AP1 – Projet 1- Pièges !

## Sommaire :

1 - Description et enjeux du projet

* 1.1 : Description du jeu
* 1.2 : Description du projet
* 1.3 : Difficultés potentielles

1. - Exploration triviale du code

* 2.1 : Structure des programmes
* 2.2 : Explication des classes

3 - Exploration avancée du code

* 3.1 : Moteur de jeu
* 3.2 : Système de scènes
* 3.3 : Affichage

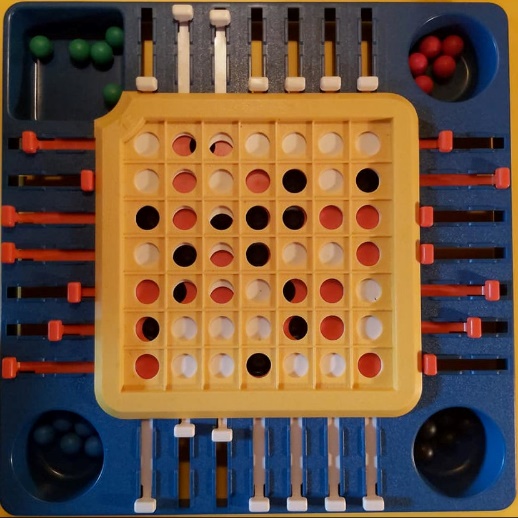
4 - Conclusion

## **1 - Description et enjeux du projet**

* 1. : Description du jeu -

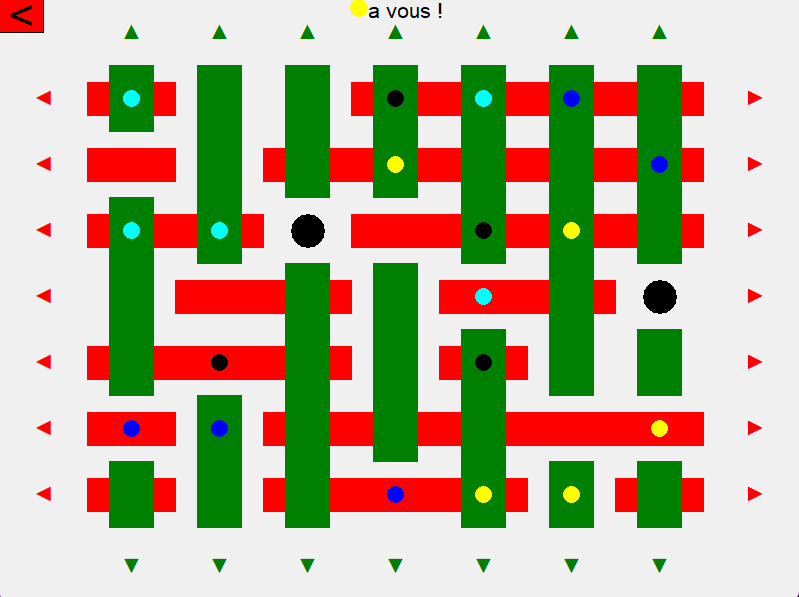
Pièges ! est un jeu de société daté de 1971, et permettant à entre 2 et 4 joueurs de s’affronter sur un plateau avec des tirettes. Ces tirettes ont des positions, et des trous dans celles-ci. Le plateau a un fond creux, et est organisé sous la forme d’un quadrillage. Les joueurs placent en début de partie des billes de différentes couleur (selon le joueur) sur le plateau, et à chaque tour, tous les joueurs peuvent bouger une tirette du plateau, de sorte à ce que les trous des tirettes fassent tomber les billes des adversaires dans le fond creux du plateau, en sachant qu’il y a deux couches de tirettes sur le plateau (des tirettes verticales et des tirettes horizontales).

L’enjeu est donc de recréer entièrement le jeu, de façon fonctionnelle en python, avec FLTK.



* 1. : Description du projet

Notre reproduction vidéoludique du jeu ressemble à cela.



