PROJET L1 – CMI 2018-2019

spécialité informatique

Les étudiants :
Peron Valentin, Fontaine Valentin et Renaut Thony
vous présentent

SOMMAIRE

INTRODUCTION

1.1 Généralités :

Durant notre première année de licence en tant que CMI, nous sommes amenés à un nouveau module (HLSE205) qui prend part au second semestre, ayant pour objectif le développement d'un jeu vidéo fonctionnel dans un temps imparti.
Ce projet est avant tout un travail de groupe, pour notre cas composé de trois personnes:

-Peron Valentin	CMI Informatique
-Fontaine Valentin	CMI Informatique
-Renaut Thony	CMI EEA

Pour ce faire, nous devons utiliser nos compétences acquises durant le premier semestre, nos connaissances personnelles et faire nos propres recherches. Nous disposons de 25h de cours réparties sur 2 mois avec un encadrement de 1h30 à 3 heures par semaine) ainsi que notre temps libre (travail personnel chez soi).

Le choix du jeu était libre (avec quelques idées et recommandations). C'est pourquoi nous avons décidé de partir sur un sujet assez prometteur et ambitieux, un petit RPG en vue « topdown » (c'est-à-dire vue du dessus et devant des objets) dans un univers médiéval et fantastique avec scénario simple et basique.

1.2 Histoire du jeu :

Vous incarner Darwin, un jeune homme courageux. Voici votre histoire :

« Vous vous réveillez un matin et vous ne vous rendez pas compte que vous êtes dans un cauchemar. Vous devez donc comprendre ce qu'il se passe en enquêtant par vous même, et chercher un moyen de s'évader de cet endroit quelque peu bizarre»

Le monde comportera 5 zones distinctes : le village, la forêt noire, la montagne blanche, le labyrinthe infernal et une dernière zone mystérieuse.

1.3 Cahier des charges :

Prérequis : le matériel obligatoire pour notre jeu est un clavier et une souris fonctionnels. De plus le jeu tourne normalement sur toute plateforme (Windows, Mac, Linux, ...) tant que l'ordinateur dispose de Python 3 et du module pygame ainsi que le module pywin32 si vous êtes sur Windows.

L'outil principal du développement de notre jeu est son éditeur, qui doit donc être à son tour fonctionnel sur toutes les plateformes.

Darwin's Nightmare :

- Le héros doit se défendre contre des zombies et un boss, il peut donc se déplacer, attaquer, perdre de la vie ou de l'armure quand il se fait attaquer.

- Il y a également un système de commerce chez le forgeron et l'alchimiste, qui proposent potions, armure et améliorations pour l'équipement et les consommables.

- Les zombies suivent le joueur, ils sont donc équipés d'un système de suivi qui prend en compte les obstacles. Pour cela ils possèdent une I.A. très simple pour contourner les murs les plus simples, mais sont cependant bloqués assez facilement si la forme du mur se complique. Si le héros tue un zombie, il récupère des pièces d'or de manière aléatoire.

- Il y a des PNJs (personne non joueuse) qui aide le joueur en donnant des informations. Le joueur peut donc interagir avec eux, et noter les informations apprises grâce à son carnet de note.

- Un inventaire est aussi présent, il peut contenir :
    - une arme de corps à corps (épée)
    - une arme à distance (arc)
    - des potions de vie et de bouclier
    - des pièces d'or
    - des ressources (du bois et du fer)
- Des coffres sont mis à disposition du joueur pour qu'il puisse ranger son équipement.

- Un système de jour / nuit est aussi présent, avec 5 minutes pour le jour et 5 minutes pour la nuit. Cette dernière réduit considérablement le champ de vision.

- Sur la carte, le joueur peut rencontrer des fontaines qui sont des points de sauvegarde de position et de réapparition.

- Une caméra suit notre héros qui se situera au centre de l'écran. Elle se déplace casse par casse.

- Le boss final est un zombie fantôme, beaucoup plus résistant que les zombies normaux. Pour le faire apparaître il faut activer les 3 reliques dispersées dans le monde.

- La police d’écriture est faite à la main, avec un système d’affichage lui aussi créé de toute pièce.

- Enfin, il y a un écran d’accueil pour pouvoir reprendre sa sauvegarde ou en recommencer une nouvelle.

