Projet : Borne d'arcade

Jason LALANDE & Léo Laurens

Sommaire

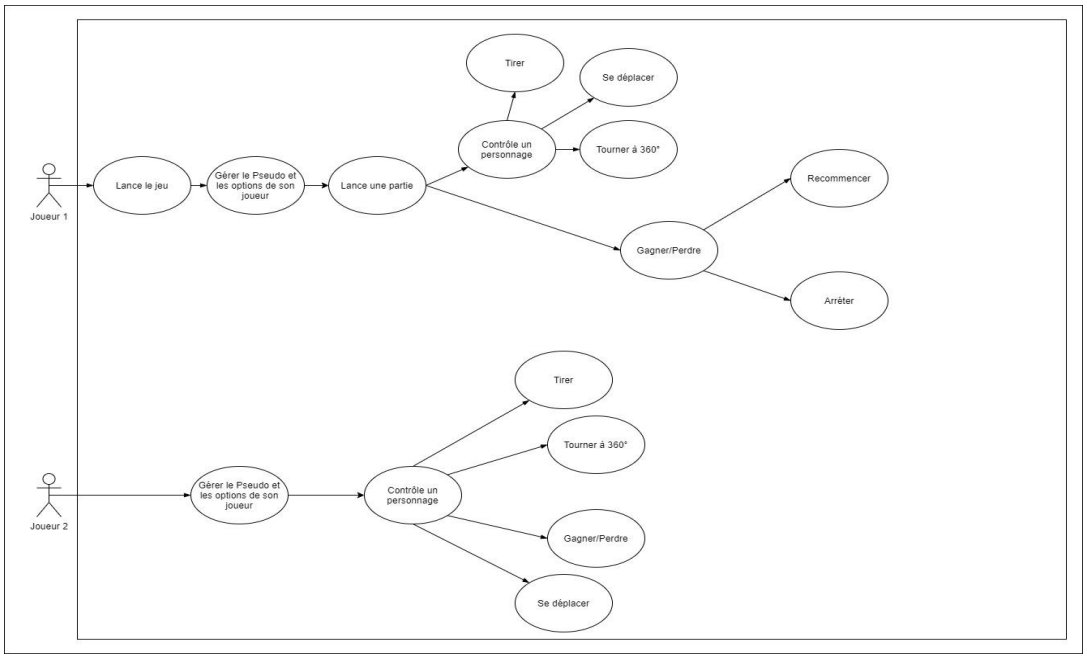
I – Introduction

Ce projet consiste à créer une mini borne d'arcade contenant un jeu simple de 1 VS 1, à l'aide d'une Raspberry pi et de boutons/joystick à assembler.

Les caractéristiques à tenir en compte sont :

- Plusieurs écrans (Menu principal, menu d'option, highscore, etc.)
- Deux joueurs possédants des caractéristiques distinctes pouvant se modifier à notre guise le tout relié à une base de données.
- Une interface utilisateur entièrement gérée par une « manette »

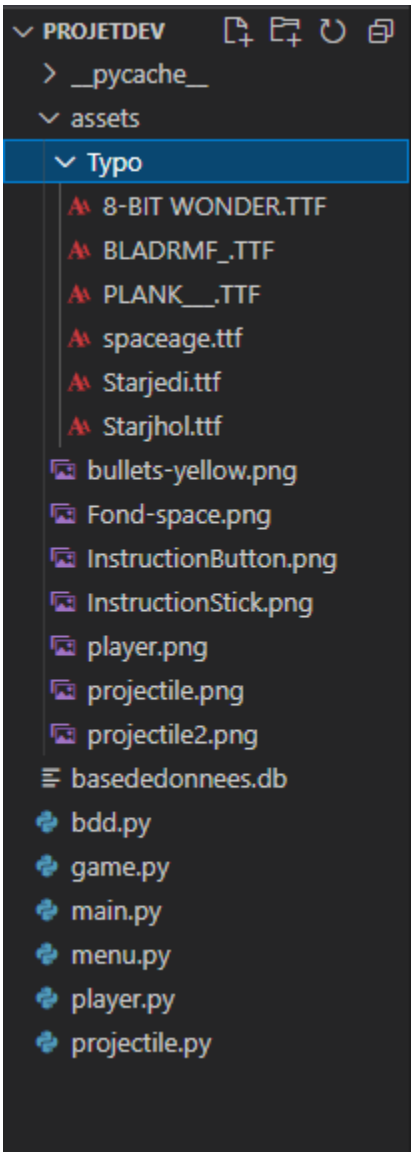
Avec pour ligne directrice de suivre ce diagramme de cas :



Au début de ce projet, nous avons décidé de partir sur le Python comme langage de programmation car bien que nous ne connaissions pas encore le langage, toutes nos recherches sur le Raspberry nous ramener à Python.

