TRAVAIL DE MATURITÉ

JEU VIDÉO D'ARCADE

CRÉATION D'UN SHOOT 'EM UP NIGHTMARE GRAVITY

NOUREDDINE GUEDDACH

COLLÈGE SAINT-MICHEL

2015-2016

TABLE DES MATIÈRES

1 INTRODUCTION

J'ai toujours aimé les jeux vidéos. Depuis tout petit, j'ai été fasciné par les graphismes (quoiqu'un peu dépassés aujourd'hui, ils me paraissaient à peine quelques années en arrière époustouflants), les scénarios, l'acuité visuelle et les réflexes visuels requis, ainsi que par les challenges qu'offrait chaque jeu.

J'avais donc parcouru une large palette de jeux, allant des jeux de sport (basket-ball, tennis et surtout football) aux jeux de course, en passant par les jeux de combat, les FPS[1], les RPG[2], et bien sûr les shoot'em up.

Un shoot'em up (de l'anglais : « descends-les tous ») est un jeu où le joueur doit affronter des vagues d'ennemis et, comme son nom l'indique, tous les éliminer, un par un (ou du moins un maximum, pour atteindre le score le plus élevé possible) tout en esquivant leurs tirs. En général, le jeu est élaboré autour d'un système de niveaux à la fin desquels le joueur doit affronter un boss (ennemi très résistant et puissant avec différents modes d'attaque ; son élimination fait passer le joueur au niveau suivant).

Un shoot'em up qui m'a beaucoup marqué, et que je n'ai jamais réussi à terminer, soit dit en passant, s'appelle R-Type Delta (issu de la série R-Type[3] et conçu pour la console PS1). Lorsque j'ai parcouru la liste des séminaires, il y a quelques mois de cela, j'ai immédiatement penché pour deux d'entre eux : « Mathémagie » et « Jeux vidéos d'arcade ». Au début, j'étais un peu réticent à choisir le deuxième pour une unique raison : la peur de ne pas pouvoir avancer par manque de connaissances dans le domaine. En effet, je n'avais encore jamais fait de réelle programmation auparavant. Naquit alors un challenge : j'étais un fan de jeux vidéo qui baignait dans un monde qu'il ne connaissait pas et qui n'avait aucune idée de son fonctionnement (très petit, je pensais que tous les billions de scénarios possibles et imaginables étaient tous « préparés » à l'avance…) ; j'ai alors décidé de faire le pas et de passer de l'autre côté, celui des développeurs. Décision que je n'ai absolument pas regrettée car je me suis vite rendu compte que c'était un monde magnifique, encore plus que celui du joueur.

Une fois ma décision prise, bien que je ne savais toujours pas si ce Travail de Maturité allait m'être attribué, je me suis mis à apprendre le langage « C++ » sur OpenClassroom[4]. L'acquisition

1 Jeu de tir à la première personne.
2 Jeu de rôle.
3 Série de jeux vidéos phare du genre shoot'em up et développée par la société Irem à partir de 1987.
4 Originellement nommé « le site du zéro », ce site propose des milliers de tutoriels essentiellement en informatique, pour tout apprendre depuis « zéro ».

des bases a été largement facilitée par mon option spécifique PAM[5] qui nous initie déjà à Mathematica[6] et donc à l'utilisation de boucles, des variables etc. Par la suite, j'ai appris que le jeu devait être programmé en Java. Je me suis donc penché sur ce langage (un peu plus intuitif et plus facile que le « C++ », je trouve) en utilisant Eclipse[7]. Dernier rebondissement, ce TM m'est attribué et on m'informe qu'un IDE[8] simplifié sera utilisé : Greenfoot. Il faut dire qu'après avoir exploré les méandres d'Éclipse, Greenfoot était comme un labyrinthe traversé par un fil d'Ariane : plusieurs fonctions basiques mais très pratiques, telles que celles servant à l'attribution d'une image à un acteur, à le faire mouvoir ou à obtenir ses coordonnées, pour n'en citer que quelques-unes, y sont déjà implémentées et peuvent donc directement être employées ; ce qui permet d'assez vite se lancer dans une réalisation concrète.

Plus qu'un travail de maturité, ce projet représentait pour moi la quête de réponses à beaucoup de questions longtemps restées sans réponses, l'occasion d'explorer ma créativité, de découvrir un nouveau monde ainsi qu'un nouveau hobby, bientôt devenu une obsession, et surtout, une grande satisfaction et beaucoup de plaisir.

5 Physique et application des maths.
6 Logiciel développé par Wolfram Research dès 1986 et permettant entre-autres de faire de lourds calculs.
7 IDE de développement en Java.
8 Integrated Development Environment : environnement contenant un certain nombre de fonctionnalités permettant de développer un logiciel dans un langage de programmation

2 PRÉSENTATION DE NIGHTMARE GRAVITY

Certes, R-Type Delta m'a beaucoup inspiré, mais je voulais faire quelque chose d'un peu différent en introduisant de la gravité et en mettant en place un mode alternatif difficile, nommé Nightmare[9]. De là, le nom de mon jeu s'est tout simplement imposé : « Nightmare Gravity ». Je devais donc allier 3 concepts : le côté shoot'em up, la base de mon jeu, la gravité (les trous noirs qui parcourent la scène se sont naturellement imposés) et, pour finir, le Nightmare.

