Conception d'un jeu vidéo rétro

Dave Unchained

Quentin Angéloz

Figure 1 – Capture d'écran

Collège Saint-Michel - Fribourg
Travail de maturité sous la direction de Yves Dubey et Yves Roisin
Rendu le 1 avril 2019

Table des matières

Table des figures

Introduction

Ce travail de maturité m'a interpellé dès que j'ai vu qu'il était proposé. Depuis tout petit, je me suis intéressé aux ordinateurs et à l'informatique ; d'une part grâce à mon père qui m'a initié aux jeux dès mon plus jeune âge et d'autre part par le fait que ma curiosité a toujours été attirée par les ordinateurs et autres consoles de jeux vidéos.

J'ai essayé de me lancer dans la programmation déjà avant la réalisation de ce TM, mais je n'ai jamais eu de réelle motivation qui m'ait poussé à aller jusqu'au bout d'une oeuvre. Je m'étais adonné à un peu de Visual Basic lorsque j'étais petit, puis vers mes onze ans, j'ai fait un très rapide survol du C avec un microcontrôleur Arduino que j'avais reçu pour mon anniversaire. J'avais suivi le manuel donné avec le microcontrôleur et réalisé tous les exemples présentés par celui-ci. Jusqu'à ce jour, Je ne m'étais pas encore lancé dans des oeuvres provenant de ma propre initiative. J'ai donc vu ce travail comme une possibilité de finalement me lancer dans la programmation et d'aboutir à une oeuvre complète.

De plus, j'ai choisi comme option spécifique "physique et application des maths" qui comporte un peu de programmation avec le logiciel Wolfram Mathematica et j'ai choisi cette année l'option complémentaire "informatique" ; tous les éléments en corrélation avec ce choix étaient présents, et jamais je ne m'étais intéressé au langage qu'est Python, ni ne l'ai pratiqué dans mon option complémentaire. Enfin, je souhaiterais dans mon avenir me lancer dans des études comportant de l'informatique, d'où mon intérêt pour ce TM.

Mon jeu est inspiré d'un mini-jeu qui se nomme "Journey of the Prairie King" présent dans le jeu "Stardew Valley", sorti le 26 février 2016 sur Windows ; un mini-jeu de type "shoot'em up" où le joueur contrôle un cowboy qui se bat contre des monstres. Ce mini-jeu est une minuscule annexe, presque anecdotique, accessible via une borne d'arcade dans le jeu original. Au fil de l'avancée du personnage, le jeu devient de plus en plus difficile et le personnage finit par rencontrer des boss contre lesquels il doit se battre.

Mon intention n'était en aucun cas de reproduire ce jeu telle une copie. Le jeu original est plus complexe que le jeu que j'ai réalisé. Je voulais juste garder le concept de base et construire quelque chose par moi-même autour, donc au final mon jeu possède quelques similitudes mais est surtout composé d'éléments issus de ma propre inspiration.

Chapitre 1

Description du jeu

Dans mon jeu, le joueur contrôle un cowboy qui voit des ennemis converger vers lui simultanément. Son but est de les tuer tous et de survivre le plus longtemps possible. Afin de les éliminer, il peut leur tirer dessus avec la souris et ils sont ainsi éliminés dans une explosion.

Les ennemis ont 4 zones d'apparition sur l'écran : en haut, en bas, à gauche et à droite avec des probabilités d'apparition plutôt semblables. Logiquement, le joueur a donc meilleur temps d'essayer de rester le plus possible au centre de l'écran afin d'éviter de se faire acculer par les ennemis qui convergent vers lui. De plus, les coins de l'écran sont des endroits où le joueur est interdit d'accès, mais pas les ennemis. Ils sont donc à première vue avantagés dans leurs mouvements. De plus, ils peuvent se déplacer dans toutes les directions possibles alors que le cowboy n'en a que huit. En contrepartie, la vitesse de base du joueur est légèrement plus grande que celle des ennemis ce qui équilibre les deux acteurs de mon jeu.

Le joueur a aussi pour s'aider des "Power-ups" qui apparaissent aléatoirement, avec des effets distincts et qui peuvent être cumulés. En voici la liste :

1. BulletSpeed : augmente la vitesse de chaque balle tirée

2. Invincibility : confère un bouclier au joueur (pour un certain temps) qui tue tous les ennemis qui le touchent, mais dès lors ne confère pas de points de score.

