Rapport Conception Logiciel
Manic Shooter

Moriniere Robin 21606393
Semih Adanur 21706678

08/01/2018 - 01/04/2018

Figure 1: Vaisseau.jpg

Contents

1 Introduction

1.1 Rappel du sujet

Énoncé :

Le but de ce projet est le développement d'un Manic Shooter. Ce jeu d'arcade consiste à piloter un vaisseau et survivre le plus longtemps possible en affrontant des vagues successives d'adversaires. La première phase de ce projet consiste à développer un prototype du jeu en insistant sur l'aspect dynamique du jeu (défilement de l'écran, des projectiles, etc.). Pour cela, il est possible d'utiliser la bibliothèque Pygame. La seconde partie du projet consistera à améliorer l'expérience de jeu, en intégrant des bonus et des adversaires présentant différents comportements ainsi qu'un générateur de niveaux avec une difficulté s'adaptant au joueur.

1.2 Exemple de Manic Shooter

Le Manic Shooter communément appelé shoot 'em up est un jeu de tir à la façon d'un Space Invaders où l'on va devoir survivre à plusieurs vagues d'ennemis en détruisant leur vaisseau tout en évitant les tirs ennemis pour ne pas perdre de vie. Ce type de jeu est sortie à la fin des années 70 principalement dans les salles d'arcade puis c'est démocratisé à la fin des années 80 pour atteindre un record de popularité. Depuis lors le jeu a été décliné sous toutes ses formes et sur toutes les plate formes comme les consoles, les ordinateurs ou encore plus récemment les téléphones portables. Voici quelques exemples de Manic shooter que nous avons pu trouver :

Figure 2: AstroMenace.jpg

Figure 3: Chicken-Invaders.jpg

1.3 Aboutissement du projet

Le projet avait pour aboutissement de créer jeu de tir, ou l'on pilote un petit vaisseau et l'on doit survivre à de nombreuses vagues d'ennemis en les détruisant pour pouvoir avancer. Pour commencer ce projet nous avions que notre imagination et quelque exemple de jeu similaire sur internet pour nous inspirer. En plus de de devoir recréer ce jeu nous devions ajouter des fonctionnalitées supplémentaires comme des bonus ou malus ou alors augmenter le nombres de vagues et d'ennemi pour rendre le jeu un peux plus complet.

1.4 Présentation du Gantt

Dans l'annex du rapport vous pourrez trouver un Gantt MindView que nous avons construit au début du projet pour pouvoir guider notre projet tout en commençant d'abord par lister toutes les choses qu'allait contenir notre projet. Le Gantt se divise en 4 parties distinctes, une première partie Organisation qui explique quel schéma d'organisation nous avons choisis avec nos premières idées sur le sujet avant de nous lancer dans le code. Ensuite une partie Développement Jeu qui englobe tout ce que nous devions créer pour obtenir un jeu basique mais fonctionnel. Après vient une partie Menu pour tout ce qui concerne le menu donc les paramètres, le score et les boutons Jouer / Quitter. Et pour finir une partie Fonctionnalités Supplémentaires où nous avons listé toutes les choses que nous étions susceptibles d'ajouter au jeu comme par exemple les bonus ou malus ou alors l'augmentation des niveaux pour obtenir des parties plus longues et donc obtenir un jeu plus complet.

2 Conception

2.1 Description du projet

2.1.1 Cahier des charges

L'objectif de ce projet était de choisir parmi une liste de jeux déjà existant celui qui nous intéressés le plus pour ensuite le concevoir de A à Z. En commençant par créer une fenêtre graphique pour accueillir l'interface du jeu puis créer tous les éléments liés au jeu, en l'occurrence nous avons choisis de concevoir un Manic Shooter donc nous avons dû diviser notre code en de nombreuses classes pour tous ce qui est de la gestion du vaisseau ou alors les gestions des ennemis avec les tirs.

Pour ce projet nous avions un temps imparti avec une date limite début avril pour finir le jeu mais aussi préparer un rapport et une soutenance pour la présentation de notre travail.

2.1.2 Diagramme des cas d'utilisation

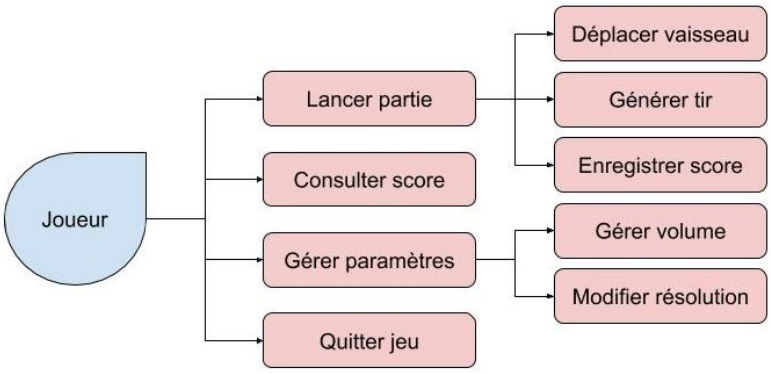


Figure 4: Diagramme.JPG

2.1.3 Cas d'utilisation détaillé

Le jeu sera un shoot em up dans l'espace, le joueur contrôlera un vaisseau qui pourra se déplacer et tirer sur des ennemis. Ces ennemis apparaîtront par vague et le joueur devra détruire tout les ennemis avant de passer à la vague suivante. Les limites de déplacement du jeu sont les bords de l'écran. Le joueur possédera 3 vies de départ, et en perdra une à chaque fois qu'il se fera toucher par un tir ennemi. Une fois la partie terminé, le joueur pourra enregistrer son score si celui-ci à atteint le top.

Le jeu affichera un menu à son exécution qui laissera au joueur le choix de jouer, regarder le tableau des scores, modifier les paramètres du jeu, ou quitter le jeu.