2.1 L'INTERFACE

Avant de commencer à programmer le jeu en soi, j'ai tout d'abord commencé par réaliser l'essentiel de l'interface, ou du moins ses onglets :

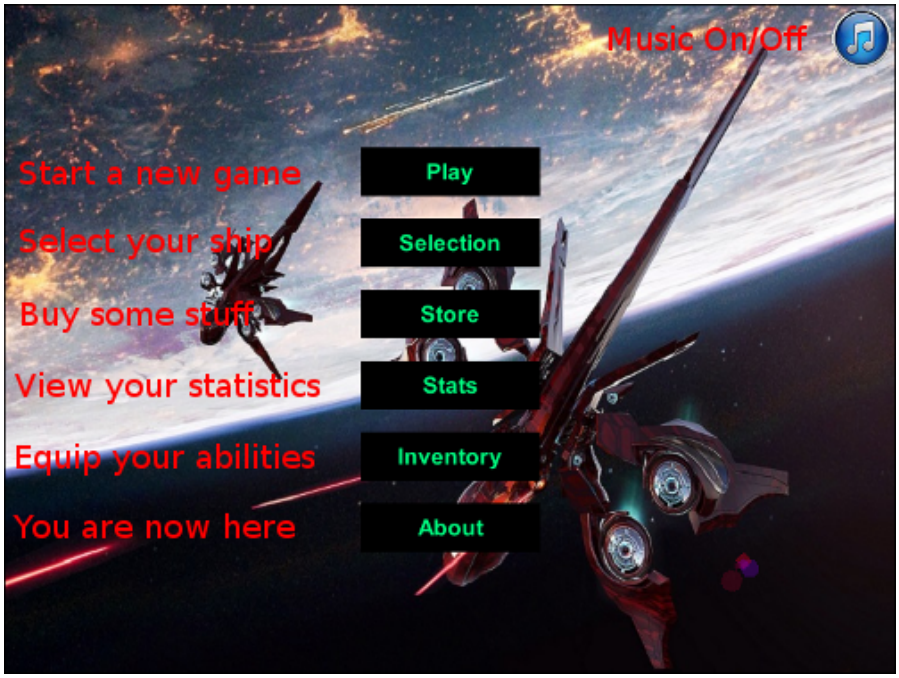

Figure 1: Interface du menu principal

9 Cauchemar en anglais

- Le menu de sélection : le joueur peut y trouver quatre vaisseaux ayant chacun différentes caractéristiques. Alors que le premier est disponible par défaut, chacun des autres est déblocable en terminant une partie avec succès.
- Le menu d'achats : le joueur a la possibilité d'y acheter des compétences comme celles de la régénération de vie ou du bonus de vitesse.
- L'inventaire : les objets achetés dans le menu d'achats apparaissent dans l'inventaire. Le joueur peut alors les « équiper » en les faisant glisser sous l'onglet « équipements », afin que ces objets soient « actifs » lors de la prochaine partie.
- L'onglet About : un tutoriel servant à guider le joueur dans ses premiers pas et contenant des informations sur la réalisation du jeu.
- L'onglet statistiques : où le joueur peut voir le nombre de Nightmares réussis, d'ennemis tués, son meilleur score etc.
- Et bien sûr, un bouton pour commencer une nouvelle partie.

2.2 LE SON

Une bande son différente est jouée à chaque niveau. Cependant, le joueur a la possibilité de la désactiver avant la partie ou via le menu pause. D'autres sons, comme les sons de tirs, d'animations, d'explosions etc. ne peuvent quant à eux pas être désactivés.

Alors que les bruitages sont tous tirés d'une bibliothèque de sons gratuits en ligne[10], j'ai réalisé les bandes son moi-même, d'abord en utilisant une version « démo » de Magix Music Maker 2015[11], puis en me servant de Rytmik Ultimate[12], qui est un logiciel beaucoup plus visuel et facile à prendre en main.

2.3 LE STOCKAGE DE DONNÉES

Dès le moment où j'ai eu l'idée de créer des objets permanents que l'on peut acquérir, un problème s'est posé : si je veux que mon jeu ait une durée de vie conséquente, il est nécessaire que l'utilisateur puisse retrouver tous ses objets acquis même après avoir quitté le jeu.

J'ai donc implémenté des méthodes pour stocker le nom d'utilisateur (pseudonyme visible pendant la partie) et son mot de passe. Jusqu'alors, le seul bémol était le nombre limité de comptes par ordinateur, c'est à dire un seul. En effet, j'employais des méthodes de la classe UserInfo de

10 « Bruitages, sons et loops gratuits », <http://www.universal-soundbank.com/>, consulté le 1e mars 2016.
11 Éditeur de musique digitale professionnel.
12 Logiciel de création musicale.

Greenfoot qui permettent de stocker un montant de données limité dans un fichier « .csv [13]» à la racine du projet.

Les choses se sont corsées une fois que j'ai exporté mon application en version exécutable « .jar » : le fichier n'était plus accessible ni en lecture ni en écriture, j'ai donc dû remplacer toutes les méthodes de sauvegarde pour qu'elles stockent dorénavant les données dans un fichier texte placé dans le répertoire personnel de l'utilisateur[14]. Un des avantages à l'emploi de cette méthode a notamment été la possibilité de créer une multitude de comptes par ordinateur, non plus un seul.

2.4 LA PARTIE

Le joueur commence une partie en cliquant sur « Play » dans le menu principal après avoir sélectionné son vaisseau.

2.4.1 L'AFFICHAGE

On peut voir ci-dessous une capture d'écran d'une partie :

Figure 2: Interface de la partie

13 « Comma-separated values » est un format informatique présentant un ensemble de valeur séparées par des virgules
14 C:\Users\…\datas.txt

En haut à gauche de l'écran, le joueur a à sa disposition les indicateurs de points de vie (A) et d'énergie[15] (B). Au milieu, il peut voir le niveau atteint (C) et en haut à droite sont affichés les crédits gagnés (D) ainsi que le temps écoulé (E) depuis le début de la partie. Soit dit en passant, les crédits sont un butin ayant une certaine probabilité d'être lâché par les ennemis et sont utilisés pour acheter de nouvelles compétences. Pour finir, tout en bas, le joueur a un aperçu des compétences (F) qu'il a préalablement équipées. De plus, le joueur a la possibilité de mettre le jeu temporairement en pause en appuyant sur la touche « P ».