L’éditeur :
- Le but de l’éditeur d’améliorer et d’accélérer l’avancement du projet car le niveau de programmation dans le groupe est très inégal. Ainsi il permet l’accès à certain outils de développement à ceux qui n’ont pas les capacités de les programmer. Il est composés de 4 applications :

- La « map » qui est la carte. On peut se déplacer avec les flèches directionnelles, poser des blocs et les supprimer. Il y a 2 modes disponibles, « pen » pour ne poser qu’un bloc à la fois, et le « bucket » pour remplir une zone d’un même bloc.
    - Il est possible d’annuler ce que l’on vient de poser (retour en arrière).
    - On peut décider d’afficher ou de cacher la grille.
    - Une « pipette » est disponible pour sélectionner un bloc déjà posé.
    - Il est possible d’enregistrer la carte dans un fichier.

- L’application des blocs, pour sélectionner les images dans la partie haute de l’application. Dans la partie basse on peut sélectionner le dossier d’images souhaité.
Les images sont triés par hauteur et par largeur.

- L’application des « layers ». Un « layer » est une couche avec un nom et un ID sur laquelle on peut poser des blocs. On peut en ajouter autant que l’on veux avec le nom que l’on veut, mais c’est définitif. En effet il est impossible de supprimer une couche.

- L’application des propriétés. C’est dans cette application que l’on peut voir les informations d’un bloc et en rajouter. Un bloc peut avoir une infinité de propriétés et il est possible de les supprimer. Une propriété doit avoir la syntaxe suivante :

nom : mode : valeur

il existe 4 modes de propagation d’une propriété :
- uniquement sur un bloc
- par remplissage (tant que les blocs voisins sont les mêmes, ils héritent de la propriété) - - sur tout les blocs visibles qui sont identiques au bloc sur lequel on a cliqué
- sur tout les blocs de la carte qui sont identiques au bloc cliqué.

ORGANISATION DU PROJET

2.1 Organisation du travail :

Nous nous sommes répartis les différentes tâches en fonction des capacités de chacun et de la difficulté envisagée de certaines parties du programme. Durant les cours dédiés au projet, nous avons faisions essentiellement des mises au point sur l'avancement et décidions de qui allait faire quoi durant la semaine. Nous recherchions également des formules (mathématiques) pour certains algorithmes.

En dehors des cours dédiés, nous avons chacun travaillé et développé chez nous en communiquant notre avancée sur un serveur discord créé spécialement pour le projet. Ainsi moins de 20 % du projet a été fait pendant les 25h de cours.

La communications c'est faite comme énoncé précédemment autour d'un serveur discord, où nous avions la possibilité de se contacter, échanger et partager très rapidement nos avancements.

De plus, nous avons utilisé l'éditeur de texte Atom pour coder car celui-ci dispose d'une fonctionnalité (à installer) qui permet de travailler simultanément à plusieurs (en ligne) sur le même programme, avec les modifications qui s'effectuent en temps réel.

Le travail a été réparti en trois parties :

- L'éditeur et chargement de la carte en jeu — Fontaine Valentin
- Éléments de gameplay et graphismes — Peron Valentin
- Graphismes et petits algorithmes — Renaut Thony

DURANT LES TD

HORS TD

TD 1

-Recherche du projet

-premier schéma explicatif

-idée du thème

-organisation des premières tâches

-Installation du module pygame pour Thony Renaut

-Réflexion sur le début du programme

-Rédaction du cahier des charges et Début du rapport.

TD 2/3

-Création de la première fenêtre du jeu

-Première classe pour le système de mouvement

-Début de la création de l'éditeur

-Premières textures réalisées

-Système d'écriture : police

-Premier test des collisions

-D'autres textures

-Création de mouvements dynamiques Avec des vecteurs, concluant mais pas Adapté, donc on reste sur du case par case

-Création des collisions

-Continuité de l'éditeur

-Création de la classe des armes

-Développement de l'éditeur, nouvelle fonctionnalités

-Quelque textures

SEMAINE DE VACANCES

-Créations de toutes les textures du jeu ainsi que la productions des sons et background

-Programmation de l'ennemie, de son système de suivie, début de l'inventaire.

-Création du boss et finissions des armes à distances, dégâts, vies,
Armures et le système de dialogue.

Durant les TD

Après les vacances, les séances de TD ont
Servi à faire la mise au point chaque
Semaine de ce qu'il restait à faire, et nous
Mettions en commun les bouts de
Programmes que chacun avait fait depuis
Le dernier TP. Par la suite nous avons
Produis le menu du jeu avec le fond
Dynamique, et également on a avancé la
Map en la modelant petit à petit avec les
Différentes régions du jeu.

L'amélioration de l'I.A du zombie puis
La fonction permettant de crée les
Les dialogues avec les pngs du jeu.

Hors TD

Avant les vacances, chacun faisait un peu
Ce qu'il voulait coder et l'avancement
n'était pas régulier.
C'est pourquoi après les vacances nous
Avons décidé de mieux organiser le travail
Que chacun faisait chez lui.
Ainsi l'avancement est devenu linéaire et
Beaucoup moins saccadé.