2.1.4 Maquette

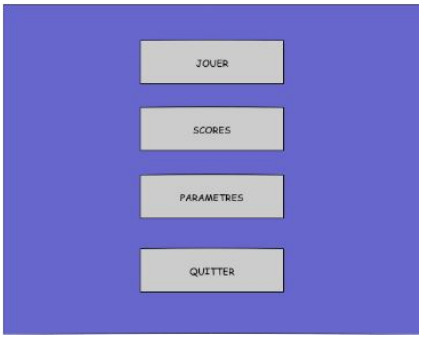


Figure 5: Menu.JPG

Figure 6: Scores.JPG

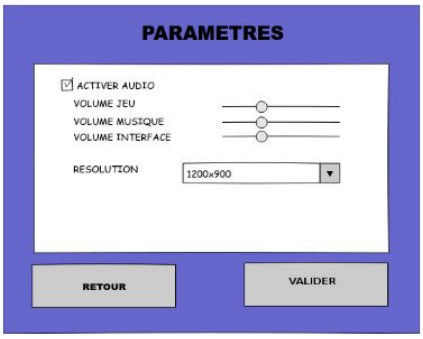


Figure 7: Paramètres.JPG

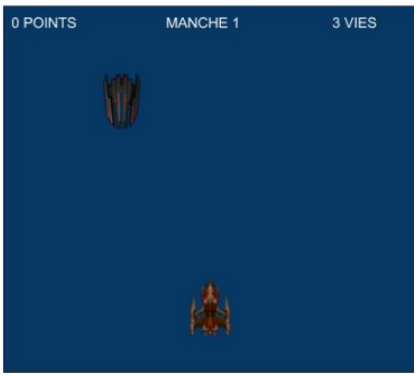


Figure 8: Jeu.JPG

3 Fonctionnalité supplémentaires

3.1 Description des fonctionnalités supplémentaires

Pour ce qui est des fonctionnalités supplémentaires nous avons ajouté deux boutons au menu de base en plus des boutons jouer et quitter, qui sont respectivement le bouton paramètre et le bouton score.

Pour commencer le bouton paramètre va permettre au joueur de régler le son du jeu à savoir l'activer ou le désactiver, mais aussi des réglages plus fins avec 3 barres de volume pour le volume de la musique, le volume du jeu et le volume de l'interface. Il y a aussi un menu déroulant où l'on peut régler la résolution de la fenêtre de jeu avec 3 choix possible :

- 1200 par 900

- 960 par 720
- 780 par 585

En ce qui concerne le bouton scores il permet au joueur d'aller voir les scores qu'il a effectués aux parties précédentes grâce à une fenêtre pop-up qui surgit soit la fin de la partie lorsque l'on a survécu à toutes les vagues d'ennemis ou alors lorsque l'on a succombé des tirs ennemis. Dans cette fenêtre pop-up on peut marquer le pseudo du joueur qui vient de réaliser la partie.

4 Développement

4.1 Architecture du projet

4.1.1 Arborescence du projet

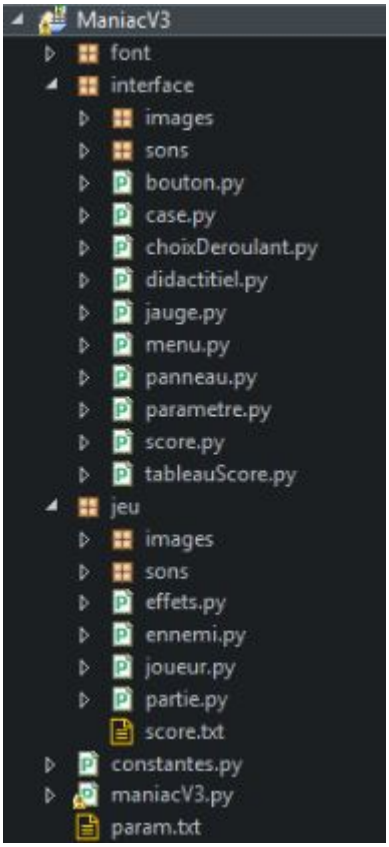


Figure 9: Arborescence.JPG

4.1.2 Description des fichiers et classes

Dans ce chapitre, nous allons faire un descriptif de chaque fichier présent dans le projet.
Fichiers "constantes.py"
Ce fichier contient les constantes du programme. Il indique au programme :

- Les liens des images, des sons...
- Les données des paramètres (récupéré depuis le fichier "param.txt"
- La composition des vagues dans le jeu.

Fichier "maniacV3.py
C'est le fichier principal du programme, celui qu'on exécute pour lancer le jeu. Il contient la boucle principal qui permet la navigation entre les différentes partie du programme.
Fichier "param.txt
Ce fichier contient les paramètres enregistrés du jeu tel que la dimension de la fenêtre, le pourcentages des différents volumes du programme et l'activation ou non de l'audio.
Package "font"
Ce package contient la police utilisé dans le programme.
Package "interface"
Ce package contient tous les fichiers liées à l'interface du jeu (menu, boutons, jauges...). Il possède aussi un dossier "son" et un dossier "image" contenant les sons et les images liées à l'interface.
Fichier "bouton.py"
Ce fichier contient la classe Bouton, cette classe va créer un objet Bouton qui pourra être afficher dans le programme comme ceci :

Figure 10: Continuer.JPG

Fichier "case.py"
Ce fichier contient la classe Case, elle permet la création d'une case avec son libellé que l'on peut cocher ou décocher. Elle est utilisé dans la partie Paramètre de l'interface :

Figure 11: Case.JPG

Fichier "choixDeroulant.py"
Contient la classe ChoixDeroulant, cette classe permet la création d'une boîte combinée. Elle est utilisé dans le programme pour le choix de la résolution.

