UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

Développement de modules d'un jeu vidéo et recherches graphiques

Rapport de stage ST50 - P2009

MONNERET Nicolas

Département Informatique
Filière Imagerie, Interaction
et Réalité Virtuelle

Phoenix Interactive
88 rue Paul Bert
69 003 Lyon
www.phoenix-i.fr

Tuteur en entreprise
BEEN Sylvain

Suiveur UTBM
MEURIE Cyril

phoenix STUDIO

utbm université de technologie Belfort-Montbéliard

Remerciements

Tout d'abord, je tiens à remercier Monsieur Pierre MOUSSON, gérant de la société Phoenix Interactive, de m'avoir accueilli au sein de son entreprise et de l'équipe DS.

Ensuite je remercie Sylvain BEEN, directeur technique, qui a été mon tuteur dans l'entreprise et qui a pris le temps de suivre et d'évaluer mon travail, ainsi que de m'aiguiller quant à la rédaction de ce rapport.

Je remercie également Jérôme COPPENS, lead gameplay programmer et Olivier RAVET, lead engine programmer, d'avoir su m'encadrer et me former, et pour les précieux conseils qu'ils m'ont apportés lors de mes travaux.

Enfin, je remercie tous les membres de l'équipe de développement DS, avec lesquels j'ai pris plaisir à travailler, pour leur sympathie, leur bonne humeur et pour l'accueil qu'ils m'ont accordé.

Sommaire

Introduction

Le projet de fin d'étude ST50 à l'UTBM permet de mettre en application les connaissances acquises durant ses études dans les conditions qui seront celles de sa future activité professionnelle.

Mon projet étant de travailler dans l'industrie du multimédia et du jeu en particulier, c'est donc tout naturellement que j'ai choisis d'effectuer mon stage dans l'entreprise Phoenix Interactive, qui est un studio de développement de jeux vidéo Lyonnais.

Le sujet sur lequel j'ai travaillé concerne le « ***développement de modules d'un jeu vidéo*** » et la recherche graphique sur Nintendo DS.

Dans un premier temps, je présenterais l'entreprise Phoenix Interactive, l'organisation, les partenaires ainsi que les précédentes créations du studio. Ensuite, je décrirais le contexte du stage, le projet auquel j'ai participé, l'équipe que j'ai intégrée et les outils et langages utilisés. Dans une troisième partie, je présenterais le travail que j'ai effectué et les différents modules que j'ai développés pour le projet avant de terminer par une description des recherches graphiques que j'ai menées au sein de l'équipe « moteur de jeu ».

Phoenix Interactive

1. Organisation

Phoenix Interactive est une société à responsabilité limitée spécialisée dans la création de jeux vidéo. L'entreprise a été créée en 2004, elle est implantée à Lyon, sixième pôle mondial du jeu vidéo et fait partie des 10 plus importants studios de développement en France parmi les 133 recensés.

Elle emploie entre 50 et 100 personnes suivant le nombre et les types de projets dont elle a la responsabilité. Dans un but commun de création, le studio fait collaborer une diversité de métiers : managers, game designers, intégrateurs, programmeurs, graphistes 2D et 3D, mais aussi testeurs, chercheurs et ergonomes.

La hiérarchie de l'entreprise est organisée par métier.
En simplifiant, on a cinq grands pôles : gestion de projet, game design, programmation, intégration et graphisme. Chacun de ces pôles possède verticalement trois niveaux ; sous la responsabilité d'un directeur, des leads sont chargés d'une équipe et travaillent sur un projet pour une plateforme donnée.

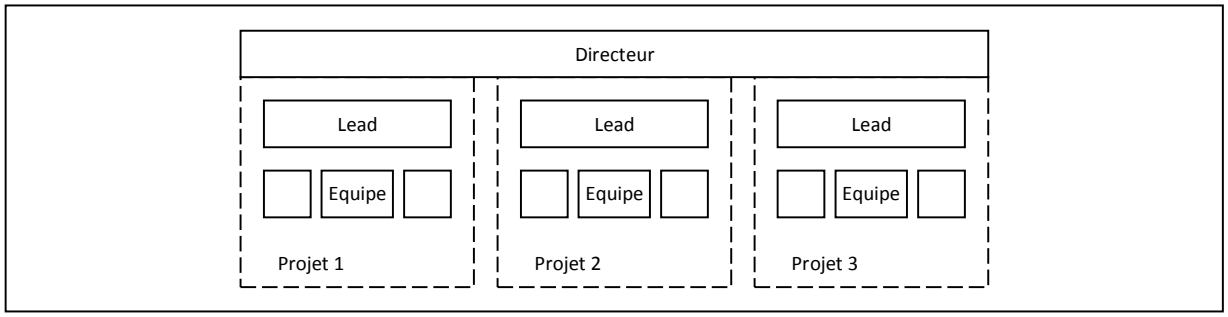

Figure 1 : hiérarchie

Aujourd'hui, l'entreprise occupe un bâtiment composé de cinq étages dont deux sont entièrement réservés à la production et un à la recherche.

Figure 2 : Les locaux de Phoenix Interactive

2. Recherche & Développement

Parallèlement aux productions, le studio lance actuellement un important programme de R&D afin d'innover aussi bien dans le domaine technologique que dans celui de l'Interface Homme-Machine (IHM) et des émotions.
Labellisé par le pôle de compétitivité ***Imaginove***, ce programme d'innovation *a pour but de développer les synergies entre les différentes filières de l'image (jeu vidéo, cinéma, audiovisuel, animation et multimédia) dans le but d'augmenter la compétitivité en matière de création, de production et de diffusion des œuvres Rhône-alpines* (Wikipedia).
Il a vocation, pour le studio, à une ouverture vers de nouveaux contenus, à la fois plus riches, plus accessibles et plus forts en émotions.
Pour cette raison, un étage parmi les cinq que compte l'entreprise est entièrement dédié à la recherche & développement.

3. Partenaires

L'entreprise travaille en collaboration avec de grands éditeurs tels qu'Ubisoft ou Disney Interactive.

Elle est également associée à des écoles comme l'ENJMIN, L'INSA, l'IIM, SUPINFOGAME ou encore SUPINFOCOM et des laboratoires de recherche comme L'INRIA et l'EPFL.

4. Plateformes

Actuellement, le studio développe des jeux sur la plupart des plateformes. Parmi celles-ci on trouve :

- PC ;
- PlayStation 3 de Sony ;
- Wii de Nintendo.


Mais le studio créé aussi des jeux sur les plateformes portables telles que :

- DS de Nintendo ;
- PSP de Sony.


5. Réalisations

Les productions actuelles de l'entreprise sont conçues pour les plateformes de nouvelle génération et visent principalement les jeunes enfants. La société est spécialisée dans le « casual gaming », c'est-à-dire des jeux aux règles typiquement simples et doté de mécanismes et de gameplay basiques afin d'être accessibles aux plus jeunes et aux joueurs occasionnels.

Les réalisations des équipes de Phoenix Interactive ont déjà reçu des récompenses prestigieuses :

- Meilleure production interactive du BAFTA (British Academy of Film and Television Arts) avec « *La Chasse au Miel de Tigrou* » (2000, Disney Interactive).
- Meilleur jeu vidéo du magazine officiel américain Playstation 2 avec « *Les aventures de Porcinet* » (2002, Disney Interactive).
- Meilleur design TV toutes chaînes européennes avec « *Fox Kids Rides* ».

Phoenix Interactive est à l'origine d'une quinzaine de jeux vendus pour la plupart sur plusieurs continents.

Les derniers jeux réalisés par le studio sont :

- *Alexandra Ledermann, le Haras de la vallée* (2007, Ubisoft, Wii/PC)
- *Alexandra Ledermann, la colline aux chevaux sauvages* (2008, Ubisoft, Wii/PC)
- *Ciné Studio Party* (2008, Ubisoft, Wii)
- *Planète nature : Au secours de l'île tropicale* (2008, Ubisoft, DS)

Figure 3 : Alexandra Ledermann

Figure 4 : Planète nature

Contexte du projet

1. Sujet du stage

L'intitulé du sujet de stage défini avant mon arrivée dans l'entreprise était celui-ci :

« Développement de modules d'un jeu vidéo ».

Celui-ci étant très large, je ne connaissais ni la plateforme sur laquelle j'allais travailler, ni sur quel projet on allait me placer.
A mon arrivée, on m'a informé que j'allais rejoindre une équipe de développement pour la plateforme Nintendo DS en tant que programmeur gameplay. J'ai donc intégré rapidement une équipe pluridisciplinaire en pleine production qui travaillait sur un projet déjà assez avancé.

2. Projet

Le projet sur lequel j'ai travaillé est un « ***casual game*** » pour la console de jeu Nintendo DS, il est destiné à tout public, mais possède en particulier une cible jeune. Il s'agit d'un « ***party game*** », c'est-à-dire un jeu au concept simple, amusant et accessible, composé de nombreux mini jeux de courte durée.
Le jeu est divisé en mini-jeux, jouables en 4 niveaux de difficultés et organisés autour de quêtes qu'il faut terminer pour débloquer toutes les fonctionnalités (features). Chaque mini jeu a pour objectif un score à atteindre et dure entre 1 et 3 minutes. Tous les mini-jeux sont jouables uniquement avec le stylet et le micro de la console.
En plus du mode aventure, le jeu contient :

- Un mode libre ou il est possible de rejouer aux mini-jeux débloqués ;
- Un mode défi ou il faut enchainer plusieurs mini-jeux à la suite sans perdre ;
- Un mode village. Dans le mode village, le but est d'obtenir le meilleur village (meilleur score) en construisant des bâtiments que l'on a débloqués précédemment. Chaque bâtiment apporte un nombre de points spécifique supplémentaire au village. Plus le joueur avance dans le mode aventure, plus il aura accès à de gros bâtiments (plus de points pour le village).