En parallèle, nous avions l'habitude d'utiliser Discord où nous utilisions tout le temps la fonction de partage d'écran qui nous a permis d'avoir une communication simple rapide et visuelle.

II – Structure Algorithmique



Notre projet est structuré ainsi, un dossier appelé « ProjetDev ».

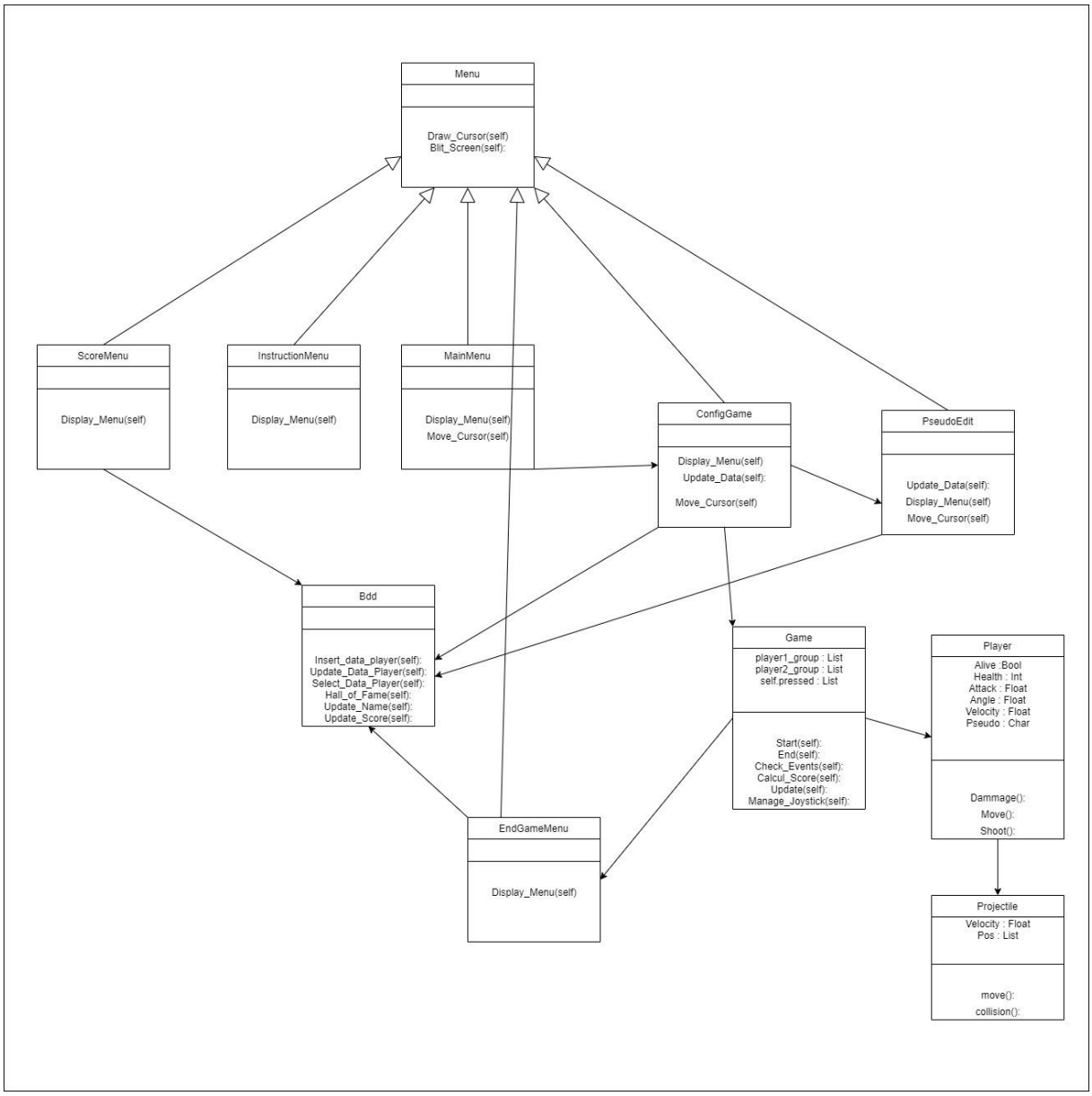
Dans ce dossier on y retrouve le « main.py » là où lance le jeu ainsi que ses fichiers complémentaires tels que « bdd.py », « game.py », « menu.py », « player.py » et « projectile.py » qui sont simplement des parties de programmation segmenter pour une meilleure compréhension et organisation du projet.

