

SHIPBATTLE



Rendu projet Dev 2021

MODOLO Thomas / MATAS Lucas

Document technique Borne d'arcade

I - Introduction.....	p.3
II - Contexte.....	p.5
III - Le projet.....	p.7
IV - Architecture.....	p.18
V - Développement.....	p.20
VI - Conclusion.....	p.23



I - Introduction

Dans ce document, nous allons voir comment nous avons réussi à mettre en œuvre notre **projet de fin d'année**.

Notre projet consistait à créer une mini borne d'arcade contenant un jeu simple de **1 Contre 1**, à l'aide d'un **Raspberry Pi** et de boutons/joysticks à assembler.

Nous avons pris ce projet car on aimait le défi, mais aussi car c'est le projet qui nous parlait le plus en termes **d'amusement** et de défis techniques.

Dans un premier temps on va voir le contexte du projet afin de mettre en situation notre projet.

Par la suite nous verrons la présentation du projet en profondeur, nous passerons également par l'architecture que nous avons mis en place.

Puis nous rentrerons en détails dans certaines parties du code qui fut un réel défi à réaliser autant en terme technique **qu'algorithme**.

Et nous finirons par la **conclusion** de ce projet ainsi que les axes **d'améliorations possibles**.



Une dernière chose. Suite au choix de ce projet nous avons dû acheter le matériel suivant :

- Kit Raspberry Pi 4 4GB de RAM



- Kit de joysticks et de boutons pour Raspberry Pi 4



II - Contexte

Notre équipe est composée de 2 personnes :



MATAS Lucas



MODOLO Thomas

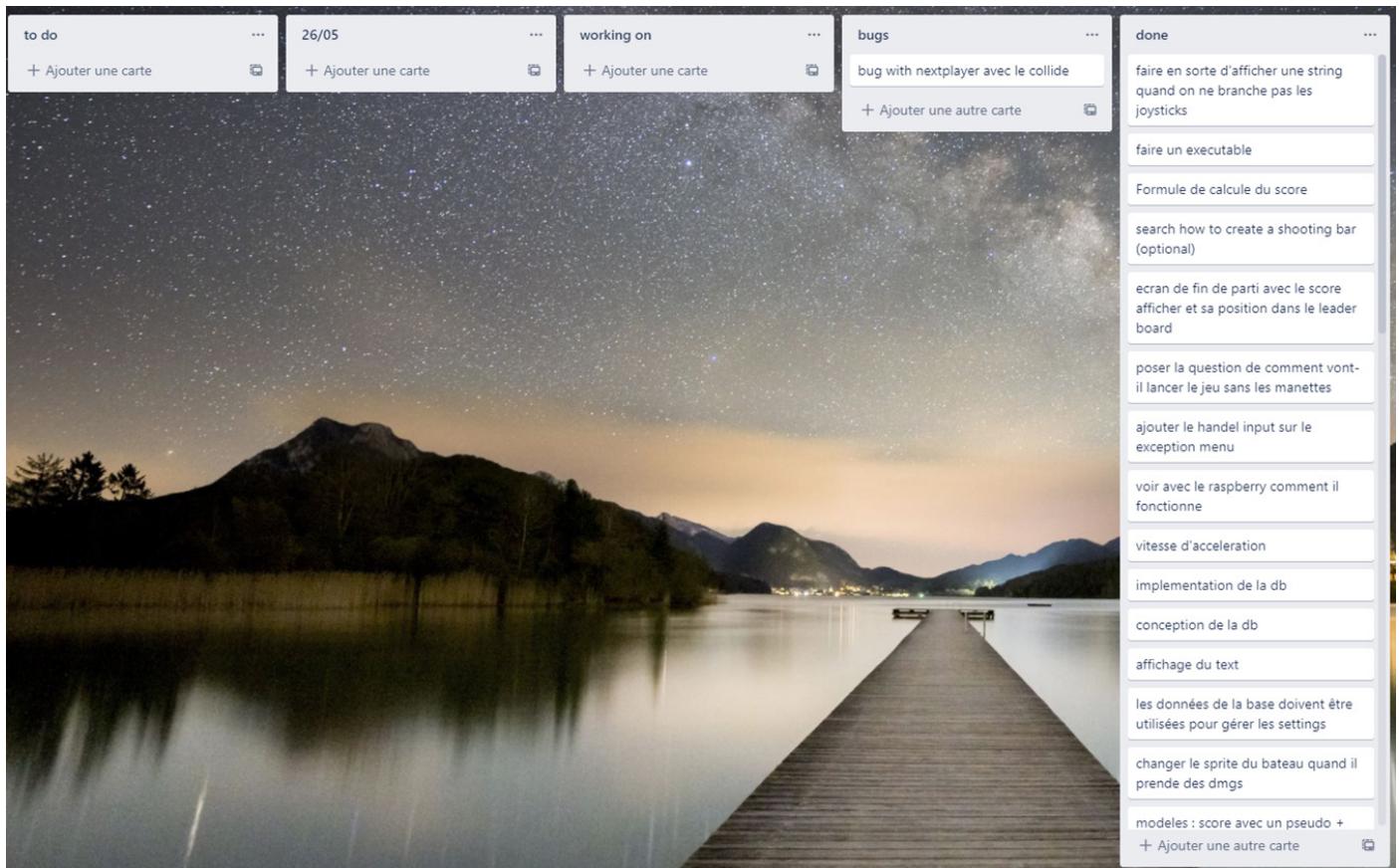
Nous travaillons ensemble depuis plus de **2 ans** sur des projets divers et variés. Toute l'expérience acquise durant ces 2 années nous ont permis de nous adapter sur la division des tâches à effectuer, ainsi que nos forces et nos faiblesses.

En effet, Lucas est plus sur la forme alors que Thomas se concentre sur le fond.



Nous nous sommes attribué les tâches en se basant sur nos **points fort** cités juste au-dessus.

De ce fait, nous nous sommes organisés avec un **Trello** mis à jour tous les jours afin que l'on puisse effectuer toutes les tâches nécessaires au **bon fonctionnement** du jeu.



Pour ce projet, lors de la lecture du cahier des charges nous nous sommes aperçus d'une problématique, les bornes d'arcades sont très/trop souvent rigides et ne permettent pas de pouvoir changer des éléments de gameplay à la guise du joueur. Nous avons donc décidé de mettre en place une borne d'arcade avec un jeu qui est **complètement modifiable**.

Si nous avons pu réaliser ce projet c'est en grande partie grâce à l'enseignement que nous avons reçu, que ce soit pour la gestion de projet (les diagrammes de classes, de séquences..), les bases de données ou encore la programmation orientée objet. Cependant, nous avons dû nous adapter sur une technologie que nous n'avions presque jamais utilisé auparavant qui n'est d'autre que le language **Python**.



III - Le projet

Le projet que nous devons réaliser est : créer une **mini borne d'arcade** contenant un simple jeu de **1 contre 1**, à l'aide d'un **Raspberry Pi** et de boutons/joysticks à assembler.

Partant de cette thématique, nous avons réalisé la liste des **fonctionnalités nécessaires** :

- Mettre en place des scènes de jeux pour les différentes actions du jeu
- Créer les modèles de données (joueur, score)
- Mettre en place une base de données
- Faire fonctionner le Raspberry Pi ainsi que les joysticks
- Créer la charte graphique ainsi que l'univers dans lequel les joueurs vont évoluer
- Ainsi que créer le jeu

Nous avons donc choisi pour le style graphique, un jeu de bataille navale où ils devront se **combattre** à l'aide de **leurs canons** jusqu'à qu'il n'en reste qu'un. On se retrouvera donc sur des décors **tropicaux** composés d'eau et de sable où les bateaux pourront naviguer.