Le jeu est entièrement développé en 2D et est basé sur un moteur de jeu développé en interne par l'équipe moteur DS.

3. Equipe

L'équipe avec laquelle j'ai collaboré est composée de plusieurs métiers sous la direction du chef de projet. On retrouve ainsi :

- des game designers, chargés de développer la forme, le concept et la jouabilité du jeu, ainsi que ses règles. Ils sont également les responsables du level design, qui consiste à créer les niveaux du jeu en insérant les éléments de gameplay aux endroits et moments adéquats ;
- des programmeurs moteur, qui développent le moteur de jeu sur lequel se base les programmeurs gameplay ;
- des programmeurs gameplay, dont j'ai fait partie, qui codent le jeu ;
- des intégrateurs, qui collectent et organisent les données graphiques et sonores et les intègrent au jeu ;
- des graphistes et animateurs, chargés de la création des visuels et des animations.

4. Plateforme

Le projet consiste à développer un jeu sur la console DS de Nintendo. La DS (pour Dual Screen) est une console portable composée de deux écrans, dont un tactile (l'écran inférieur). Tout le gameplay du jeu est basé sur la manipulation du stylet avec l'écran tactile.

Figure 5 : Nintendo DS Lite

Lors du développement du jeu, deux versions de la console étaient disponibles sur le marché : la DS et la DS Lite. Leurs spécifications sont visibles sur la figure 6. Aujourd'hui, une troisième version a été créée par Nintendo : la DSi, qui possède deux caméras ainsi que des écrans de tailles supérieures et un processeur plus rapide.

Spécifications de la Nintendo DS et de la Nintendo DS Lite			
Spécification		Description (DS / DS Lite)	
CPU		ARM946E-S - 32-bit RISC (67 MHz) ARM7TDMI - 32-bit RISC (33 MHz)	
Mémoire	RAM	4 Mo	
	VRAM	656 Ko	
Alimentation		Batterie Lithium-ion 6 à 10 heures d'autonomie 4 heures de charge	Batterie Lithium-ion 5 à 19 heures d'autonomie 3 heures de charge
Écrans		LCD semi-transparent 3.0" Rétro-éclairage	LCD transparent 3.0" 4 niveaux de luminosité
Affichage		Résolution de 256 x 192 pixels par écran 260 000 couleurs	
Fonctionnalités		Croix directionnelle + 8 boutons Écran tactile Microphone Haut-parleurs stéréo Prise casque Réseau sans fil IEEE 802.11 Ports cartouche DS / GBA	
Dimensions		14.9 cm x 8.5 cm x 2.9 cm	13.3 cm x 7.4 cm x 2.2 cm
Poids		280 g	218 g

Figure 6 : spécifications Nintendo DS

5. Outils

Les outils que j'ai pu utiliser lors du projet sont principalement ceux du kit de développement fourni par Nintendo pour la DS. Toute la partie programmation est effectuée en langage ***C++*** avec l'environnement de développement ***CodeWarrior***.
Le kit de développement est également composé de l'***émulateur IS Nitro*** (d'Intelligent System, filiale de Nintendo), qui permet de tester le jeu en exécutant le programme sur une DS.

Figure 7 : émulateur IS Nitro

J'ai également été amené à programmer en langage ***Lua***. En effet, le moteur de jeu inclut le scripting et permet d'utiliser des fichiers externes en langage Lua afin de définir les scènes du jeu ou les menus.

6. Milestones

Durant toute la durée de la production, nous avions un certain nombre de points de contrôle à passer et de versions du projet à envoyer à l'éditeur. Ces étapes sont très importantes et permettent de rendre compte du travail effectué et de la présence des fonctionnalités de jeu demandées par l'éditeur.

Etant arrivé dans le studio le 2 février, le premier milestone auquel j'ai été confronté a été la FPP, première version jouable du jeu. Toutes les versions qui ont suivies jusqu'à la Master candidate ont été des itérations successives de l'avancement du jeu.
Les master candidates NOE et NOA sont les versions finales (respectivement européenne et américaine) candidates à validation par le constructeur (Nintendo dans notre cas). Si une version n'est pas validée, il faut, pour l'équipe de développement, corriger les erreurs et tenter une nouvelle validation.
Juste avant mon départ, l'équipe à eut l'agréable surprise d'apprendre que la version européenne avait été validée dès la première soumission, ce qui est assez rare pour être souligné.
La dernière version est la gold master, c'est celle qui est destinée à être commercialisée.

Travail réalisé pour le projet

Ma première semaine de travail a consisté à lire de la documentation et à m'informer sur le projet en cours. J'ai lu de nombreux documents, concernant la programmation sur Nintendo DS, le moteur de jeu et le jeu en lui-même.

Le travail que j'ai réalisé pour ce projet au sein de l'équipe DS de Phoenix se divise en quatre grandes étapes après cette première semaine :

- Développement de mini-jeux
- Développement du menu principal
- Développement de l'échange de codes pour le mode village
- Débogage

Ces étapes n'ont pas été linéaires et se sont entrecoupées, de même, d'autres travaux ont également été effectués durant ces étapes, le planning suivant sert donc juste à donner un ordre d'idée du temps passé dans chaque développement et ne reflète pas vraiment la réalité.

Semaines	Travaux réalisés
1	Lecture de documents
2, 3, 4, 5, 6	Mini-jeux
7, 8, 9	Menu principal
10, 11, 12	Echange de codes
13, 14, 15	Travaux divers
16, 17, 18	Débogage
19, 20, 21, 22, 23, 24	Recherches graphiques

Figure 8 : planning

1. Mini jeux

Après avoir pris connaissance du projet, on m'a demandé d'analyser et de développer plusieurs mini-jeux. Au final, sur la totalité des mini-jeux, j'en ai programmé sept.
Tous les mini-jeux ont été développés par itérations successives :

- Analyse du jeu et de la méthode qui sera utilisée pour son développement ;
- Programmation du jeu avec des ressources temporaires ;
- Intégration des ressources graphiques ;
- Intégration des sons ;
- Intégration des effets graphiques (particules).

En parallèle de ces itérations, un réglage du gameplay et des valeurs des différentes variables est effectué après chaque test du game designer.

Un mini-jeu est en fait une scène qui est décrite dans un fichier Lua externe. Pour créer un mini-jeu, il faut donc commencer par créer un fichier Lua qui contient la scène (et tous ses paramètres) et tous les objets qui la composent. Ces objets peuvent être une caméra, des lumières, des personnages, des objets de jeu. Il est possible également d'associer à ces objets une ressource graphique (2D ou 3D) qui va permettre de le visualiser, ainsi qu'une IA, si nécessaire. Tous les objets dotés d'une IA vont pouvoir être programmés et avoir un comportement propre.
On associe à la scène elle-même une IA dans laquelle on va coder le comportement du mini-jeu correspondant.
Lors du chargement d'une scène, le moteur de jeu va se charger de créer tous les objets décrits dans ce fichier, ils vont ainsi pouvoir être récupérés en code par le programmeur gameplay.

Tous les jeux sont basés sur des machines à état, pour la boucle principale de fonctionnement du jeu, mais également pour les objets et personnages contenus dans le jeu. Ce mode de fonctionnement est particulièrement adapté est permet de simplifier la compréhension et le codage du gameplay.