De plus nous y retrouvons le dossier « assets » qui regroupe, comme son nom l'indique, les assets du jeu. On y verra donc toutes les images en .png utilisées ainsi qu'un dossier regroupant plusieurs typographies.

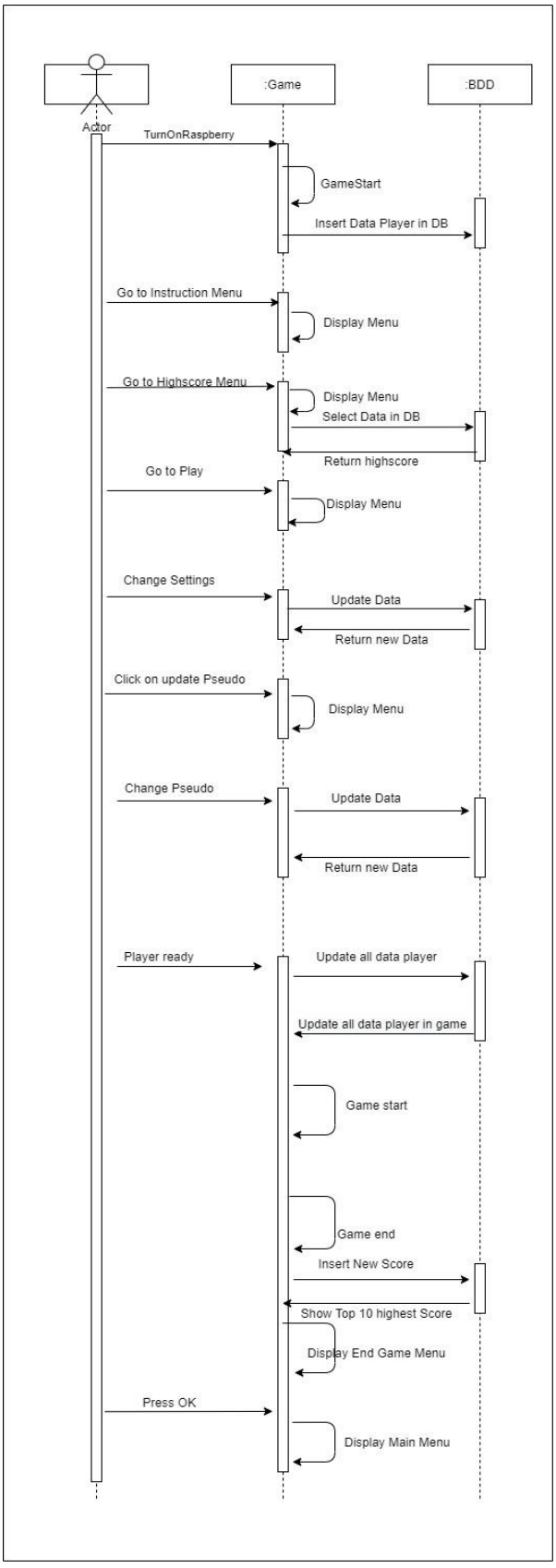
III – Architecture

Voici quelques diagrammes qui ont permis de bien structurer notre projet ainsi que tous les écrans que vous allez rencontrer dans le jeu:

- Notre diagramme de classe :