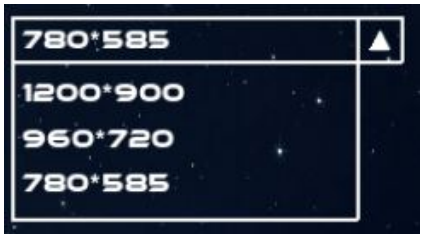


Figure 12: Choix.JPG

Fichier "didactitiel.py"
Contient la classe Didacticiel, elle permet l'affichage de la page de didacticiel avant le début du jeu
Fichier "jauge.py"
Contient la classe Jauge, elle permet l'affichage d'une jauge avec son taux de remplissage :

Figure 13: Jauge.JPG

Fichier "menu.py"
Ce fichier contient la classe Menu, elle dispose le menu principal qui est affiché au lancement du jeu :

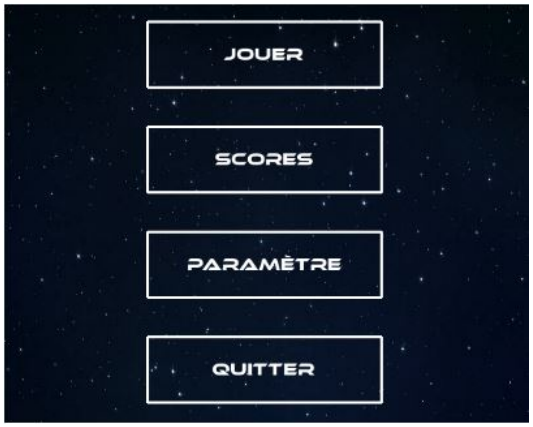


Figure 14: Menu.JPG

Fichier "panneau.py"
Contient la classe Panneau, elle permet de créer un contour avec un titre, utilisé dans la vue des scores et des paramètres :

Figure 15: Panneau.JPG

Fichier "parametre.py"
Contient la classe Paramètre, elle permet de créer la vue des paramètres :

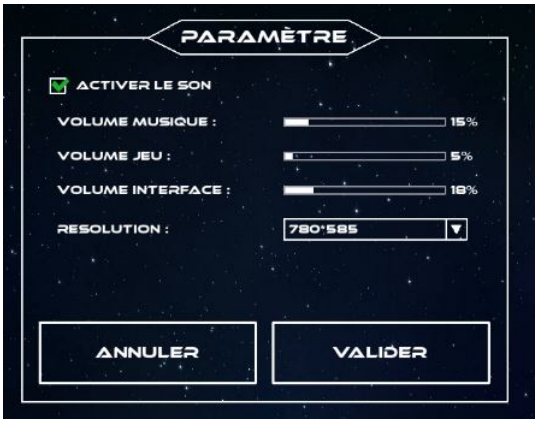


Figure 16: Parametre.JPG

Fichier "effets.py"
Ce fichiers contient deux classes : la classe Impact et la classe Explosion. Ces deux classes fonctionnent de la même façon, la classe Impact sert à animer une petite explosion à l'impact des tirs, et la classe Explosion anime l'explosion des vaisseaux. Exemple lorsque le joueur reçoit un tir :

Figure 17: Effets.JPG

Fichier "score.py" et fichier "tableauScore.py"
Ces deux fichiers contiennent respectivement la classe Score et TableauScore, la première permet de créer la vue des scores, et la seconde exclusivement le tableau des scores en récupérant les données à partir du fichier “score.txt” présent dans le package “jeu”.

Figure 18: TabScore.JPG

Package "jeu"
Ce package contient tous les fichiers liées au déroulement d’une partie. Il possède aussi un dossier “son” et un dossier “image” contenant les sons et les images liées au jeu.
Fichier "ennemi.py"
Ce fichier contient la classe Ennemi et TirEnnemi. La classe Ennemi s’occupe de créer les sprites des ennemis. La classe TirEnnemi créer les tirs des ennemis.
Fichier "joueur.py"
Ce fichier contient la classe Joueur et Tir. La classe Joueur s’occupe de créer le sprite du joueur. La classe Tir créer les tirs du joueurs..
Fichier "partie.py"
Contient la classe Partie. Cette classe va gérer la partie qui a été lancé par l’utilisateur.
Fichier "score.txt"
Ce fichier texte contient les scores enregistrés dans le jeu

4.2 Éléments techniques

4.2.1 Bibliothèque utilisés

Pour le programme, nous allons utiliser la bibliothèque **Pygame** pour la création du jeu, **sys** pour déclencher la fermeture du programme, et **random** pour la génération de nombres aléatoires.

5 Problème et solutions apportées

5.1 Expérimentations et usages

En lançant le programme, l'utilisateur arrive au menu principal du jeu :

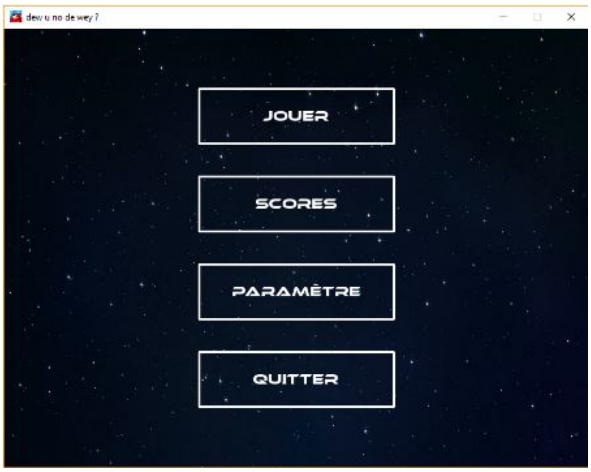


Figure 19: Jeu.JPG

En cliquant sur le bouton des scores, le programme va afficher le tableau des scores enregistrés par l'utilisateur :

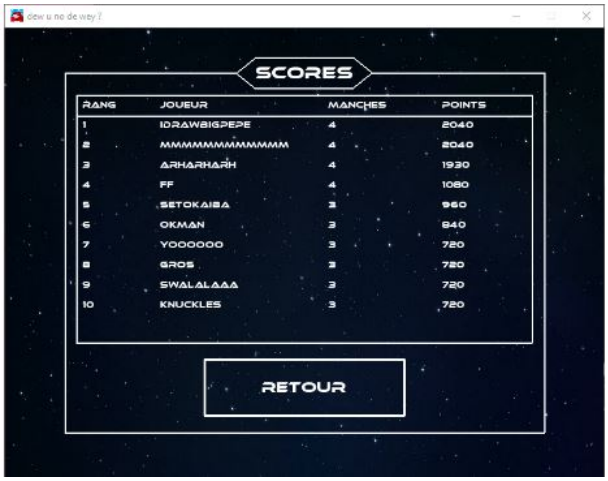


Figure 20: Jeu.JPG

Il suffit à l'utilisateur de cliquer sur le bouton Retour pour revenir au menu principal.