Le monde défilant dans lequel se déroule le jeu change à chaque niveau. Il a été réalisé grâce à un logiciel graphique en ligne très simple, mais donnant de très jolis résultats : Silkweave[16].

2.4.2 LE ROBOT

Tout au long de la partie, le joueur a un compagnon de voyage nommé R-Bot et qui lui donne de précieuses indications avant un combat de fin de niveau sur la manière par laquelle affronter le boss ou qui s'affole quand il est sur le point de perdre. Le robot trahit le joueur à la fin de la partie et le laisse se faire emprisonner dans un labyrinthe (voir la section '2.8').

Figure 3: R-Bot - robot donnant des informations au joueur

15 L'énergie est nécessaire pour utiliser des compétences.

16 <http://weavesilk.com/>, logiciel gratuit en ligne permettant de réaliser de jolis décors avec de simples cliques de la souris.

2.4.3 LES NIVEAUX

Le jeu s'étale sur neuf niveaux conventionnels avec une phase de pur shoot'em up et un boss[17] de fin que le joueur doit battre pour accéder au palier suivant. Avant chaque combat de ce genre, le robot apparaît pour donner de précieuses informations au joueur sur la manière d'affronter le boss, en l'informant notamment de ses différents modes d'attaques, de ses points faibles ou de ses coups mortels. La durée des niveaux croît au fil de la partie, mais elle n'excède jamais les trois minutes.

2.5 LES VAISSEAUX

Tous les vaisseaux sont différents mais ils ont tous la faculté de se déplacer, de façon plus ou moins rapides, ont un mode de tir principal ainsi qu'une compétence secondaire dépensant de l'énergie. Les déplacements se font grâce aux touches directionnelles du clavier ou « ASDW », la souris étant utilisée pour viser et tirer. La barre d'« espace » sert à utiliser les compétences.

2.6 LES ENNEMIS

Il existe plusieurs classes d'ennemis, chacune ayant des compétences différentes et étant plus ou moins puissante et résistante. Plus le niveau est élevé, plus les ennemis sont nombreux et infligent davantage de dégâts. Outre leurs différences, tous les ennemis ont une certaine chance de lâcher des objets après leur destruction, que ce soient des crédits, des potions de vie, ou encore des vies bonus pour le mode Nightmare (voir la section '2.7' ci-dessous). Si l'on peut les considérer comme des ennemis, les trous noirs viennent s'ajouter à cette liste, d'autant plus que leur nombre augmente au fil des niveaux et qu'ils deviennent de plus en plus mortels.

2.7 LE NIGHTMARE

Cet aspect du jeu a été développé lentement, longtemps laissé de côté par souci de cohérence avec le reste. Le Nightmare n'a cessé d'évoluer tout au long de la réalisation. Au tout début, l'idée était de créer 3 défis de difficulté croissante avec plusieurs variantes de chaque défi. Tel était le concept : quand le vaisseau du joueur entrait en collision avec un trou noir, il était systématiquement téléporté vers l'un des trois Nightmares ou défis. Chacun des trois types offrait alors différents avantages, tels que des dégâts bonus ou une résistance accrue aux dégâts ennemis etc.

17 Dans tous les jeux vidéos, un boss désigne un ennemi beaucoup plus puissant que les autres et apparaissant à la fin d'un niveau. Le détruire permet en général au joueur de passer au palier suivant.

Après de longues réflexions, il s'est avéré qu'avec ce système, un bon joueur pouvait enchaîner les Nightmares, s'y familiariser, et le jeu se transformerait en une suite de défis qui deviendraient faciles et rébarbatifs à force de les refaire. Le shoot'em up n'aurait donc plus aucun intérêt, d'autant plus que les trous noirs ne sont en principe pas créés pour s'y précipiter mais pour la difficulté de les éviter, ce qui fait toute l'originalité du jeu. De plus, il fallait que les Nightmares restent un mode secondaire. J'en ai donc réduit le nombre à un seul, qui consiste en la résolution d'un puzzle unique en son genre. Pour y réussir, le joueur doit faire pivoter les dizaines de disques présents sur scène (en cliquant dessus) afin qu'ils soient tous reliés entre eux et qu'ils forment un seul réseau.

Mais alors, les Nightmares sont devenus encore moins intéressants puisqu'il n'y en a maintenant plus qu'une seule variante ? Non. Tout d'abord, chaque puzzle est généré à 100 % aléatoirement, il y a donc des millions de puzzles différents. Ensuite, comme dit précédemment, le joueur est censé éviter les trous noirs. Pour cela, j'ai fait en sorte que le joueur réfléchisse à deux fois avant de s'y précipiter : une fois qu'il entre en collision avec un trou noir, il a un pourcentage de chance de pénétrer le Nightmare ou de tout simplement perdre la partie. La formule est la suivante :

$$\textit{Chance de survie (\%)} = 100\ \% - (10 * \textit{le niveau du joueur})\ \%$$

Ainsi, plus le niveau du joueur est élevé, moins il a de chances de survivre. Par exemple, au niveau neuf, le joueur n'a plus que 10 % de chance de survivre à la collision avec un trou noir, alors qu'elle est de 80 % au niveau 2. Mais alors, quel bénéfice est-ce que le joueur pourrait-il bien en tirer, d'autant plus que la réussite n'offre plus aucun bonus pendant la partie? La réponse est que le score est décuplé pour chaque Nightmare réussi ! Car oui, le score est un aspect très important des « shoot'em up », et j'ai voulu exploité cet aspect-là de ce type de jeux. Encore faut-il réussir le Nightmare, car le puzzle est passablement compliqué et le temps est compté.