3. SpeedUp : augmente la vitesse de déplacement du personnage

4. Big Bullets : Augmente la taille des projectiles tirés, ainsi il devient plus simple de toucher les ennemis

5. Boss Power-up : Power-up spécial qui n'apparaît que s'il y a un combat contre un boss. Il est nécessaire pour blesser ce dernier et prend l'apparence d'une balle rouge. Il ne dure cependant que quelques instants et ne permet de toucher le boss que quelques fois. Son taux d'apparition est plus fréquent que celui d'un "Power-up" d'un autre type.

Les "Power-ups" apparaissent sur l'écran avec une position au hasard et durent 200 ticks ; passé ce délai, ils disparaissent. Si le joueur marche sur l'un d'entre eux, il en gagne l'effet. Un "Power-up" peut apparaître sur les bordures que le personnage ne peut pas traverser mais il pourra quand même être récupéré si l'on se déplace en bordure. Ils sont tous bénéfiques et d'une grande aide pour le joueur.

Si un ennemi touche le personnage, celui-ci meurt et perd une vie sauf s'il a le bonus "invincibility" actif. Il commence avec trois vies et lorsqu'il n'en a plus c'est "Game Over". Le joueur peut regagner une vie chaque 50 ennemis standards tués. Le but global du jeu est de faire le meilleur score possible. Chaque ennemi de base tué vaut 1 point. Il existe deux types d'ennemis : Des ennemis de base qui ne font qu'avancer vers le joueur et un autre type d'ennemis qui, en plus de converger vers le joueur, tire contre celui-ci. Lorsqu'une de leurs balles touchent le joueur, il perd une vie. Ces ennemis ont trois vies au début de la partie et apparaissent moins fréquemment que les ennemis standards.

Malgré ce principe simpliste, il existe une progression dans mon jeu par le fait qu'il existe plusieurs ennemis mais aussi que la difficulté augmente graduellement. En effet, tous les 30 ennemis tués, le personnage est invité à se rendre dans une nouvelle zone, qui sera plus compliquée car les ennemis apparaîtront plus fréquemment. De plus, après cinq zones nettoyées de tous les ennemis, le joueur doit affronter un boss, que l'on ne peut que blesser avec un "Power-up" spécial (voir ci-dessous) qui n'apparaît que si le boss est en vie (avec un taux d'apparition plus fréquent qu'un Power-up standard). Une fois celui-ci vaincu,le joueur continue de jouer avec la difficulté qui continue d'augmenter car les ennemis auront un point de vie supplémentaire après chaque boss vaincu. Chaque boss vaut 30 points. Le prochain boss verra par ailleurs chaque fois sa vie doubler et sa vitesse légèrement augmenter. Le jeu commence assez facilement mais au fur et à mesure de l'avancée du joueur, il devient plus intéressant de par sa difficulté croissante.

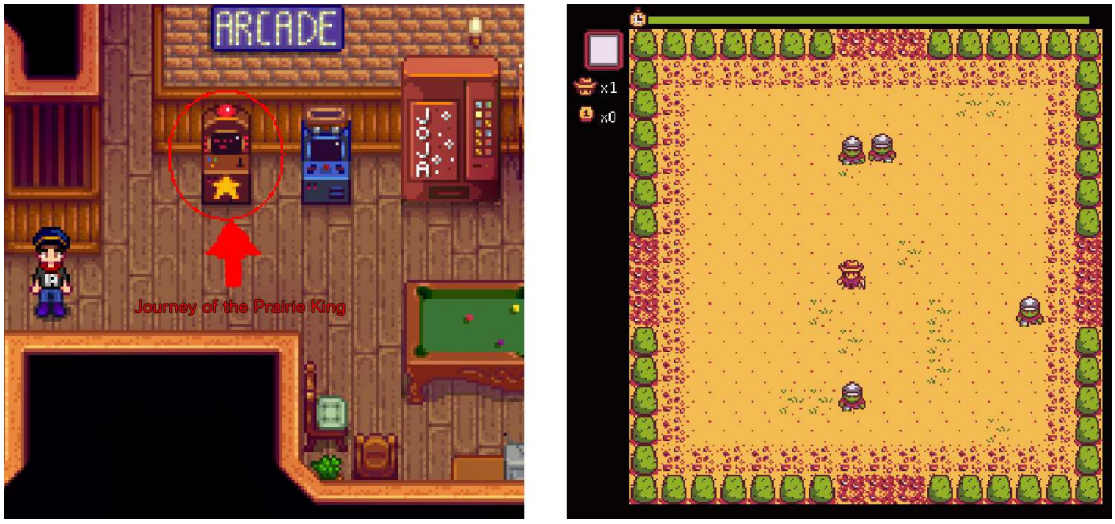

FIGURE 2 – Borne d'accès au jeu dans "Stardew valley" et jeu d'inspiration

Chapitre 2

Débuts et difficultés rencontrées

J'ai eu quelques difficultés à commencer la partie de programmation. Jamais auparavant je n'étais parti de rien et arrivé à quelque chose de concret. Heureusement, les exemples de mes précepteurs m'ont grandement aidé et m'ont donné plusieurs éléments qui ont énormément facilité mon raisonnement par la suite.