Au centre, nous reconnaissons bien le terrain de jeu, avec les tirettes, leurs trous, et les billes des joueurs. Sur cette image, nous pouvons voir 4 joueurs (le bleu foncé, le bleu clair, le noir et le jaune)

En haut, le tour du prochain joueur est indiqué, sachant qu’a chaque tour, le joueur peut seulement changer la position des tirettes, grâce aux flèches tout autour du plateau (les triangles bleus et rouges autour du plateau)

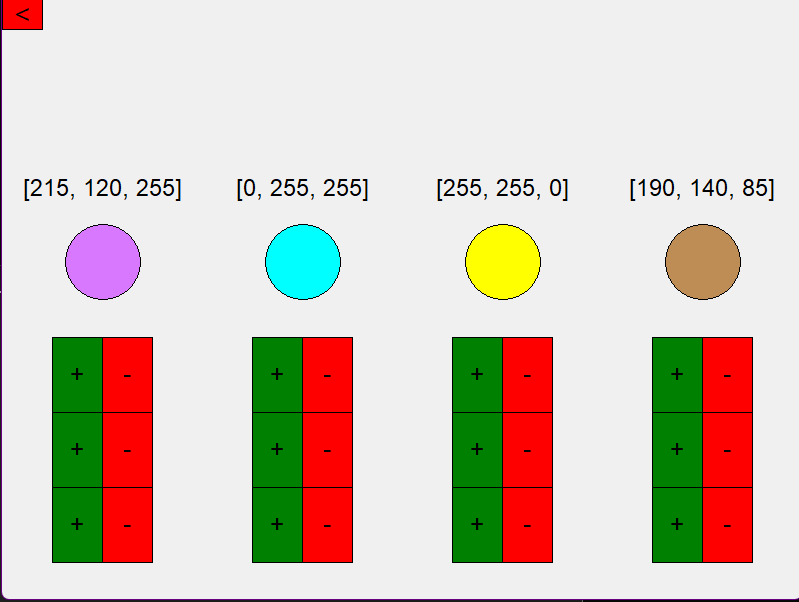
Les endroits ou deux trous sont présents sont représentés par des gros cercles noirs. Les trous comme cela sont présents lorsque deux trous de tirettes se superposent.

Le jeu dispose d’un menu, qui s’ouvre quand le programme est lancé



Les boutons permettent de changer la couleur des joueurs (donc de leur billes) et de jouer au jeu, mais aussi d’accéder aux paramètres

Ce menu permet de changer le nombre de joueurs, et de changer les dimensions de la fenêtre.



* 1. : Difficultés potentielles -

Sur ce projet, l’objectif est de permettre l’affichage d’une grille de jeu, et de prendre en compte les différentes positions des tirettes afin de déterminer si une bille en un point est sur deux trous de tirettes

Le programme du jeu doit donc déterminer pour chaque bille la position des deux tirettes dont la position de la bille est l’intersection, et regarder si les trous de ces deux tirettes sont à cette position. Si oui, alors la bille est supprimée.

Pour ce qui est des structures de données, la grille est représentée sous forme d’une liste de liste, mais le challenge est donc d’implémenter la gestion des tirettes, car celles-ci ne peuvent pas être mises dans une liste de liste de la taille du plateau.

Pour ce qui est de l’affichage, les difficultés rencontrées peuvent être l’affichage dynamique des tirettes, qui changent de position à chaque fois qu’un bouton est pressé. Le jeu doit donc afficher la nouvelle position de la tirette modifiée, en prenant en compte si la tirette modifiée crée (ou bouche) un nouveau trou, et si oui en l’affichant, pour pouvoir avoir une indication claire au joueur.

