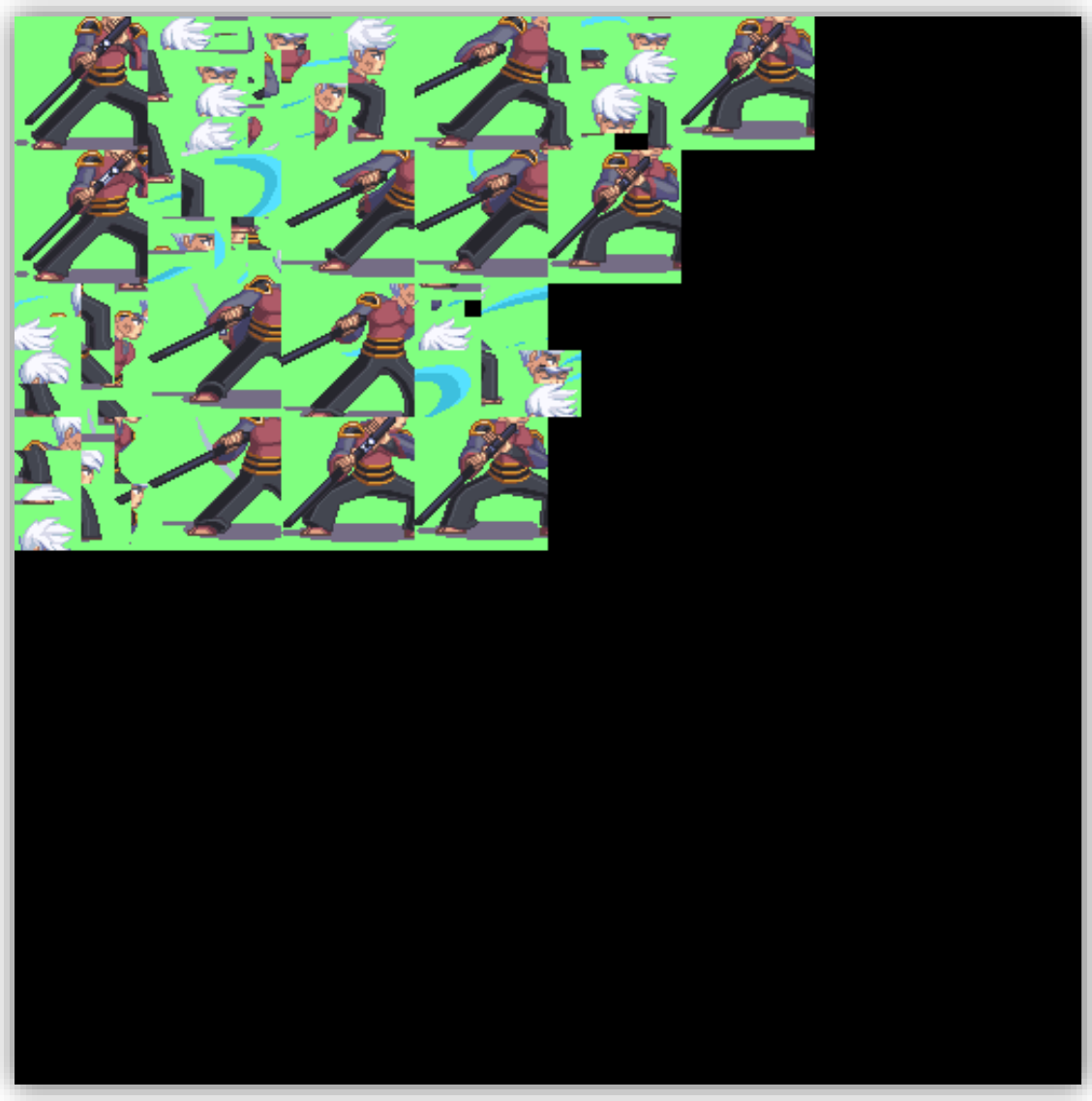
Ce qui nous amène à ma deuxième tâche qui est la suite logique de la première.

3.1.2. Mission 2.

Une fois les bases assimilées, je me suis vu confier ma première tâche utile à la société : l'ajout une nouvelle fonction dans cet outil de capture de sprites. En fonction du format original du fichier Photoshop (en palette ou RGB), je devais créer un nouveau format intermédiaire qui viendrait ensuite s'incorporer au fichier DAT.