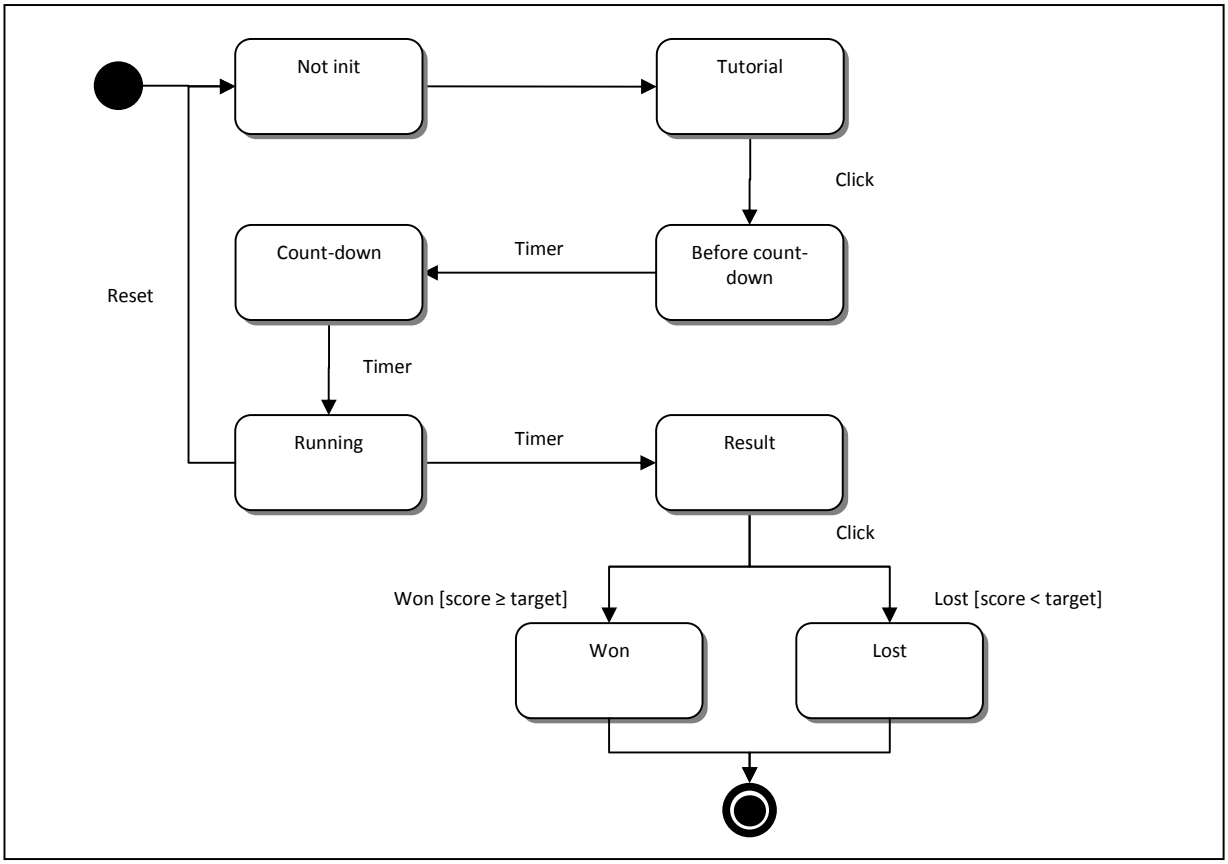

Figure 9 : machine à état des mini-jeux

Un mini-jeu est toujours basé sur la machine à état décrite dans la figure 5.

- L'état ***Not Init*** est l'état de départ du mini-jeu.
- Pendant l'état ***Tutorial***, on montre au joueur une vidéo et une explication du fonctionnement du jeu.
- ***Before count-down*** permet de jouer une animation d'introduction ou d'entrée des personnages.
- Dans l'état ***Count-down***, on affiche un compteur de 3 à 0 puis le jeu est lancé.
- L'état ***Running*** est l'état principal du jeu, c'est dans celui-ci que le jeu en lui-même se déroule.
- ***Result*** permet d'afficher un décompte du score et le résultat final au mini-jeu.
- En fonction de la défaite ou de la victoire du joueur au mini-jeu, on va passer dans l'état ***Won*** ou ***Lose***.

Mini-jeu 1 - attraper des objets

Dans le premier jeu que j'ai développé, le joueur déplace horizontalement un personnage en bas de l'écran à l'aide du stylet en cliquant la où il veut que le celui-ci se dirige. Le but est de

ramasser des objets qui tombent de quatre positions différentes avant que ceux-ci ne touchent le sol. Les objets qui tombent peuvent être des bonus ou des malus (à ramasser ou laisser tomber).
La principale difficulté pour ce mini-jeu a été de programmer le déplacement du personnage afin que celui-ci ralentisse quand il arrive à destination et ne se stoppe pas immédiatement (diminution progressive de la vitesse). La seconde difficulté a été de contrôler l'animation de déplacement du personnage afin que celle-ci s'adapte à sa vitesse.

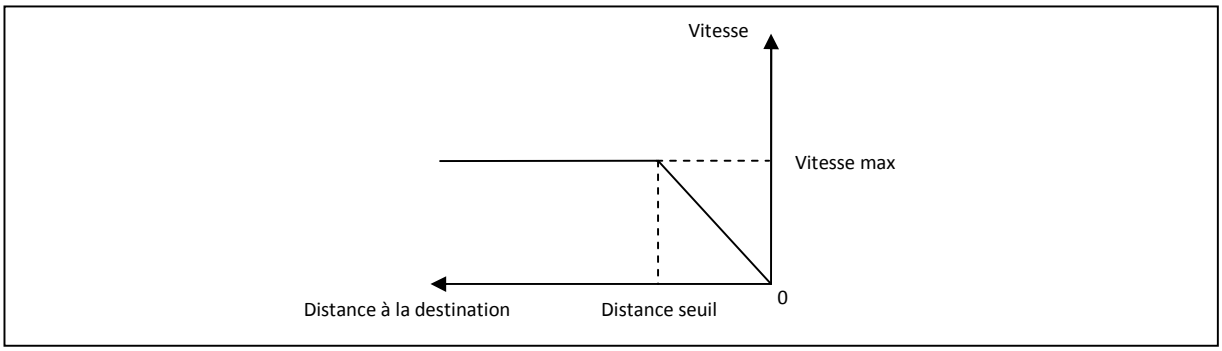

Figure 10 : vitesse en fonction de la distance à la destination

Mini-jeu 2 – scrolling

Dans le second jeu, des personnages avancent de gauche à droite dans un décor (scrolling), le joueur doit faire sauter ou s'agenouiller ces personnages afin que ceux-ci évitent des obstacles et ramassent des bonus.
Ici, la difficulté a été de concilier les animations et le gameplay. En effet, si on base le jeu uniquement sur les animations (attente de la fin de l'animation de saut pour pouvoir se baisser par exemple), celui-ci perd tout son répondant en termes de gameplay. Il a donc fallut trouver des solutions pour accorder les deux, en coupant certaines animations à des moments clés par exemple.
Une autre difficulté a été de régler les boîtes englobantes des personnages afin de gérer au mieux les collisions avec les obstacles.

Mini-jeu 3 - entourer les personnages

Ce jeu se déroule sur un écran fixe, le joueur doit entourer à l'aide de son stylet des personnages qui passent à l'écran en suivant des splines, afin de les capturer.
Ce jeu a nécessité un développement et une recherche algorithmique spécifiques, d'une part pour la reconnaissance de la forme tracée au stylet (encerclement), d'autre part pour l'affichage sur l'écran du tracé.

Pour l'encerclement, nous sauvegardons les points du stylet à intervalle de distance régulier. Si le segment formé par le nouveau point et le point précédent intersecte le tracé créé avant ou si le nouveau point se situe proche d'un autre point, nous pouvons fermer le tracé. Nous obtenons alors à la fin une liste de points qui forment un polygone.

L'algorithme de reconnaissance de l'encerclement utilisé est celui-ci :

```
soit la liste des points sauvegardés l
soit un point p donné par le stylet
si la distance entre p et le point précédent p-1 est supérieure à un seuil
alors
      on insère p à la fin de l
      pour chaque point p' de la liste l
            si p est proche de p' alors
                  on considère qu'il y a une boucle entre p' et p
            sinon
                  soit s le segment formé par p et p-1
                  soit s' le segment formé par p' et p'+1
                  si s intersecte s' alors
                        on considère qu'il y a une boucle entre p'+1 et p-1
                  fin si
            fin si
      fin pour
fin si
```

Comment savoir si un objet est à l'intérieur du cercle ?
Pour chaque objet présent à l'écran, on va effectuer un lancer de rayon vers la droite. On teste l'intersection de ce rayon avec chaque segment formé par deux points consécutifs du tracé.
Si le rayon a intersecté un nombre pair de segment, alors, il est à l'extérieur du cercle, sinon, il est à l'intérieur.

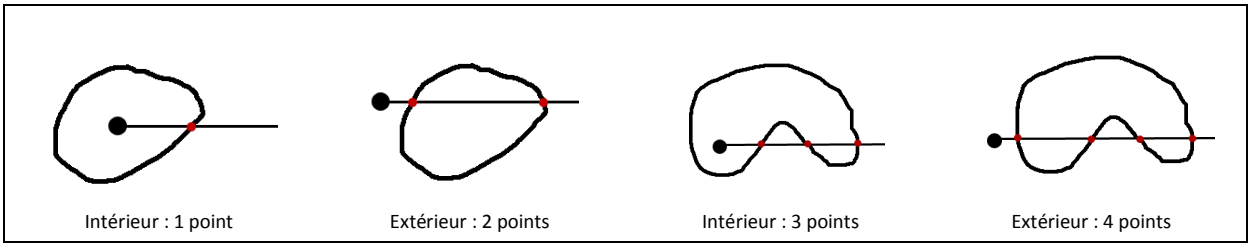

Figure 11 : déterminer si un point est à l'extérieur ou à l'intérieur d'un polygone

De base, rien n'est prévu au niveau du moteur de jeu pour dessiner sur l'écran d'une DS. Il a donc fallut trouver une solution pour résoudre ce problème.
Il existe une fonction pour transférer directement en mémoire vidéo (VRAM) une zone mémoire (depuis la RAM). Nous avons donc placé une texture transparente dans la VRAM qui est plaquée sur toute la zone de l'écran. Il ne nous reste alors qu'à modifier cette zone mémoire en transférant un tableau de valeurs stocké dans la RAM. Pour dessiner le tracé, nous dessinons en fait dans le tableau qui sera transféré à chaque frame.