2.8 LA DERNIÈRE PHASE DU JEU

Vu tous les changements apportés au mode Nightmare, et pour avoir un certain scénario dans le jeu, en plus du fait que j'avais déjà programmé une grande partie de ce qui devait être le 2e type de Nightmares, j'ai décidé de réserver ce dernier pour un ultime défi à la fin de la partie : une fois tous les niveaux terminés, et après qu'une cinématique relatant que le héros s'est fait capturer et emprisonner dans un labyrinthe, ayant été trahi par le robot (voir section '2.4.2'), le joueur doit en chercher la sortie en faisant attention à ne pas être tué, tout en collectant les pièces qui lui permettront d'acquérir le prochain vaisseau. Il peut en effet y rencontrer deux types de créatures : des sentinelles assez passives, mais aussi des araignées qui essaieront de le traquer et de le tuer partout où il ira. Pour simplifier la tâche du joueur, il est muni d'une arme utile pour se frayer un passage à travers ce dédale. Une fois la sortie atteinte, il est libéré et, après une nouvelle brève cinématique, la partie est enfin terminée ! Voici une capture d'écran représentant une partie du labyrinthe :

Figure 4: Le labyrinthe représentant la dernière phase de la partie

On atteint donc gentiment la fin de cette partie dédiée à la présentation générale de Nightmare Gravity et on passe sans plus tarder à une analyse plus en profondeur du code source !

3 PRÉSENTATION DU CODE SOURCE

Bien entendu, toutes les fonctionnalités qu'offre mon jeu sont synonymes de centaines de pages de code, nous nous contenterons cependant d'analyser plus en détail certaines parties.

3.1 STRUCTURE GÉNÉRALE DU JEU

Avant d'analyser certaines parties, voici la structure générale de Nightmare Gravity :

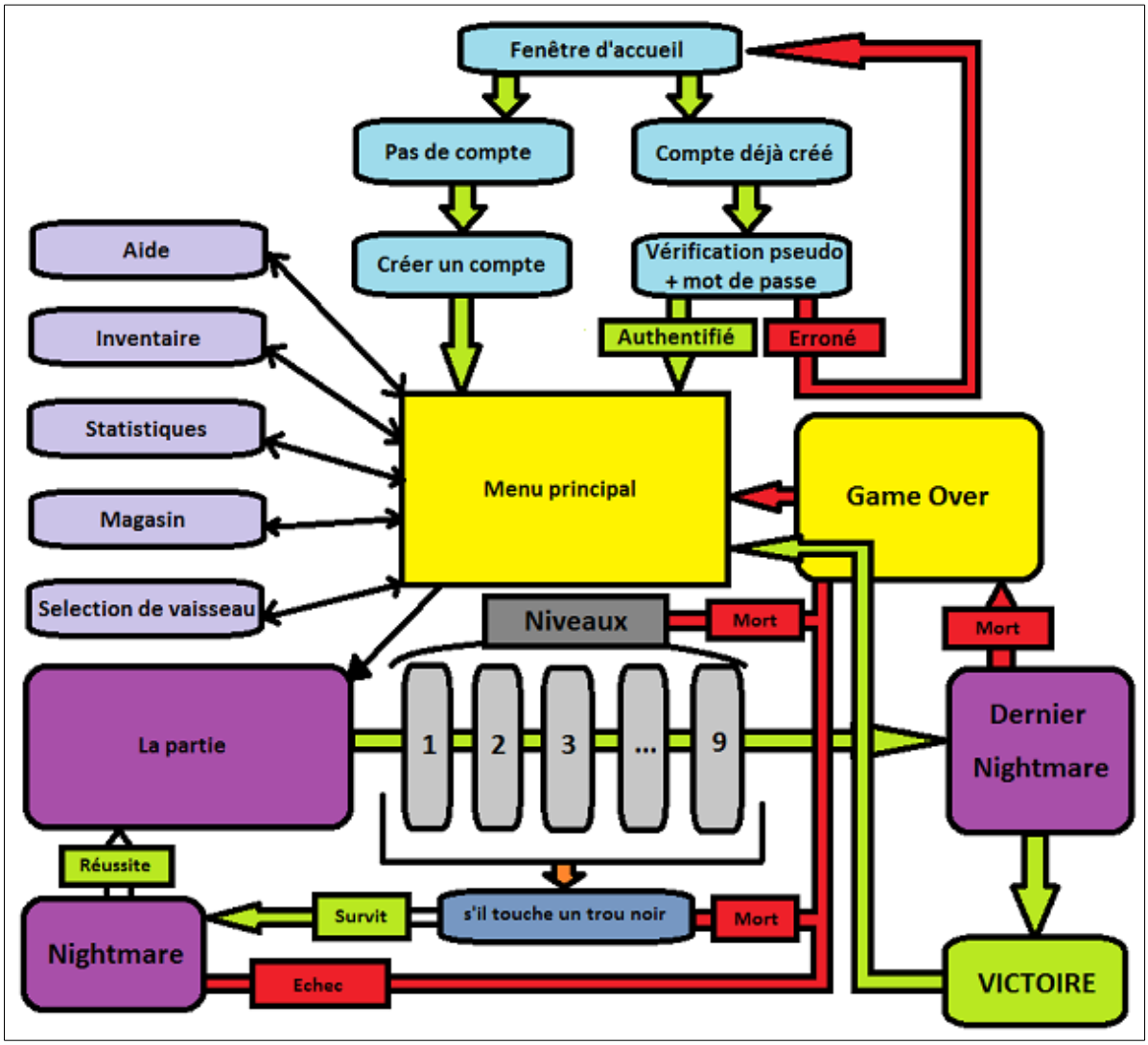

Figure 5: Schéma représentant la structure générale du jeu

Le jeu débute avec une fenêtre d'accueil, où le joueur a le choix de se connecter avec un compte déjà existant ou d'en créer un nouveau. Après identification, il passe au menu principal où il a à sa disposition d'autres onglets (Aide, Inventaire, Statistiques etc.).