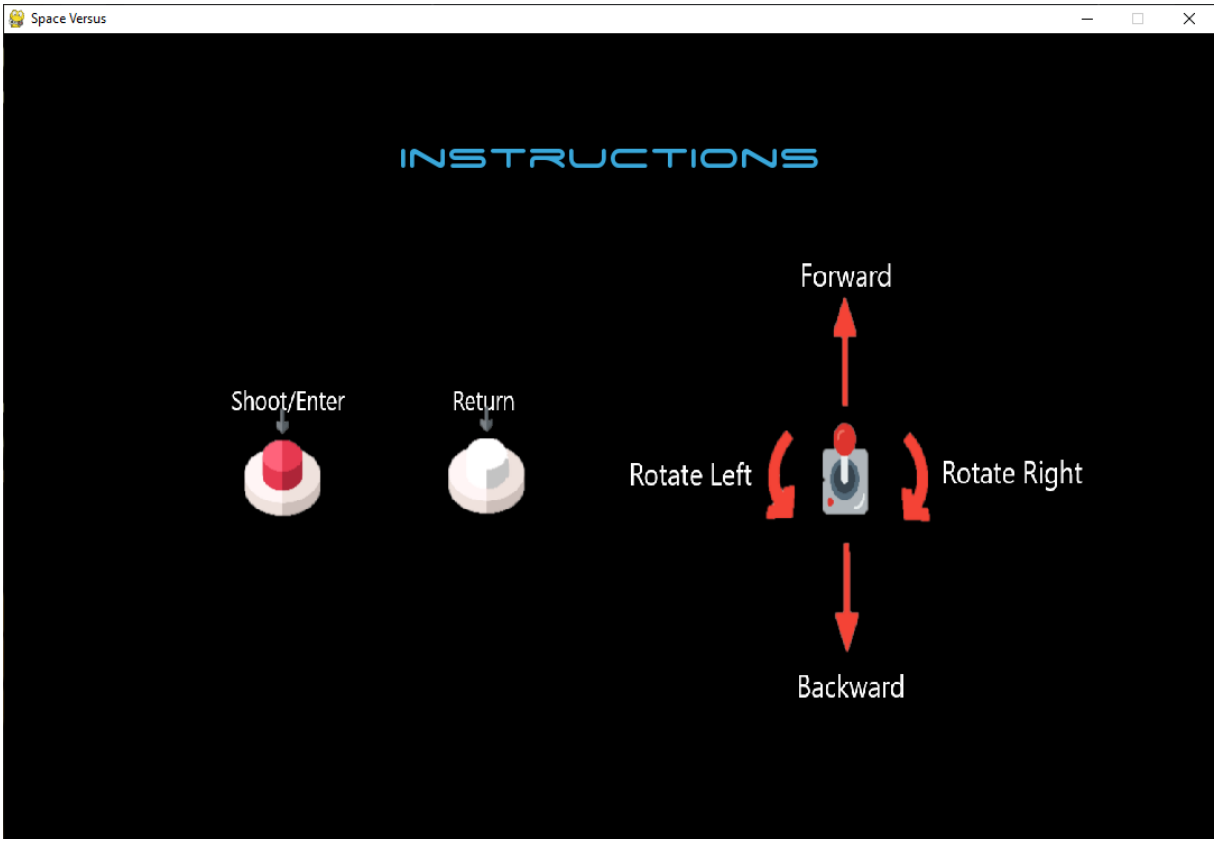


- Notre diagramme de séquence :



- Menu Principal :