L'utilisateur peut accéder aux paramètres en cliquant sur le bouton Paramètre du menu principal, il peut ensuite modifier les paramètres et valider, ou annuler pour garder les paramètres actuels :

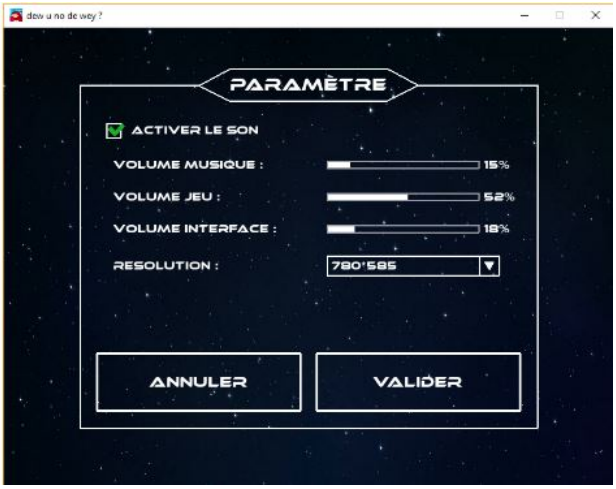


Figure 21: Jeu.JPG

Lorsqu'il clique sur le bouton Jouer, une page du didacticiel va s'afficher l'utilisateur devra cliquer sur Continuer pour démarrer la partie :

Figure 22: Jeu.JPG

Une fois le jeu lancer, l'utilisateur pourra déplacer son vaisseau avec les flèches directionnelles et tirer avec la touche "F" :

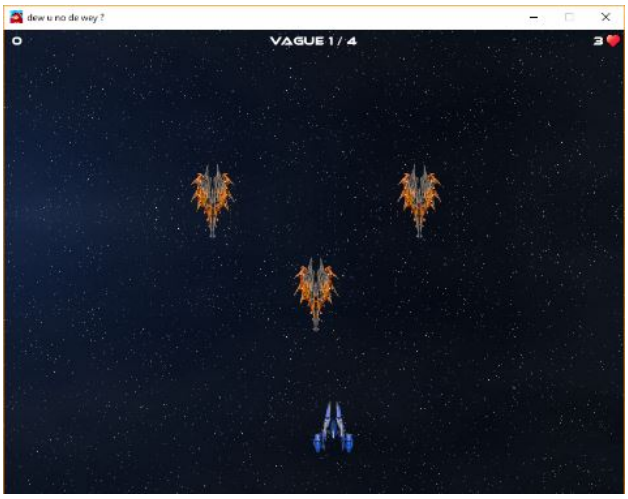


Figure 23: Jeu.JPG

S'il n'a plus de vie ou qu'il a passé toute les vagues, le jeu se termine et demande à l'utilisateur d'entrer son pseudo pour enregistrer son score (seulement s'il a atteint le top du tableau des scores). Pour terminer la partie, l'utilisateur devra cliquer sur le bouton Terminer et ainsi revenir au menu principal :

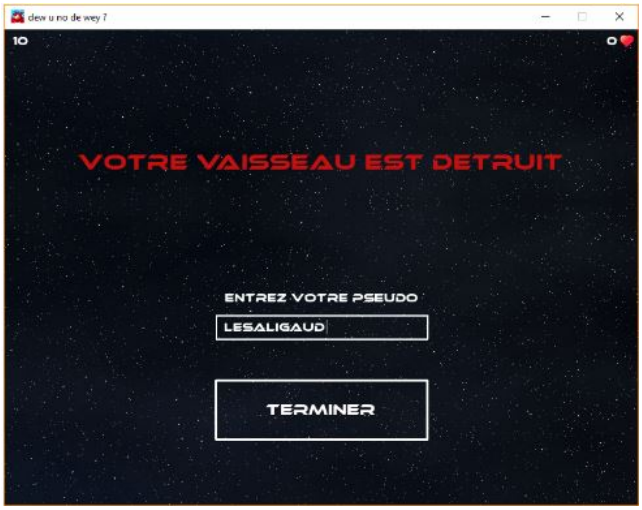


Figure 24: Jeu.JPG

5.2 Problèmes / Solutions

Au début du projet, nous avions défini que pour que le joueur tire avec son vaisseau il devait appuyer sur la touche Espace, car dans la plupart des jeux de ce type c'est à cette touche que l'on attribue les tirs. Cependant lors des phases de test, nous avons constaté que lorsque le joueur tirait il ne pouvait plus se déplacer en diagonale, et lorsqu'il se déplaçait en diagonale il ne pouvait plus tirer.

Ce problème vient de la fonction qu'on utilise pour récupérer l'état des touches pressés, fonction qui est incluse dans la bibliothèque pygame. Nous avons constaté que pour certaines touches la fonction n'arrivait pas à capter simultanément 3 touches pressés ou plus, dans notre cas il n'arrivait pas à capter à la fois 2 flèches directionnels et la touche Espace.

Nous avons donc essayé plusieurs touches pouvant être capté par la fonction tout en maintenant 2 flèches directionnels enfoncées, et avons décidé que la touche F remplacerait la touche Espace pour lancer des tirs.

6 Conclusion

6.1 Récapitulatif des fonctionnalités

Le programme actuel est stable, il permet de jouer, d'enregistrer son score, de visionner le score des meilleurs joueurs, de changer les paramètres sonores ainsi que la résolution de la fenêtre.

6.2 Ce que le projet nous à apporté

Ce projet nous a beaucoup appris sur le développement d'un programme en équipe, ainsi que sur la manipulation des classes et l'externalisation du code.