Une fois une partie débutée, il doit en franchir les neuf niveaux, après quoi il accède au dernier Nightmare qui mène à la victoire, ou bien il meurt suite aux dégâts infligés par les ennemis, ou encore après une fâcheuse collision avec un trou noir. La fin de chaque partie, qu'elle soit gagnée ou perdue, passe par la scène « Game Over » où sont notamment affichés le score et le nombre d'ennemis tués.

Afin de mieux comprendre la suite, il faut savoir que l'IDE de Greenfoot contient deux classes mères fondamentales :

- World : classe contenant toutes les méthodes et fonctions nécessaires à la création d'une scène, à fixer ses dimensions, la manipuler, y rajouter des acteurs etc.
- Actor : classe contenant toutes les méthodes et fonctions nécessaire à la création d'un acteur, mais aussi à gérer ses déplacements, ses collisions, son interaction avec la scène etc.

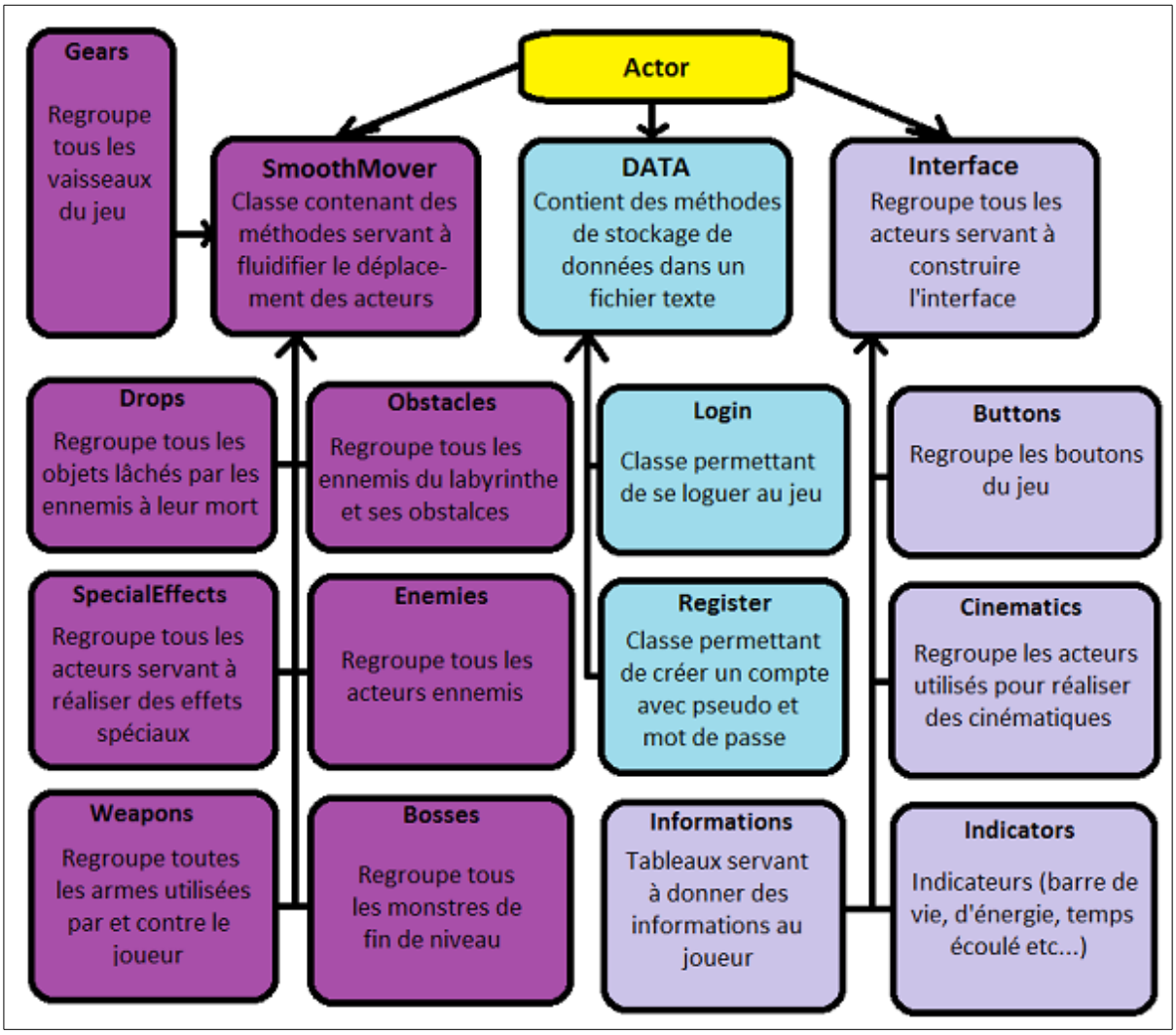

Figure 6: Schéma des principales classes utilisées au sein du projet

Ainsi, Nightmare Gravity est composé de plusieurs scènes : « Menu principal », « Nightmare », « Game » etc. (quatorze en tout) sur lesquelles évoluent des acteurs ; certains ont un comportement automatisé et géré par l'ordinateur, comme les ennemis par exemple, d'autres sont commandés par le joueur (via la souris ou les touches du clavier), comme le vaisseau.

Le jeu étant développé en Java, l'optimisation du code pour que l'exécution ne ralentisse pas à cause de phases durant lesquelles des centaines d'acteurs évoluent sur scène n'a pas été une mince affaire. En effet, tous les calculs nécessaires au calcul des dégâts ou au déplacement des acteurs se font via la machine virtuelle Java qui a un espace de mémoire vive allouée assez réduit et ne sont donc pas directement exécutés par le processeur, ce qui diminue drastiquement le nombre d'images par seconde, ou peut même dans certains cas, causer l'arrêt total de l'exécution du projet. Il y a donc quelques animations que j'avais entièrement programmées mais qui ont dû être supprimées au dernier moment à cause de ces inconvénients, que je n'ai malheureusement pas pu résoudre, même après des heures d'acharnement.