Il y a eu la continuer de ce qui à été fait en
TD et d'autre fonction vital au jeu tels que
Le cycle jours nuit qui est très important,
Le système de dégats.

Beaucoup de temps a été dédier à l'inventair
Du joueur, les coffres puis les marchants
Du jeu.

TD suivant

2.2 Choix des outils de développement :

Durant la première année de licence, nos bases en informatique se forge autour du C/C++ et de manière logique, nous aurions dû nous tourner vers ce langage pour notre projet. Cependant, nous avons décider d'utiliser Python pour sa simplicité et son accessibilité sur les différentes plateformes, et aussi parce que nous étions plus à l'aise avec ce langage où nous avions déjà de très bonne bases pour deux d'entre nous.

Thony ayant un niveau un peu moins élevé du fait qu'il soit en EEA et non en informatique, la syntaxe et la simplicité de Python ont pu l'aider à programmer, avec les nombreux tutoriels sur internet.

Nous avons en effet pris Python pour sa syntaxe car avec un peu d'entraînement on arrive beaucoup plus vite à faire des progrès qu'avec d'autre langage. Ainsi la programmation orientée objet facilement accessible.

Pour notre jeu et surtout son interface graphique, nous utilisons le module pygame qui une fois importé nous permet d'utiliser pleins de fonctionnalités liées à l'affichage des textures à l'écran. Pygame est aussi une bibliothèque libre de droits et multiplate-formes, ce qui facilite le développement et la distribution de note jeu. De plus pygame possède une documentation très détaillée sur son propre site qui explique le fonctionnement de toutes les commandes possibles, suivi d'exemples.

Pour programmer, nous étions au début sur sublime text, car c'est l'éditeur que l'on utilise à la l'université, mais après avoir découvert Atom, nous avons migré sur ce dernier. Atom est un éditeur de texte moderne, rapide et extrêmement puissant. En effet il possède de nombreux add-ons, pour le rendre plus utile, comme avoir la possibilité de coder en simultané sur le même code. C'est pour cette raison que nous avons utilisé cet outil.

En comparaison au C/C++, python ne crée pas d'exécutable, il faut lancer le programme main à partir de l'invite de commande du système d'exploitation utilisé. Python n'est pas un langage compilé mais interprété, ce qui peut le rendre moins performants dans certains cas.

Pour ce qui est des graphismes, ils ont été fait en grande majorité sur Piskel, un site très utile pour créer rapidement des pixel art. Nous avons utilisé Bfxr pour les sons, et Medibang Paint Pro pour l'écran d'accueil.

ANALYSE DU PROJET

3.1 Réflexions préalables:

Pour réussir à programmer ce que l'on voulait, nous sommes passés par la case papier. En effet, la plupart des algorithmes et des fonctions du projet sont assez voir très complexes. C'est pourquoi avant de coder et avant même la première heure de cours (en Janvier, car nous savions le projet que nous voulions faire), nous avons réfléchi sur papier pour savoir à quoi ressemblerai le projet final.

Pour le jeu en lui-même ce procédé à peut être été un peu moins utilisé mais pour l'éditeur cela représente plus de 20 heures. En effet, créer un petit logiciel était une première pour nous, et un gros travail de réflexion a été fait pour en arriver à l'état actuel de l'éditeur, qui est une version qui nous ressemble. Par « qui nous ressemble » nous entendons un logiciel qui est fait de nos propres mains et dont le visuel correspond à une image qui nous est propre, et non une simple copie d'un logiciel déjà existant. Cela peut sembler facile mais lorsqu'on n'y a jamais réfléchi c'est très difficile de penser à quelque chose qui ne se ressemble pas à quelque chose d'existant.

3.2 Structure du projet :

La base de notre projet est l'éditeur, sans lui il n'y a aucun jeu, car c'est cet éditeur qui permet la réunion de la carte, des différentes propriétés des blocs et le programme principal contenant les éléments de jeu.

L'éditeur lui-même repose sur swapcore, une petite API (application Programming Interface) que nous avons créé. Swapcore est le squelette, la base de tout le jeu. Cette API est composé de 2 fichiers, celui des constantes et celui des classes (le kernel).
L'éditeur est lui aussi composé d'un classes, avec un dernier fichier main qui est le programme principale qui se charge de tout assembler et de recevoir les éventements clavier / souris. C'est également dans le dossier de l'éditeur que la sauvegarde de la carte se trouve.