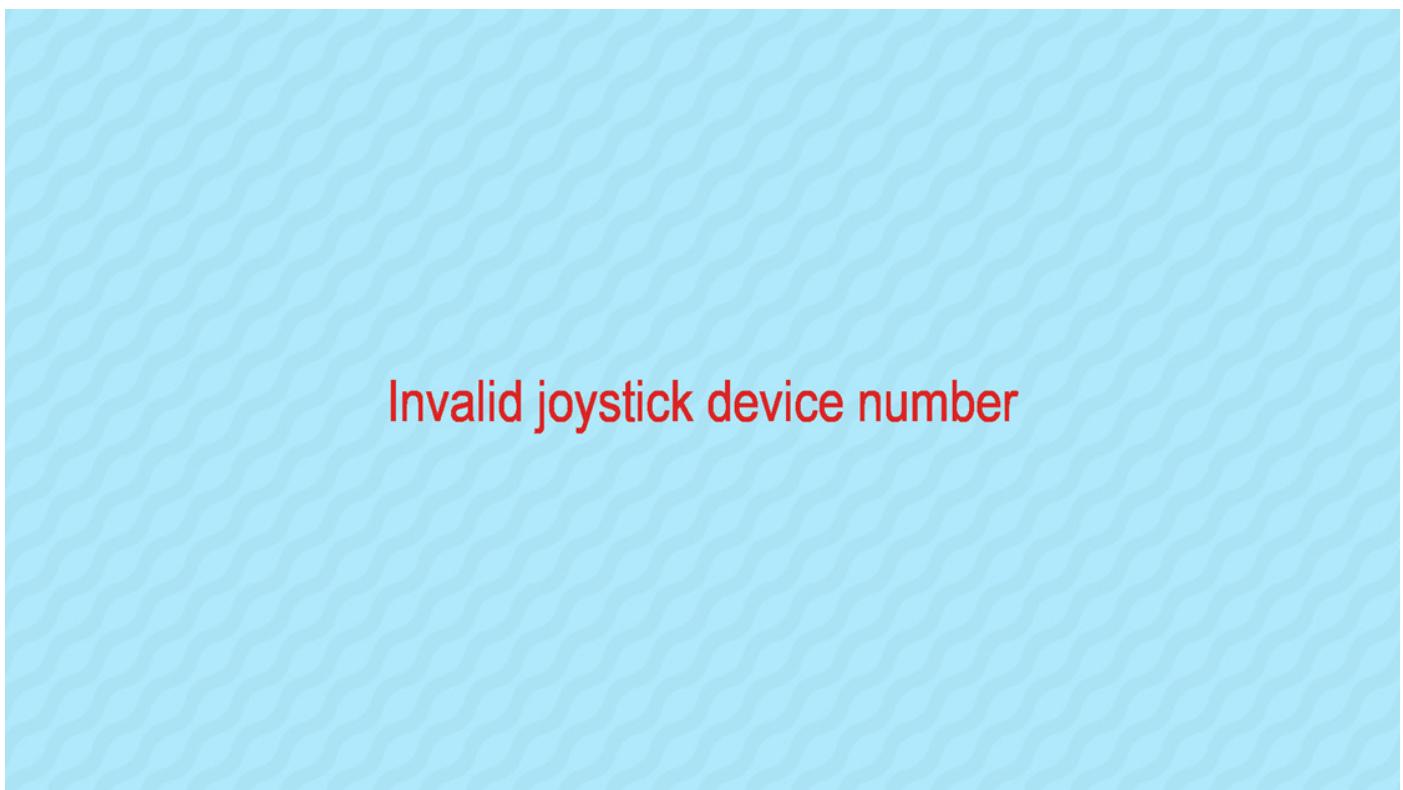
Tout ça viendra avec un système de **classement** ainsi que des paramètres totalement modulables à sa guise, tout ça sur un Raspberry Pi.



Voici sur quoi on tombe quand on lance le Raspberry Pi. Le joueur pourra naviguer dans les différents menus à l'aide du joystick et des boutons.



À noter que nos manettes sont **essentielles** pour jouer au jeux. Sinon vous tomberez sur ce menu

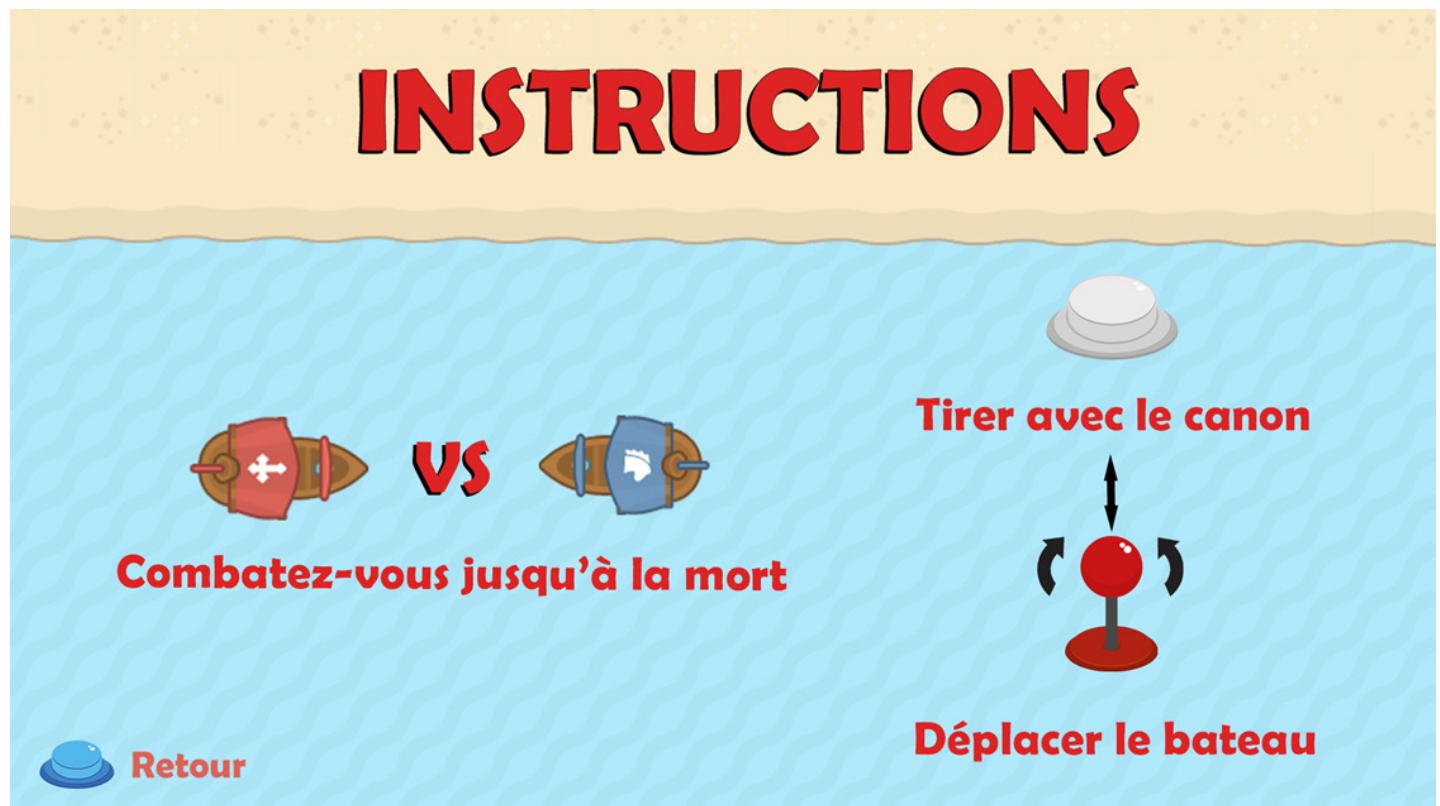


Voici les 3 menus possibles depuis l'écran d'accueil :

- Jouer : Vous permet d'accéder à l'écran de paramètres des joueurs.