3.1.1 LA GRAVITATION :

```
/**
 * Applique la formule de Newton de la gravitation à cet acteur
 */
public void applyGravity(int mass)
{
    if(getWorld()!= null)
    {
        List<Traps> traps = getWorld().getObjects(Traps.class);
        int size = traps.size();
        if(size>0)
        {
            for(Traps trap : traps)
            {
                double dx = trap.getExactX() - this.getExactX();
                double dy = trap.getExactY() - this.getExactY();
                Vector force = new Vector (dx, dy);
                double distance = Math.sqrt (dx*dx + dy*dy);
                double strength = GRAVITY * mass * trap.mass / (distance * distance);
                double acceleration = strength / mass;
                force.setLength (acceleration);
                myAddForce (force);
                move();
            }
        }
    }
}
```

Figure 7: La méthode applyGravity() de la classe Gears

Ci-dessous, on peut voir un extrait de code qui met en œuvre la gravité influençant les vaisseaux en présence d'un trou noir (symbolisé ici par la classe « Trap ») :

Cette méthode prend un nombre entier « mass » en paramètre (mass représente la masse du vaisseau sur lequel la gravitation va s'appliquer). La boucle conditionnelle « if(getWorld() != null) » permet simplement de passer outre la méthode si la scène est changée (ce qui ferait planter le programme). On entre ensuite dans le vif du sujet : la méthode commence par créer une liste de tous les trous noirs présents sur scène (« Traps trap : traps » se lit : « pour chaque trap de la classe Traps de l'ensemble des traps sur scène ») ; si cette liste n'est pas vide (« if(size>0) »), elle va appliquer une force attractive sur le vaisseau du joueur. Une fois le fonctionnement général posé, place à la physique :

La variable « strength » stocke la force gravitationnelle donnée par :

$$F_{A/B} = (G * M_A * M_B)/d^2$$

Où :

- G = constante de gravitation (« GRAVITY »)
- M_A = masse de cet acteur (« mass »)
- M_B = masse du trou noir (« trap.mass »)
- d^2 = distance qui sépare le vaisseau du trou noir (« distance »)

On applique ensuite la 2e Loi de Newton dont l'énoncé est :

$$a = F/M$$

Où :

- a = accélération
- F = Force appliquée
- M = masse de l'objet

Pour finir, on ajoute le vecteur accélération ainsi obtenu au vaisseau, qui sera donc attiré par le trou noir. Comme quoi quand on fait de la programmation, on ne se contente pas simplement d'écrire du code, mais que l'on est parfois amené à utiliser des formules physiques ou mathématiques afin de réaliser certaines tâches. D'autres fonctions qui ont été primordiales à la réalisation de ce projet sont les fonctions trigonométriques. En effet, tout ce qui est de l'ordre des distances, des angles, de la rotation etc. fait toujours appel à celles-ci.

3.1.2 LE LABYRINTHE :

La construction du labyrinthe a été un véritable défi : je voulais absolument qu'il soit très grand. Le problème était que la scène est beaucoup plus petite que sa taille voulue. Il a donc fallu créer un mécanisme de défilement (scrolling) :