2.1 Mes débuts

Ma première expérience avec Python et Pygame a eu lieu lors de ma première réunion de travail de maturité. Mon professeur nous avait montré un exemple où l'on pouvait faire bouger un carré sur un écran. Le code était simpliste, mais n'ayant jamais pratiqué auparavant, il m'a fallu bien l'analyser pour le comprendre. Je connaissais déjà le principe des classes et des méthodes, mais je n'avais jamais vu d'autre langage qui utilisait les attributs 'self'. Par la suite, j'ai commencé gentiment, j'ai continué de suivre les cours et en premier lieu, essayé de modifier quelque peu les exemples que mes professeurs me donnaient. Ainsi, j'ai essayé d'animer mon propre personnage et de le faire se déplacer. De cette manière, mon long apprentissage avait commencé. Je n'étais pas très rapide à la réalisation, mais j'ai persévéré et, à force d'obstination, je suis toujours arrivé à mes fins, d'une manière ou d'une autre.

2.2 Principales difficultés

Ma première grande difficulté a été de faire tirer mon personnage. Ensuite j'ai eu de la peine à gérer mes collisions à l'intérieur de plusieurs groupes. C'était des principes qui étaient tous nouveaux pour moi et j'ai dû grandement faire appel à Internet et à des tutoriels. Je tenais cependant à y arriver par moi-même, nonobstant l'évidente perte de temps que cela engendrait, l'apprentissage et la compréhension ont été en premier lieu mes priorités et, une fois les principes de base acquis, Les algorithmes nécessaires à la réalisation de mon jeu me sont venus à l'esprit beaucoup plus vite. Je savais quoi faire et comment le faire en pseudocode.

Une de mes grandes difficultés a été sans aucun doute la compréhension de la gestion de collisions entre groupes. Pour cela, j'ai eu pas mal de difficultés à comprendre et dans un premier lieu différencier les types de collisions que Pygame nous propose. J'ai eu particulièrement de la peine à générer les masques de mon personnage, étant donné que son image dépend de deux dictionnaires, à savoir l'un donnant des informations sur sa direction et l'autre traitant laquelle des trois images propre à l'animation est oblitérée sur l'écran.

Une autre part de mon travail qui m'a pris du temps était la réalisation de mes graphismes. J'ai été assisté et conseillé pour cela par ma soeur Chloé Angéloz, qui est en dernière année au collège Saint-Michel dans la classe 4B2. Je les ai réalisés à l'aide du logiciel japonais Medibang Paint. Il est à la base plutôt conçu et pensé pour dessiner des mangas mais il m'a été conseillé par ma soeur car elle est une grande adepte de celui-ci. J'ai donc réalisé la plupart de mes graphismes moi-même. Certains d'entre eux sont néanmoins l'oeuvre de ma soeur (notamment l'image de titre). Les graphismes qui n'ont pas été réalisés par moi proviennent du site internet "Opengameart.com". C'est un site où des gens postent des images, musiques, et autres. Toutes les contributions publiées sur ce site sont libres à la réutilisation avec mention de l'auteur.

J'ai aussi voulu me lancer dans la réalisation de mes propres musiques. Ne m'étant jamais essayé à la composition musicale auparavant, et bien que je pratique de la guitare acoustique, je ne m'attendais pas à un résultat transcendant. J'ai donc décidé de faire de mon mieux dans la mesure du possible. Pour réaliser mes musiques, je me suis aidé du logiciel audacity et FL Studio Fruity Edition. J'ai utilisé les échantillons fournis de base avec ce logiciel. La réalisation de mes musiques a été longue et fastidieuse pour un résultat passable, mais je ne regrette pas ce choix car elles viennent de moi et renforcent l'idée que ce jeu est ma pure création. J'ai cependant quand même décidé de changer la musique qui se joue pendant une partie car je ne la trouvais vraiment pas satisfaisante. J'ai donc opté pour une musique que j'ai trouvée sur "OpenGameart.com", comme quelques-uns de mes graphismes.

Chapitre 3

Parties intéressantes

Au fil de mon apprentissage, je me suis vite rendu compte des difficultés principales de la programmation, à savoir d'une part la compréhension des processus nécessaires à la réalisation d'un algorithme et d'autre part l'apprentissage des structures propres au langage qu'est Python. Dès lors, il m'a fallu beaucoup de temps pour réaliser certaines parties de mon code, mais l'exploit s'est avéré d'autant plus gratifiant quand je suis arrivé finalement à réaliser mes souhaits.