- Comment jouer : l'écran nous montre comment jouer au jeu et à quoi servent les boutons/joysticks



- L'écran des scores : affiche les 10 premiers meilleurs scores



Comme vous pouvez le voir sur les différentes captures d'écrans les menus restent simples à naviguer et tout ça se fait à l'aide du bouton bleu : Retour, et le bouton blanc : Accepter

Tous les menus ont été faits à l'aide des sprites suivant :



Maintenant, revenons-en à notre menu des paramètres



Voici tous les paramètres que vous pourrez changer :

- Le bateau
- La manœuvrabilité
- La vitesse du bateau
- La vitesse du projectile
- Vos points de vie
- Le délai entre les tirs
- Les dégâts des projectiles

Pour passer au menu suivant, les deux joueurs devront être «READY». Tous les paramètres seront ensuite mis à jour dans la base de données.

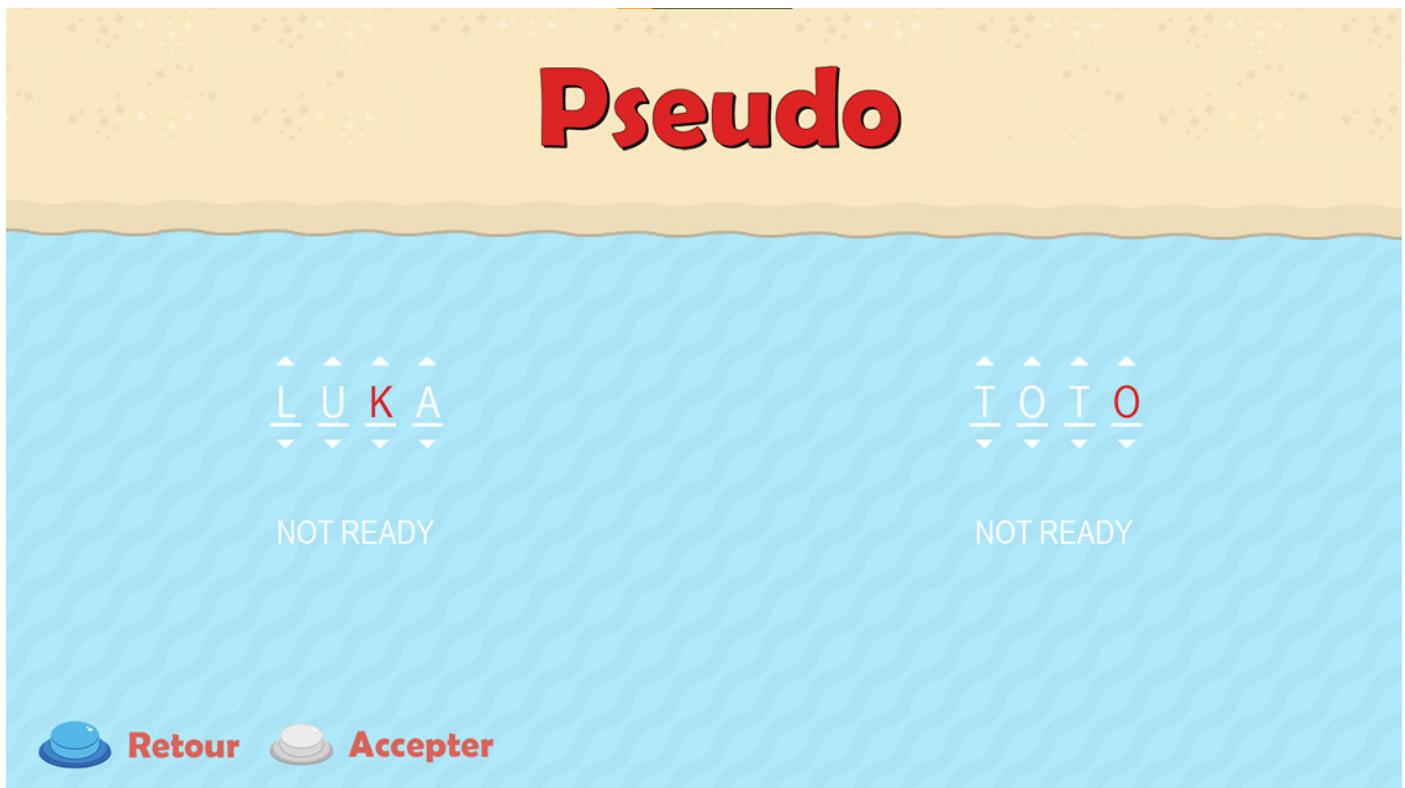


Voici les bateaux disponibles :

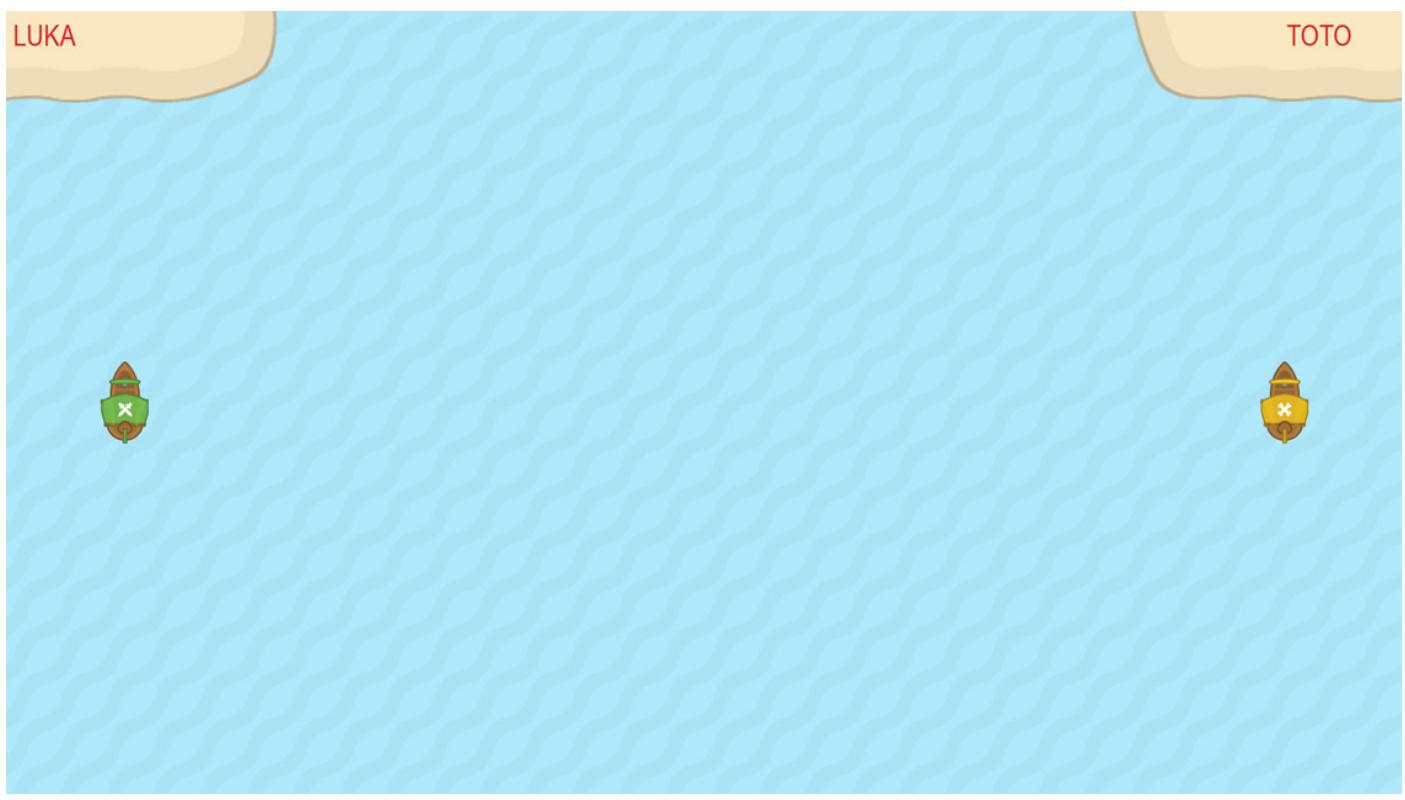


Une fois les paramètres validés vous devrez ensuite choisir un pseudo. Les lettres vont de A à Z et tout se fait avec le joystick.