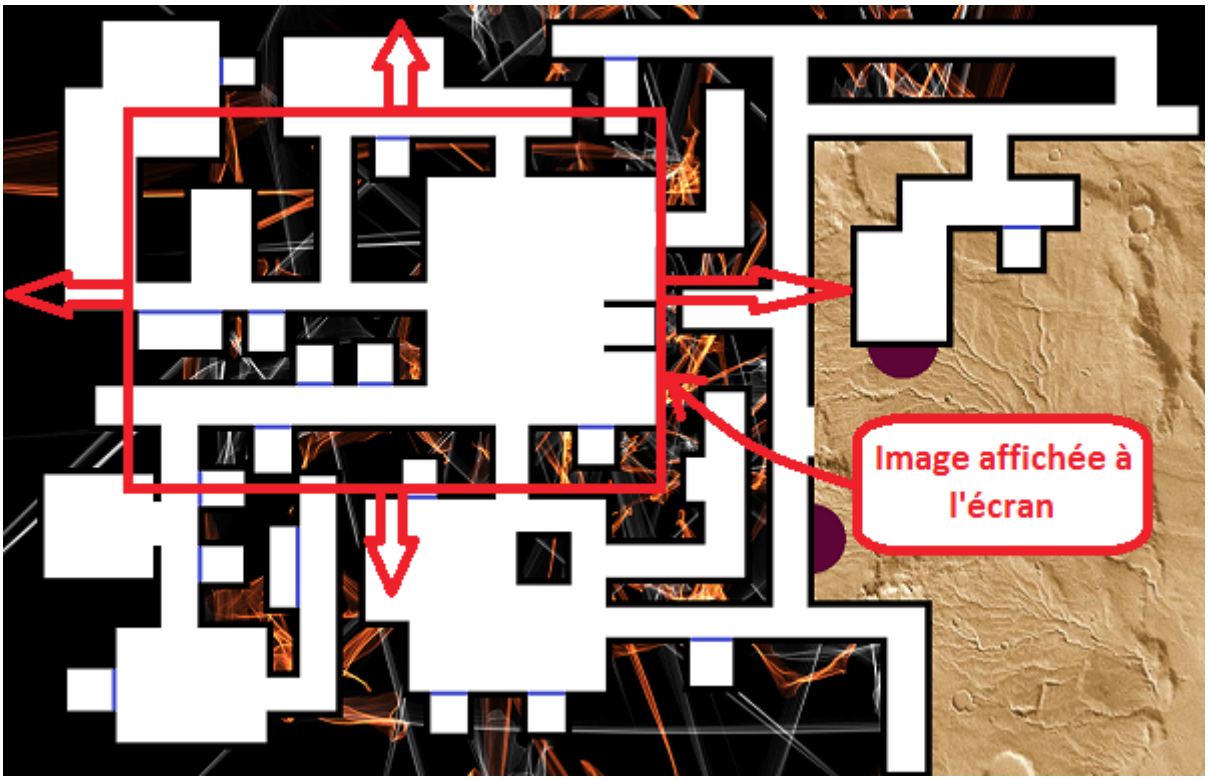

Figure 8: Labyrinthe auquel doit survivre le joueur pour terminer la partie

L'image ci-dessus illustre le labyrinthe entier et la petite partie affichée à l'écran ; pour ce qui est du code source, voici un extrait de la méthode moveMap() qui se trouve dans la classe MediumNightmare (scène du labyrinthe) et qui sert justement à faire défiler la carte :

```
/**
 * Méthode permettant de faire défiler la carte.
 */
public void moveMap()
{
    MouseInfo mouse = Greenfoot.getMouseInfo();
    Roamer roamer = (Roamer) getObjects(Roamer.class).get(0);
    if(mouse!= null)
    {
        if(directions()==1)
        {
            if(xPosition>= -(image.getWidth()-750) && roamer.getX()>=getWidth()/2)
            {
                visible.drawImage(image, xPosition-speed, yPosition);
                xPosition -= speed;
                if(getObjects(Obstacle.class).isEmpty()) return;
                else
                {
                    List<Obstacle> obstacles = getObjects(Obstacle.class);
                    for(Obstacle obstacle : obstacles)
                    {
                        obstacle.setLocation(obstacle.getX()-speed, obstacle.getY());
                    }
                }
                moved = true;
            }
        }
```

Figure 9: Méthode moveMap() de la scène MediumNightmare

La méthode commence par récupérer le premier objet de type souris (mouse) trouvé sur scène et, le cas échéant, récupère les informations qu'il fournit, ainsi qu'un objet Roamer (personnage du jeu), afin d'obtenir ses coordonnées. Elle vérifie ensuite si la souris est bien sur scène (if(mouse != null)), sans quoi le programme va essayer de trouver ses données alors qu'elles lui sont tout simplement inaccessibles ! A préciser que cette partie de la méthode teste le déplacement vers la droite (ce qui correspond au cas où la fonction directions() déclarée également dans la classe MediumNightmare renvoie la valeur 1).

On arrive donc au cœur de la méthode et le système de coordonnées : celle-ci commence tout d'abord par tester si la variable xPosition (qui stocke le nombre de pixels de déplacement horizontal) n'a pas atteint sa valeur maximale ; si ce n'est pas le cas, le déplacement peut donc être effectué. Comment ? Plus qu'un déplacement, c'est en réalité un cadrage : c'est comme si l'on déplaçait une loupe de quelques pixels, en l'occurrence, en imprimant l'image du labyrinthe concernée sur l'écran de l'utilisateur et en remettant la nouvelle valeur de xPosition en mémoire, après qu'elle ait été décrémentée de la valeur de la variable speed (qui renvoie la vitesse de défilement du laby-

rinthe). Tous les déplacements, qu'ils soient horizontaux ou verticaux sont réalisés de la même façon, en incrémentant ou décrémentant, respectivement, la variable xPosition (pour l'axe des X) ou la variable yPosition (pour l'axe des Y).

Pour finir, la méthode décale également tous les acteurs pour qu'ils restent au mêmes coordonnées relatives au labyrinthe.

3.1.3 LE COMPTE D'UTILISATEUR

Dans Nightmare Gravity, l'utilisateur a la possibilité de créer un compte et de sauvegarder ses données via une sauvegarde locale[18]. Pour cela, et avant d'implémenter les méthodes de stockage, il a fallu créer des méthodes pour que l'utilisateur puisse inscrire son pseudo. En voici un exemple :

```
/**
 * Méthode qui écrit le pseudo et le stocke dans une variable,
 * certains caractères sont interdits
 * et la longueur du pseudo est limitée
 */
public void writeID()
{
    keyName = Greenfoot.getKey();
    if(keyName == null) return;
    if (keyName != null)
    {
        if(keyName.equals("backspace"))
        {
            if(ID.length()>0)
            {
                ID = ID.substring(0, ID.length()-1);
            }
            else if(ID.length() == 0)
            {
                return;
            }
        }
        else if(keyName.length()<2)
        {
            if(ID.length()<MAX_SIZE_ID)
            {
                ID+=keyName;
            }
        }
    }
}
```

Figure 10: Méthode writeID() de la classe Background

18 Les données ne sont pas stockées en ligne mais sur l'ordinateur de l'utilisateur

La méthode ci-dessus nommée writeID() et permettant d'écrire l'identifiant, comme son nom l'indique, se base sur la fonction getKey() de Greenfoot qui retourne la dernière touchée enfoncée par l'utilisateur. Elle commence donc par détecter si celui-ci a touché à son clavier ou pas. Si c'est bien le cas, il y a alors trois issues possibles :

- La touche est celle de retour (« backspace ») : si le mot comporte au moins un caractère, la méthode en efface le dernier (en prenant la chaîne de caractère actuelle et en la privant de sa dernière lettre via la fonction « substring » de Java) ; sinon, elle ne fait rien (le mot-clé return sert notamment à forcer la sortie d'une méthode en Java).

- La touche est un input spécial dont le nombre de caractères excède le nombre « un » : par exemple, la touche espace s'appelle « space » (ce qui représente 5 caractères, c'est donc un caractère interdit). Dans ce cas-là, la méthode ne fait rien non plus.