Pour ce qui est du jeu, c'est la même structure que l'éditeur (main et classes). Dans le fichier classes se trouvent les éléments de jeu (inventaire, joueur, zombie, caméra, …)

Enfin, le dossier data regroupe toutes les ressources qui ne sont pas du code, c'est-à-dire les images, la police d'écriture, les sons et la sauvegarde du joueur.

DEVELOPPEMENT

4.1 Généralités sur le gameplay et I.A :

La partie gameplay et I.A est celle a lancer pour avoir le jeu, elle est composé d'un programme main, un init et un classes qui sont importés dans le main.

Par ailleurs, dans « classes » ont import d'autres fonctions qui proviennent de l'éditeur, comme par exemple le kernel.Area qui est une surface avec des coordonnées.

C'est dans le main.py contenant toutes les classes pour le gameplay, que la map est chargée partiellement et non dans son intégralité car étant trop grande, cela impliquerait des moments de latences et des FPS trop faibles pour apprécier le jeu, et ferait tout simplement planter le programme qui ne supporterai pas une surface si grande.

C'est donc notre classe Camera qui charge uniquement un espace autour du joueur, bloc par bloc et chunk par chunk dès qu'on avance. En réalité la partie de la carte qui est chargée est plus grande que celle que nous voyons pour avoir la place de faire apparaître les zombies.

La classe ZombieSpawn se charge de faire apparaître les zombies en dehors de votre champ de vision et de amener à vous.

Le système de sauvegarde de la map, et des données du joueur lors des checkpoints sont tous deux différents. La map est codées dans un système binaire grâce au module pickle, tandis que les données du joueur sont des valeurs décimales mises à la suite dans un fichier .txt et ne sont donc pas binaires.

Le programme commence par le init.py, grâce à lui, on peut directement changer les paramètres d'affichage, est set_caption (nom de la fenêtre) puis importer les bibliothèques souhaitées et autres.
On va donc importer l'init dans le fichier classes, et importer classes dans le main.

```python
import swapeng.swapeng016.swapcore as swapcore
import pygame
from pygame.constants import *
import pickle
import random
import time
import os


RUNNING = True
PROGRAM_WIDTH, PROGRAM_HEIGHT = 1280, 720
PROGRAM_DIMENSIONS = (PROGRAM_WIDTH, PROGRAM_HEIGHT)
ORIGIN = (0,0)


pygame.init()
pygame.mixer.init(22050, -16, 2, 4096)
screen = pygame.display.set_mode(PROGRAM_DIMENSIONS)
pygame.display.set_caption("Darwin's Nightmare")

```

Classes.py contient l'ensembles des class et des def, dont celles mentionné ci-dessous.

```
 36   class Title: ...
114
115   class jourNuit(): ...
170
171   def draw_lives(surf, x, y, lives, img): ...
179
180   def shield(surf, x, y, shield, img): ...
187
188   def store_data_to_file(filename, *data): ...
192
193   def load_data_from_file(filename, amount=None, index=0): ...
212
213
214   class Clock: ...
242
243
244   class Images: ...
268
269
270   class Sounds: ...
274
275
276   class Player(swapcore.kernel.Area): ...
286
287
288   class GUI(swapcore.kernel.Area): ...
547
548
549   class NoteBook(swapcore.kernel.Area): ...
630
631
632   class Camera: ...
931
```

La première classe est celle de l'écran d'accueil / ou écran titre, qui est composé de 4 définitions. Elle va tout simplement gérer le mouvement du background, et les événements qui s'y produisent afin de les envoyés vers d'autres classes.

Pour démarrer, on crée une variable « on_title », en position True, qui va nous servir de switch dans la boucle principale.

Pour la classe jourNuit(), on appelle dans le main() seulement la partie update() de la classe, car c'est celle qui va gérer les autres fonctions. Pour celle-ci, on utilise le module time qui permet de jouer avec le temps pour avoir nos conditions. Une journée basique dure en réalité 10 minutes, avec 5 minutes de nuit et 5 minutes de jour. On commence par appeler la fonction « commencer_nuit » qui va initialiser notre timer principal et crée ce halo lumineux qui se centre sur le personnage principal.

Pour avoir l'effet transparent, on a créé une surface sur laquelle on a blit un cercle. Pour creuser cette surface à partir de notre sercle, il suffit de mettre en troisième paramètre dans le blit : « special_flag=BEND_MULT » Cette commande est propre à pygame, ont la trouvé en cherchant sur la documentation du site.

Les définitions qui suivent sont assez basiques et servent à blit les coeurs du personnage et son shield en fonction de la classe player et de son nombre de vies enregistrer dans le dossier de sauvegarde gérer par la fonction store_data_to_file().