La méthode de traçage a été la suivante : pour chaque point échantillonné du tracé, deux autres points sont définis de part et d'autre. On trace alors un quadrilatère avec ces deux points et les deux précédents. Un triangle est dessiné avec le dernier point de la liste afin de montrer la pointe du tracé.

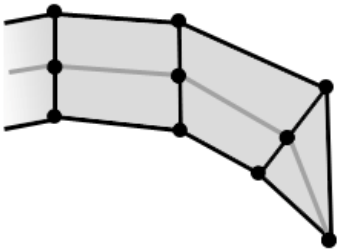

Figure 12 : méthode de traçage

Pour dessiner segments, quadrilatères et triangles, j'ai dû mettre en place des algorithmes de rastérisation en 2D (dessin à partir de points).

Mini-jeu 4 - catapulte

Dans ce mini-jeu, le rôle du joueur est de diriger une catapulte qui se situe au centre de l'écran afin de tirer sur des ennemis qui traversent la scène. En touchant l'écran avec le stylet, il va armer et orienter la catapulte, en relâchant le stylet, la catapulte va tirer un projectile. Le but du jeu est de toucher un maximum d'ennemis.

Mini-jeu 5 - gâteaux

Le but de ce jeu est de confectionner des gâteaux. Le joueur doit apporter les ingrédients à un personnage. Quand le personnage a assez d'ingrédient, il se déplace en direction du gâteau. Il faut alors le pointer avec le stylet pour lui faire préparer le gâteau.
Le développement de ce jeu n'a pas été compliqué de par l'architecture, mais il a nécessité de nombreuses animations qu'il a fallut régler de manière très pointue afin d'obtenir un résultat satisfaisant (animation de déplacement, de prise d'ingrédients, de préparation du gâteau, etc.).

Mini-jeu 6 - cartes

Ce mini-jeu est composé de plusieurs tableaux avec des cartes. Le but, ici, est de retrouver une carte cible parmi plusieurs cartes affichées à l'écran. Une carte avec un motif spécifique est montrée au joueur dans une zone en haut de l'écran. En dessous, de nombreuses cartes sont affichées et peuvent être en mouvement. Le joueur doit retrouver la carte recherchée le plus rapidement possible et cliquer dessus avec le stylet.

La difficulté, ici, a été de faire en sorte que le level design ne soit pas trop important et que la création des tableaux ne soit pas trop fastidieuse. En effet, décrire toutes les cartes ainsi que leur déplacement peut vite créer de nombreuses lignes de code source.

Mini-jeu 7 - tir

Ici, trois personnages sont dans un stand de tir, ils sont représentés en bas de l'écran. En face d'eux, des cibles apparaissent et disparaissent aléatoirement sur deux hauteurs. En appuyant au stylet sur un des trois personnages, le joueur va faire apparaitre un viseur en face de lui, qui va osciller verticalement de manière sinusoïdale. Il faut alors relâcher le stylet quand le viseur est en face d'une cible afin que le personnage envoie un projectile sur celle-ci. Les cibles peuvent être amies ou ennemis. Pour obtenir un maximum de points, il ne faut pas toucher les cibles amies.

Cette première partie m'a permis de me familiariser avec le projet, le moteur de jeu et le développement sur Nintendo DS. J'ai dû aussi apprendre à concevoir des machines à états, utilisées fréquemment dans ce genre de programmation.
Malgré le fait que tous les jeux soient basés sur la même architecture, leur nombre et leurs spécifications propres m'ont permis de toucher du doigt de nombreux mécanismes de jeu comme le déplacement de personnages et la reconnaissance de mouvements au stylet.

2. Menu général

La seconde grande partie de mon stage fut le développement du menu principal du jeu. Celui-ci commence après la vidéo d'introduction et permet d'accéder à tous les modes de jeux via une succession d'écrans.

Fonctionnement des menus

Le moteur de jeu gère les menus dans une pile, l'enchainement des écrans est donc réalisé simplement avec des instructions push et pop. L'instruction push permet de placer un menu sur la pile et l'instruction pop permet de retirer le menu situé en haut de la pile. Le menu affiché à l'écran est toujours celui qui se situe sur le haut de la pile. Cette façon de faire permet un déplacement logique de l'utilisateur dans l'arborescence des différents menus et une programmation simplifiée.

Le moteur nous permet de créer plusieurs éléments dans les menus (widgets) :

- Boutons cliquables
- Images
- Textes
- Groupes de widgets

Un menu est entièrement défini dans un fichier Lua. Pour créer un nouveau menu, il suffit donc de le décrire dans un fichier Lua externe, et définir tous les widgets qui le composent (position, taille, etc.). Le moteur se charge d'interpréter ces fichiers de définition et de créer automatiquement les objets correspondants au sein du programme.

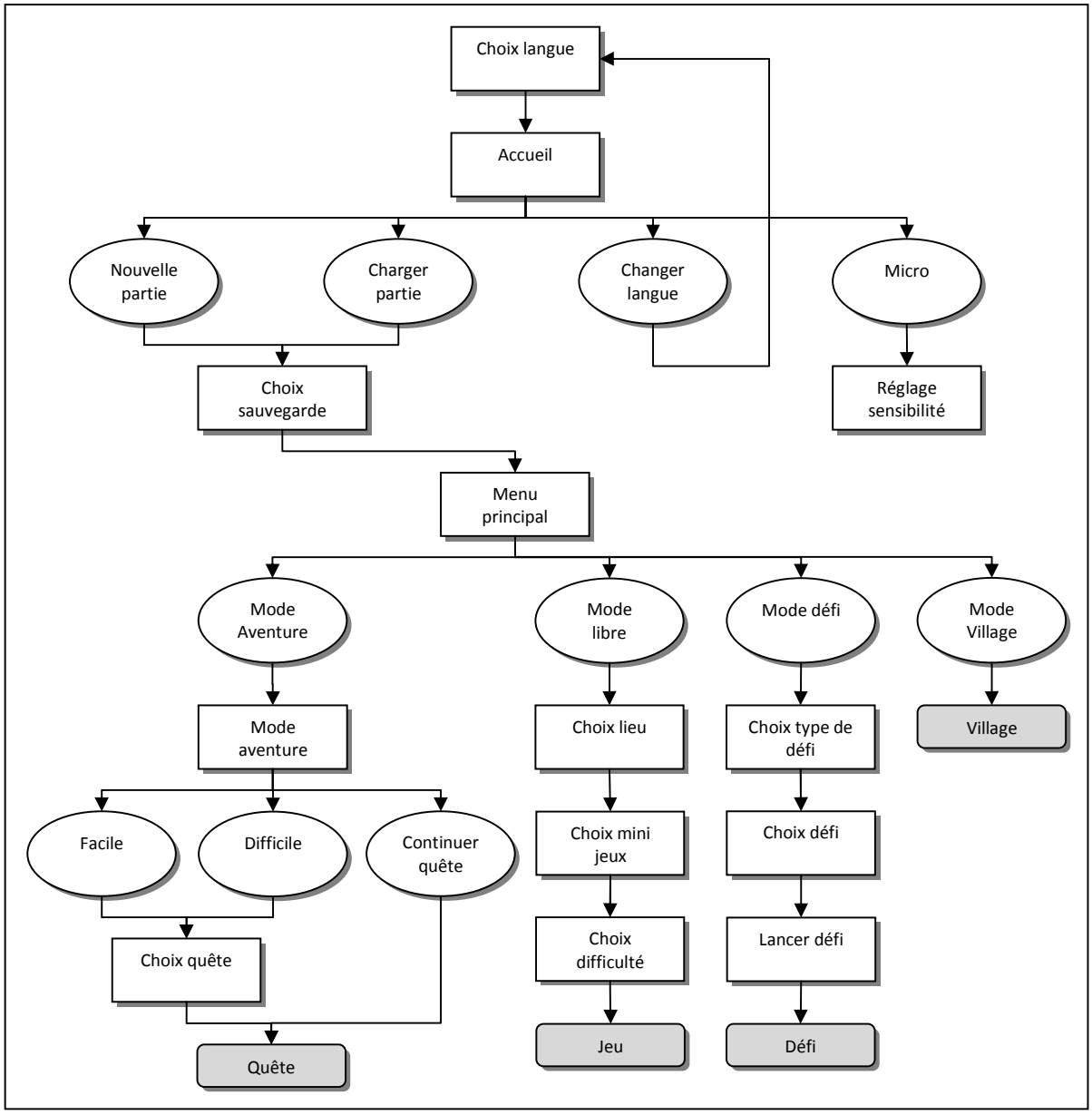

Figure 13 : diagramme du menu principal

Développement

Dans un premier temps, il a fallut que j'étudie l'architecture globale du jeu afin de savoir comment intégrer le menu principal.
Après réflexion, nous avons convenu que le menu général serait considéré comme un mode de jeu, et serait un état à part entière dans la machine à état globale du jeu (tout comme le mode aventure, le mode défi, le mode libre et le mode village).

J'ai ainsi créé un manager de menu principal (singleton), dont le but et de savoir quel est l'état du menu à chaque instant. C'est par lui que le jeu passe pour interagir avec le menu, pour définir sur quel page le joueur doit entrer, etc. C'est lui également qui contient les variables internes au menu comme le mode de jeu courant par exemple, et qui se charge d'initialiser la pile des menus.