- La touche est un caractère normal (lettre, chiffre, symbole à un seul caractère…) : dans ce cas-là, la méthode rajoute le symbole tapé à la chaîne de caractères constituant l'identifiant jusqu'à ce que celle-ci atteigne sa taille maximale prédéfinie par la constante « MAX_SIZE_ID ».

Et bien sûr, le tout est affiché à l'écran. Une autre méthode similaire sert à écrire le mot de passe (en astérisques à l'écran). Ensuite, ces données sont comparées à celles du fichier de stockage (un fichier texte) et l'on peut ainsi déterminer si le joueur a correctement entré son identifiant s'il a l'intention de se connecter, ou lui créer un nouveau compte s'il n'en a pas encore.

4 CONCLUSION

Ce travail de maturité était un vrai plaisir. Il était riche en nouveautés, en découvertes, truffé de problèmes, gorgé de solutions. Il était le berceau d'une passion, le début d'un rêve, l'accomplissement d'un premier pas.

Depuis les premiers jours, j'ai été captivé, pendant quelques mois j'ai été prisonnier d'un beau cauchemar. Ainsi, plus de 450 heures se sont écoulées en travail acharné, en frustrations, en réflexions, en échecs mais surtout en réussites. Le résultat ? Un jeu qui m'apporte pleine satisfaction.

En effet, ce travail m'a permis de découvrir un nouveau monde aux horizons encore inexplorés, de me familiariser à l'utilisation de logiciels graphiques tels que Gimp ou Blender, de découvrir l'univers de la production musicale au travers de Rytmik Ultimate ainsi que d'apprendre une nouvelle langue. Il m'a également appris à ne jamais renoncer, à persévérer, à avoir un esprit méthodique, organisé et à trouver des solutions là où il n'y en a à priori pas.

J'ai également trouvé la réponse à de nombreuses questions liées à l'univers du jeu vidéo qui me torturent l'esprit depuis trop longtemps, compris quel était le travail d'un développeur et fait face à beaucoup des difficultés que ce métier lui impose.

Pour finir, je tiens à dire que je ne considère pas cette conclusion comme la fin de Nightmare Gravity, mais plutôt comme une pause en vue d'une nouvelle lancée. En effet, je ne compte pas m'arrêter ici, je veux dorénavant sortir du cadre de Greenfoot, découvrir d'autres langages de programmation, les maîtriser, développer d'autres applications, de nouveaux jeux de plus en plus complexes et pourquoi pas, refaire un jour une refonte complète de ce jeu en y rajoutant du nouveau contenu (compétences supplémentaires, d'autres niveaux et éventuellement un mode multijoueur), mais également en résolvant les problèmes liés aux animations.

5 REMERCIEMENTS

J'aimerais tout particulièrement remercier :

- La communauté du forum Greenfoot, et spécialement un membre portant le pseudo de « danpost » pour toutes ses réponses à mes questions.
- Mon tuteur : Laurent Bardy, pour les nombreuses réunions consacrées à ce travail ainsi que pour ses précieux conseils.
- Un ami et camarade de classe : Paul Bureth, pour avoir testé le jeu à plusieurs reprises et m'avoir donné plusieurs bons conseils.
- Ma famille qui m'a beaucoup encouragé pendant mes heures de travail.
- Et tous ceux qui m'ont aidé de quelque manière que ce soit à réaliser ce travail de maturité.

6 WEBOGRAPHIE

« Forum de Greenfoot », <http://www.greenfoot.org/topics>, consulté le 14 mars 2016.

« Forum de Stack Overflow », <http://stackoverflow.com/questions/tagged/java>, consulté le 10 mars 2016.

« Open Classrooms », *Apperenez à programmer en Java,* <https://openclassrooms.com/courses/apprenez-a-programmer-en-java>, consulté le 12 janvier 2016.

« Programmation orientée objet avec Greenfoot », <http://www.mathematiques.ch/~salvadore/2M/OC_info/programGreenfoot.pdf>, consulté le 20 septembre 2015.

« Documentation sur le package Greenfoot », <http://www.greenfoot.org/files/javadoc/>, consulté le 10 mars 2016.

ORACLE, *Java Platform, Standard Edition 7 API Speicification,* <https://docs.oracle.com/javase/7/docs/api/>, consulté le 10 mars 2016.

7 SOURCE DES FIGURES

Figure de la page de titre :

R-TYPE, *R-13A Cerberus*, <http://vignette4.wikia.nocookie.net/rtype/images/2/22/R13_Delta.png/revision/latest?cb=20110209024943>, consulté le 12 mars 2016.

Arrière-plan de la page de titre :

WALL321, *Abstract waves*, <http://www.wall321.com/thumbnails/detail/20120308/light%20abstract%20waves%20sound%201600x1200%20wallpaper_www.wall321.com_50.jpg>, consulté le 12 mars 2016

Figure 1 :

Capture d'écran du menu principale

Figure 2 :

Capture d'écran d'une partie

Figure 3 :

DRIBBLE, *Friendly Robot*, <https://d13yacurqjgara.cloudfront.net/users/16073/screenshots/761948/friendly-robot.jpg>, consulté le 12 mars 2016

Figure 4 :

Capture d'écran du labyrinthe (vision réduite)

Figure 5 :

Schéma de la structure générale du jeu, réalisé avec Paint

Figure 6 :

Schéma des classes du projet, réalisé avec Paint

Figure 7 :

Capture d'écran du code source montrant la méthode applyGravity() de la classe Gears

Figure 8 :

Image du labyrinthe réalisée avec Paint et Silkweave

Figure 9 :

Capture d'écran du code source montrant la méthode moveMap() de la classe MediumNightmare

Figure 10 :

Capture d'écran du code source montrant la méthode writeID() de la classe Background

8 ANNEXES

- Un CD contenant :
  - Le projet en version Greenfoot.
  - Le projet en version exécutable (jar).
  - Les sources des images et sons utilisés au sein du projet.