Rapport du jeu Manic Shooter

Tomy Demazeau Nicolas Masson

Sommaire

1 Les objectifs du projet

1.1 Description du concept derrière l'application

Pour pouvoir enoncer correctement les objectifs du projet que nous avons effectué, il est bon de revenir sur un point important : Qu'est-ce qu'un manic shooter ?

Parmi la multitude de types de jeu qui existent dans le petit monde du jeu-vidéo, certain se sont forgés avec le temps et les innovations des créateurs, d'autres ont émergés dès les tout débuts, les jeux d'arcades. En effet, avant l'apparition des plateformes nous permettant de jouer à la maison comme aujourd'hui (consoles de salon, consoles portables ou même les ordinateurs) les jeux se sont vus distribués sur des plateformes d'arcades. Des jeux restant assez simples, on y retrouvait les jeux de voitures mais également ce qui nous interesse, les 'Shoot them up'.

Les Shoot them up sont des jeux dérivés de jeu d'action dans lesquels le joueur se retrouve à diriger un véhicule ou un personnage et devant détruire les ennemis apparaissant à l'écran. Les apparences et les contextes de ces jeux n'étant bien entendu qu'un pretexte pour justifier un gameplay souvent nerveux. De son coté brutal, les shoot them up pouvaient laisser libre cours à une difficulté exponentielle, laissant les joueurs les moins téméraires sur la touche. Ce style de jeu ne laissait donc peu, voir aucune place aux scénarios et aux développements d'une quelconque histoire, ils étaient donc conçus pour des parties courtes mais très captivantes, incitant ainsi le joueur à remettre une pièce pour rejouer.

Partant de ce style, de nombreuses variations ont vu le jour. Certaines n'étant que des changements très discrets sur par exemple la mobilité (multi-axiale, monodirectionnelle), revenant presque à les classer en sous-genre. Pour ne citer que le plus connu : le 'arena shooter' proposant une liberté totale de mouvement au joueur, le plaçant au centre de l'ecran et faisant apparaitre des vagues d'ennemis pouvant arriver de toutes parts. Qu'en est t'il du Manic shooter ? nous y venons.

Le Manic shooter de son nom anglophone 'bullet hell' (enfer des balles), tire son épingle du jeu notamment du fait que le nombre de projectiles prend le pas sur le nombre d'ennemis au cours d'une partie. Tous l'interêt du jeu résidant dans l'esquive des balles et la survie. En effet la plupart des manic shooter n'exigent pas au joueur la destruction d'un ennemi mais plutôt la capacité à esquiver les tirs de ses assaillants. De plus les entités ennemies sont géneralement plus lentes que sur d'autres styles de shooter.

Les objectifs qui ont été établis sont plus ou moins différents selon le projet choisi par les groupes, cependant les points important restent commun à n'importe quel projet. Ces objectifs commun se résument en quelques points :

— Pouvoir mener un projet en groupe, savoir répartir la charge de travail équitablement.
— Savoir utiliser un logiciel de versions, ici Forge.
— Pouvoir utiliser des librairies externes, se familiariser avec leurs particularités et les implémenter au mieux à son projet.
— Avoir une certaine vision globale et ainsi voir les points importants de son projet.
— Savoir présenter son projet à des personnes extérieures à celui-ci.
— Savoir écrire un rapport complet et détaillé sur son projet.

1.2 Ce qu'il fallait faire

Pour les projets de manic shooter, quelques points ont été énoncés lors de la description des différents projets possibles, notamment :

— Mettre en place un système de vagues d'ennemis.
— Prévilégier un gameplay dynamique.
— Intégrer des éléments utiles à l'expérience de jeu (Bonus, amélioration...).
— Mettre en place un système de niveau qui puisse s'adapter aux performances du joueur.

En partant de ces consignes, nous devions nous organiser pour rassembler des idées. Il fallait en effet choisir des parties pris qu'en à la forme que prendra notre jeu, la résolution et la taille de la fenêtre, l'agencement de l'interface, le sens de défilement du jeu et la mobilité du joueur. Il y a bien sûr de nombreuses autres facettes à prendre plus ou moins en compte selon les premiers choix fait sur notre jeu.