Ensuite, mon travail a consisté à créer et optimiser les fichiers Lua en créant des fonctions génériques pour afficher chaque type de boutons, icônes, ou encarts. Grâce à ces fonctions contenues dans un fichier commun séparé, nous pouvons avoir des fichiers de menus relativement petits et simples à mettre en place pour le développeur.
Dans ce fichier commun, nous stockons également toutes les variables globales de positions, de tailles, de numéros de frames, afin de pouvoir les changer rapidement lorsqu'il faut modifier l'état d'un widget.

Une fois le menu créé dans le fichier Lua, il faut le récupérer dans le programme du jeu pour coder les actions de chaque bouton. Chaque click sur un bouton lance un évènement qu'il faut récupérer et auquel on peut associer des actions (ouvrir un nouveau menu, revenir au menu précédent, lancer un mode de jeu, etc.).
Nous pouvons également, en code, créer des widgets dynamiquement lors de l'ouverture d'un menu et les supprimer lors de sa fermeture. Cela permet de libérer la mémoire vidéo entre chaque page de menu (Ce qui est très important lorsqu'on travaille sur une petite plate-forme telle que la Nintendo DS, qui possède très peu de mémoire). Les allocations dynamiques ne sont pas descriptibles dans les fichiers Lua, pour cette raison, nous avons été obligés de le faire au sein même du code source.

Animation

Les fichiers Lua n'étant que des fichiers de description, l'animation des widgets ne peut se faire que dans le code source du jeu. Une autre partie de mon travail sur les menus a donc été de chercher un moyen simple pour animer toutes les pages sans pour autant réécrire le même code dans chaque classe de menu et pour chaque widget.
La solution trouvée a consisté à créer une classe mère de toutes les classes de menu qui se charge d'animer les widgets contenus dans la page.
Pour chaque menu, il suffit ainsi de définir quel sont les widgets qui vont être animés automatiquement, quels sont leur point de départ et leur point d'arrivée (Les animations sont des déplacements linéaires d'un point à un autre de l'écran) et quelle est leur vitesse de déplacement. Une méthode de la superclasse va de cette manière permettre d'effectuer l'animation lorsqu'elle sera appelée depuis une classe fille.
En plus de cela, la classe mère contient de nombreuses méthodes pour connaître l'état de l'animation et pour animer un seul widget indépendamment des autres.
Pour simplifier encore le code source et donner encore plus de flexibilité au programme, nous laissons la possibilité de définir le mouvement des widgets via une variable dans les fichiers Lua.

Ainsi, il est très facile de modifier les animations en changeant cette variable dans le fichier externe sans avoir à toucher au code du programme.

Grace à cette méthode, nous avons pu très facilement et rapidement mettre en place un système de menus animés. Lorsqu'un menu est affiché, nous faisons entrer les widgets en les déplaçant de l'extérieur de l'écran vers leur position dans l'écran. Lorsque l'on quitte le menu, tous les widgets effectuent le déplacement dans le sens inverse et sortent de l'écran.

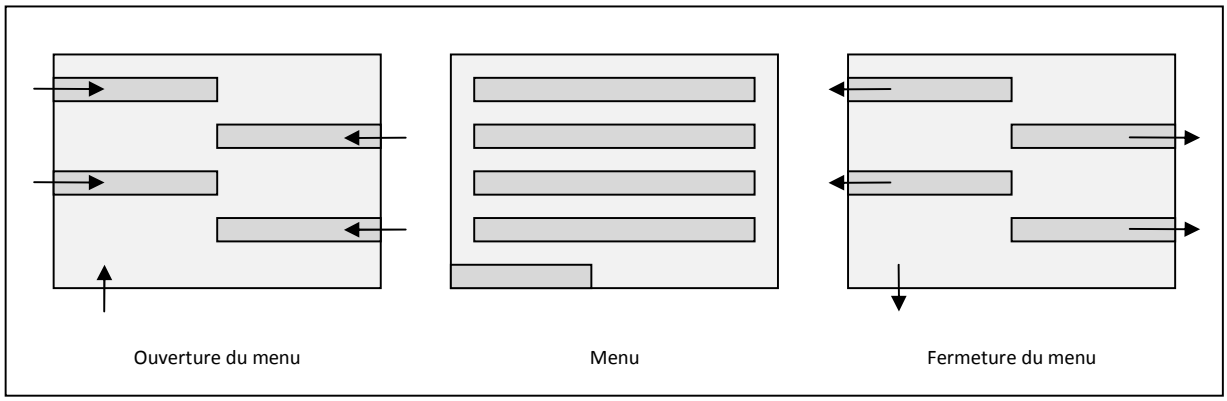

Figure 14 : animation des widgets des menus

Cette étape du projet m'a permis d'effectuer mes premières intégrations dans l'architecture du jeu en créant le menu principal. J'ai également pris connaissance du fonctionnement de menus par piles qui simplifie grandement la conception.
Pour finir, le menu principal permet d'accéder à toutes les fonctionnalités du jeu et d'unifier tout ce qui a été créé précédemment.

3. Echange de code

Dans le mode village, le joueur doit construire des bâtiments grâce aux plans et à l'argent qu'il a gagné en mode aventure. Le village occupe plusieurs écrans de DS, le joueur peut naviguer parmi ces écrans pour visualiser l'état de ses constructions. Chaque écran possède un certain nombre d'emplacements de construction prédéfinis.
Dans le cahier des charges de ce mode de jeu, il est spécifié que le joueur doit pouvoir échanger des plans de bâtiments avec ses amis (entre deux DS) et doit pouvoir également visualiser le village de ses amis sur son écran.

Après avoir analysé cette demande, nous avons convenu que ces échanges se feraient avec des codes secrets.

Pour la visualisation du village, un code alphanumérique est généré à partir de l'état du village. Le joueur qui va taper ce code sur sa DS va ainsi pouvoir visualiser le village correspondant. Le code généré peut être utilisé par n'importe qui, cela ne pose pas de problème particulier puisqu'il ne s'agit que de visualisation (pas de triche possible).

Pour le don de plans de bâtiments, il a fallut trouver une solution pour être certain que le code généré ne puisse être utilisé que par le joueur ami.
La solution consiste en un échange de code. Le jeu qui va recevoir un plan va ainsi générer un code secret. Ce code secret va être entré par le joueur qui donne le plan et le code final va être créé à partir de ce premier code. Dans le jeu receveur, nous allons ainsi pouvoir vérifier que le premier code correspond bien.
De cette manière, nous pouvons être certain qu'il y'a bien un échange entre seulement deux joueurs et que le code ne va être utilisable qu'une seule fois, ce qui évite les possibilités de triche.

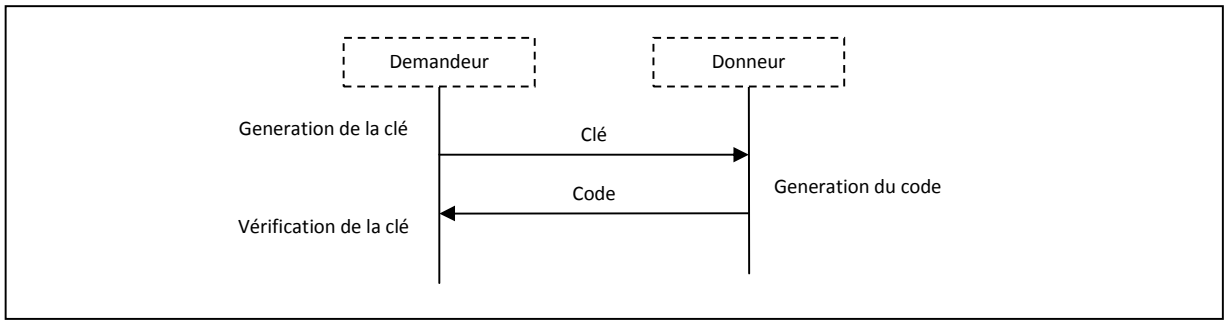

Figure 15 : échange de code

Durant toute cette partie, mon objectif a été de mettre en place ces différents codes et leurs mécanismes d'échanges.

Analyse du village

La première étape a été d'analyser la construction du village pour pouvoir déterminer le nombre de caractères alphanumériques dont nous aurions besoin pour le code.
Le village est composé de trente huit emplacements de construction, chaque emplacement ne peut recevoir qu'un seul type de bâtiment et chaque type de bâtiment possède entre quatre et six bâtiments.
A chaque emplacement, on a donc un nombre de quatre à six bâtiments possibles. Le code doit être assez grand pour pouvoir contenir toutes les possibilités de villages différents.
Après quelques calculs, nous en arrivons à la conclusion qu'il faudra 16 octets (128 bits) pour pouvoir contenir tout le code, ce qui nous laisse $3{,}4.10^{38}$ possibilités, c'est au dessus de ce que nous avons besoin réellement, mais nous devons garder une marge pour inclure un CRC.
En passant d'un code binaire à un code alphanumérique en base 62 (26 lettres minuscules + 26 lettres majuscules + 10 chiffres), il apparaît que nous aurons besoin de 20 caractères pour pouvoir représenter toutes les possibilités ($62^{20} < 2^{128}$).