Les principaux défis de cette tâche étaient la compréhension du code source de l'outil (qui a connu plusieurs mises à jour et dont le code est forcément un peu chargé) et la mise en place d'un système de rangement des sprites. Ce système de rangement se nomme « quad-tree » : il permet d'optimiser l'espace utilisé dans la planche de sprites (voir sur internet pour plus d'informations).

Après m'avoir expliqué le fonctionnement de ce système, j'ai imaginé un algorithme le reproduisant.



Rendu du quad-tree. Les pixels transparents sont affichés en vert pour visualiser l'espace utilisé.

Il m'a fallu environ 2 semaines pour venir à bout de cette tâche qui fût la plus difficile de mon stage.

Les tâches suivantes étaient plus courtes.

3.1.3. Missions 3.

J'ai réalisé un algorithme de « choix » du meilleur mode OpenGL pour la création d'une vue parmi une liste. La conception de cet algorithme était intéressante (avec le recul, plutôt simple). Durée : moins de 2 jours.

Solution employée : système de sélection par priorisation des paramètres.

1.1.1. Mission 4.

Intégration de la bibliothèque Flurry au moteur suite à la demande de Square Enix qui souhaitait récupérer des statistiques quant à l'utilisation de son jeu.

Le plus difficile a été de faire le lien entre la bibliothèque en Java et le code du jeu en C, notamment à cause des différences vis-à-vis des différents types de données entre ces 2 langages.

Solution employée : utilisation de Java Native Interface (JNI) pour faire l'interface entre les 2 langages. Ainsi le Java peut appeler du C et le C peut appeler du Java.

Une autre difficulté a été de savoir si la bibliothèque fonctionnait bien (les données récupérées mettaient 24h avant de s'afficher sur l'interface web de Flurry).

De plus, il a été demandé que cette bibliothèque puisse être désactivée sans modifier le code, ce qui était un autre challenge car il n'y a pas de compilation conditionnelle en Java.

Solution employée : création d'une bibliothèque JAR contenant les mêmes méthodes que l'originale, mais sans implémentation. Pour désactiver l'une ou l'autre, il suffit de changer le nom de l'extension pour qu'elle ne soit pas récupérée à la compilation.

Au total, pour que tout fonctionne, il m'aura fallu presque une semaine.

1.1.2. Mission 5.

J'ai aussi intégré une fonction d'alerte dans le moteur. Cela ouvrait une Alert Java depuis le code en C et affichait un message envoyé depuis le C. J'ai utilisé la même technique que pour Flurry.

Début Septembre est arrivé Guillaume SWIATEK qui a poursuivi mon travail sur les bibliothèques. J'ai ainsi pu entamer la deuxième partie de mon stage et commencer quelque chose de différent : le développement du gameplay.

1.2. Développement de gameplay (Septembre-Octobre).

À partir de fin Août j'ai donc commencé un tout autre travail.

Le projet sur lequel j'ai été affecté est un portage sur smartphone de la franchise « The House of the Dead » de SEGA, sortie initialement sur borne d'arcade en 1996. Le jeu de référence utilisé pour l'adaptation est « House of the Dead : Overkil »l sur PS3.

Avant que je commence, un prototype de test avait déjà été mis en place par Eric dans le but de tester les performances du moteur et les capacités des devices. À ce stade, le jeu était composé d'une caméra qui se déplaçait dans un hôpital. Des zombies inactifs apparaissaient tournés vers le joueur. Peu après, ils marchaient jusqu'au joueur. Aucun menu, aucune collision, bref que de l'affichage.

1.2.1. Commencement.

Ma première tâche a consisté à intégrer un réticule de visée contrôlé au doigt nommé Virtual D-Pad, ce qui fût réalisé dans la matinée. J'ai poursuivi en ajoutant 3 autres modes de contrôles (dont un de mon invention).

Par la suite, j'ai ajouté différentes fonctionnalités en suivant les demandes de mon maître de stage et du chef de projet tout en ajoutant d'autres fonctionnalités de ma propre initiative en me basant le cahier des charges établi peu à peu par un game designer.

Ces premières fonctionnalités concernaient principalement l'interface du jeu et les interactions du joueur avec la machine (écran tactile, accéléromètre).

Pour afficher l'interface, je réutilisais les données récupérées avec la commande « m » et je me servais donc indirectement de mon ancien travail.