Encore une fois, les deux joueurs devront être «READY» pour lancer la partie



Après cela vous pouvez enfin jouer



Quand un joueur touche son adversaire une effet d'explosion apparaîtra, ce qui indiquera qu'il a pris des dégâts



Au fil de la partie, le bateau se dégradera selon les dégâts qui lui ont été infligés. En plus de cela, les pirates passeront par-dessus bord



Voici les différents stades de dégradations pour un bateau :

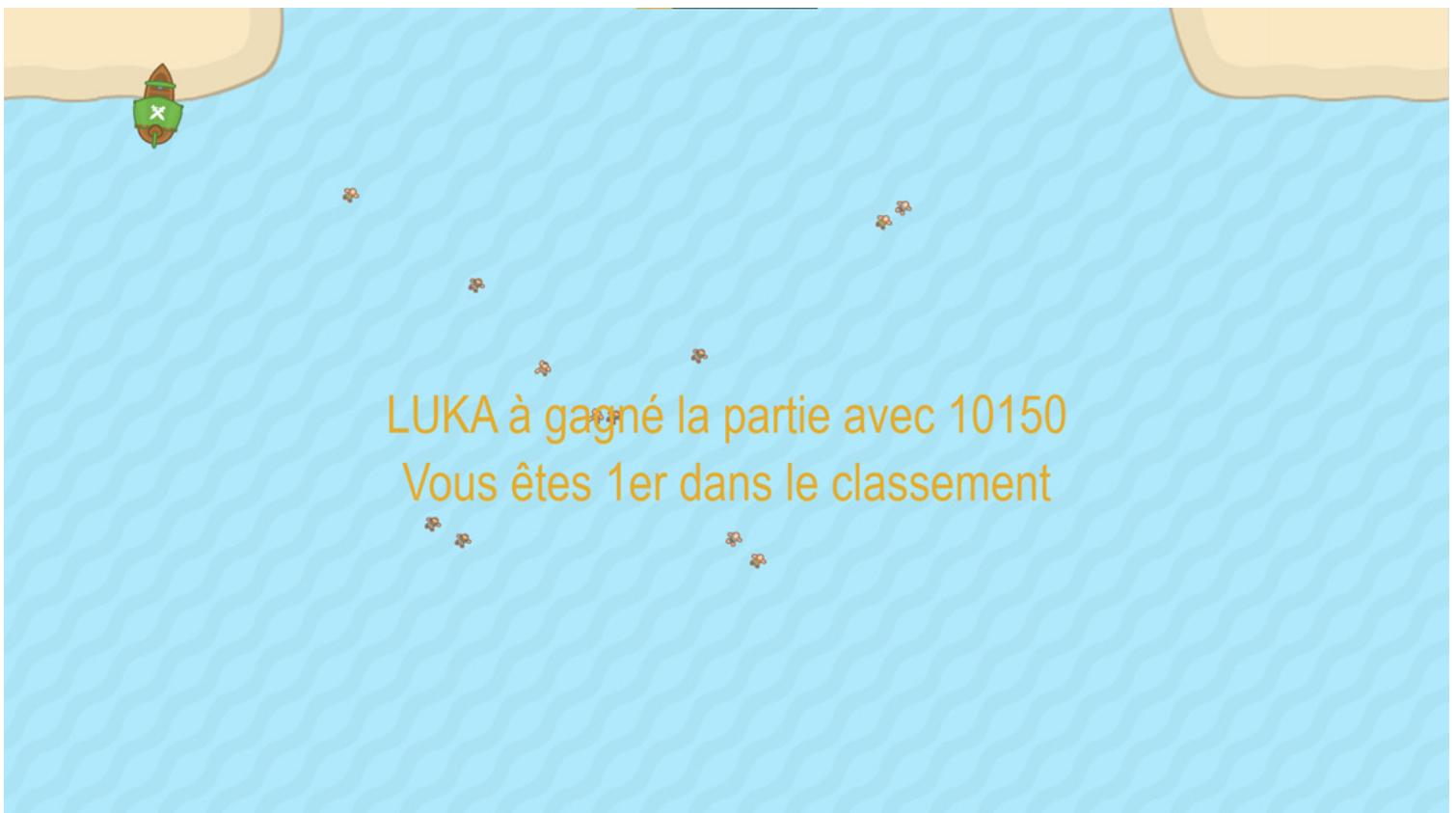


Et voici les différents sprites de pirates :

