PEW PEW

Un projet d'Informatique et Sciences du Numérique

Entièrement réalisé par :

Hugo BLAISON

Arno RAMM

Introduction

Depuis le début de l'année, nous avions déjà l'idée de programmer un petit jeu vidéo d'arcade dans le cadre de notre projet de fin d'année, cependant il nous fallait trouver un concept relativement simple à réaliser pour notre niveau et surtout trouver un jeu qui serait agréable à tester pour n'importe quel joueur. Nous nous sommes d'abord orienté vers un jeu de type "Worms" dans lequel les joueurs pourraient se tirer dessus avec différents types d'armes. Cependant, la réalisation de la trajectoire des projectiles s'avérait très compliquée ...

Nous avons alors décidé de développer PewPew : un jeu de tir convivial, jouable par deux joueurs sur un même clavier, dans lequel ils prennent le contrôle d'un vaisseau spatial pour tenter de détruire celui de l'autre. PewPew possède trois modes de jeu différents (partie rapide, partie en trois manches et en cinq manches), un mode entraînement et il existe une version française ainsi qu'une version anglaise.

Le projet a été réalisé par Hugo Blaison, élève de Terminale S7, qui s'est chargé des graphismes et des fonctions primaires telles que celles nécessaires aux mouvements des projectiles et des joueurs, aux collisions et aux barres de vie/dégâts…

Notre groupe réunit également Arno Ramm, élève de Terminale S10, qui s'est quant à lui occupé des différents modes de jeu (les parties rapides, le mode entraînement, trois manches et cinq manches), du développement et de l'organisation du menu, du système de musique et des pseudos ainsi que de la résolution de certains bugs, dont un majeur (plus communément appelé "bug de la fenêtre").

Présentation de notre projet

Avec notre jeu d'arcade, nous voulions à la fois donner l'impression d'un jeu bien réalisé, amusant, avec une interface propre et claire mais également garder un concept relativement simple et réalisable à notre niveau.

Lorsque nous avons vu l'interface Tkinter en classe, il était donc évident que nous allions nous pencher sur un jeu en 2D avec une ou plusieurs cibles et un système de tir, mais à l'époque rien n'était encore sûr. C'est en novembre que nous avons finalement décidé que notre jeu porterait sur un système de 1 vs 1 où chacun des 2 joueurs se servirait d'une partie d'un clavier pour diriger un vaisseau spatial et tirer sur son adversaire.

Avant de commencer quoi que ce soit, nous avons créé une liste des fonctions que nous voulions utiliser tel que les fonctions de mouvements, de tir, de collisions etc… Ainsi que nos idée principales pour les concevoir et les implanter dans le jeu. Une fois tout cela au clair, nous avons lancé nos premiers essais.

Quand à la répartition des tâches, nous n'avons pas établi de règle stricte. C'est à dire que celui d'entre nous qui décidait de programmer une nouvelle fonctionnalité le signalait simplement à l'autre et demandait de l'aide s'il le faut. Ainsi tout au long du projet nous nous sommes à la fois envoyés des dizaines de versions de notre code (à chaque changement majeur) mais aussi rassemblés chez l'un ou l'autre pour réellement travailler en groupe et de manière synchronisée.

Réalisation du projet

Nos premiers tests concrets sur Python se résumèrent principalement à utiliser ce que nous avions faits dans nos exercices pour coder une fenêtre avec des dimensions correctes, un canvas avec une limite entre la zone de jeu et la zone d'information (où se situeront les barres de vie) et les prémices de nos vaisseaux, à savoir de simples rectangles.

Version 1.1 de Pew Pew, à cette époque les seuls interactions possibles étaient le mouvement du rectangle avec les touches directionnelles du clavier

Afin de faire bouger un élément sur l'écran (ici à gauche par exemple) nous avons utilisé le code :

```
if Action1.keysym=="Left" and Coordonees[0]>50:

        ZoneJeu.move("Joueur1", -20,0)
```

Qui fait donc bouger le vaisseau de 20 unités d'abscisses vers la gauche.

Notre second défi après avoir créé un deuxième vaisseau fût de permettre à ceux-ci de tirer un projectile, c'est ici que nous avons rencontré un premier problème puisque la vitesse des tirs ne cessaient d'accélérer après chaque pression de touche. Voici ici le code originellement utilisé:

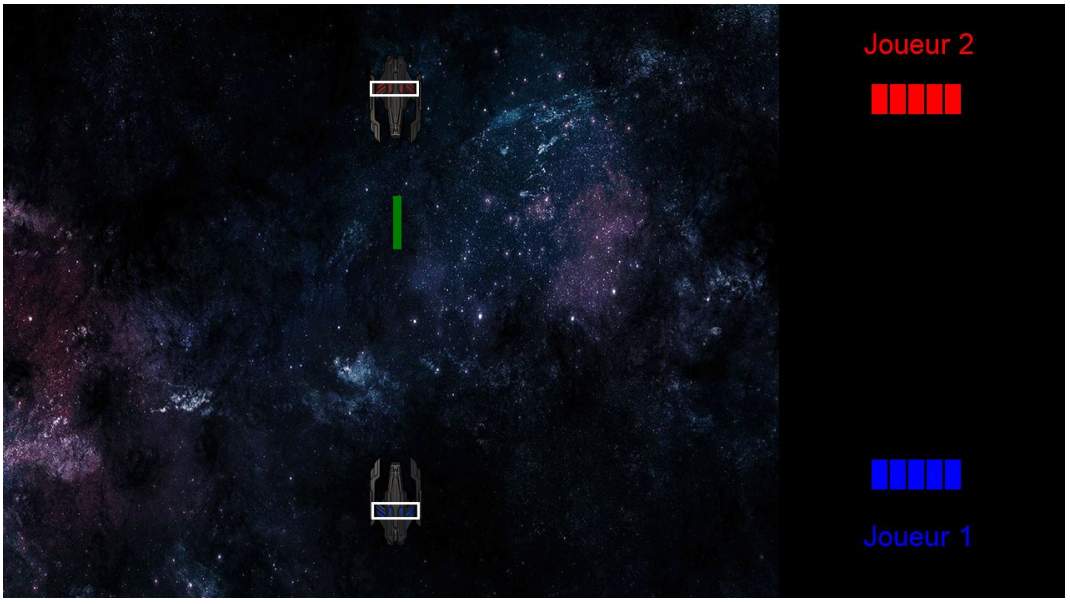
```
# Fonction de tir

def Tir():
    Coordonnes=ZoneJeu.coords("Joueur1")
    ZoneJeu.create_line(Coordonees[0], Coordonees[1]+20,
Coordonees[2], Coordonees[3], fill="red", width=5, tag="Laser")
    MouvementTir("Laser")

def MouvementTir(Projectile):
    ZoneJeu.after(100,MouvementTir,"Laser")
    ZoneJeu.move(Projectile,0, -5)
```

La première fonction servant à créer le projectile et la seconde à le faire avancer

Pour remédier à ce problème nous avons plus tard décidé de donner un seul tir aux joueurs tant que le projectile n'a pas atteint sa cible ou n'est pas sorti de l'écran. Il s'agissait bien-sûr d'un choix pratique car de cette façon le problème disparaissait, mais avant tout d'un choix par rapport à la jouabilité puisque cette modification a rendu le jeu bien plus stratégique.

Cependant, un système de tir ne serait pas complet sans des barres de vie et un système de détection des dégâts, ces derniers furent l'objet de notre troisième projet. Il s'agit sans doute de la partie de notre jeu où nous avons eu le plus de mal puisque Tkinter n'était pas forcément la meilleure interface pour détecter des collisions entre différents objets mais nous y sommes finalement parvenus.

Avant de commencer à programmer ces collisions il nous fallait d'abord des indicateurs de vie, nous avons opté ici pour des petits rectangles, 5 par joueurs, dont chacuns symboliseraient un point de vie et qui disparaîtraient en cas de collisions (voir photo ci-dessous).



Ici sont représentées en blanches les zones de collisions qui nous servent de point de repère (c'est à dire que lorsqu'elles sont atteintes, une barre de vie disparaîtra)

Afin d'étudier les collisions de ces zones blanches, nous avons fait recours à la fonction "find_overlapping" qui permet de retourner, sous forme de liste, les objets avec lesquels un autre objet est en collision. Il suffisait alors de compter le nombre d'éléments dans cette liste et s'il venait à augmenter, alors cela voudrait dire qu'un nouvel objet serait entré en contact avec notre case blanche; une barre de vie serait alors retirée. A ce système il a bien-sûr fallu ajouter des fonctionnalités (par exemple le fait qu'un seul projectile ne peut faire qu'un seul point de dégât, d'où la création d'une variable "AncienneCollision").

Voici ici un exemple de condition de collision pour le premier point de vie du joueur 1:

```
    if NombreCollisionJoueur1 > 3 and
AncienneCollision!=CollisionsJoueur1[3] and Degats1==5:

        ZoneJeu.delete(VieJoueurUn5)
        ZoneJeu.after_cancel(id2)
        ZoneJeu.delete("Laser2")
        DelaiTir2=0
        AncienneCollision=CollisionsJoueur1[3]
        Degats1=Degats1-1
```

Remarques :
- Ici “Degats1” désigne l’état de la barre de vie du joueur 1, si elle est égale à 5 cela veut dire que la barre est pleine
- “VieJoueurUn5” est le nom du premier rectangle de la barre de vie du joueur 1

Une fois notre mode de jeu 1 contre 1 entièrement opérationnel, notre idée était de créer un menu où le joueur pourrait choisir entre une partie rapide, des matchs en 3 ou 5 manches, un mode entraînement ou encore un tutoriel. La base de tout ce projet est donc la fonction “Menu” qui regroupe la création de tous les boutons utiles à la navigation, il n’est donc pas vraiment intéressant d’afficher une partie du code ici.