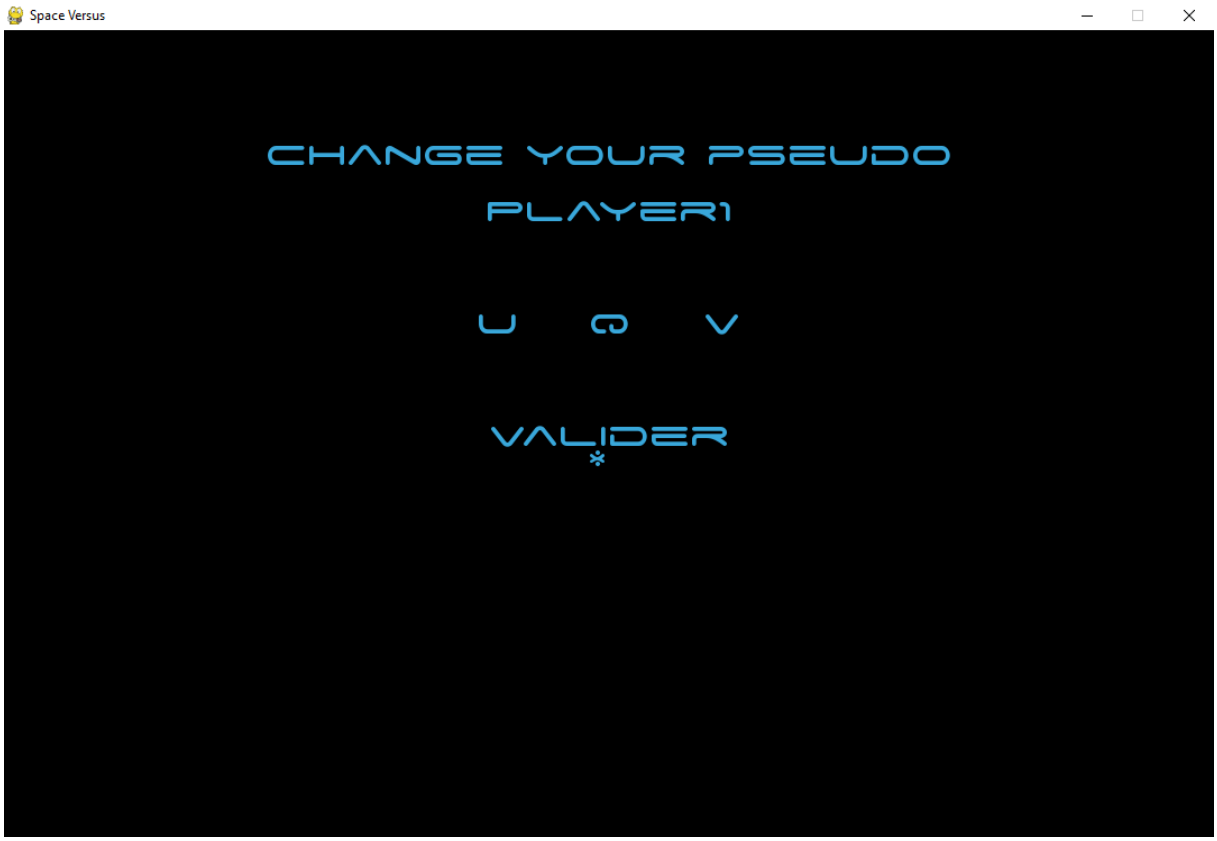
- Menu d'instruction :