Les deux classes Images et Sounds sont là pour initialiser tous les fichiers et nous permettent d'avoir un programme carré et organisé.

La classe player fait partie d'une des classes mères du programme. On y relit la camera et tout autre fonction. À partir de la classe player on gère ses points de santé, son argent, son armure, son inventaire, sa position, etc.

La classe Gui est là pour la gestion et la création de l'inventaire et du shop qui assez complexe, et qui a été compliqué à programmer, car il a fallut prendre en compte beaucoup de choses, surtout les gestions de la souris, faire attention à ne pas être sur une bordure, privilégiés les espaces vides de l'inventaire. Chaque case a ses propres coordonnées en chiffre unitaire.

Le système de suivi est simple à concevoir, les mobs se déplacent en diagonale initialement vers le joueur, avec la distance qui est simplement calculée entre les deux, et qui update à chaque nouvelle frame du jeu, donc si le joueur ne se déplace pas, le zombie avancera quand même. De plus, pour qu'ils ne foncent pas vers un mur ou un obstacle, on a fait en sorte que s'ils nous voient plus (direction coupée par une collision), ils retourneront au dernier emplacement où ils ont apercu pour la dernière fois le joueur, et avancer dans le sens initial à ce point.

Le fonctionnement du main : se base sur une boucle while avec la variable « RUNNING » qui tant qu'elle est vrai, fait tourner le programme. Donc dès que l'on souhaite stopper le programme (quand on clique sur la flèche ou en appuyant sur echap), ça retourne False, qui passe RUNNING en état False et met fin au programme car la ligne suivante est « pygame.quit() », c'est ligne ferme l'instance et le processus en cours.

Dans cette boucle pour appeler nos fonctions ou par exemple notre écran d'accueil, on dit que RUNNING = Title_screen(…) pour modifier l'instance. Puis on return True dans la classe Title_screen pour lancer la fonction suivante et donc démarrer la partie.

4.2 Éditeur :

L'éditeur est la partie la plus complexe du projet intégralement super visé par Fontaine Valentin. Il y a la même formation qu'au dessus, c'est-à-dire une fichier classes.py, init.py et un main.py, en plus du dossier swapcore qui contient le kernel.py (qui est l'API).

Il est composé dans sa majorité des classes App sont les application layers, map, properties et Blocks.

La police que l'on utilise dans le jeu a été codée à la main, chaque lettre et chiffre sont écrits dans un fichier avec un code spécifique, puis lu par notre programme qui traduit ces lignes et les affiches. C'est comme ça que notre Police est faite. Pour être plus précis et concret voici l'explication :

a52012 2014 2016 0005 0343 

Le premier caractère de la chaîne est la lettre qui doit être dessinée. Le deuxième est la longueur en pixel de lettre (pas besoin de préciser la hauteur, toutes les lettres font 9 pixels de haut : 5 pixels pour le corps, 2 pixels pour les lettres hautes et 2 pixels pour le bas). Le reste de la chaîne est composé d'instructions de 4 chiffres pour désigner (dans l'ordre):

- la longueur de la ligne à dessiner
- 1 (-1 car 0 désigne déjà un rectangle de 1 pixel) - la hauteur
- 1 (si on veut dessiner une ligne verticalement)
- la coordonnée x
- la coordonnée y

Pour la lettre a l’algorithme comprend donc :

2012 : dessine une ligne de 3 par 1 en (1, 2)
2014 : dessine une ligne de 3 par 1 en (1, 4)
2016 : dessine une ligne de 3 par 1 en (1, 6)
0005 : dessine une ligne de 1 par 1 en (0, 5) (ligne de 1 par 1 = 1 pixel)
0343 : dessine une ligne de 1 par 3 en (4, 3)

Ce qui correspond exactement au a de l’image ci-dessus.

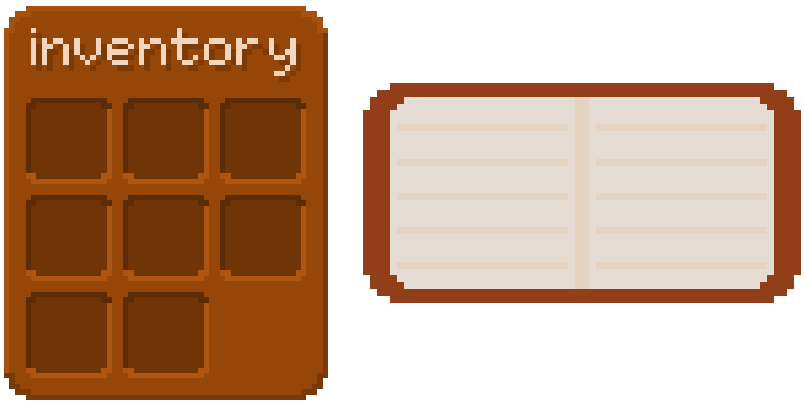
4.3 Interface du joueur :

Le joueur a accès à plusieurs informations. Tout d’abord son nombre de vies sur le bord haut à gauche. Les vies en rouge sont ses cœurs et ceux en gris, son armure.