6.3 Ce que l'on à pas pu réaliser

Voici des idées trouvées qui n'ont pas eu le temps d'être développé :

- Faire en sorte que les points de vie des ennemis soient différents
- Créer un système d'apparition de bonus/malus sur la zone de jeu que le joueur pourrait attraper.
- Ajouter d'autres trajectoires de déplacement des ennemis (autre qu'en diagonale)
- Pouvoir personnaliser le vaisseau du joueur via la page des paramètres
- Créer des tirs d'ennemis différents selon leurs types.

7 Annexe

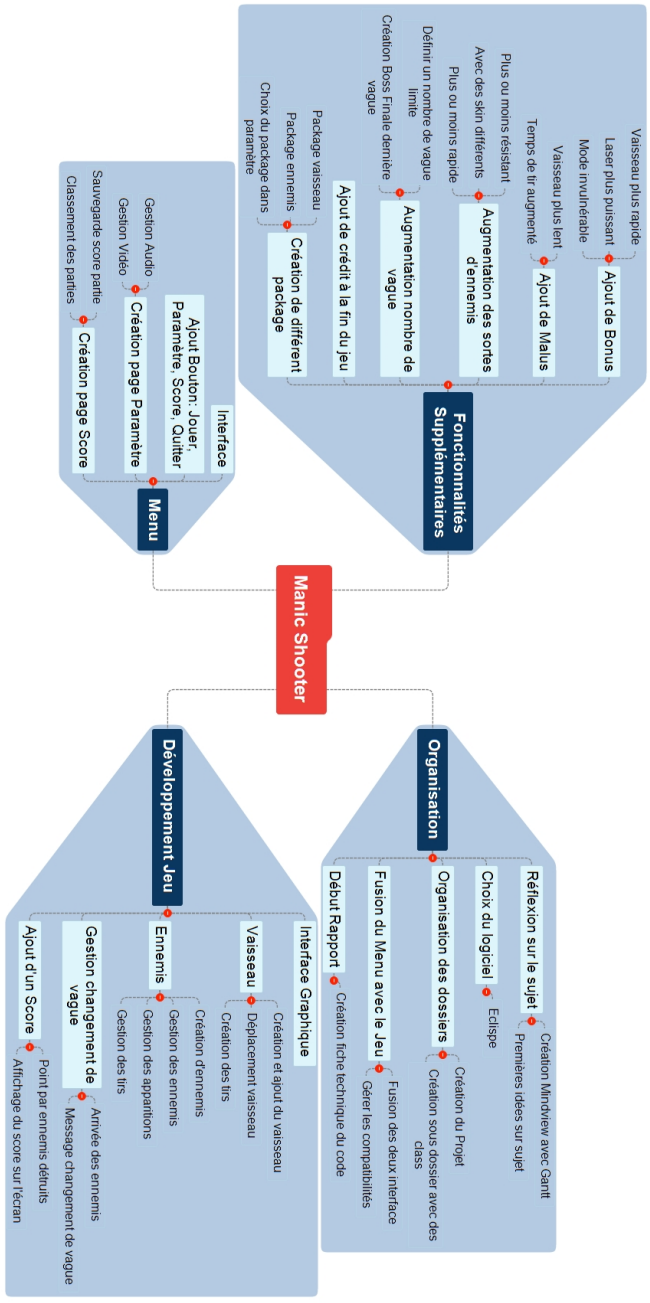


Figure 25: MindView1.jpg

Dossier Technique Maniac

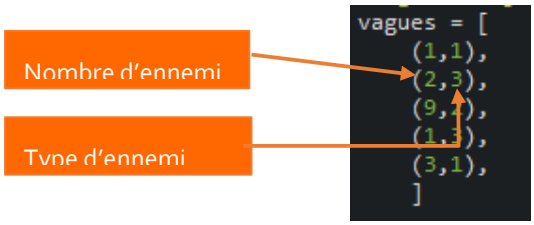
Table des matières

1 Racine du programme

1.1 Fichier « constantes.py »

Ce fichier contient les constantes du programme. Il indique au programme :

- Les liens des images, des sons…
- La longueur et la largeur de la fenêtre
- La composition des vagues dans le jeu avec la liste vagues, cette liste contient des tuiles possédant 2 valeurs (nombre d'ennemi, type d'ennemi), chaque tuile correspond à une vague (1ere tuile correspond à la vague 1, 2eme tuile à la vague 2, etc.)



1.2 Fichier « maniacV3.py »

C'est le fichier principal du programme, celui qu'on exécute pour lancer le jeu.

Ces données sont :

- Une variable fenetre, c'est la fenêtre du jeu
- Une variable continuer, c'est un booléen qui indique à la boucle principale de s'exécuter tant qu'il vaut True
- Une variable menuPrincipal, c'est un objet de la classe **MenuPrincipal** (classe située dans le fichier menu.py)
- Une variable viewScore, c'est un objet de la classe **Score** (située dans le fichier score.py)

```
while continuer:
    menuPrincipal.run()
```

Le fichier contient la boucle principale du jeu, lors de son lancement on affiche le menuPrincipal en utilisant sa fonction **run()** (voir classe **MenuPrincipal**).