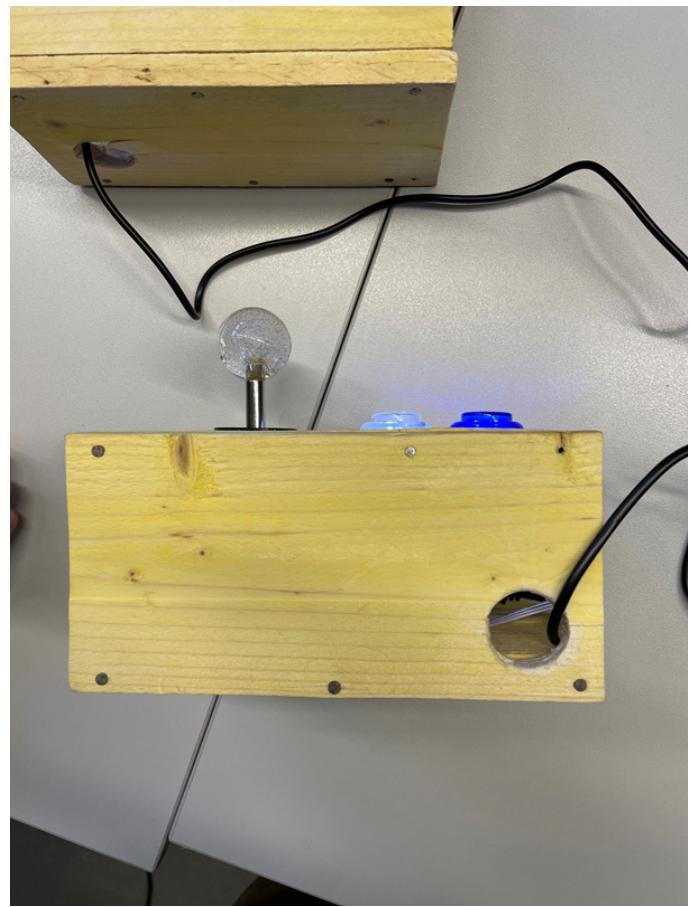
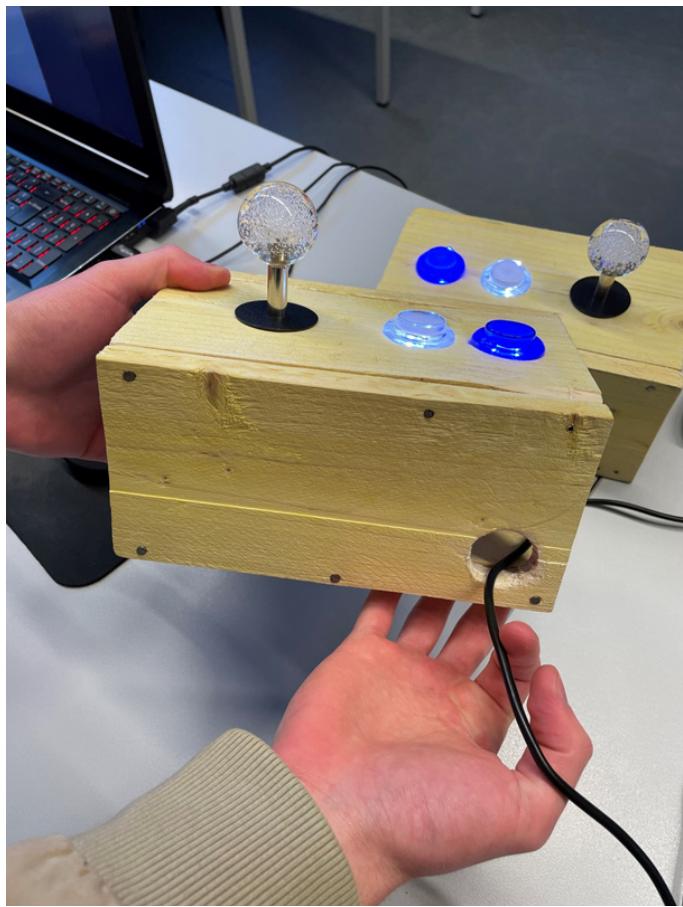
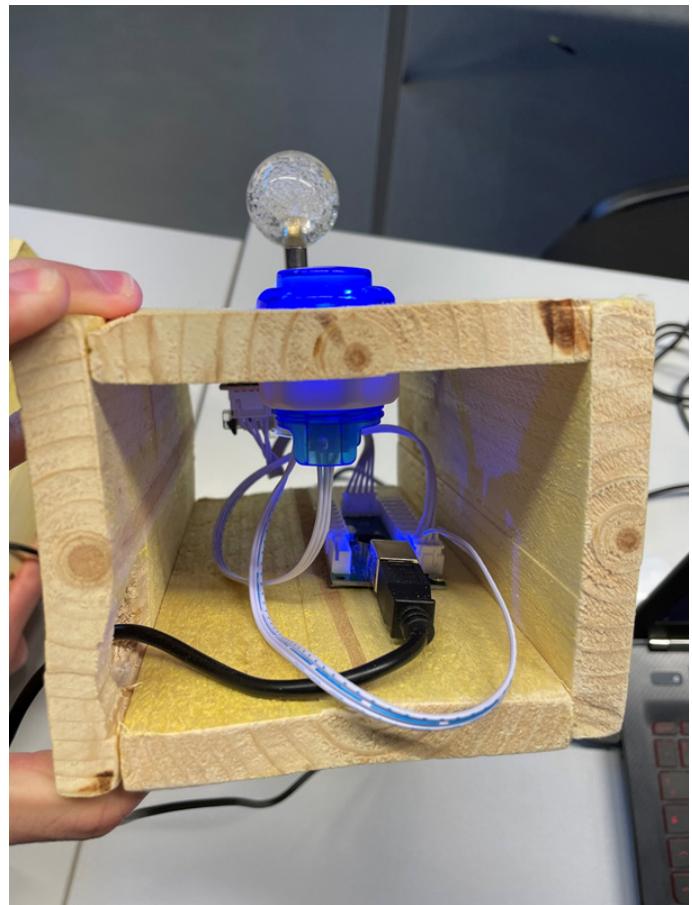
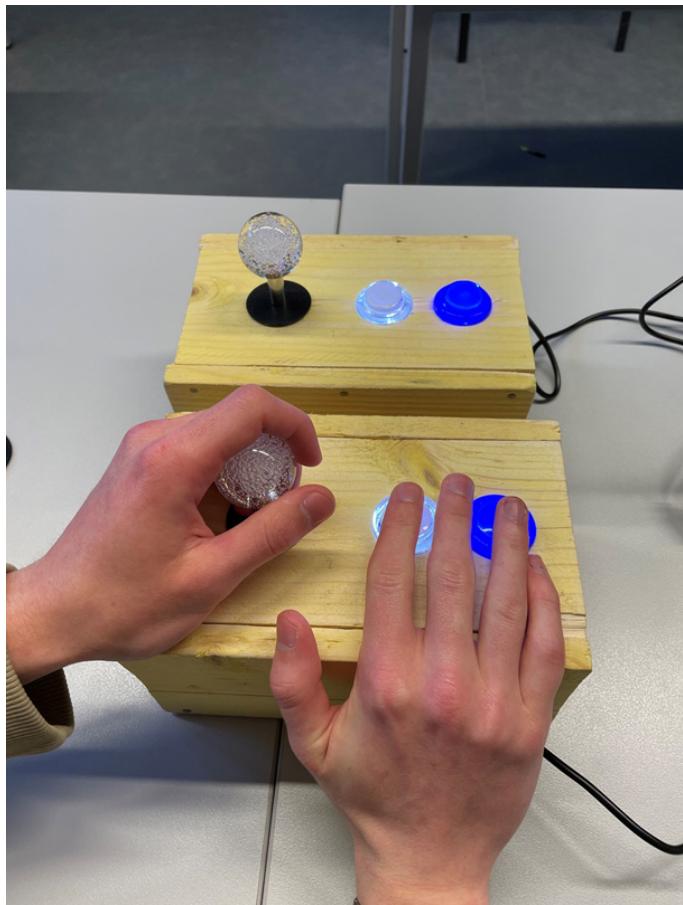


Une fois la partie terminée, le vainqueur sera affiché à l'écran, ainsi que sa position dans le classement et son score