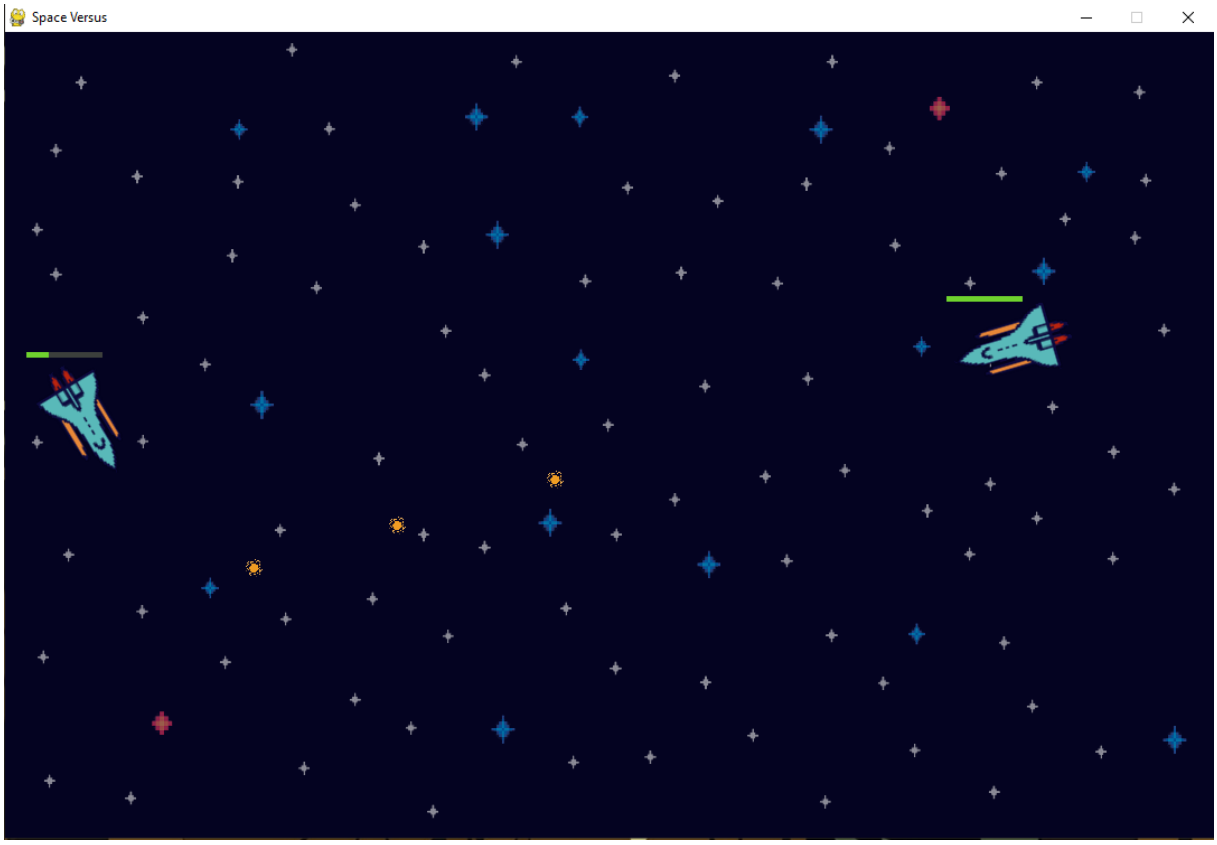
- Menu des meilleurs scores :