Quand la nuit tombe, la visibilité est réduite. En effet on ne voit plus qu’a travers un petit cercle, le reste étant noir

C’est durant la nuit que l’on risque le plus de ce faire surprendre par les zombies, qui eux, vont apparaître en dehors de votre champs de vision et arriver dangereusement sur vous.

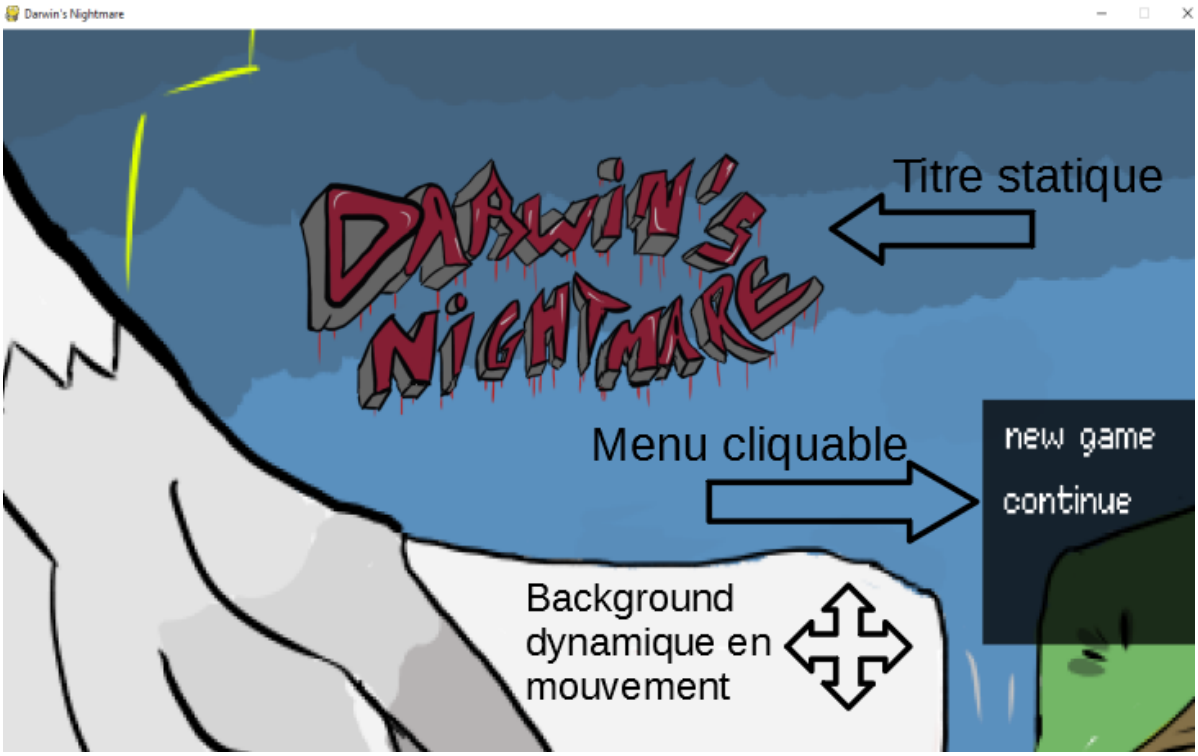
Ensuite votre inventaire fais partie de votre interface, et est accessible à tout moment, tout comme votre carnet :



4.4 Graphique :

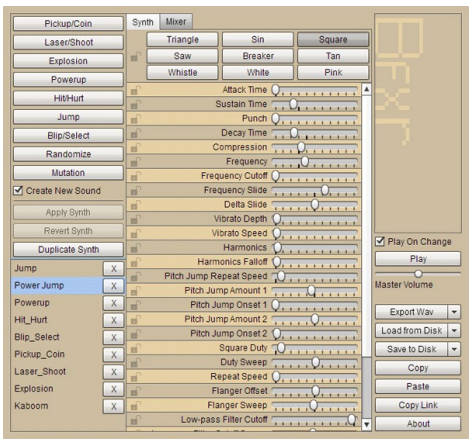
L’aspect graphique du jeu est basé sur le pixel art, et le jeu rétro. C’est pourquoi, Piskel nous a été très utile pour crée nos textures dont certaines sont très légèrement inspirée de jeux existants. Au total nous dépassons le nombres des 100 textures.