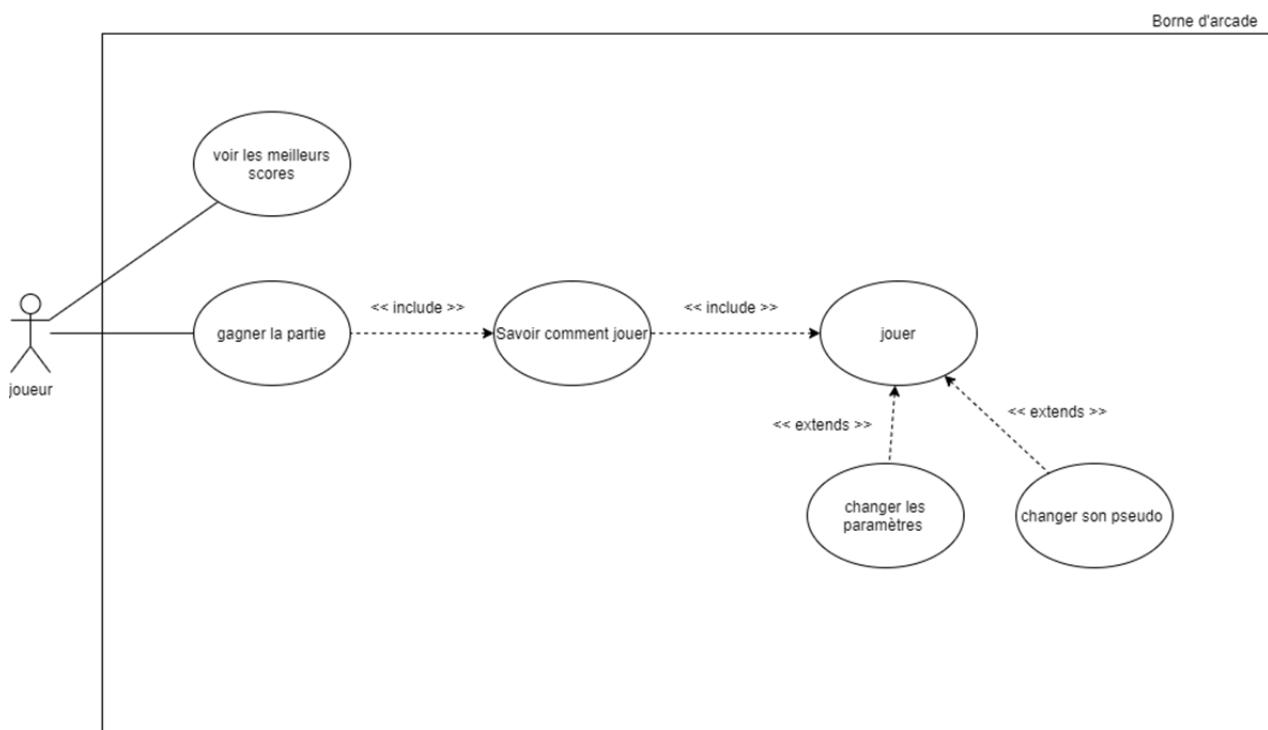
Le score sera calculé en fonction du temps de la partie et des paramètres qui ont été appliqués.



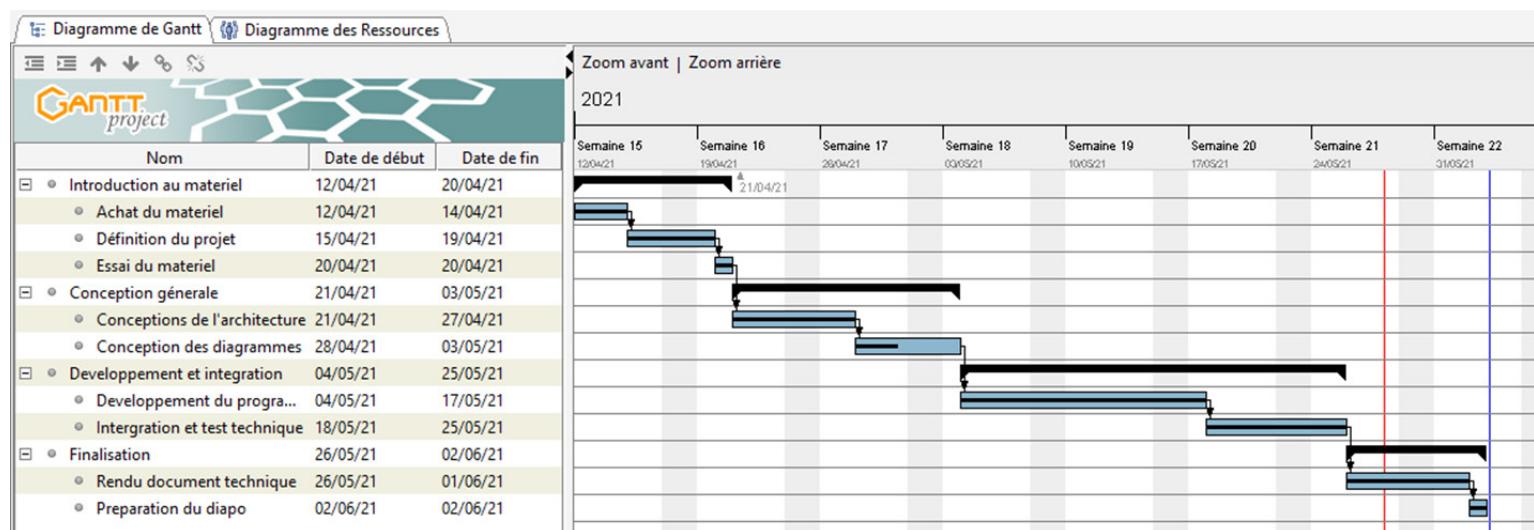
Voici les quelques photos des manettes que Thomas a fait spécialement pour le jeu :



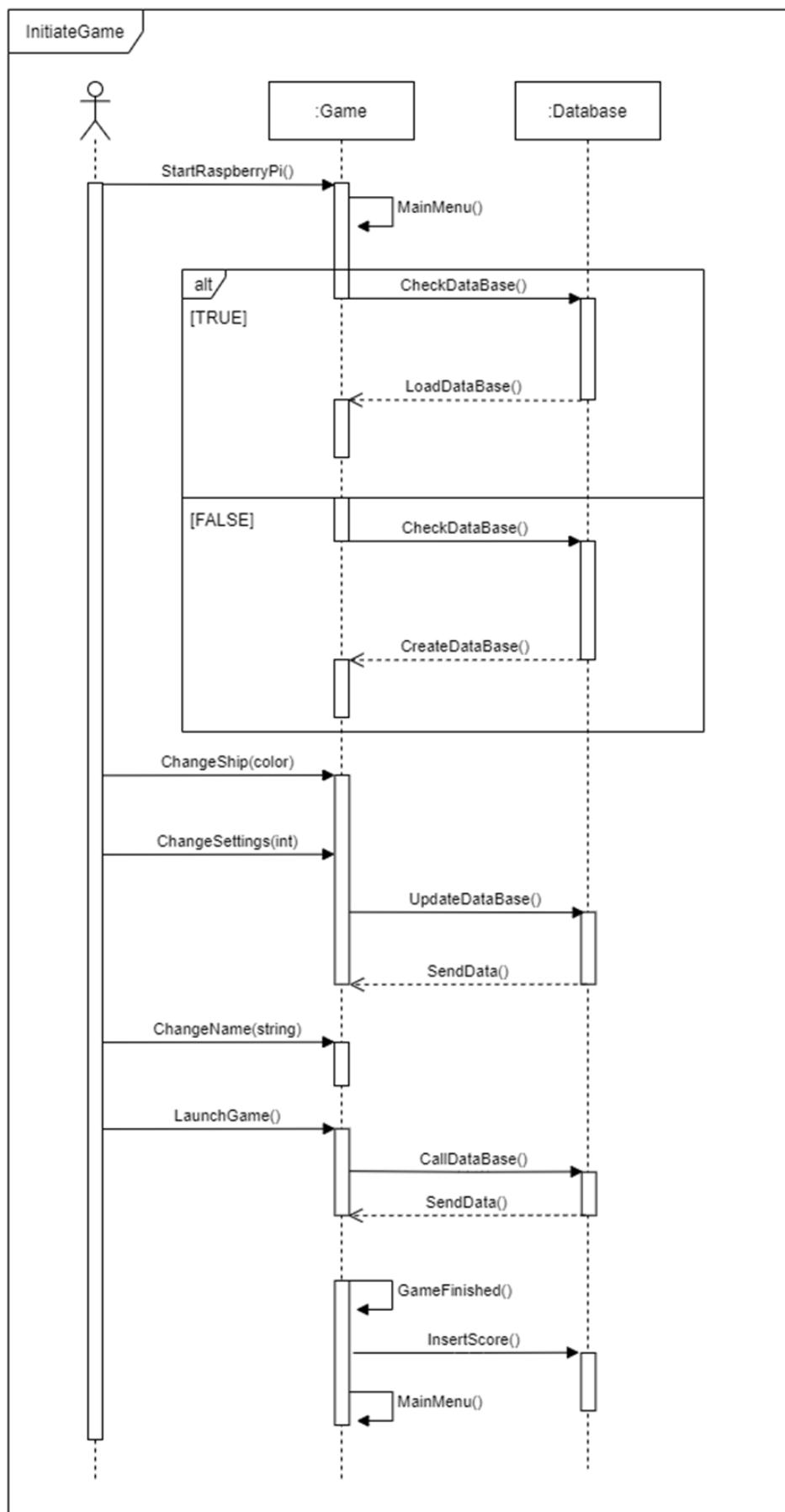
Afin de mieux analyser les **besoins de l'utilisateur**, nous avons mis au point un **diagramme usecase** qui nous a permis de mettre le doigt sur les principes fondamentaux de notre projet.