3.1 Convergence des ennemis

J'ai choisi de faire converger les ennemis vers mon personnage de cette façon :

```
norme = math.sqrt(math.pow(perso.x - self.x,2) + math.pow(perso.y - self.y,2))
```

Ceci étant concrètement :

$$\sqrt{(perso.x - self.x)^2 + (perso.y - self.y)^2}$$

Cette simple ligne détermine la norme du vecteur entre le personnage et l'ennemi. Pour ce faire, je fais appel à plusieurs fonctions de python.math, à savoir :

1. math.sqrt(x) : retourne la racine carrée de x
2. math.pow(x,y) : retourne x élevé à la puissance y

C'est comme si je possédais 2 points dans l'espace et que je calculais la norme du vecteur entre ces deux points à partir de la composante en x et la composante y de ce vecteur. Puisque je suis dans un repère orthonormé, l'angle entre mes vecteurs sur l'axe des x et l'axe des y est de 90° ; donc je peux calculer la norme du vecteur à l'aide du théorème de Pythagore, à savoir :

`La somme des carrés des cathètes est égale au carré de l'hypoténuse:`

$c^2 = a^2 + b^2 \Leftrightarrow c = \sqrt{a^2 + b^2}$

De cette manière j'ai pu en déduire c, qui n'est ici rien d'autre que la norme du vecteur qui nous intéresse. J'applique donc cela dans mon cas particulier : pour ce faire, je calcule la composante horizontale de mon vecteur, à savoir :

```
perso.x - self.x
```

cela calcule la différence de la position de mon personnage et la position de l'ennemi (self retournant ici l'ennemi, car je suis à l'intérieur de sa classe) sur l'axe des abscisses. Je fais la même chose pour l'axe des ordonnées :

```
perso.y, self.y
```

à partir de cela je n'ai plus qu'à appliquer le théorème décrit plus haut. Donc concrètement, je calcule la racine carrée de la somme de la différence de la position en x de mon personnage avec la position en x de l'ennemi (cette différence) élevée au carré et la différence de la position en y de mon personnage avec la position en y de l'ennemi élevée au carré.

Par la suite, je calcule les attributs concrets en terme de déplacement de l'ennemi :

```
self.speedx =(perso.x - self.x)/ norme*ennemyspeed
self.speedy = (perso.y - self.y)/norme*ennemyspeed
```

Je calcule ici sa vitesse en x et en y respectivement ; pour cela je calcule le vecteur entre mon personnage et l'ennemi pour les deux composantes x et y divisé par la norme calculée auparavant multipliée par la variable "ennemyspeed" qui modifie la vitesse de convergence des ennemis. Cette variable est altérée à chaque fois qu'un boss est éliminé. En effet, elle est mentionnée au début du fichier :

```
self.ennemyspeed = 6 + level/1.5
```

Cet attribut dépend d'une seule variable, nommée "level". Cette variable est incrémentée de 1 à chaque fois qu'un boss est éliminé. Concrètement, elle représente le niveau de difficulté. Elle est initialisée à 1. Ainsi au niveau 2, l'attribut "self.ennemyspeed" vaut $7,\overline{3}$.
J'ai donc calculé de cette manière le déplacement horizontal et vertical de mon ennemi.