Les scéances dédiées à la conception ce sont déroulées de façon à ce que nous puissions concevoir étape par étape notre jeu. Tout d'abord il a fallu prendre en main l'outil de versions decentralisés Forge, savoir récuperer les fichiers de notre dépot et pars la suite pouvoir envoyer nos modifications et gerer les différents conflits possibles. Ensuite nous avons dû nous familiariser avec la librairie Pygame contenant bon nombre de méthode facilitant ainsi la mise en place de nos projets de jeu.

Nous avons du créer le jeu entièrement. Pour pouvoir créer le jeu ils nous a fallu créer les différents graphisme du jeu comme le fond de chaque niveaux, les ennemis ou même les apparences du joueur. Nous avons très vité

décidé de partir sur un jeu à quatre niveau bien distinct. Pour chaque niveau nous avons décidé de créer des graphismes différents, mais aussi d'avoir une difficulté différente suivant le niveau. Donc bien évidement, le niveau un est simple. Le dernier niveau, le quatre est quant-à-lui beaucoup plus complèxe. Pour augmenté la difficulté du jeu, nous avons voulu augmenté le nombre d'ennemis et modifié leur place pour destabilisé le joueur et rendre le jeu plus compliqué.

1.3 Ce qui existe déjà

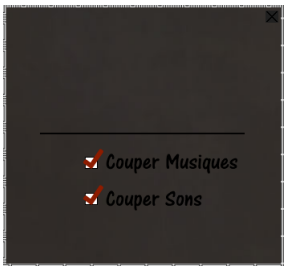
Pour les projets soumis à notre promotion pour la conception logicielle, on nous a proposé à chacun des outils, que se soit une librairie, un moteur de jeu ou un IDE (un environnement de developpement). Pour les projets de Manic Shooter, nous avons Pygame à notre disposition.

pygame est une bibliotèque libre qui permet d'aider au développement de jeu vidéo avec le langage de programmation Python.
Elle permet de programmer la partie multimédia (graphismes, son et entrées au clavier, à la souris ou au joystick), sans se heurter aux difficultés des langages de bas niveaux comme le C et ses dérivés. Cela se fonde sur la supposition que la partie multimédia, souvent la plus contraignante à programmer dans un tel jeu, est suffisamment indépendante de la logique même du jeu pour qu'on puisse utiliser un langage de haut niveau (en l'occurrence le Python) pour la structure du jeu. (citation Wikipédia)

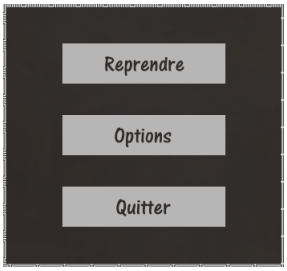
2 Fonctionnalités implémentées

2.1 Description des fonctionnalités

Dans le jeu que nous avons créé, nous pouvons tirer sur les ennemis, dans certains niveaux les ennemis peuvent aussi nous tirer dessus. Pour nous tirer dessus, ils utilisent un schéma de tir que nous avons au préalable créé. Nous avons aussi créé un menu option pour pouvoir couper les sons ou la musique du jeu mais également les deux.



Nous avons aussi implémenté un menu pause. Depuis ce menu nous pouvons acceder au menu option cité plus haut, quitter la partie ou même recommencer le niveau.



2.2 Organisation du projet

Au début du projet, nous groupe était constitué de trois membres. Nous nous sommes retrouvé à deux très rapidement. Lorsque nous faisions un niveau, nous le préparions entièrement avant de passer au niveau suivant. Nous voulions vraiment finir correctement le niveau en cours pour ne pas avoir à revenir dessus.