Pour placer les sprites, je récupérais des points sur les couches alpha des PSD qui forment un squelette. Ainsi, il suffit de modifier le PSD pour modifier l'interface du jeu, ce qui permet aux graphistes et aux game designers de modifier le jeu rapidement et sans avoir à en faire la demande aux développeurs.

1.2.2. Développement rapide.

Mi-Septembre sont arrivés en même temps un associate producer et game designer pour travailler avec moi sur le projet en cours. De plus, début Octobre, une partie de l'équipe travaillant sur un autre projet a été redirigée sur celui pour lequel je travaillais. Ainsi le jeu avançait à une vitesse élevée et gagnait en contenu chaque jour.

Après avoir réalisé de l'interface et géré les interactions, je me suis lancé dans du gameplay pur : système de scoring, gestion des collisions, récupération de données depuis des bases de donnée, comportement des zombies, etc.

La première étape importante (et la seule à laquelle j'ai pu assister durant la période de mon stage) a été la First Playable Publishable/Preview/Prototype (FPP). Auparavant avaient été envoyé une version « prototype » puis une version « work in progress » la semaine précédant la FPP et à chaque fois le retour de SEGA était positif.

Mi-Octobre est arrivé Pierre RIOM qui poursuivra mon travail sur House of the Dead après mon départ.

1.2.3. Quelques problématiques.

Le développement gameplay était ma partie préférée du stage. Elle aura posé de nombreux défis divers et variés. En voici les plus marquants :

- **Problème de programmation** : Collisions 3D.

Gérer les collisions avec les zombies n'était pas une mince affaire. Sur le plan conceptuel, la solution a été vite trouvée : pour représenter les zones de collisions, on sert des os du squelette du zombie et on forme une capsule (composée par un segment et un rayon) autour de ce dernier. La liste des os et leur rayon sont stockés dans une base de données. Reste à déterminer la collision entre la ligne formée par la trajectoire de la balle et ces capsules.

Voici la solution. Tout d'abord, on prolonge le segment de l'os pour former une droite dans l'espace, puis on calcule le plus petit segment entre ces 2 droites (En effet, 2 droites ne se coupent quasiment jamais dans l'espace. Leur « intersection » est donc un segment.). C'est moi qui ai intégré la fonction qui permet de réaliser ce calcul dans le moteur (mais ce n'est pas moi qui l'ai conçu, voir http://paulbourke.net/geometry/pointlineplane/). Ensuite, à l'aide d'une deuxième fonction du moteur, on récupère le point du plus court segment situé sur la trajectoire de la balle et on cherche le point le plus proche entre ce dernier et le segment de l'os. Si la distance entre ces 2 points est plus faible que le rayon de la capsule, alors il y a impact de la balle sur le zombie !

Pour compléter ce système de collisions, il reste ensuite à déterminer si la balle a percuté le décor (qui ne possède pas d'os) avant le zombie. Pour cela j'ai utilisé une troisième fonction du moteur et déplacé la balle petit à petit le long de sa trajectoire (seulement la logique, pas l'affichage car tout se déroule en $1/60^{ème}$ de seconde, soit une trame). À chaque itération on observe si elle percute un élément de décor. Si un élément de décor est trouvé, on compare les distances arme-décor et arme-zombie pour déterminer si l'impact avec le zombie prend effet.

Combiné à la base de données qui fournit la liste des os en fonction du shape d'un élément, on obtient un système automatique et facilement paramétrable qui gère les collisions de manière transparente pour la majorité des éléments 3D du jeu (zombie, civils, objets à ramasser, etc.).

- **Problème de programmation** : Système de démembrement.

Une fois le système de collision mis en place, il a été demandé de trouver sur quel membre avait lieu l'impact pour le retirer et de ce fait démembrer le zombie. Il est possible de démembrer le bras gauche, le droit et la tête pour un total de $2^3 = 8$ shapes différents. Le problème était de pouvoir sélectionner le bon shape de zombie de la meilleure manière possible. La solution de simplicité aurait consisté à créer un grand tableau de correspondances mais cela est peu pratique et volumineux. J'ai donc eu l'idée d'organiser les shapes dans un ordre bien précis. On enlève d'abord le bras gauche, puis le droit, puis la tête. Le moteur renvoyant les shapes sous formes d'un define (numéro) il suffisait ensuite d'ajouter $2^0 = 1$ au numéro du shape pour un bras gauche démembré, $2^1 = 2$ pour un bras droit et enfin $2^2 = 4$ pour la tête. Ainsi, en une ligne de code le problème était résolu de manière simple, fiable, flexible et rapide.