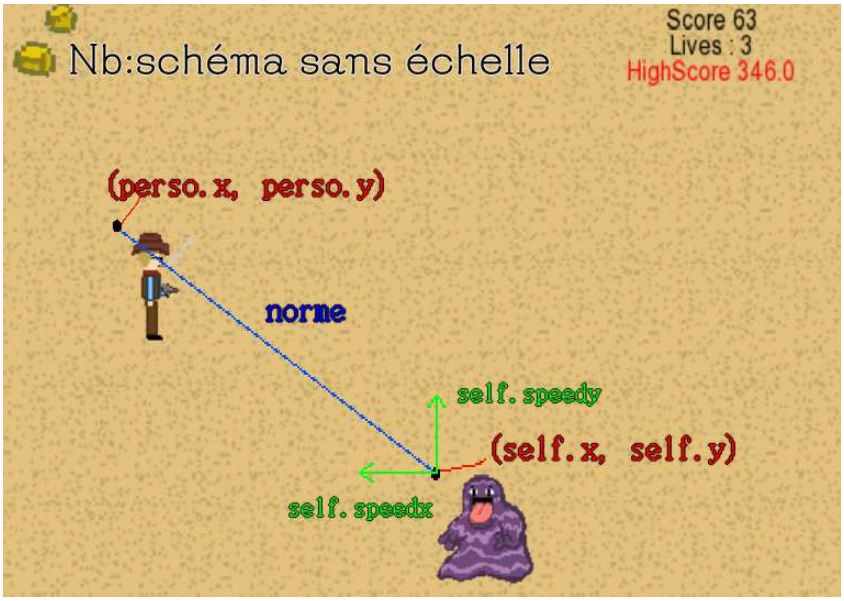

FIGURE 3 – Schéma explicatif

3.2 Déplacement du boss

Mon jeu contient un combat contre un boss. Tous les cinq déplacements d'écran, le joueur doit affronter un ennemis atypique qui prend la forme d'un Minotaure. Celui-ci fonctionne d'une manière similaire à celle d'un ennemi. Nous allons nous intéresser à sa manière de se déplacer,qui est caractérisée par une charge du Minotaure vers le joueur.
J'ai ici deux méthodes, la première va calculer la direction dans laquelle le boss va se déplacer. Cela se fait exactement de la même manière que les ennemis de base (cf. 2.1). La seconde méthode, quant à elle, est bien plus intéressante. Celle-ci gère son déplacement, qui se fait en deux parties. En premier lieu, le Minotaure recule légèrement pendant un cours laps de temps, puis il se rue sur le joueur :

```
    def get_direction(self):
# calcule la norme du vecteur entre lui et le personnage
        norme = math.sqrt(math.pow(perso.x - self.x,2)
            + math.pow(perso.y - self.y,2))
        self.speedx =(perso.x - self.x)/ norme*self.bossspeed
        self.speedy = (perso.y - self.y)/norme*self.bossspeed
        self.last_charge = pygame.time.get_ticks()


# son déplacement
    def move(self):
# si il est apparu moins depuis - de 60 ticks, il ne bouge pas
        if self.time_spawn < 60:
                pass
# si il a fini son déplacement, il s'arrête
        if self.charge_counter == 30:
                self.moving = False
                self.animation_time = 3
```

Afin de savoir s'il doit encore se déplacer, il regarde si "self.charge_counter" est égal à 30. Le cas échéant, sa valeur booléenne "self.moving" est sur "False" ce qui redéfinit son image sur son image fixe et signifie qu'il a fini de se déplacer. Il va donc s'arrêter dans sa méthode "update". La condition else qui suit n'est là que pour l'empêcher de sortir de l'écran.

```
            else:
```

(Je ne fais que l'empêcher de sortir des bordures ici.)

```
# l'animation ou il se prépare à charger
                else:
                    if self.charge_counter < 10:
                        self.animation_time = 6
```

```
                        self.x = self.x - self.speedx/ 30
                        self.y = self.y - self.speedy/30
                        self.charge_counter = self.charge_counter + 1
# la charge
                    elif self.charge_counter < 30:
                        self.animation_time = 3
                        self.x = self.x + self.speedx
                        self.y = self.y + self.speedy
                        self.charge_counter = self.charge_counter + 1
```

Ici, il agit en deux temps. En premier lieu, la variable "self.charge_counter" est inférieure à 10, dès lors, il joue son animation de préparation, augmentant le délai entre chaque image et le faisant se déplacer d'une courte distance dans le sens contraire du vecteur de déplacement qu'il a calculé. Ensuite, tant que cette variable est plus grande que 9 et plus petite que 30, il se déplace vraiment. Sa vitesse d'animation est rétablie à sa vitesse de course et il se déplace à sa vitesse de course définie, sa direction étant donnée par le vecteur qu'il a calculé auparavant.

```
# appelle les methodes une a une
    def update(self):
        now = pygame.time.get_ticks()
        if now - self.last_charge > self.chargedelay:
            if self.time_spawn < 60:
                self.moving = False

                pass
            else:
                self.get_direction()
                self.charge_counter = 0
                self.moving = True

            (...)
```

Finalement, dans sa méthode update, il regarde quand était sa dernière charge et s'il est temps pour lui de recommencer, il recommence son déplacement.

FIGURE 4 – Différents ennemis

Chapitre 4

Améliorations depuis la version intermédiaire

J'ai ajouté et corrigé plusieurs choses à la suite des retours que j'ai reçus de ma version intermédiaire. J'ai ajouté beaucoup de fonctionnalités qui n'étaient pas encore présentes et globalement peaufiné les mécaniques de mon jeu.