Cette analyse effectuée, nous savons maintenant qu'il nous faudra gérer des grands nombres (16 octets) et que le code final que l'utilisateur échangera aura une taille de 20 caractères.

Gestion des grands nombres

Après avoir effectué l'analyse préliminaire, j'ai donc pu commencer à créer une classe pour gérer les grands nombres. Cette classe nous permet de manipuler des nombres de 16 octets (128 bits) non signés. Elle est composée de quatre entiers non signés de 32 bits.
Dans cette classe, j'ai implanté toutes les méthodes nécessaires à son utilisation :

- Opérations classiques :
  - Addition
  - Soustraction
  - Multiplication
  - Division
  - Modulo (reste de division)
- Opérations bits-à-bits :
  - Et
  - Ou
  - Ou exclusif
  - Décalage à gauche
  - Décalage à droite

Tous ces opérateurs sont basés sur des calculs entre nombres entiers de 32 bits non signés, pour cette raison, ils sont très efficaces et de complexité algorithmique très faible.
Avec toutes ces méthodes, nous allons pouvoir aisément manipuler notre code de village.

Recherches sur les bases arithmétiques

Pour encoder et décoder le village, il faudra effectuer des opérations arithmétiques, notamment de changement de base pour insérer ou extraire un chiffre du code.

Un nombre dans une base b donnée s'écrit sous la forme d'additions des puissances successives de cette base. Le nombre $c_n...c_2c_1c_0$ *en base* b*, constitué des chiffres* c_n*, ...,* c_2*,* c_1*,* c_0*, peut aussi s'écrire sous la forme* $c_nb^n + ... + c_2b^2 + c_1b^1 + c_0b^0$*, c'est-à-dire un polynôme dont les coefficients sont les chiffres et l'inconnue est la base. (Wikipedia)*

Dans notre cas, le nombre final est le code, b est la base dans laquelle nous travaillons et les c_i sont les informations contenues dans le code.

$$0 \leq c_i < b$$

La base b va donc être choisie en fonction des informations que l'ont voudra stocker dans le code.

A partir de cette définition, il est possible de mettre en place deux méthodes, l'une pour insérer un chiffre (une information) d'une certaine base dans le code, l'autre pour extraire un chiffre (une information) du code.

Pour insérer une information dans le code (un chiffre à droite dans le polynôme), il faut déplacer d'abord tout le polynôme vers la gauche, c'est-à-dire augmenter son degré de 1. Pour déplacer le polynôme vers la gauche, il suffit de multiplier celui-ci par la base b.

$$c_j b^i * b = c_j b^{i+1}$$

Toutes les informations vont ainsi être déplacées vers la gauche du code.
La puissance 0 se retrouve donc libre, on peut insérer une nouvelle information (un nouveau chiffre c_0) en effectuant une simple addition.

Pour extraire une information du code, nous allons procéder à l'inverse de l'insertion. Le but étant d'extraire le c_0. Pour obtenir cette information, il faut effectuer la division entière du polynôme par la base.

$$c_n b^n + \cdots + c_2 b^2 + c_1 b^1 + c_0 b^0 = qb + c_0$$

Le reste c_0 de la division entière du code par la base est l'information recherchée.
Pour trouver c_1, il faut effectuer la même opération à partir de q, qui est le polynôme après division.

$$q = c_n b^{n-1} + \cdots + c_2 b^1 + c_1 b^0 = q'b + c_1$$

On réitère ainsi l'opération jusqu'à extraire toutes les informations contenues dans le code.

Encodage

Le but de l'encodage est de créer le code de 128 bits à partir de l'état du village. Pour cela, nous parcourons tous les emplacements du village. Pour chaque emplacement, on connaît le nombre de bâtiments différents qu'il est possible de construire, nous allons travailler dans la base arithmétique correspondante.
L'idée est d'insérer dans le code, à l'aide de la méthode arithmétique décrite précédemment, le chiffre correspondant au numéro du bâtiment contenu dans l'emplacement. Ce chiffre est compris entre zéro et le nombre de bâtiments possibles pour cette case.

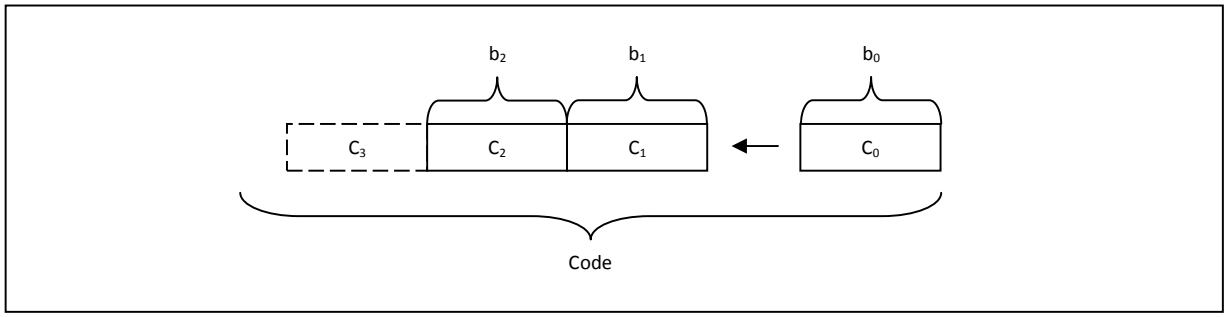

Figure 16 : insertion du bâtiment dans le code

Exemple : pour l'emplacement 3, il y'a 5 possibilités de bâtiments différents. Le joueur a construit ici le bâtiment numéro 2 parmi les 5 possibles. Nous travaillons donc en base 5 et nous insérons le chiffre 2 à la droite du code.

En traitant de cette manière tous les emplacements, nous obtenons notre code de 128 bits qui contient le numéro de bâtiment de chaque emplacement du village.

```
Soit c le code de 128 bits
c <- 0
Pour chaque emplacement
      Soit b le nombre de bâtiments possibles (la base)
      Soit n le numéro du bâtiment contenu dans l'emplacement (0 ≤ n < b)
      c <- c x b (décalage à gauche)
      c <- c + n (insertion du bâtiment)
Fin pour
```

Grâce à cette méthode, tous les bits du code sont utiles et aucun espace n'est perdu pour le stockage des données.

Une fois ce code généré, nous devons lui ajouter un CRC (Contrôle cyclique de redondance) afin de pouvoir vérifier lors de l'échange si le code est valide et s'il ne contient pas d'erreur. Le polynôme générateur est choisi sur 16 bits afin de pouvoir être inséré à la droite du code sans provoquer une perte de données (le code a été prévu assez grand pour pouvoir contenir un CRC de 16 bits).

Le CRC est calculé en effectuant le reste de la division du code du village par le polynôme générateur de 16 bits, puis est inséré à la droite du code. Lors de l'échange de code, nous pourrons ainsi vérifier, en recalculant et en comparant les CRC, que le code ne contient pas d'erreur.

Pour complexifier encore un peu plus le code, nous le cryptons avec une clé stockée dans le programme. Cela permet de cacher au joueur le fonctionnement du code et les différentes parties qui le composent.

Enfin, la dernière étape consiste à convertir le nombre de 128 bits en une chaîne de 20 caractères qui puisse être présentée au joueur. Pour faire cette opération, il suffit de changer de base arithmétique en passant à une base 62 (alphabet minuscule et majuscule et chiffres).
Les chiffres en base 62 sont obtenu à partir de la méthode arithmétique d'extraction décrite précédemment, en divisant successivement le code par 62. Le reste de la division correspond au caractère recherché.

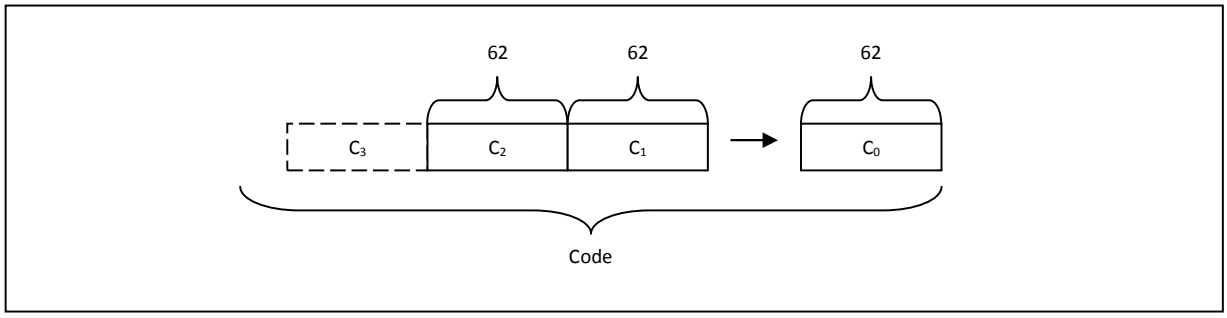

Figure 17 : extraction des caractères du code

A la fin, nous obtenons bien une suite de 20 caractères alphanumériques qui contient le code du village avec, sur chaque emplacement, le bâtiment correspondant.

Décodage

Pour décoder le code, il faut effectuer toutes les opérations qui ont permis le codage dans le sens inverse.