- Menu Option du jeu :

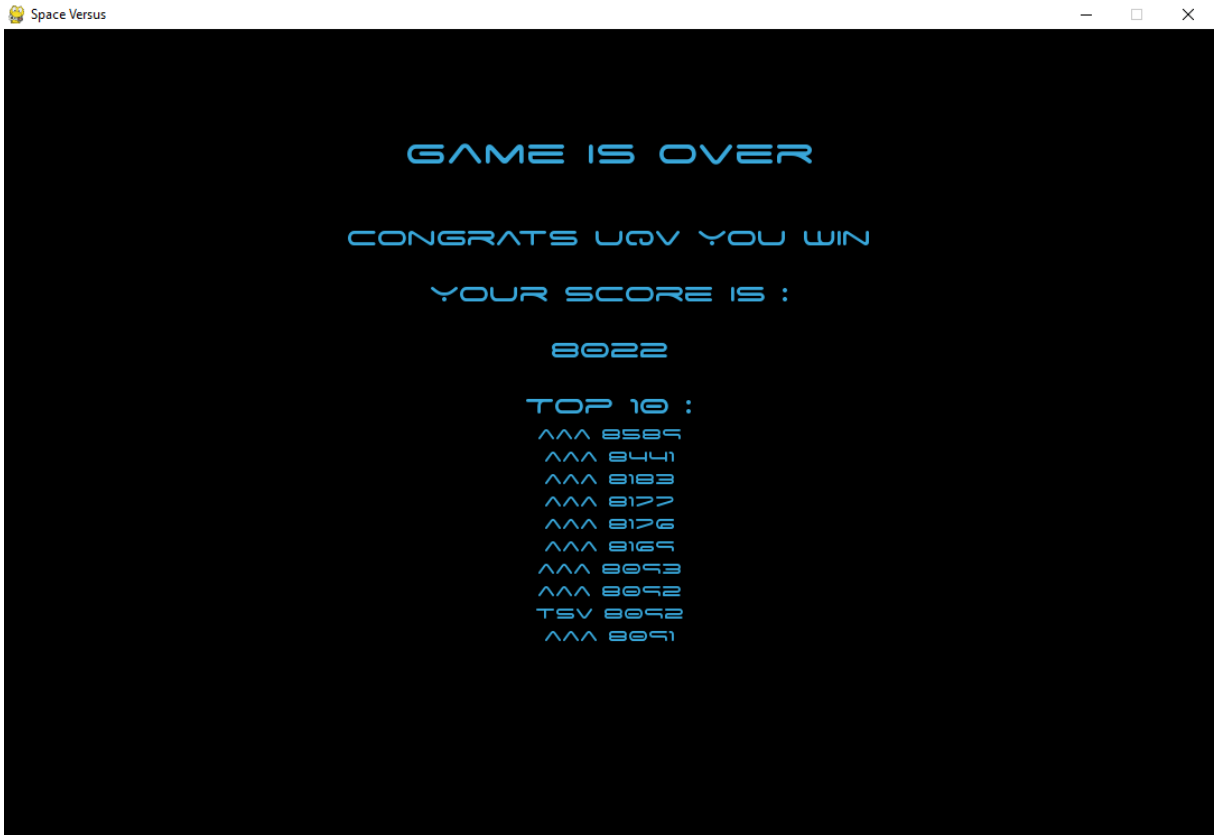
- Menu de changement de pseudo



- Partie en cours :



- Ecran partie terminée :



IV – Fonctionnalités Majeures

Pour vous montrer un peu de notre code, nous allons vous expliquer comment nous avons procédé pour se déplacer dans le menu des options de jeu (avant de lancer la partie).

Premièrement, nous avons créé une liste recensent les états du futur curseur, puis nous leur avons attribué la position x et y du rectangle qui leur est attribué sur l'affichage.

```
self.list_setting_p1=[["pseudo"], ["velocity"], ["angle_speed"], ["attack"], ["health"],["valider"]]
```

```
for idx,data in enumerate(donnees):
    if idx != 0:
        self.game.draw_text(f"{data}", 25, self.game.screen_dimension[0] / 2 - 150, self.game.screen_dimension[1] / 2 - y + 20)
        self.list_setting_p1[i] = self.game.screen_dimension[0] / 2 - 150, self.game.screen_dimension[1] / 2 - y + 20
        y -= 20
        i += 1
```

Précision que quand nous apparaissons sur le menu l'état de base est « Pseudo ».

De ce fait quand le jeu capte que le joystick est vers le bas et change d'état.

Ce changement d'état permet de réattribuer une nouvelle position au curseur, dans ce cas-là « Velocity », et ainsi de suite.

```python
def move_cursor(self, cursor, list, cursor_state, idx):
    if self.game.list_keys[idx][1]:
        if cursor_state == 'Pseudo':
            cursor.midtop = (list[1][0] + self.offset, list[1][1])
            cursor_state = 'Velocity'
            self.cursor_list[idx] = cursor_state

        elif cursor_state == 'Velocity':
            cursor.midtop = (list[2][0] + self.offset, list[2][1])
            cursor_state = 'Angle_speed'
            self.cursor_list[idx] = cursor_state

        elif cursor_state == 'Angle_speed':
            cursor.midtop = (list[3][0] + self.offset, list[3][1])
            cursor_state = 'Attack'
            self.cursor_list[idx] = cursor_state
        elif cursor_state == 'Attack':
            cursor.midtop = (list[4][0] + self.offset, list[4][1])
            cursor_state = 'Health'
            self.cursor_list[idx] = cursor_state

        elif cursor_state == 'Health':
            cursor.midtop = (list[5][0] + self.offset, list[5][1])
            cursor_state = 'Valider'
            self.cursor_list[idx] = cursor_state

        elif cursor_state == 'Valider':
            cursor.midtop = (list[0][0] + self.offset, list[0][1])
            cursor_state = 'Pseudo'
            self.cursor_list[idx] = cursor_state
```