4.1 Gestion des tirs à la souris

Une des premières améliorations que j'ai menées à bien était de remanier le système de tir de mon personnage. J'utilisais à la base les touches 'w','a','s','d','q','e',"¡" et 'c' afin de tirer dans 8 directions prédéfinies. à savoir les quatre points cardinaux et leurs entre-deux. Ce système fonctionnait mais était très peu intuitif et manquait de précision. Il rendait mon jeu bien plus compliqué et par la même occasion gâchait un peu l'expérience. Sur les conseils de mes maîtres, j'ai donc opté pour un autre périphérique qui gèrerait les tirs, à savoir la souris. Dorénavant, le personnage regarde en direction de la position de la souris et lorsque le joueur presse le clic gauche de la souris, une balle est tirée, effectuant le chemin le plus court entre le joueur et la position de la souris au moment du tir. Cela implique donc que le joueur n'est plus limité à huit directions pour tirer mais bel et bien une infinité. Dans la classe Perso :

```
# retourne la position horizontale de la souris
            mousex = pygame.mouse.get_pos()[0]
# retourne la position verticale de la souris
            mousey = pygame.mouse.get_pos()[1]
            angle = math.degrees(math.atan2(mousey- self.y, mousex- self.x))
```

Je définis "mousex" et "mousey" comme la position du curseur de ma souris. Ensuite, je calcule l'angle que fait le vecteur entre le personnage et le curseur de la souris avec l'horizontale à l'aide de la fonction atan2(x,y). Cette fonction retourne l'angle (en radians) entre la partie positive de l'axe des x du plan et le point de ce plan (x,y). Il est à noter que l'angle retourné est entre -π et π. Il est positif pour les angles de sens trigonométrique et négatif pour les angles de sens horaire. Etant donné que l'angle retourné est en radians, je le transforme en degrés avec math.degrees car il m'est plus aisé de travailler en degrés dans ce cas. Après avoir calculé cet angle, il ne me reste plus qu'à définir la direction de mon personnage en fonction de la valeur de l'angle. Afin de réaliser cela, je dois calculer le vecteur entre la position de mon

personnage et la position de la souris afin d'obtenir le sens et la direction de celui-ci, puis je le divise par sa norme afin d'obtenir un vecteur unitaire.
Dans la classe Bullet :

```
mouse_x = pygame.mouse.get_pos()[0]
mouse_y = pygame.mouse.get_pos()[1]
 self.speedx = bulletspeed* (mouse_x -self.start_posx) /
 math.hypot(mouse_x - self.start_posx, mouse_y - self.start_posy)
 self.speedy = bulletspeed* (mouse_y -self.start_posy) /
 math.hypot(mouse_x - self.start_posx, mouse_y - self.start_posy)
```

Ces quelques lignes de code définissent ici la direction de la balle. Je calcule ici sa vitesse de déplacement respectivement sur l'axe des abscisses et des ordonnées. Pour cela, je calcule le vecteur entre la position de ma souris et la position d'apparition de la balle en x (pour "self.speedx" et en y (pour "self.speedy") que je divise par la norme du vecteur entre le curseur et le personnage. Pour calculer celui-ci, j'utilise la fonction math.hypot(x,y) qui calcule l'hypoténuse entre deux vecteurs normaux. Cela est équivalent à : $\sqrt{x^2 + y^2}$. Cela me donne un vecteur unitaire que je multiplie par la variable "bulletspeed", afin de faire avancer la balle à la vitesse souhaitée.

4.2 Scrolling

Une des autres améliorations sur laquelle j'ai dû travailler a été d'ajouter du scrolling. Cela ajoute un peu de diversité au décor et le jeu est ainsi moins répétitif. J'ai opté pour du défilement écran par écran, donc rien de progressif. Le principe est le suivant : tous les 30 ennemis tués, les ennemis cessent d'apparaître et le joueur est invité à se rendre sur la droite de l'écran. Ceci fait, le scrolling est déclenché :

```
# triple condition qui déclenche le scrolling
 if (scrollcount >= 30 and perso.x >= scr_width - 64 and flechedisplay == True):
              scroll = True
              scroll_delay = 0.1
```

ici, "scrollcount" est une variable contenant le nombre d'ennemis tués, réinitialisée après chaque écran défilé.
la position en x du personnage doit être suffisamment à droite de l'écran pour déclencher cela.
La variable "flechedisplay" est une valeur qui détermine s'il y a une flèche qui clignote à droite de l'écran. Elle est indispensable ici car sans sa présence, on pourrait changer d'écran même si le boss est en vie.
Intéressons-nous au scrolling en lui-même :

```
# en cas de scroll
            if scroll == True:
# vide les groupes
                    all_sprites.empty()
                    bullets.empty()
                    powerups.empty()
                    boss_powerups.empty()
                    explosions.empty()
                    ennemybullets.empty
```

```
                    ennemys.empty()
                    boss_group.empty()
# désactive la flèche
                    flechedisplay = False
                    minotaure_alive = False

# scroll tous les scroll_delay ( soit 0,1 ticks)
                    now = pygame.time.get_ticks()
                    if now - last_scroll > scroll_delay:
                         last_scroll = now
# décale l'image de fond de 50
                         backgroundImage.scroll(-55)
# corrige la position de mon personnage
                         perso.x -= 54
                         scr.blit(backgroundImage,(0,0))
```