On commence donc par créer le code à partir de la chaîne de 20 caractères. Toujours avec la même méthode arithmétique en base 62, nous insérons successivement dans le code les 20 caractères.

La seconde étape est le décryptage à l'aide de la clé utilisée lors du cryptage.

Ensuite, il faut extraire le CRC contenu à la droite du code et recalculer celui du code restant. Si les deux CRC sont égaux, il n'y a pas eu d'erreur, nous pouvons continuer, sinon, il faut afficher un message pour dire au joueur que le code n'est pas valide.

La dernière étape est de parcourir tout le code et d'extraire toutes les informations sur chaque emplacement afin de reconstruire le village en plaçant les bons bâtiments aux bons endroits. L'extraction des informations est toujours effectuée avec la même méthode arithmétique.

```
Soit c le code de 128 bits contenant les informations
Pour chaque emplacement
      Soit b le nombre de bâtiments possibles (la base)
      Soit n le numéro du bâtiment contenu dans l'emplacement (0 ≤ n < b)
      n <- c % b (reste de la division de c par b)
      c <- c / b (décalage à droite)
Fin pour
```

Plans de bâtiment

Pour l'échange de plan de bâtiment, le principe global reste le même, excepté le fait que le code doit contenir une clé générée par la DS receveuse.
Le nombre de plans étant égal à une vingtaine, il n'est pas nécessaire de travailler avec des grands nombres de 128 bits. Pour simplifier les traitements, la compréhension et la facilité d'utilisation, le code d'échange sera numérique uniquement et sa taille sera de 6 caractères, nous pourrons ainsi travailler en base 10.
Le code est composé de :

- 3 chiffres pour la clé générée
- 2 chiffres pour le numéro du plan
- 1 chiffre pour un CRC

La première étape est de générer la clé à 3 chiffres sur la console qui va recevoir un plan. Cette clé est générée de manière aléatoire.
Lorsque le joueur receveur a généré et donné sa clé et que le joueur donneur l'a rentrée et a sélectionné le plan qu'il voulait donner, nous pouvons générer un code à 5 chiffres contenant ces informations.
A partir de ce code à 5 chiffres, un CRC de 1 chiffre est créé et ajouté au code.

L'étape suivante est le cryptage avec une clé à 6 chiffres contenue dans le programme.

Une fois le code généré, le joueur receveur peut le rentrer dans sa DS. A partir de là, nous passons au décodage en décryptant le code à l'aide de la clé de cryptage.

Nous vérifions ensuite le CRC afin d'éviter les erreurs de code, et la clé à 3 chiffres générée précédemment sur cette DS. Si ces deux vérifications sont réussies, nous pouvons extraire le numéro à 2 chiffres correspondant au plan qui a été donné.

Toute cette partie sur la génération et l'échange de code m'a donné l'opportunité d'effectuer des recherches sur l'arithmétique et l'utilisation de CRC pour détecter les erreurs, domaines sur lesquels je n'étais pas à l'aise par le passé. J'ai donc énormément appris durant cette période pour pouvoir mener à bien les tâches qui m'étaient assignées. Cette étape a été, selon moi, l'une des plus intéressantes de ce stage.

4. Débogage

La dernière partie de ma contribution au projet a concerné le débogage de l'application. Cette étape est très importante dans un but de validation du jeu par l'éditeur et le constructeur, elle est donc très bien organisée.
Durant toute la fin du développement du jeu, une équipe de testeurs externe au studio (employée par l'éditeur) est mise à contribution. Elle est chargée de traquer le moindre bug et de le reporter dans une base de données accessible à tous. Chaque personne travaillant sur le projet possède des droits différents sur cette base. Dans cette base, les bugs sont répertoriés par sévérité :

- S0 : bug critique ou bloquant ;
- S1 : bug important ;
- S2 : bug moyen ;
- S3 : bug n'apportant pas de gène pour l'utilisateur ;

Et par métier :

- Programmation ;
- Game design ;
- Graphisme.

Les bugs possèdent un état :

- Ouvert ;
- Corrigé ;
- Impossible à reproduire ;
- Ne sera pas corrigé ;
- Besoin de plus d'information ;
- Corrigé et validé ;
- Fermé.

Chaque bug est également assigné à une personne, cependant il n'est pas question de se limiter à ses propres bugs, il faut tenter d'en corriger le plus possible (selon son métier évidemment).
Pour cette raison, j'ai été amené à lire, comprendre et corriger du code source qui avait été écrit par d'autres, ce qui est très formateur. A la fin de cette période, chaque programmeur a ainsi une connaissance parfaite du projet et de son architecture dans son ensemble.

L'étape de débogage est une phase très importante du développement d'un logiciel que je ne connaissais pas, j'ai donc beaucoup appris, du fonctionnement d'une base de bug au procédé très méthodique de correction.

5. Autres tâches

Bien que les tâches précédemment décrites aient été le fil conducteur de ma participation au projet, mon travail ne s'est pas limité à ces grandes parties. J'ai également programmé d'autres fonctionnalités moins importantes en tailles et en temps de développement.

Parmi elle, on peut citer le HUD des mini-jeux (Head Up Display : écran haut pendant le jeu contenant des informations comme le score). Cet écran, qui est le même quelque soit le mini-jeu, a été modifié de nombreuses fois au cours du développement pour pouvoir s'accorder avec le game design.

Avec le HUD s'est accompagnée la gestion des combos pendant les mini-jeux et leur représentation. Les combos sont également génériques à tous les jeux. Durant une partie, si le joueur amasse des points sans perdre, il obtient un bonus qui va lui permettre d'atteindre plus rapidement le score cible.

Pendant le développement du menu principal, j'ai également codé le mode défi. Dans ce mode, plusieurs jeux sont lancés à la suite et le joueur a pour objectif de les terminer tous sans perdre une seule fois. J'ai intégré ce mode dans l'architecture générale du jeu et programmé son fonctionnement ainsi que les différents menus qui le composent.

J'ai également participé à la création des menus du mode village, et plus particulièrement les menus d'échange de codes. Une des difficultés de ce menu est la création d'un clavier virtuel sur lequel le joueur tape son code. Un clavier implique de nombreux objets affichés à l'écran (touches du clavier, boutons et décorations), ce qui n'est pas compatible avec la faible mémoire d'une DS. De plus, il faut récupérer chaque pression sur une touche du clavier et afficher une surbrillance dessus. Ce menu a donc été bien réfléchi à l'avance et optimisé afin que le résultat soit à la hauteur des attentes du game design et du graphisme.

Dans un jeu sur console, il y'a un certain nombre de points sur lesquels on ne peut pas transiger et qui doivent obligatoirement apparaître dans la version finale. Ceux-ci sont imposés et vérifiés par le constructeur (ici Nintendo). Ces fonctionnalités sont d'ailleurs clairement énoncées dans un document spécifique envoyé à chaque studio de développement. Chez Nintendo, ces points sont appelés Lotchecks, ils portent un nom différent chez les autres constructeurs comme Sony. Parmi ces points obligatoires, on retrouve les messages d'erreurs lors de sauvegardes erronés ou d'erreur de lecture de la cartouche de jeu.
Durant quelques semaines, mon travail a été de programmer le chargement et la sauvegarde des données de la partie du joueur et de vérifier que les Lotchecks correspondants étaient bien respectés. Ces vérifications doivent être faites avec le plus grand sérieux et apportent une certaine responsabilité puisque pouvant être cause de non validation du jeu.

Durant près de dix-huit semaines, j'ai donc participé à la réalisation d'un jeu sur Nintendo DS en apportant le savoir-faire acquis à l'UTBM. J'ai programmé sept mini-jeux, créé le menu principal et son système d'animation et développé tout le système d'échange de codes. J'ai également participé activement au débogage du logiciel avant la soumission du jeu à Nintendo et effectué quelques autres tâches comme le mode défi, le mode libre, le HUD et les sauvegardes.
Le respect des délais et des méthodes de travail nous ont permis d'envoyer chaque version à temps à l'éditeur, tout en conservant une certaine qualité dans le logiciel. Une preuve de la qualité de notre travail a été la validation immédiate par Nintendo de la version européenne.

Après le projet

Le projet auquel j'ai participé s'est terminé bien avant la fin de mon stage. Lorsque plus aucune tâche n'était à effectuer sur le jeu, j'ai été transféré dans l'équipe moteur de jeu DS.

Mon travail au sein de l'équipe moteur a consisté à effectuer des recherches graphiques sur l'effet de plasma sous-marin, le toon shading, les effets de caustiques (effets de lumière formés avec l'eau) et la création d'un tracé du stylet à l'écran.

1. Effet plasma

Le premier effet graphique recherché est un effet animé de plasma, qui, appliqué sur une texture d'eau en arrière-plan, donne une impression de mouvement d'eau et de lumière.
Le principe est d'afficher une grille de polygones rectangulaires et de faire évoluer la couleur des vertex qui la composent de manière sinusoïdale. La couleur des polygones est interpolée lors du rendu en fonction de la couleur des vertex qui le définissent. Sur DS, le rendu est effectué selon la méthode d'illumination de Gouraud (illumination par interpolation linéaire de couleurs).