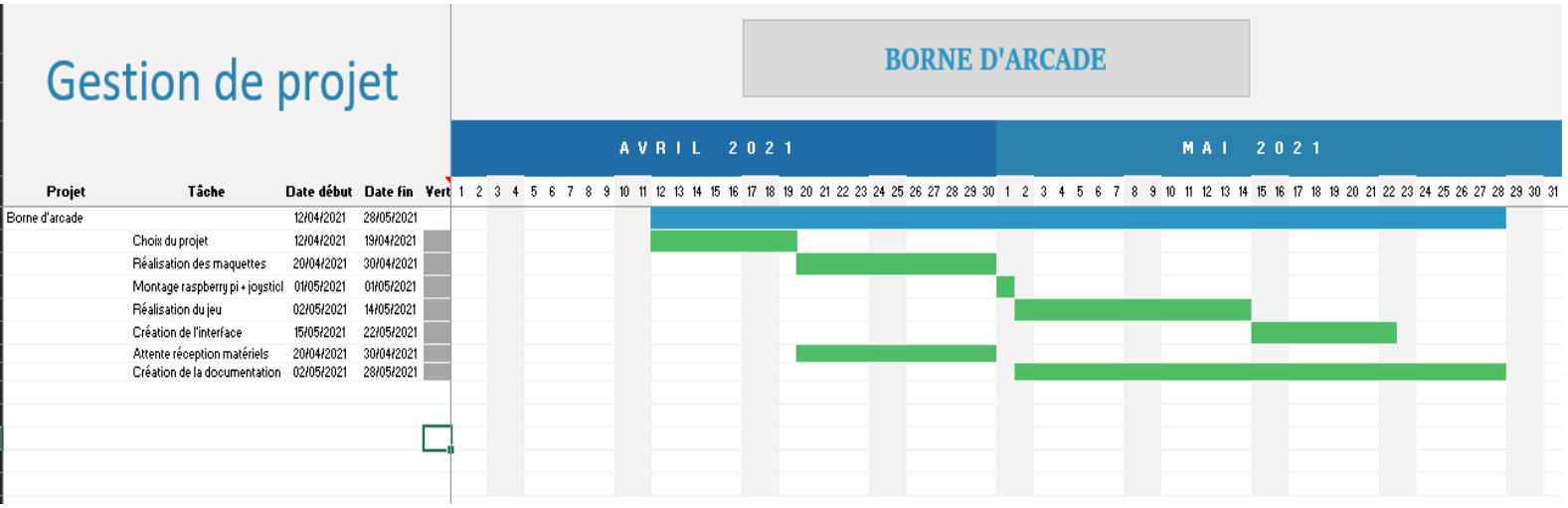
De plus vous pouvez voir cette ligne `self.cursor_list[idx] = cursor_state`, ceci nous permet d'avoir une variable qui nous servira pour mettre à jour la base de données concernant les statistiques du joueur.

```python
def gestion_menu(self, event, idx, instance_id, player):
    if event.dict['axis'] == 0 and round(event.dict['value']) == 1 and event.dict['instance_id'] == instance_id:
        if (self.cursor_list[idx] != "Valider" and self.cursor_list[idx] != "Pseudo"):
            self.game.list_keys[idx][4] = True

            self.bdd.update_data_player(player, self.cursor_list[idx], 1)

    if event.dict['axis'] == 0 and round(event.dict['value']) == -1 and event.dict['instance_id'] == instance_id:
        if (self.cursor_list[idx] != "Valider" and self.cursor_list[idx] != "Pseudo"):
            self.game.list_keys[idx][5] = True

            self.bdd.update_data_player(player, self.cursor_list[idx], -1)
```

Ci-dessus, nous voyons donc la fonction qui selon l'état du curseur nous permet de faire une requête qui augmente ou réduit les statistiques du joueur, pour sa future partie.

V – Rôle de chacun

Dans ce projet nous avons toujours travaillé ensemble, l'un partageant son écran et programmait pendant que l'autre faisait des recherches et des propositions pour résoudre tel ou tel problème. Cela de façon alternée. Nous faisions en sorte que toutes les semaines de nouveaux objectifs soient définis et à chaque fin de session un debrief était fait pour savoir ce que nous devrions faire le lendemain tout en respectant le GANTT que nous avions créé.



VI – Synthèse

Le but premier de ce projet était de réaliser un jeu de type « Versus », faisant appel à des réflexes, de la rapidité d'exécution et du game-sens. Les objectifs exposés dans l'introduction ont bien été réalisé.

On y retrouve un « Menu Principal », un écran de configuration de la partie et du pseudo, un écran high score, un écran de fin de partie et évidemment un écran où se déroule la partie.

Les joueurs peuvent modifier à volonté leurs caractéristiques ce qui influera sur le score final selon la difficulté pour le joueur vainqueur.

Nous n'avons pas spécialement rencontré de grosse problématiques, mis à part la compréhension des vecteurs pour déplacer le joueur ainsi que d'adapter le programme sur le Raspberry.

Le jeu est codé intégralement en Python et relié à une base de données. Vous pouvez retrouver l'entièreté du code sur GitHub.

https://github.com/Leo-Lrs/Spave-Versus.git