En premier lieu, je vide tous les groupes, puis je désactive l'affichage de la flèche et je définis "minotaure_alive" sur "False" (bien que techniquement ce ne soit jamais nécessaire). Ensuite, je calcule s'il est temps de décaler l'image. Le cas échéant, je décale mon image de fond de 55 pixels vers la gauche. Pour cela j'utilise la fonction pygame.Surface.scroll(dx =0,dy = 0) qui décale la surface appelée de Δx pixels vers la droite et Δy pixels vers le bas. Puisque j'utilise ici une valeur négative, je décale la surface vers la droite. Je dois aussi déplacer mon personnage sur la droite, sinon il resterait à droite de l'écran à la fin du défilement. Comme le scrolling se déclenche si mon personnage est à "scr_widht" -64 et pour éviter un décalage de mon image entre chaque écran, il est déplacé un petit peu moins que l'écran (-54). Finalement, j'affiche l'image de fond qui vient de subir un défilement sur l'écran. Il me reste cependant à arrêter le défilement :

```
if perso.x <=  0:
             ennemys_killed.empty()
             scroll = False
             minotaure_killed = False
             now = pygame.time.get_ticks()
             perso.x += 30

# réduit le délai d'apparition des ennemis
             spawn_delay -= 30
             afterscroll = True

# bouclier qui dure moins longtemps pour éviter de se faire tuer instantanément
 par un ennemi qui apparait sur la droite
             activepowerups.append('invincibility')
             last_invincibility = now - 10000

# incrémente la variable pour qu'au 5e scroll, l'image originale soit affichée
             times_scrolled += 1
             if times_scrolled == 5:

                backgroundImage=pygame.image.load('fond_main_new.png').convert()
                backgroundImage = pygame.transform.scale(backgroundImage,
```

```
                (scr_width*6, scr_height))
                                times_scrolled = 0
# augmente la difficulté d'un niveau
                                level += 1
```

Le défilement s'arrête lorsque la position de mon personnage sur l'axe des abscisses équivaut à 0. Dès lors, le personnage est légèrement avancé pour cause d'esthétisme. Ensuite, le temps entre chaque vague d'ennemis est réduit de 30 ticks. Ensuite, je donne un bouclier au joueur pendant un cours laps de temps afin qu'il soit invulnérable pendant un certain temps juste après le scrolling. Je fais ceci afin d'éviter qu'il se fasse immédiatement tuer par un ennemi malicieux qui apparaîtrait dans un endroit impossible à esquiver. Finalement, j'incrémente la variable "times_scrolled" de 1. Ceci est fait pour que lorsque l'on est arrivé au terme de la longueur de l'image (épuisé tous les défilements), le défilement recommence depuis le début. De plus, tous les 5 écrans défilés, la difficulté augmente de 1 par l'incrémentation de la variable "level" de 1.

Chapitre 5

Conclusion

Pour conclure, j'ai bien développé mes connaissances dans le langage qu'est Python. J'ai pris beaucoup de plaisir à réaliser ce travail, et l'apprentissage n'était en aucun cas un fardeau. J'ai bien entendu rencontré des difficultés, mais avec un peu de persévérance, j'ai toujours fini par résoudre le problème. Tout au long du travail, je dois avouer que les cours de Monsieur Dubey m'ont grandement aidés. Il nous a présenté des exemples et fourni une sorte de boîte à outils qui contenait des choses que j'ai utilisées tout au long de la réalisation de mon jeu. Ceci couplé à sa sympathie naturelle ainsi qu'à celle de Monsieur Roisin, venir au réunions à toujours été fort intéressant. Bien entendu, ces cours seuls n'étaient pas suffisants. J'ai aussi regardé quelques vidéos sur YouTube et analysé de nombreux exemples sur internet, notamment quelques jeux ayant été postés sur le site internet de Pygame. Ce travail de maturité n'a pas seulement développé mes compétences en programmation. J'ai aussi appris à utiliser d'autres programmes comme LaTeX, ou du moins en survol. De plus, je me suis essayé au graphisme par ordinateur, bien que cela m'ait nécessité quelque aide de ma soeur, il m'a plu de dessiner mes propres images. Encore, j'ai composé moi-même quelques musiques, même si je n'ai pas les notions théoriques nécessaires à la réalisation de compositions musicales avancées, le simple fait de me dire que je les ai faites moi-même me satisfait. En définitive, je suis content d'avoir pu suivre ce séminaire. Il était fort intéressant et m'a enrichi non seulement techniquement, mais aussi dans ma capacité à planifier. Ce long travail ne m'a que plus inspiré à me lancer dans un domaine en lien avec l'informatique dans la suite de mes études.