3 Elements techniques

3.1 Librairie tiere : Json

Nous avons utilisé une librairie qui n'est pas implémenté dans Python. Cette librairie s'appelle Json. Nous avons utilisé cette librairie pour créer les niveaux. Le fichier Json nous permet de créer nos niveaux dans un fichier différent et de faire appel au fichier Json voulu (en fonction du niveau souhaité) à l'aide d'une fonction que nous avons créé et appelé "loadLevel" qui appel le fichier Json souhaité et retourne le fichier ouvert voulu. Grâce au fichier Json on peut choisir des sprites différents pour chaque niveau. On peut choisir de changer la musique du niveau ou même l'apparence des ennemis... Nous pouvons changer toute l'apparence du niveau grâce au fichier Json.

3.2 Animation des images

Pour permettre l'animation des ennemies, du personnage principal ou des projectiles, nous avons créé une fonction nommée "Set Area" dans le fichier Player.py". Dans cette fonction, on prend l'image voulue, qui est composée de plusieurs sprites sur la même image. Au début de la fonction, on créé une zone dans l'image (pygame.Rect), qui ne prend en compte qu'un seul sprite. Par la suite, on fait bouger cette zone pour permettre l'affiche de l'image suivante. Pour que cette zone ce déplace, il faut que le temps (le délai choisi)

entre chaque image soit dépassé. Le temps passé depuis la dernière image est trouvable via l'instruction `pygame.time.get_ticks()`. Il faut que le résultat soit inférieur au temps, en milliseconde, qui est passé depuis l'initialisation. Si c'est inférieur alors l'image se déplace, l'animation se créer et le temps depuis la dernière image revient à zéro. Si on arrive au bout de l'image, la fonction reprend la première image au début. Donc grâce à cette fonction, l'image est animé, cela donne un aspect plus vivant au jeu.

3.3 Gestion des mouvements

Dans notre jeu nous avons créé plusieurs mouvements. Tout d'abord, il y a le mouvement horizontal du joueur. Pour que le joueur puisse se déplacer de gauche à droite ou de droite à gauche, nous avons créé une fonction qui vérifie sur quel bouton le joueur appuie. Si le joueur appuie sur la flèche de droite, soit `K_RIGHT` en language Pygame, le joueur se deplace à droite à l'aide de la commande image.setPosition(x, 0). Cependant, la commande est légèrement différente si le joueur appuie sur la flèche gauche, soit `K_LEFT` en language Pygame. En effet, la commande devient image.setPosition(-x, 0). La fonction qui permet la réalisation de ce mouvement est le fichier "Player.py".

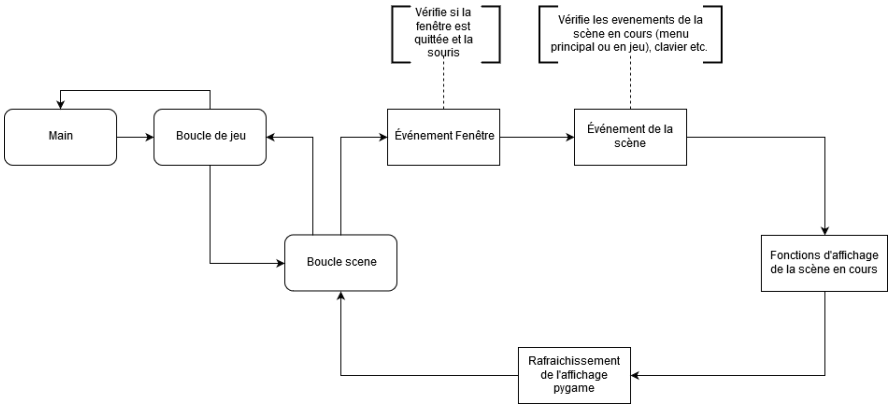
Le mouvement horizontal du joueur n'est pas le seul mouvement que l'on a créé dans le jeu. Effectivement, les ennemis aussi on leurs propres mouvements. Il y a tout d'abord les ennemis "classique" qui descendent tout droit. Ils bougent de façon verticale et disparaissent lorsqu'ils sont touchés par un projectile émis par le joueur. Nous avons une fonction qui les fait apparaître un peu au dessus de la fenêtre de jeu. Nous avons aussi une fonction qui vérifie à chaque instant leur position. C'est cette fonction qui nous permet de les faire disparaitres lorsque les ennemis sortent de la fenêtre, donc lorsqu'aucun projectile ne les a touchés.