Voici un aperçu de l’écran d’accueil :



Son et bruitages :

Pour le son et les bruitages, ils sont réalisés à partir de bfxr un logiciel qui crée des son à partir de fréquences en 8bit. Pour le reste c’est du bruitages fait maison à partir d’un enregistreur vocal, puis des modifications audio.



4.5 Difficultés rencontrées :

Lors de la réalisation du projet, nous avons rencontrés plusieurs grosses difficultés.
Premièrement, pour l'éditeur, la réalisation de l'AppManager a été très difficile. Il s'agit de la classe qui gère l'ouverture des applications de l'éditeur, reçoit les évènements clavier / souris et les redirigent vers l'application qui a le focus. C'est le côté théorique et algorithme de cette classe qui a été très difficile à réaliser.

Ensuite, la création de la barre de défilement a été aussi très difficile car même si au premier abord cela peut paraître trivial, ça ne l'est pas du tout car il y a énormément de cas d'utilisation différents. Par exemple lorsque de l'écriture du nom d'une propriété, la barre de défilement doit être mise à jour à chaque nouvelle lettre.

Pour la partie gameplay, c'est l'IA qui n'a pas été évidente à mettre en place, surtout pour permettre aux zombies de passer un obstacle.

Aussi, la transition entre le jour et la nuit a posé des problèmes car elle au début elle ne se faisait pas du tout.

Enfin, l'inventaire étant assez complexe, ça a été difficile de faire en sorte de l'on puisse bien acheter les objets et qu'ils se placent dans l'inventaire et ne reste pas dans le magasin.

MANUEL D’UTILISATION

5.1 Darwin’s Nightmare :

5.1.1 Menu :

Au lancement du jeu (en faisant **python3 main.py** à partir de l’invite de commande ouvert à partir du dossier du jeu), vous atterrissez dans le menu principal, avec une musique de fond, le titre du jeu et le fond animé qui se déplace, puis le menu se situant en bas à droite comme ci-dessous :

« new game » vous fera commencer le jeu à zéro, puis « continue » va démarrer votre dernière sauvegarde, ou une nouvelle partie si jamais vous n’avez pas sauvegardé avant.

5.1.2 En jeu :

Une fois dans le jeu, vous serez livré à vous même, le but sera de récolter des informations aux près des différents PNJs du jeu qui auront différents dialogues et indices à vous donner, pour pouvoir finir votre quête.

Les différentes commandes :

-Pour se déplacer : les flèches directionnelles ou W A S D
-Pour l’inventaire : la touche « i »
-Pour le carnet de note : la touche « n »
-Pour tirer / attaquer : clique droit / gauche
-Pour utiliser une potion : F pour la vie, G pour le bouclier
-Pour interagir avec quelque chose : « e »

Au départ vous débuterez avec seulement vos poings et un seul cœur de vie. Mais vous allez rencontrez des ennemies et les tuer vous donneront des pièces de manière aléatoire (entre 2 et 5).

Ces pièces d’or vous serviront par la suite à améliorer votre nombre de vie (jusqu’à 3 maximum, votre nombre d’armure (3 cœurs gris maximum). Vous pourrez également dépenser votre argent et vos ressources dans les consommables et l’amélioration de ces derniers. En effet les potions de vie rouge vous redonnerons 1 cœurs de vie alors que la potion de vie dorée vous en redonnera trois.

Voici le magasin de l’alchimiste ainsi quelque visuels de ressources:

Pour acheter, il faut composer une des recette indiqués dans les cases blanches, si la recette est reconnue, alors l'objet final apparaîtra sur la troisième case. Vous pourrez alors cliquer dessus pour l'acheter.

Dans les différents coffres du jeu, il y aura des matériaux comme ceux ci-dessous, à savoir du fer et du bois, qui vous permettrons d'acheter des consommables :

Dans le jeu pour sauvegarder votre progression, vous devez trouver des fontaines. Ce vos points de réapparition si vous mourrez.

5.2 Éditeur :

L'éditeur permet de créer votre propre monde et jouer sur vos créations. Dans le dossier « backup », il y a une carte vierge que vous pouvez modifier à votre guise.

Il suffit soit de modifier la carte actuelle, soit de la remplacer par celle dans le « backup » (elle se nomme « blank », et il faut la renommer en « map » si vous l'utilisez).

Ensuite comme pour lancer le jeu, il suffit à partir de l'invite de commande lancer à partir du dossier de l'éditeur et taper «python3 main.py».