L'objet menuPrincipal contient une variable start_selected qui est un booléen qui prend True comme valeur lorsque l'utilisateur clique sur le bouton « Jouer » du menu. Dans cette condition, lorsqu'il prend la valeur True, le programme créer un objet partie de la classe **Partie**, en donnant comme paramètre la fenêtre du jeu (cela permettra aux fonctions de l'objet de manipuler la fenêtre). On appelle ensuite la fonction **run()** de partie pour lancer la boucle du jeu. Puis on remet start_selected a False.

```
if menuPrincipal.start_selected: #Si on c
    partie = Partie(fenetre)
    partie.run()
    menuPrincipal.start_selected = False
```

Comme pour start_selected, l'objet menuPrincipal possède une variable score_selected. Lorsque l'utilisateur clique sur le bouton « Scores » alors on entre dans la condition. On fait repasser score_selected à False, puis on affiche la page des scores en mettant à jour au préalable les données du tableau des scores.

```
if menuPrincipal.score_selected:
    menuPrincipal.score_selected = False
    viewScore.tableau.majDonneesTableau()
    viewScore.run()
```

2 Package « Jeu »

2.1 Fichier « effets.py »

Ce fichier contient les classes qui vont animés les effets du jeu (explosions, impact). Dès le chargement du fichier, le programme va initialiser le mixer de Pygame et récupérer les sons qui seront jouées pour les effet.

```
5  pygame.mixer.pre_init(44100,-16,2,2048)
6  pygame.mixer.init(44100, -16, 2, 2048)
7  sonImpact = pygame.mixer.Sound(son_impact)
8  sonExplosion = pygame.mixer.Sound(son_explosion)
```

2.1.1 Classe Impact

Cette classe génère une animation lorsqu'un tir touche sa cible.

2.1.1.1 Fonction __init__

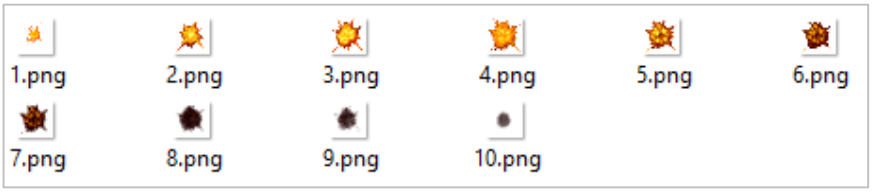
Cette fonction est le constructeur de la classe, elle prend en paramètre pos qui est la position où doit avoir lieu l'animation. D'abord on va lancer sonImpact qui est le son de l'impact entre un tir et un vaisseau.

```
12    def __init__(self,pos):
13        pygame.sprite.Sprite.__init__(self)
14        sonImpact.play()
```

Ensuite on définit la variable self.imgNum qui est le numéro de l'image à afficher, puis self.image qui va récupérer une image ayant comme nom le numéro que contient la variable self.imgNum.

```
15        self.imgNum = 1
16        self.image = pygame.image.load(image_impact.format(self.imgNum)).convert_alpha()
```

Explication : dans le dossier « images/impact » du package jeu, nous pouvons voir qu'il y a plusieurs images présentes avec comme nom un numéro.



Avec la commande image_impact.format(self.imgNum) qui est l'équivalent de 'jeu/images/impact/{0}.png'.format(self.imgNum) (voir la variable image_impact du fichier « constantes.py »), on remplace grâce à la fonction **format** le « **{0}** » par une chaine de caractère qui est donné dans le paramètre de **format** (ici self.imgNum).

Comme on a défini self.imgNum étant égale à 1, notre chaine qui est 'jeu/images/impact/{0}.png' va devenir grâce à **format** 'jeu/images/impact/1.png' On a donc là un lien vers l'image « 1.png » du dossier impact. Donc self.image va contenir l'image « 1.png ». Nous allons montrer dans la fonction **update()** l'utilité de cette méthode.

Dans la ligne suivante nous déclarons la variable self.rect qui récupère l'objet **Rect** de l'image présent dans self.image (voir la documentation pygame sur les **Rect**). Puis nous déclarons self.rect.center qui définit la position de l'image en prenant son centre pour que celui-ci prenne la position pos donnée en paramètre.

```
17        self.rect = self.image.get_rect()
18        self.rect.center = pos
```

2.1.1.2 Fonction Update

Nous allons maintenant modifier la fonction **update**, cette fonction se lance automatiquement à chaque boucle du jeu sans qu'on ait besoin de l'appeler dans le programme, tous les objets **Sprite** de pygame possèdent cette fonction.

```
20    def update(self, *args):
21        pygame.sprite.Sprite.update(self, *args)
```

Cette fonction va d'abord implémenter de 1 la variable self.imgNum, puis va redéfinir la variable self.image en utilisant la même méthode vu au-dessus, c'est-à-dire :

```
22        self.imgNum += 1
23        self.image = pygame.image.load(image_impact.format(self.imgNum)).convert_alpha()
```

Comme self.imgNum a été implémenter de 1, nous chargerons l'image « 2 .png » après le passage de la 1ere boucle du jeu, puis « 3.png » après la boucle suivante, et ainsi de suite, cela permet récupérer toujours l'image suivante du dossier impact et de créer l'animation de l'impact dans le jeu.

A la ligne suivante nous avons la condition suivante :

```
24        if self.imgNum == 10 :
25            self.kill()
```

Si nous arrivons à la 10ème image (donc la dernière du dossier impact), nous utilisons la fonction **kill** pour que l'objet construit s'auto-détruise (voir doc pygame sprite -> kill()).

2.1.2 Classe Explosion

Cette classe fonctionne de la même façon que la classe **Impact** vu au-dessus, juste le lien du dossier contenant les images qui change.

2.2 Fichier « ennemi.py »

Ce fichier regroupe les classes liées aux ennemis du jeu.

2.2.1 Classe Ennemi

Cette classe génère un ennemi.

2.2.1.1 Fonction __itit__

Dans ce constructeur, nous demandons de déclarer en paramètre pos qui est une tuile (x,y) contenant la position de l'ennemi à sa création, et typeEnnemi qui est un numéro qui va correspondre à un certain type d'ennemi. Le constructeur créera une variable self.type qui copiera la donnée de typeEnnemi. Ensuite nous récupérons l'image de l'ennemi avec la même méthode vu dans la classe **Impact**, en mettant la variable self.type dans la fonction **format**. Nous récupérons son objet **Rect**, la position du centre de l'image via son **Rect**, et nous définissons la variable self.life qui correspondra aux vies restante de l'ennemi. Enfin nous définissons le délai de tir (valeur aléatoire entre -100 et 0).

```
def __init__(self,pos,typeEnnemi):
    pygame.sprite.Sprite.__init__(self)
    self.type = typeEnnemi
    self.image = pygame.image.load(image_ennemi.format(self.type)).convert_alpha()
    self.rect = self.image.get_rect()
    self.rect.center = pos
    self.life = 2
    self.delai_tir = random.randrange(-100,0)
```