Il y a un autre comportement que nous avons créé pour les ennemis. Nous avons créé une liste de mouvements spéciaux pour les ennemis qui apparaissent plus rarement. Nous avons plusieurs possibilités de mouvements pour ce type d'ennemis. Nous avons tout d'abord créé les mouvements un par un à l'aide d'une fonction. Les fonctions que nous avons créées sont toutes différentes. Les ennemis peuvent se déplacer en biais, de façon verticale ou horitale, ou même tirer. Toutes les fonctions nommées "mouvement" ce trouvent dans le fichier "Mob.py".

4 Architecture du projet

4.1 Diagramme des modules et classes

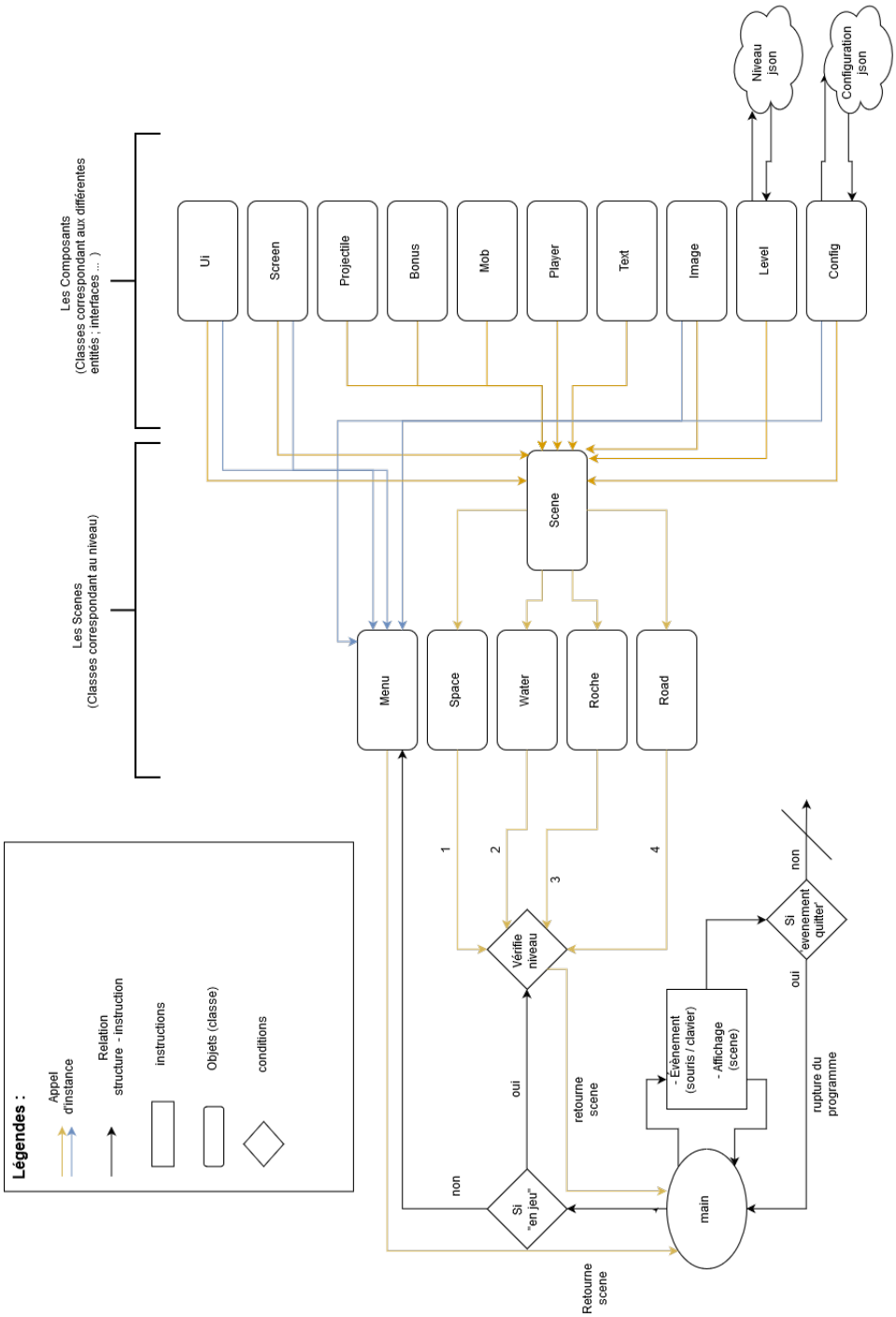
L'organigramme général du programme :