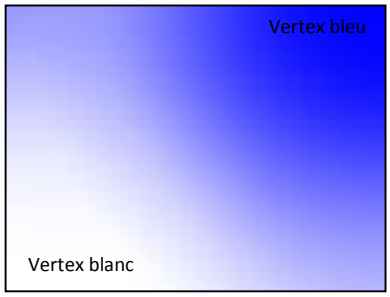

Figure 18 : couleur des polygones selon les vertex

On affiche ensuite la texture d'arrière plan sur la grille, ce qui va avoir pour effet de mélanger la couleur de la grille avec la couleur de la texture.
En faisant évoluer la couleur de chaque vertex du blanc au bleu suivant une sinusoïde de phase et de fréquence aléatoires, nous obtenons un effet de mouvement appelé effet plasma.

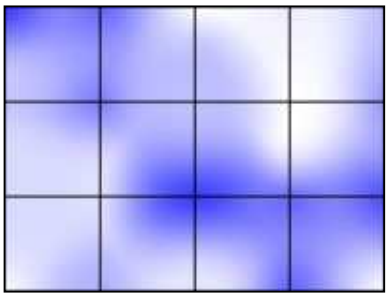

Figure 19 : effet plasma (animé)

Figure 20 : texture d'arrière plan

2. Toon shading

Le second effet testé a été le toon shading. Le toon shading est une technique de rendu présent sur DS qui consiste, lors du calcul d'illumination, à remplacer une plage de valeurs par une couleur définie dans une table de trente deux valeurs. Le but est d'obtenir un effet cartoon.

Figure 21 : toon shading

J'ai effectué plusieurs tests de modification dynamique des valeurs de la table de toon shading de manière sinusoïdale afin de simuler un effet plasma tel que précédemment.
Ces différents tests n'ont aboutis à rien de concluant concernant les effets sous-marin, mais nous ont tout de même permis de tester le fonctionnement de cette méthode de rendu.

3. Effet de caustiques

La troisième phase de recherche a concerné les effets de caustiques. Ceux-ci sont le résultat du reflet de la lumière à la surface de l'eau.

Figure 22 : effet de caustiques

Afin d'obtenir cet effet, nous avons superposé une texture animée qui représente les caustiques et les textures du sol et des objets. De la même manière qu'avec la grille de plasma, les deux textures superposées sont mélangées pour n'en former au final qu'une seule.

Afin d'obtenir un effet encore plus réaliste, nous projetons la texture de caustiques sur les objets. En projetant la texture, celle-ci reste sur place lorsque les objets se déplacent, elle est fonction de la position dans la scène et ne reste pas « collée » aux objets. Cela correspond à la réalité où la lumière est projetée sur les objets depuis la surface.
Cette méthode implique de changer la matrice de textures afin que les coordonnées de texture soient générées à partir des coordonnées x et z de la scène (cf. figure 23).

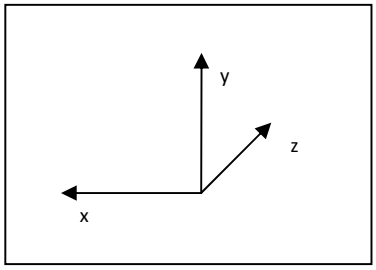

Figure 23 : repère DS

Cependant, pour pouvoir utiliser ce procédé sur DS, il faut obligatoirement superposer deux polygones identiques possédant chacun une texture (un polygone normal et le polygone de caustiques), ce qui implique de doubler quasiment tous les polygones de la scène (en réalité, on va doubler ceux qui se trouvent sur le dessus des objets). La console portable ne possédant pas une énorme capacité mémoire (2000 polygones par scène maximum), cette technique est donc très coûteuse, elle devra être utilisée avec parcimonie.

4. Tracé du stylet

Le dernier point sur lequel j'ai travaillé a été le dessin d'un tracé dans une scène en 3D afin de pouvoir visualiser le déplacement du stylet à l'écran.

Mon objectif a été de reprendre l'algorithme utilisé pour le mini-jeu dont j'étais responsable lors de la production précédente (encerclement avec dessin dans une texture en 2D) et de l'adapter à une scène en 3D.
Dans mon précédent algorithme, je dessinais dans une texture. Ici, les polygones sont dessinés directement par la DS dans la scène. Le reste de l'algorithme reste le même : échantillonnage des points donnés par le stylet en fonction de la distance et création des polygones correspondants.
J'ai amélioré cet algorithme en permettant de paramétrer la longueur des triangles à chaque extrémité en rétrécissant peu à peu la longueur du segment qui défini les polygones (cf. figure 24). Les polygones du tracé sont tous créés à partir de segments parallèles, ce qui donne un effet calligraphique. Je l'ai également amélioré en permettant à l'utilisateur un ré-échantillonnage entre deux points lorsque ceux-ci sont trop éloignés.
Pour finir, une texture est appliquée et répétée sur l'ensemble des polygones du tracé.

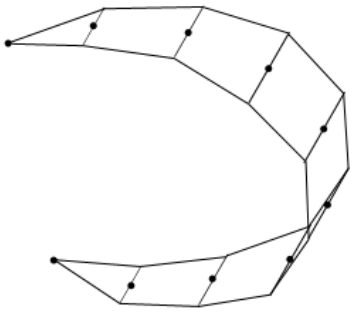

Figure 24 : tracé du courant

Cette phase de recherche m'a donnée l'occasion de bien comprendre les fonctionnalités de la console de par la lecture de la documentation officielle de Nintendo, mais aussi par une programmation bas niveau. J'ai pu appréhender également le moteur de jeu et son architecture dans son ensemble. De plus, une approche très bas niveau des problèmes est très intéressante et instructive et permet de bien comprendre les mécanismes de génération de scène et de rendu 3D et 2D. Durant cette période, j'ai donc beaucoup appris quand à l'architecture de la console de jeu et d'un moteur de jeu.

Conclusion

Mon stage dans l'équipe DS du studio de développement Phoenix Interactive m'a permis de mettre à contribution les compétences acquises lors de mon cursus dans le domaine particulièrement intéressant qu'est le développement de jeux vidéo.

J'ai ainsi pu approfondir mes connaissances dans le fonctionnement global d'un studio de création, et en particulier d'une équipe de développement travaillant sur un projet. Ce travail en équipe a permis de me familiariser avec les règles telles que le respect des délais, la capacité à s'informer et à travailler en groupe, mais aussi l'initiative et l'autonomie.

Le développement informatique spécifique qu'est celui pour console de jeux m'a permis d'appréhender l'architecture logicielle d'un jeu vidéo et le fonctionnement d'un moteur de jeu. Les restrictions dues à la console portable DS ont également mis à l'épreuve mon sens de l'optimisation, à la fois mémoire et temporelle, mais aussi la qualité de ma programmation en langage C++.

Ce stage à aussi été l'occasion de travailler dans une équipe composée de plusieurs métiers très différents. Le contact avec des graphistes, des game designers, des programmeurs est réellement enrichissant, tant au niveau technique qu'au niveau humain et culturel. Pour finir, sur le plan humain, ce stage s'est particulièrement bien déroulé, l'intégration dans l'équipe a été rapide et les échanges ont été nombreux, ce qui m'a permis de mener à bien ce projet dans les meilleures conditions.

Bibliographie

Architecture de la DS :

Nitro Programming manual, Nintendo (confidential)

Effets graphiques (caustiques) :

Graphics gems, Andrew S. Glassner

La physique dans les jeux :

Game physics, David H. Eberly

L'arithmétique :

http://fr.wikipedia.org/wiki/Base_%28arithm%C3%A9tique%29

Les CRC:

http://en.wikipedia.org/wiki/Cyclic_redundancy_check

http://dvsoft.developpez.com/Articles/CRC/

http://fr.wikibooks.org/wiki/Architecture_des_ordinateurs/Codes_d%C3%A9tecteurs_et_correcteurs_d%27erreurs

Mots clefs

SSII / Services informatiques – Jeu vidéo - Informatique - Développements logiciels - Multimédia - Logiciel de jeu

MONNERET Nicolas

Rapport de stage ST50 - P2009

Résumé

Phoenix est un studio de développement de jeux vidéo basé à Lyon. L'entreprise créée des jeux sur PC, mais aussi sur des plateformes telles que PS3, Wii ou Nintendo DS. L'objectif de mon travail au sein du studio a été de développer des modules d'un jeu vidéo sur Nintendo DS. J'ai donc intégré une équipe pluridisciplinaire en tant que programmeur gameplay, pour participer à la création d'un jeu de type « party game », composé de divers mini-jeux. Le travail que l'on m'a confié peut se diviser en quatre grandes parties. La première a consisté à programmer sept mini-jeux, de l'architecture jusqu'à l'intégration des ressources. Ensuite, j'ai développé le menu principal du jeu et son système d'animation. La troisième étape a été la création d'un système d'échange d'objets de jeu entre plusieurs DS par génération et échange de codes. Pour finir, la dernière partie concernant ce projet a été le débogage de l'application avant sa validation par l'éditeur. Par la suite, j'ai été transféré dans l'équipe « moteur de jeu DS » ou j'ai effectué des recherches graphiques, notamment sur les effets de caustiques sous-marins, de plasma et sur l'affichage du tracé du stylet dans une scène 3D.

Phoenix Interactive
88 rue Paul Bert
69 003 Lyon
www.phoenix-i.fr