Pour accéder au mode de jeu en lui même (ainsi que le mode en 3 et 5 manches) nous avons créer une fonction “Lancer” qui retire tous les boutons du menu pour laisser place aux 2 vaisseaux, les barres de vie et les pseudos. Elle permet également de faire appel à la fonction “ComparaisonDegats” qui permet de détecter si un des vaisseaux est endommagé. Pour finir, nous avons développé une fonction “Retour” qui permet de stopper la fonction “ComparaisonDegats” et faire à nouveau appel à la fonction “Menu”.

L’entrée des pseudos par les joueurs (une idée que nous avons implanté sur un coup de tête puisqu’elle n’était pas dans nos objectifs initiaux) se fait aussi dans le menu à l’aide de simples labels Tkinter:

```
def Getpseudo1():

    global Nom1,Pseudo1,VariablePseudo1

    Nom1=Pseudo1.get()

    for i in range(len(Nom1)):
        Pseudo1.delete(0,None)

    VariablePseudo1=1
```

Notre dernière addition au jeu est un mode entraînement qui permet au joueur de contrôler un vaisseau et de tirer sur un ennemi généré à une position aléatoire. A chaque tir réussi, il gagne un point de score qui s’ajoute à son compteur. Cette fonction était très simple à programmer puisque nous avons principalement ré-utilisé certaines méthodes. Le système de score est simplement une variable qui augmente de 1 à chaque vaisseau détruit et qui est affiché par un label Tkinter.

Conclusion

La version finale de PewPew correspond presque parfaitement à l'image que nous nous étions fait du jeu au début du développement. Nous avons réussi à développer un jeu jouable par deux joueurs sur le même clavier afin de le rendre convivial à la manière des Shoot'em up un peu "old school" (tel que Space Invaders). Tous les modes de jeu initialement prévus sont jouables, et nous avons même réussi à mettre en place l'ambiance rétro/arcade que nous avions imaginée (bien que la cohabitation Tkinter/pygame fût assez laborieuse). Malheureusement, nous n'avons pas eu le temps de développer un système de score global par manque de temps : nous souhaitions que, lorsqu'un joueur gagne un match, des points lui soient attribués et reliés son pseudo pour ensuite effectuer un classement des meilleurs joueurs (cette fonctionnalité était nommée " Panthéon des héros" dans les versions antérieures puis a été supprimée par la suite).

Nous sommes plutôt satisfaits de l'allure du projet final, puisque l'écart entre les choses que nous voulions réaliser et celles qui l'ont été est léger. Il reste cependant 2 sources d'erreurs (erreurs minimes qui n'impactent pas le jeu) dans notre code:

- Les touches liées au déplacement sont activées lorsque les joueurs sont en partie, et désactivées lorsqu'ils sont dans le menu. Cependant, l'écran de fin de partie est considéré comme intégré à la partie, et les touches sont toujours liées à la fonction de déplacement : ainsi, si le joueur appuie sur les touches ZQSD sur l'écran de fin, la console affichera des erreurs (car python essaye d'appliquer un déplacement à un vaisseau n'existant plus sur l'écran de fin). Cette erreur pourrait être fixée mais cela prendra plusieurs heures car il faudrait changer plusieurs conditions dans les fonctions primaires, ce qui demanderait de nombreux changements collatéraux.

- La musique (aussi bien celle du menu que celle du jeu en lui même) se coupe totalement lorsque le joueur navigue trop vite entre les différents modes/menus. Après quelques recherches sur internet, nous avons appris que le problème vient pas de nous mais de Pygame. En effet celui-ci semble être incapable de suivre lorsque trop de pistes sont lancées rapidement.

Nous souhaiterions poursuivre un peu notre projet, par exemple pendant notre temps libre cet été, en proposant par exemple des nouveaux modes de jeu (2 vs 2) ou même un mode joueur contre IA. Nous aimerions également poster notre code en ligne afin de pouvoir recueillir des avis ou témoignages de différents joueurs, Arno aurait également voulu enregistrer et interpréter lui même la musique du jeu au piano mais cela ne s’est pas réalisé par manque de temps.

Références

Sites utilisés : http://balirosfabert.fr/
https://openclassrooms.com/
https://stackoverflow.com/

Nous avons réalisé nous même l'intégralité du code, ainsi que les images (sur Photoshop) à l'exception de l'image servant de fond d'écran dans les parties (qui est une image libre de droits). Les musiques du menu, de jeu et de l'écran de victoire proviennent respectivement de la bande son du film Interstellar (composée par Hans Zimmer), des jeux vidéo League of Legends et Final Fantasy. Le fameux bruitage caractéristique "PewPew" est quant à lui un bruitage libre de droit trouvé après de longues recherches sur Internet.