Dans cet organigramme représentant la première 'couche' du déroulement du jeu, nous pouvons voir ce qu'il se passe dans la boucle génerée par le fichier principal de notre script, le fichier 'main.py'. Propre à la base de pygame, le fichier principal lance l'ouverture de la fenêtre et met en place la boucle de jeu. Notre boucle dans le fichier main fait appel à un objet que l'on appellera 'scene', l'objet représentant toute l'organisation de la fenêtre en fonction de l'étape à laquelle le script est (Le menu du jeu, le niveau 1, etc).

Au sein même de la scene en cours, on distingue une boucle pour chaque scene possible (menu ou jeu). Elle regroupe les événements de la fenêtre, c'est à dire une vérification si le joueur clique sur la croix de fermeture de la fenêtre ou bien si il clique à l'interieur de la fenêtre, ce qui signifiera qu'il souhaite activer l'un des boutons du menu par exemple.
Elle regroupe également les événements liés aux contenus de la scène en cours (principalement en partie), autrement dit les evenements au clavier pour le joueur, les déplacements des entités (ennemis, tirs...)

l'organigramme détaillé lors du processus :



Voici l'organigramme détaillé de l'organisation des modules et des classes. Il y a en tout quatre modules (deux représentés) dans notre programme :

- le module 'Ressources' comprennant toutes les éléments dont nous avons besoin pour le jeu, il se divise explicitement en trois sous-modules, 'Songs' (les musiques et sons) ; 'Sprites' (les images) et 'Fonts' (les polices d'écritures).
- le module 'Scenes' comprennant toutes les classes représentant les scenes du jeu (le menu, les différents niveaux)
- le module 'Componants' regroupant toutes les classes représentant les composants de notre jeu (le joueur, les ennemis, l'interface utilisateur, les images, les textes, les bonus...)
- le module 'Datas' qui regroupe les fichiers json servant aux stockages des données des noveaux et des configurations.

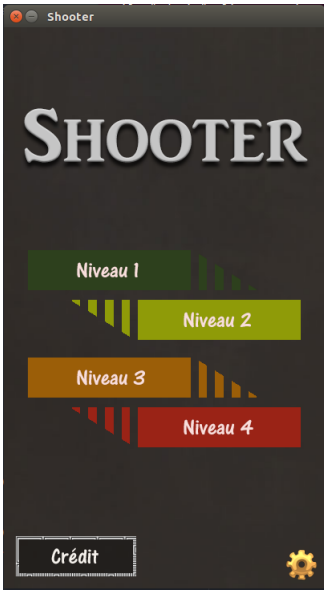
Notre programme est fait pour éviter les importation à répétition, chaques éléments des modules, importe que ce dont-il a besoin. Le fichier principal 'main.py' importe les différentes scènes, les scènes à leur tour importe les composants dont-ils ont besoin. Les classes Level et Config récuperent leurs fichiers json respectifs qui stockent les données des niveaux (le temps, le nombre d'ennemis, etc) et les données liées aux options (sons coupés ou non, musique coupées ou non, etc).

La boucle d'evenements reliée au fichier main sur l'organigramme représente le déroulement de la première couche vu plus haut, nous pouvons maintenant voir comment le script instancie la bonne classe (ici la scène). Le script vérifie si l'état est 'en jeu' (si le joueur a cliqué au préalabke sur un niveau), si c'est le cas il récupère le numéro du niveau et appelle la scène en question, si ce n'est pas le cas il appelle la scène du menu.