Afin de tenir les délais, nous avons donc mis au point un **diagramme de Gantt** pour structurer le projet

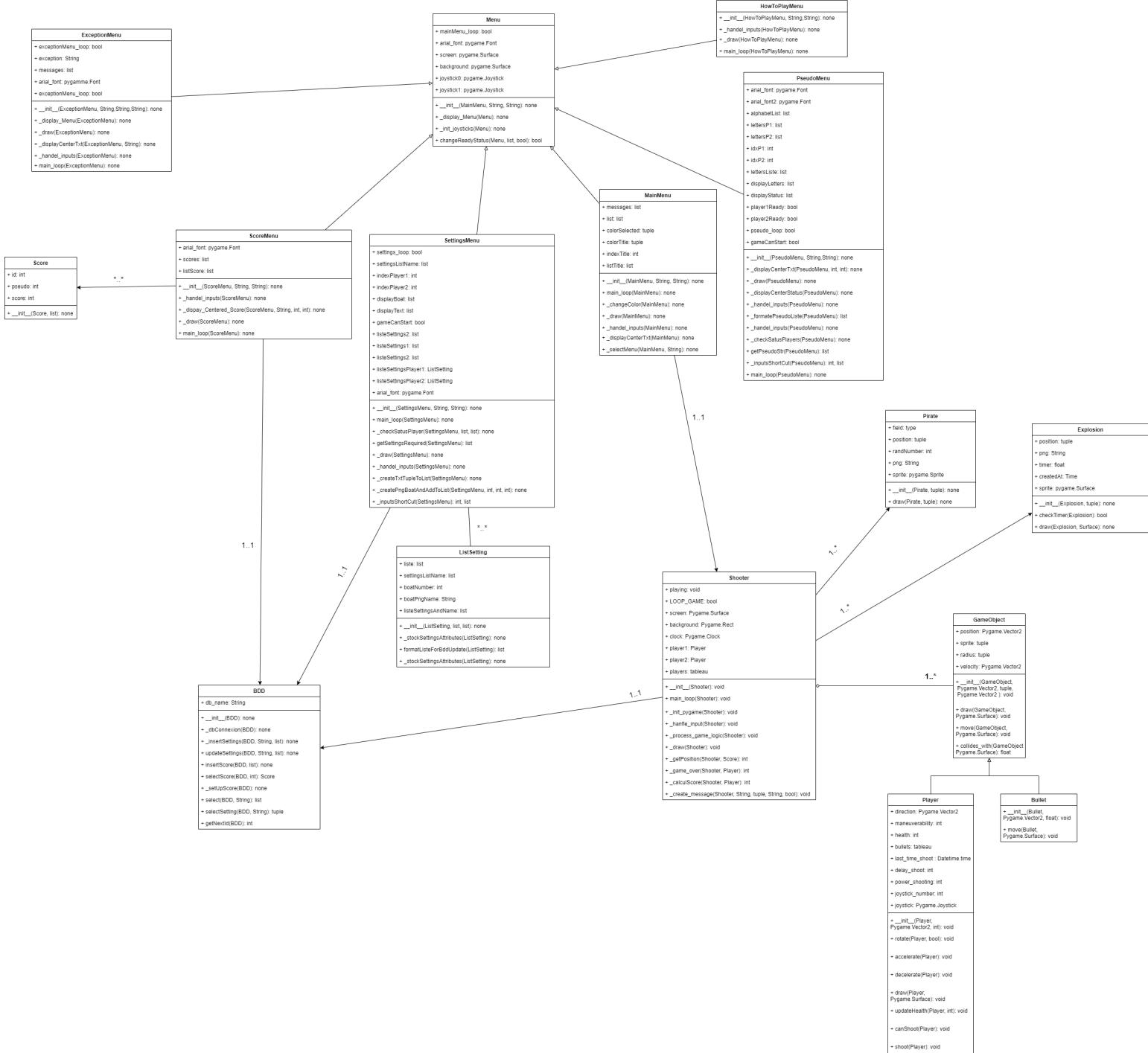


Voici le diagramme de séquence qui nous a donné une ligne directive :



IV - Architecture

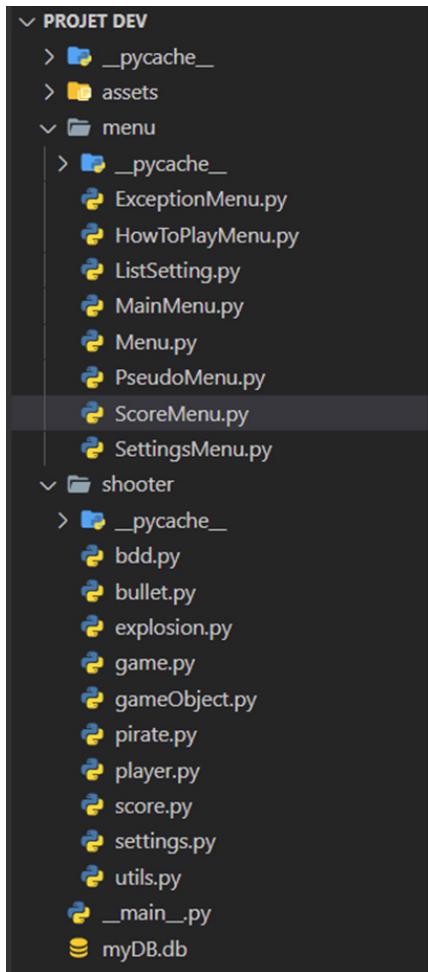
Nous avons réalisé un **diagramme de classe** pour bien structurer le code et travailler sur des parties différentes du code :



Sur ce diagramme vous pouvez voir les différentes classes de ce projet, en plus de l'organisation des classes, nous avons instauré une certaine rigueur dans l'écriture des fonctions.

En effet, vous pouvez voir que certaines fonctions sont précédées d'un « `_` », ce signe nous permet de savoir si la fonction est utilisable en dehors de la classe même ou si elle n'est utilisable qu'en interne.

Nous avons fait une **arborescence de fichier** comme celle-ci car cela nous paraissait plus cohérent :



Nous avons donc 3 fichiers :

- **Shooter** : contenant tous les fichiers du jeu
- **Menu** : contenant tous les fichiers relier au menu
- **Assets** : contenant tous les sprites

Comme vous pouvez le voir, tous les fichiers sont en Python. En effet, tout ce projet a été réalisé en python. Python a une très bonne librairie pour faire des jeux : Pygame.

Pygame nous a permis d'aller beaucoup plus vite dans la construction du jeu, de par la façon dont les fenêtres d'affichages sont générées ou encore la gestion des joysticks.

De plus python est facile à prendre en main, en quelques semaines nous avons réussi à apprêhender le langage et à créer le jeu.