2.2.1.2 Fonction update

Dans cette fonction nous allons seulement faire en sorte que le délai du tir s'incrémente une fois que l'ennemi est apparu entièrement sur l'écran. Si le haut de l'image à passer le 0 des ordonnées de la fenêtre, alors on incrémente de 1 le délai du tir.

```
def update(self):
    if self.rect.top > 0 :
        self.delai_tir += 1
```

2.2.2 Classe TirEnnemi

2.2.2.1 Fonction __init__

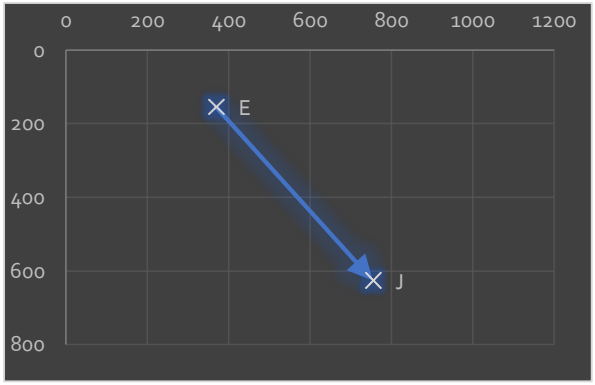
Ce constructeur demande en paramètre posEnnemi (position de l'ennemi), et posJoueur (position du joueur). Il va définir l'image, son **Rect**, et va position l'image au centre/bas de l'ennemi qui tire.

```
def __init__(self,posEnnemi,posJoueur):
    pygame.sprite.Sprite.__init__(self)
    self.image = pygame.image.load(image_tir_ennemi).convert()
    self.rect = self.image.get_rect()
    self.rect.x = posEnnemi.centerx
    self.rect.y = posEnnemi.bottom
```

Ensuite la variable self.vitesse est créée, elle va définir la vitesse de déplacement du tir.

```
self.vitesse = 6
```

On doit maintenant créer les variables qui indiqueront vers où et à quelle vitesse le tir doit se déplacer. Pour ça faisons un schéma de la fenêtre du jeu en affichant la largeur et la longueur (ici 1200x800) :



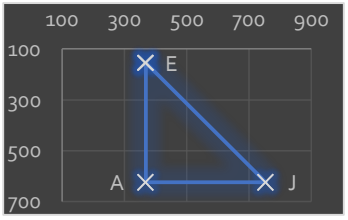
On place un point qui correspond à l'ennemi E et un autre qui correspond au joueur J. D'abord il faut calculer la trajectoire, pour cela il suffit de calculer le vecteur $\overrightarrow{EJ}$. Dans notre code cela va donner :

```
x = posJoueur.centerx-posEnnemi.centerx
y = posJoueur.centery-posEnnemi.centery
```

On calcule le x et y de $\overrightarrow{EJ}$ en prenant le centre des vaisseaux.

Maintenant qu'on a notre trajectoire, il faut régler la vitesse. Pour que la vitesse du tir soit la même quelques soit les positions des vaisseaux, il faut que la longueur de $\overrightarrow{EJ}$ soit fixe. Nous avons défini au-dessus self.vitesse qui définissait la vitesse de déplacement du tir, nous allons donc faire en sorte que la longueur de $\overrightarrow{EJ}$ soit égale à self.vitesse.

D'abord il faut calculer la longueur de $\overrightarrow{EJ}$ car elle nous est inconnu. Pour cela on peut utiliser le théorème de Pythagore, en ajoutant un point A on obtient alors un triangle rectangle.



Il suffit maintenant de calculer la longueur de l'hypoténuse.

Dans notre code cela va donner : (libraire math requis)

```
distE_J = math.hypot(x,y)
```

On met donc ce résultat dans une variable distE_J.

Maintenant qu'on a notre longueur, on la divise par self.vitesse, on

appellera le résultat d.

```
d = distE_J/self.vitesse
```

Et enfin on divise le vecteur $\overrightarrow{EJ}$ par d pour que la longueur du vecteur soit égale à self.vitesse. On met le coordonnées du vecteur dans les variables self.trajH et self.trajV qui seront appelés lors du déplacement du tir dans la fonction update.

```
self.trajH = x/d
self.trajV = y/d
```

2.2.2.2 Fonction update

Dans cette fonction comme dit au-dessus nous allons déplacer le tir selon la trajectoires calculés :

```
def update(self):
    self.rect = self.rect.move(self.trajH,self.trajV)
```

Puis nous supprimons le tir lorsque celui-ci quitte la fenêtre :

```
if (self.rect.top > longueur) or (self.rect.left > largeur) or (self.rect.right < 0) :
    self.kill()
```

2.2.3 Fonction placementEnnemi

Cette fonction ne fait pas parti des classes du fichier, elle va placer les ennemis d'une vague dans le jeu selon leurs nombres. La fonction définit ze qui est la zone ou les ennemis apparaissent, c'est une tuile qui possède comme valeur la largeur et la longueur/2-1200 (la longueur doit être négatif pour que l'ennemi apparaît au-dessus de la zone visible du jeu).

Par exemple si la largeur de la fenêtre est égale à 1200 et la longueur 800, ze sera égale à (1200,800/2-1200) donc (1200,-800).

Si le nombre d'ennemi est égale à 1, la fonction va placer une tuile dans la liste ayant comme valeur le centre de l'écran (600,-800).

Ainsi de suite on place les ennemis selon leurs nombres en ajoutant des tuiles (largeur/longueur) de la position initiale de l'ennemi.

2.3 Fichier « joueur.py »

2.3.1 Classse Joueur

Cette classe va créer le vaisseau du joueur.