5 Expérimentations et usages

5.1 Utilisation du jeu

Lorsqu'on lance le fichier "main", une fenêtre se lance. Sur cette scène, on peut faire différentes choses, choisir le niveau (de 1 à 4), régler le son et la musique du jeu ou voir le crédit. Si l'on clique sur le niveau 1, on voit apparaître le niveau 1 du jeu. Un niveau relativement simple. Le niveau de l'espace. Les ennemis sont des météorites et nous sommes un vaisseau spacial. Pour le niveau 2, il s'agit du niveau de la mer, notre personnage est un bateau et nous devons détruire un bateau qui ressemble à des Drakkar Viking. Le thème du niveau 3 est les grottes. Les ennemis sont maintenant des rochers et notre vaisseau ressemble énormément au vaisseau de Star Wars 1. Ainsi pour le niveau 4 qui est le niveau le plus dur de notre jeu, nous avons choisi une simple route. Les ennemis sont des voitures qui roulent sur la route. Notre personnage est un tank, pour justifier les tirs de missiles.



5.2 Mesures de performance

Voici les mesures prises sur un système d'exploitation Windows lorsque notre jeu est sur le menu principal :

Nom	8% Processeur	58% Mémoire	0% Disque	0% Réseau	10% Processe...
Hôte de service : Service de stra...	0%	22,3 Mo	0 Mo/s	0 Mbits/s	0%
Hôte de service : lanceur de pro...	0%	20,8 Mo	0 Mo/s	0 Mbits/s	0%
Console Emulator (x64) (5)	0%	17,8 Mo	0 Mo/s	0 Mbits/s	0%
Indexeur Microsoft Windows Se...	0%	17,7 Mo	0 Mo/s	0 Mbits/s	0%
Python	1,5%	13,4 Mo	0 Mo/s	0 Mbits/s	0%
Shooter					

Enfin, voici les mesures prises lors d'un partie :

Python	10,5%	69,0 Mo	0,3 Mo/s	0 Mbits/s	0%
Shooter					

Comme nous pouvons le voir, le jeu ne prend quasiment aucunes ressources et la mémoire reste constante, ce qui signifie qu'il n'y a pas de fuites

de mémoires.

6 Conclusion

6.1 Mode d'emploi

Pour utiliser le jeu, il faut tout d'abord utiliser la commande ./main dans le dossier manic-shooter-bnt/Jeu/. Sélectionner un niveau à l'aide la souris. Pour déplacer votre joueur (seulement de gauche à droite) il faut utiliser les flèches. La flèche gauche pour aller à gauche et la droite pour aller à droite. Pour tirer, et détruire les ennemis, utilisez la touche espace.

6.2 Récapitulatif des fonctionnalités principales

Dans notre jeu, il y a plusieurs fonctionnalités principales. Les fichiers Json qui permettent de lire les informations des différents niveaux du jeu est une fonctionnalité très importante de notre jeu. Notre méthode pour les mouvements des entités est aussi importantes. L'animation de ces mêmes entités est aussi un élément spécifique. Voici donc les principales fonctionnalités de notre jeu.

6.3 Propositions d'améliorations

Pour amélioré notre jeu, pour qu'il soit vraiment bien pensé et conçu, il faudrait sûrement modifié les graphismes des niveaux ainsi que reprendre totalement et sérieusement la disposition et le comportement des ennemis (qui est ici quasi inexistant), il suffirait pour cela que de modifications des fichiers json liées aux niveaux. Du Côté du code, l'optimisation (l'architecture) est plutôt bien réalisé, mais il manque quelque optimisation telle que :

- Regrouper certaines méthodes communes à un bon nombre d'objet, par des classes abstraites ou des interfaces (si cela est possible en python).
- Revoir le système de comportement des entités ("patterns" dans nos fichiers json et les methodes pour interpreter ces informations) qui est plutôt contraignant à l'heure actuelle.
- Une possible suppression des doublons d'importation pygame.

Nous aurions bien entendu voulons un meilleur projet final (comme tout le monde), des tirs plus interessant, des hitboxes (zones de collision) plus précises, etc. Les idées ne sont pas en reste mais notre temps l'est malheureusement.