Chapitre 6

Références bibliographiques

Marley Adair, Brian Carling. 2016. *Python Hunting*. Pearson.

site internet 01 (Horst JENS). Consulté le 13 février 2019. *Step 014 - Pygame sprites*. http ://thepythongamebook.com/en :pygame :step014 ?do=index.

site Internet 02 (Paul Craven). Consulté le 13 février 2019. *Chapter 14 : Bullets*. https ://www.youtube.com/watch ?v=PpdJjaiLX6A.

site Internet 03 (Christophe Bertrand). Consulté le 13 février 2019. *Sprites*. https ://sciences-du-numerique.fr/tuto-pygame/sprites.html.

site internet 04 (Dan737). Consulté le 20 février 2019. *Gérer les collisions avec pygame*. https ://openclassrooms.com/forum/sujet/gerer-les-collisions-avec-pygame-1.

site internet 05 (usr2564301). Consulté le 20 février 2019. *Collision not working for pygame Sprites*. https ://stackoverflow.com/questions/27498478/collision-not-working-for-pygame-sprites.

site internet 06 (martineau). Consulté le 25 février 2019. *How to use pygame.surface.scroll() ?* https ://stackoverflow.com/questions/44610428/how-to-use-pygame-surface-scroll.

site internet 07 (DJMcMayhem). Consulté le 7 mars 2019. *Bullets aren't shooting at the EXACT position of mouse :(*. https ://stackoverflow.com/questions/33221308/bullets-arent-shooting-at-the-exact-position-of-mouse.

site internet 08 (KidsCanCode). Consulté le 20 février 2019. *Shmup Part 1*. https ://github.com/kidscancode/gamedev/blob/master/shmup/shmup-1.py.

site internet 09 (KidsCanCode). Consulté le 24 février 2019. *Shmup Part 8*. https ://github.com/kidscancode/gamedev/blob/master/shmup/shmup-8.py.

site internet 10 (Python Software Foundation). Consulté en mars 2019. *Mathematical functions*. https ://docs.python.org/2/library/math.html.

site internet 11 (L. Kocbach). Consulté en mars 2019. *LaTeX Math Symbols*. http ://web.ift.uib.no/Teori/KURS/WRK/TeX/symALL.html.

site internet 12 (Johannes Bo). Consulté en mars 2019. *LaTeX/-Mathématiques*.

site internet 13 (Sogomn). Consulté le 15 mars 2019. *Explosion*. https ://opengameart.org/content/explosion-3.

site internet 14 (GrumpyDiamond). Consulté en mars 2019. *SandTileset 16x16*. https ://opengameart.org/content/more-sandtileset-16x16.

SITE INTERNET 15 (SHOLEV). Consulté en mars 2019. *Shield Aura Effect*. https ://opengameart.org/content/shield-aura-effect.

SITE INTERNET 16 (GUMICHAN01). Consulté en mars 2019. *Shield Aura Effect*. https ://opengameart.org/content/missiles-bullets-and-bomb-set.

SITE INTERNET 17 (INANZEN). Consulté en mars 2019. *Arrow keys, wsad, mouse icon*. https ://opengameart.org/content/arrow-keys-wsad-mouse-icon.

SITE INTERNET 18 (RALLIX). Consulté en mars 2019. *Keyboard keys*. https ://opengameart.org/content/keyboard-keys-1.

SITE INTERNET 19 (MICHAEL KLIER). Consulté le 15 mars 2019. *Large Monster*. https ://opengameart.org/content/large-monster.

SITE INTERNET 20 (PHOENIX1291). Consulté le 15 mars 2019. *Sound effects Mini Pack1.5*. https ://opengameart.org/content/sound-effects-mini-pack15.

Déclaration sur l'honneur

Nom et prénom :	Quentin Angéloz
Adresse :	Chemin de la Forêt 2
	1720 Corminboeuf

Je certifie que le travail Conception d'un jeu vidéo : Dave Unchained a été réalisé conformément aux conditions relatives à la réalisation du Travail de Maturité.

Lieu, date et signature :