Pour ne pas poser de problème avec ce système, les os (et donc les collisions) de ce membre étaient retirés du squelette de la victime (on ne peut pas démembrer 3 fois le bras gauche par exemple).

- **Problème mathématique :** L'accéléromètre.

Un autre problème aura été la gestion d'un mode de tir à l'accéléromètre. Les valeurs renvoyées par les 3 axes de ce dernier étaient comprises entre -1 et 1 mais la courbe de progression n'était pas linéaire ce qui posait problème quant à la jouabilité. Il a donc fallu linéariser cette courbe. Grâce à l'aide de Pierre qui m'a donné le déclic, j'ai utilisé la fonction arccos qui a rendu la courbe parfaitement linéaire et permis d'exploiter des angles.

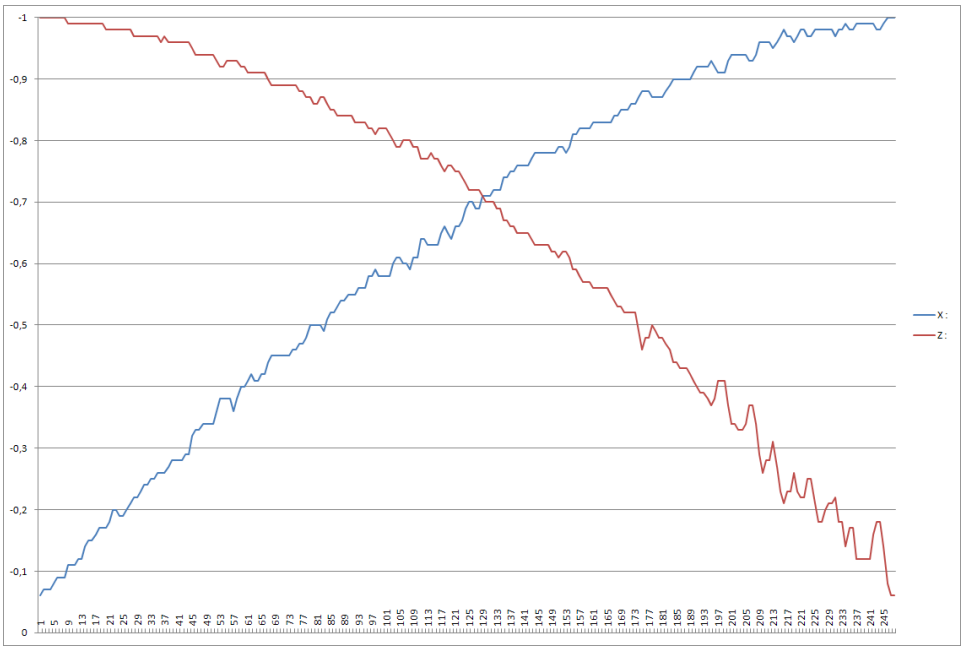
Mais les angles renvoyés sont compris en 0° et 180° et le joueur doit potentiellement pouvoir jouer sur les 360° (debout, allongé, dans une montagne russe, etc.). Il a donc fallu calibrer l'appareil à différents moment du jeu (lancement du jeu, menu pause, changement de contrôles) et déterminer sur quel axe seraient lues les valeurs utiles. Pour éviter l'effet « retour en arrière » l'axe choisi était le plus proche de 90° lors du calibrage.

Ensuite, il a fallu appliquer une marge de manœuvre pour que le joueur puisse atteindre les bords de l'écran sans complètement tourner l'appareil et encore une fois éviter l'effet « retour en arrière ».

Enfin, comme les valeurs brutes de l'accéléromètre varient souvent, il a fallu adoucir le déplacement du réticule de visée. La solution adoptée a été le calcul de la moyenne des valeurs des 3 axes sur les X derniers enregistrements. Ainsi il y a un (très faible) décalage entre le mouvement du device et le déplacement du viseur, mais celui-ci ne tremble plus.

Un système d'accroche du viseur sur un zombie lorsque celui s'en approchait n'a finalement pas été implémenté.

La solution finale est jouable mais peu probablement encore être améliorée.