2.3.1.1 Fonction __init__

D'abord le constructeur charge l'image, son **Rect** et définit les positions du joueur.

```
def __init__(self):
    pygame.sprite.Sprite.__init__(self)
    self.image = pygame.image.load(image_perso).convert_alpha()
    self.rect = self.image.get_rect()
    self.rect.x = largeur/2-35
    self.rect.y = longueur-120
```

Ensuite, on initialise les variables qui vont indiquer :

- Le délai du tir
- La vie du joueur
- Sa vitesse de déplacement
- Son score

```
self.delai_tir = 0
self.vie = 3
self.vitesse = 5
self.score = 0
```

Et on créer un groupe qui contiendra les **Sprite** de tir du joueur :

```
self.grp_tir = pygame.sprite.Group()
```

2.3.1.2 Fonction controleJoueur

Cette fonction va gérer les contrôles du joueur. D'abord on va gérer le déplacement, on initialise une variable v et h à 0, ces variables vont indiquer de combien de pixel le joueur devra se déplacer verticalement et horizontalement.

```
v = 0
h = 0
```

Ensuite on va récupérer dans k un tableau contenant l'état des touches pressés (True si pressé, False sinon).

```
k = pygame.key.get_pressed();
```

Puis selon la touche pressée, on modifie la valeur de v et h pour définir une direction :

```
if k[K_DOWN] and (self.rect.bottom < longueur):    #Si "flèche bas"
    v += self.vitesse
if k[K_UP] and (self.rect.top > 0):    #Si "flèche haut"
    v -= self.vitesse
if k[K_LEFT] and (self.rect.left > 0):    #Si "flèche gauche"
    h -= self.vitesse
if k[K_RIGHT] and (self.rect.right < largeur):    #Si "flèche droite"
    h += self.vitesse
```

Et on déplace de v et h pixels le joueur : `self.rect = self.rect.move(h,v)`

Ensuite on regarde si la touche de tir à été pressé (ici la touche « k ») ET si le délai de tir du joueur est supérieur ou égale à 30, si la condition est bonne on créer un objet de la classe **Tir** et on l'ajoute au groupe de tir du joueur, sinon on incrémente de 1 le délai du tir.

```
if k[K_f] and self.delai_tir >= 30 :
    tir = Tir(self.rect)
    self.grp_tir.add(tir)
    self.delai_tir = 0
else :
    self.delai_tir += 1
```

2.3.2 Classe Tir

```
class Tir(pygame.sprite.Sprite):
    def __init__(self,position):
        pygame.sprite.Sprite.__init__(self)
        sonTir.play()
        self.image = pygame.image.load(image_tir).convert()
        self.rect = self.image.get_rect()
        self.rect.x = position.centerx
        self.rect.y = position.top

    def update(self):
        self.rect = self.rect.move(0,-25)
        if (self.rect.top < 0) :
            self.kill()
```

Cette classe fonctionne de la même manière que la classe **TirEnnemi**, à la différence ici que le tir part du joueur et qu'il se dirige verticalement vers le haut.

2.4 Fichier « partie.py »

2.4.1 Classe Partie

Cette classe va gérer une partie lancée par l'utilisateur.

2.4.1.1 Fonction __init__

Ce constructeur demande en paramètre la fenêtre du jeu. On va mettre cette variable dans **self.fenetre**, nous nous serviront à présent de **self.fenetre** dans cette classe pour faire appel à la fenêtre.

```
#Constructeur
def __init__(self,f):

    #Données membres
    self.fenetre = f
```

On déclare **self.background** qui va charger l'image de fond du jeu, puis **self.moveBckgr** qui va être un compteur qui servira pour le défilement du fond (voir fonction **defilementBackground**).

```
self.background = pygame.image.load(image_fond).convert()
self.moveBckgr = 0
```

On déclare **self.joueur** qui va être un objet de la classe **Joueur**, puis **self.grp_joueur** qui va être un groupe contenant le sprite **self.joueur**.

```
self.joueur = Joueur()
self.grp_joueur = pygame.sprite.GroupSingle()
self.grp_joueur.add(self.joueur)
```

On créer aussi le groupe qui va contenir les sprites ennemis, **self.depEnnemi** contient la tuile qui va définir les déplacements ennemis, elle sera modifiée au fur et à mesure des déplacements dans la fonction **deplacerGroupeEnnemi**. On instancie aussi le groupe qui contiendra les sprites des tirs ennemis. De même pour les sprites des explosions et des impacts on aura un groupe qui les contiendront.

```
self.grp_ennemi = pygame.sprite.Group()
self.depEnnemi = (2,0)
self.grp_tir_ennemi = pygame.sprite.Group()
self.grp_explosion = pygame.sprite.Group()
```

La variable **self.vagueEnCours** indique à quel vague se situe le joueur, elle sera modifié au fur et à mesure dans la boucle du jeu, et **self.delai_vagueSuivant** est un compteur qui va indiquer combien de temps c'est écoulé depuis la fin de la dernière vague (explication dans la boucle du jeu).

```
self.vagueEnCours = 0
self.delai_vagueSuivant = 0
```

Pour finir, **self.police** récupère la police qui sera utilisé pour afficher du texte, et **self.musique** est la musique du jeu.

```
self.musique = pygame.mixer.Sound(son_jeu)
self.musique.set_volume(0.6)
self.musique.play()
```

2.4.1.2 Fonction genererVague

Cette fonction demande en paramètre le nombre d'ennemi et le type d'ennemi à générer. La variable **lp** est défini par la fonction **placementEnnemi** qui va renvoyer une liste de coordonnées de chaque ennemi à son apparition.

```
def genererVague(self,nbEnnemi,typeEnnemi): #f
    lp = placementEnnemi(nbEnnemi,typeEnnemi)
```

Ensuite chaque ennemi est crée en donnant comme paramètre les coordonnées corespondante de la liste, ainsi que son type. Puis on ajoute l'ennemi créé dans le groupe des ennemis.

```
for i in range(nbEnnemi) :
    ennemi = Ennemi(lp[i],typeEnnemi)
    self.grp_ennemi.add(ennemi)
```

Enfin une fois tout les ennemis ajoutés on met à jour le groupe.

```
self.grp_ennemi.update()
```