Une fois dans l’éditeur, vous pouvez :

-Déplacer la carte avec les flèches directionnelles
-Clique gauche pour poser le bloc et clique droit pour le supprimer
-B pour le remplissage et P pour le stylo (bloc par bloc)
-O pour la pipette (prend le bloc sur lequel vous cliquez à votre layer)
-S pour sauvegarder

Ensuite vous pouvez tout simplement naviguer entre les différents blocs grâce au menu, et changer de layer. Un layer est tout simplement une couche, vous démarrer au layer 1 et vous pouvez en ajouter autant que vous voulez en donnant le nom que vous voulez. Néanmoins chaque layer rajoute un certain nombre de bytes à la taille du fichier de sauvegarde de la carte, il faut donc être prudent.

Exemple d’utilisation des layers:
- layer 1 : l’herbe
- layer 2 : objets divers comme les arbres
- layer 3 : les toits des maisons

Pour ajouter une propriété sur un bloc il suffit d’ouvrir l’application dédié, appuyer sur la touche « ctrl » pour afficher les propriétés d’un bloc. Ensuite cliquez sur le bouton vert pour ajouter une propriété. Vous devez alors entrer son nom suivant cette syntaxe :

nom : mode : valeur

le mode est au choix : o (1 bloc) / b (remplissage) / a (bloc visible) / v (tout les blocs)

Voici une vision globale de l’éditeur :

CONCLUSION

6.1 Perspectives :

Par rapport au cahier des charges, tout a été fini dans les temps, mêmes les sessions de « debugging » ont pu aboutir avant la version final.

Concernant ce que l'on pourrait ajouter au jeu avec plus de temps, ce serait d'abord plus d'animations pour les zombies, un déplacement fluide et non saccadé (toujours case par case mais la transition entre 2 blocs se ferait au fur et à mesure, pas instantanément) ainsi qu'un boss plus travaillé et des textures plus soignés. Avoir plus d'armes, un mode deux joueurs et différents modes de difficulté pourrait également être intéressant.

Concernant l'éditeur, nous pourrions ajouter des raccourcis clavier supplémentaires, comme pouvoir ouvrir et fermer une application avec le clavier. Seulement au vu de la construction actuelle de l'API sur laquelle repose l'éditeur cela ne serai pas facilement faisable et impliquerai de réécrire complètement certaines parties du programme.

6.2 Conclusions :

6.2.1 Conclusions du projet (JEU) :

Pour commencer par ce dont nous ne sommes pas satisfaits, il y a les textures qui ne sont pas spécialement jolies pour certaines. Aussi l'IA des zombies n'est pas géniale car ils sont très vite bloqués si l'obstacle n'est pas juste un bloc.
De plus, concernant les bugs, il y a les zombies qui peuvent parfois apparaître sur les toits, certaines collisions sont mal placées (le joueur peut se heurter à des blocs invisibles). Enfin, les clics au niveau du menu sont assez aléatoires, mais nous comptons fixer ce dernier avant l'oral.

Concernant les outils utilisés, nous sommes convaincus que choisir Python a vraiment été le meilleur choix car jamais nous n'aurions pu réaliser un tel projet en C++. Cependant nous pensons que l'analyse du projet avant la réalisation n'a pas été assez prise au sérieux, car nous avons dû reprogrammer et adapter notre code pour que nous puissions ajouter ce que nous voulions.

La création de l'éditeur a été un réel défis très plaisant et vraiment producteur pour l'amélioration de notre niveau en python. Puis il nous a été très utile pour la création de la map est du jeu.

Finalement, nous sommes assez fières dans l'ensemble de notre projet en voyant tout ce que l'on a réussi à faire en quelques mois.

6.2.2 Conclusion du projet (GROUPE) :

Notre plus gros ennemi a été le temps. En effet finir le projet dans le temps imparti a été un réel défi pour chacun des membres du groupe. Pour contrer ce temps assez cours, nous avons dû définir très précisément qui devait coder tel partie, et c'est cette organisation qui nous a vraiment aidé à terminer le projet.

Ce projet à vraiment été très enrichissant, d'une part pour les 2 moins bon d'entre nous car ils ont pu être aidé par le troisième pour certain algorithme et améliorer leur compréhension de la programmation orienté objet et leur logique.

D'autre part, pour le troisième qui possède plus bases, il a dû apprendre à bien expliciter son code et sa logique, ce qui n'est pas forcément facile. En effet il faut arriver à expliquer avec des mots sa logique et sa pensée, et pour les algorithme les plus complexes c'est assez difficile.

Pour conclure, nous avons tous « grandis » en programmation et avons énormément amélioré notre efficacité en travail de groupe lors de la réalisation de ce projet.