Courbes brutes des axes X et Z de l'accéléromètre avant lissage.

2. En dehors de mes missions.

Au-delà de ce qu'il m'a apporté en compétences, ce stage m'aura permis de découvrir le monde du jeu vidéo et de me cultiver. La liste serait longue, mais voici en voici quelques extraits :

- J'ai pu voir à quoi ressemble le manuel développeur de la Game Boy Color. Il contient plein d'informations bas niveau sur la console, mais sur cette vieille console, rien était fait : le développeur devait tout faire à main.

- Découverte (en surface) des finances d'une société du jeu vidéo. Ce qu'il faut retenir : les salaires représentent la plus grosse partie de la facture. En effet l'amortissement des machines, la location du bâtiment, etc. représente peu à côté. De plus, un tel type de société d'utilise presque pas de consommables.

  Au niveau des contrats, il existe différents modèles. La plupart du temps, une somme d'argent est fournie au lancement du développement. Ensuite, le studio peut toucher des royautés sur les ventes du jeu. Ces commissions peuvent être versées tout de suite, à partir d'un certain montant ou une fois que la somme initialement payée a été remboursée. Il peut aussi arriver (comme avec Nintendo) que si un jeu ne rapporte pas assez, que les royautés d'un jeu suivant ne soient versées qu'une fois celles du premier jeu remboursées. À chaque contrat ses règles.

- J'ai appris que parfois quelques octets valent de l'or ! En effet, mon maître de stage m'a raconté que pour un précédent jeu il avait apporté des optimisations (notamment le stockage de la position d'un ennemi sur 1 bit : début ou fin) qui ont permis de gagner quelques octets de mémoire. Mais ces quelques octets ont permis à l'éditeur de produire le jeu sur une version de cartouche DS un niveau au-dessous celle qui était initialement prévue et ainsi faire une économie de 1$ par cartouche vendue. Ce chiffre peut paraître faible, mais il représente des centaines de milliers de dollars sur le total des ventes. Comme quoi il faut penser à optimiser chaque octet sur les plus faibles consoles !

- Quand un studio de développement travaille sur une licence, il reçoit les assets déjà existants. En règle générale, il s'agit d'un « closing kit » contenant les sons, modèles 3D, animations, sprites et autres du jeu précédent. Mais lorsqu'il s'agit d'un film (comme le jeu Battleship pour citer un cas de Magic Pockets), alors là le studio reçoit carrément les modèles 3D utilisés dans le film ! Reste aux graphistes à réduire le nombre de polygones de quelques dizaines de milliers à quelques centaines. Par exemple, Magic Pockets va bientôt recevoir les modèles 3D d'une série animée américaine pour le développement d'un futur jeu.

- Découverte de consoles d'antan, dont le « Virtual Boy » que l'on pourrait considérer comme l'ancêtre de la 3DS, car il est la première console portable de Nintendo à afficher des images en 3 dimensions (même si elles ne sont composées que de nuances de rouge) ! Cette console est composée de « lunettes » (afficheur rétroéclairé) ainsi que d'une manette filaire qui accueille la cartouche de jeu dans un format proche de celui de la Game Boy. Un petit support permet de poser l'afficheur sur une table et le tout fonctionne avec des piles. Il n'existe que très peu de modèles en France.

Différentes vues de la console, en cours d'utilisation à droite grâce à l'aide d'un geek qui a tenu à garder l'anonymat (d'où les bandes noires devant les yeux).

- Une borne d'arcade avec un écran de plus de 30 pouces sur laquelle on peut jouer à Metal Slug (sans mettre de pièces évidemment !). Partout dans les locaux on peut trouver des goodies (posters, figurines, livres, …) sur le thème de l'art vidéo ludique.

3. Bilan.

Cette expérience dans le monde du jeu vidéo fût très enrichissante, clairement ma meilleure expérience professionnelle à ce jour.

Je me suis amélioré en C et en logique de manière générale. J'ai aussi étoffé ma culture sur l'univers de la conception de jeux vidéo.

Cela m'aura aussi permis de rencontrer des gens de ce milieu exerçants une large variété de métier : graphistes 2D/3D, game designers, chef de projets, ... et ainsi m'immerger dans une production de bout en bout.

Je remercie toute l'équipe de Magic Pockets de m'avoir accueillie chaleureusement et contribué à ce que mon stage se déroule au mieux.

Merci ☺