V - Développement

Nous allons maintenant vous présenter quelques morceaux de code qui sont important :

```
def _createTxtTupleToList(self, colorIdx, liste, surfaceX, surfaceY):
    for idx, setting in enumerate(liste):
        if idx == colorIdx:
            color = RED
        else:
            color = WHITE
        if idx != 0:
            strTxt = str(setting[0])
            txt = self.arial_font.render(strTxt, True, color)
            txtCenter = txt.get_rect().center
            surfaceY = surfaceY + 80
            myTuple = (txt, txtCenter, (surfaceX, surfaceY))
            self.displayText.append(myTuple)
```

On se retrouve dans la fonction `_createTxtTupleToList` qui se trouve dans la classe **SettingsMenu**.

Cette fonction nous permet de créer une chaîne de caractère qui va s'afficher à l'écran à un endroit bien précis, ainsi qu'avec une couleur définie. Or il n'affiche pas la chaîne de caractères directement à l'écran, il stocke toutes les informations dans une liste.

Cette liste sera réutilisée dans la partie dédiée à l'affichage :

```
def _draw(self):
    self.screen.blit(self.background, (0, 0))

    self._createTxtTupleToList(self.indexPlayer1, self.listeSettingsPlayer1.listeSettingsAndName, SCREEN_WIDTH* 1.80/5, 300)
    self._createTxtTupleToList(self.indexPlayer2, self.listeSettingsPlayer2.listeSettingsAndName, SCREEN_WIDTH* 4.25/5, 300)
    self._createPngBoatAndAddToList(self.listeSettingsPlayer1.listeSettingsAndName[0][0], SCREEN_WIDTH * 0.75/5, SCREEN_HEIGHT/2)
    self._createPngBoatAndAddToList(self.listeSettingsPlayer2.listeSettingsAndName[0][0], SCREEN_WIDTH * 3.25/5, SCREEN_HEIGHT/2)

    for txt in self.displayText:
        self.screen.blit(txt[0], ((txt[2][0]) - txt[1][0], txt[2][1] - txt[1][1]))
```

Dans cette fonction `_draw`, tout d'abord on affiche le background et on appelle la fonction `_createTxtTupleToList`, on voit un peu plus bas qu'il y a une boucle `for` qui itère sur la liste `displayText`.

La liste `displayText` qui est rempli grâce à la fonction `_createTxtTupleToList`, on va alors grâce à cette liste pouvoir afficher le texte à l'écran avec la fonction `blit` qui provient de Pygame.



Dans chacune de nos classes, nous avons une **classe _handel_inputs** qui nous permet de gérer tous les déplacements des joueurs ainsi que les boutons pressés :

```
event in pygame.event.get():
if event.type == pygame.QUIT:
    quit()

if event.type == pygame.JOYBUTTONDOWN:
    if event.dict["button"] == 1:
        self.pseudo_loop = False
    if event.dict["button"] == 0 and event.dict["instance_id"] == 0:
        self.player1Ready = super().changeReadyStatus(self.player1Ready)
    if event.dict["button"] == 0 and event.dict["instance_id"] == 1:
        self.player2Ready = super().changeReadyStatus(self.player2Ready)

if event.type == pygame.JOYAXISMOTION:
    self.idxP1, self.lettersP1 = self._inputsShortCut(event, self.idxP1, self.lettersP1)
```

Ici nous sommes dans la classe **PseudoMenu** et on peut remarquer que nous faisons une boucle sur **pygame.event.get** qui est une fonction de Pygame afin de voir toutes les actions qui sont faites dans la partie.

Grâce à cette fonction, nous faisons des instructions sur les évènements qui nous intéresse et on appelle les fonctions adéquates.

Nous utilisons le mot **super()** qui est un appel à l'élément parent qui hérite de la classe. Ici la classe **PseudoMenu** hérite de la classe **Menu** et donc permet d'avoir la fonction **changeReadyStatus** de la classe **Menu**.

Il y a aussi une fonction **_inputsShortCut**. Cette fonction est un raccourci pour la gestion des déplacements dans le menu des pseudos.



La fonction suivante découpe en 2 grandes parties : L'axe x et l'axe y.

```
def _inputsShortCut(self, event, idxPlayer, listeLetter, instance_id):
    # handel un deplacement vers le bas
    if event.dict['axis'] == 1 and round(event.dict['value']) == 1 and event.dict['instance_id'] == instance_id:
        idxLetter = listeLetter[idxPlayer]
        if idxLetter == 0:
            idxLetter = 25
        else :
            idxLetter -= 1
        listeLetter[idxPlayer] = idxLetter
    # handel un deplacement vers le haut
    if event.dict['axis'] == 1 and round(event.dict['value']) == -1 and event.dict['instance_id'] == instance_id:
        idxLetter = listeLetter[idxPlayer]
        if idxLetter == 25:
            idxLetter = 0
        else :
            idxLetter += 1
        listeLetter[idxPlayer] = idxLetter
    # handel un deplacement vers la droite
    if event.dict['axis'] == 0 and round(event.dict['value']) == 1 and event.dict['instance_id'] == instance_id:
        if idxPlayer == 3:
            idxPlayer = 0
        else:
            idxPlayer += 1
    # handel un deplacement vers la gauche
    if event.dict['axis'] == 0 and round(event.dict['value']) == -1 and event.dict['instance_id'] == instance_id:
        if idxPlayer == 0:
            idxPlayer = 3
        else:
            idxPlayer -= 1
    return idxPlayer, listeLetter
```

L'axe x représente la droite et la gauche, pour toutes directions, on remarque qu'il y a un changement d'index. Ce changement d'index signifie la **position du joueur dans le menu du choix du pseudo**.

L'axe y quant à lui correspond à la **direction du haut et du bas**. Dans chaque «if» correspondant à la direction, on va chercher **la lettre** qui est stockée à l'**index où se trouve le joueur**, l'**index de l'axe X** puis on va chercher sur quelle lettre le joueur se situe (si c'est un **A ou un B**). Si le joueur fait un déplacement vers le haut et qu'il était sur la **lettre A** alors l'**index de la lettre du joueur va être incrémentée** et la lettre stockée **va être B**.



VI - Conclusion

Ce projet fut une belle expérience sur le **plan humain** avec la **gestion du temps**, de travail, d'écoute de chacun, mais aussi sur le **plan technique** car nous avons appris le langage **Python** et il nous sera utile pour plus tard.

Cependant, cette aventure fut semée d'embûches. Au début du projet le jeu devait être codé en **C++**. Nous l'appréciions dû aux connaissances que **notre cursus** nous a apporté. Mais après des jours de recherches pour essayer d'avoir les entrées de nos joysticks fonctionnelles et reconnues, nous avons compris **qu'aucune librairie** ne pourrait reconnaître nos manettes. Il était pour nous **impossible de le coder** car cela serait trop compliqué et nous aurions perdu beaucoup de temps. C'est pour ces raisons que nous avons choisis **Python**.

Un autre problème qui nous a pris du temps fut la **conception des classes liées aux menus**. En effet, il fallait penser à des menus qui devait être **intuitif, compréhensible et facile à utiliser** avec la configuration de manettes que nous avions. Pour cela nous avons pris plusieurs jours pour réfléchir à comment le problème pouvait être tourné. De même pour l'architecture des menus, que toutes les classes héritent de la classe **Menu** et de tout son contenu.

Nous avons aussi des **axes d'améliorations possibles**, comme rajouter un mode de **jeu solo**, des **bonus** qui arrivent aléatoirement dans la partie ou encore **différents modes de jeux** qui changeraient du 1 contre 1.

Sincèrement quand on y a joué, c'était **très amusant**. On l'a fait tester à notre famille et ils se sont bien amusés aussi. Pour nous **c'est l'essentiel**.

Merci d'avoir lu !