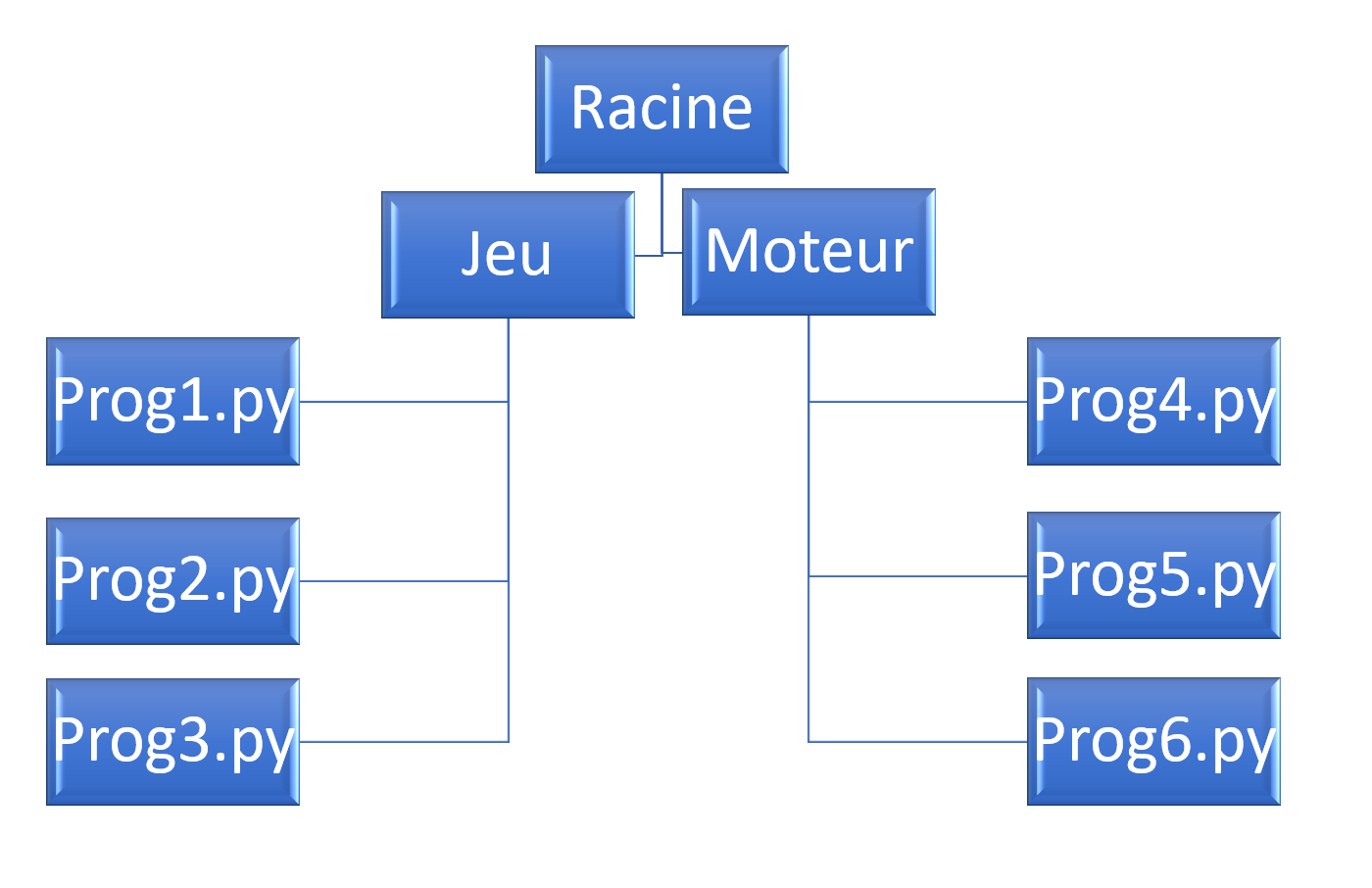
## **2 - Exploration triviale du code**

* 1. : Structure du programme

Le programme est entièrement codé en Python. Le code est organisé de la façon suivante :

Le code est divisé en deux parties distinctes, une s’occupant de gérer le fonctionnement du jeu (jeu), et l’autre de l’afficher (moteur). Le programme principal s’occupe de relier tous les sous programmes du jeu, donc les programmes du jeu et du moteur. FLTK, le module permettant de tout afficher, est inclus dans le jeu.

Une arborescence du projet pertinente serait la suivante :



* 1. : Explication des classes

Dans ce projet, nous avons choisi d’utiliser dans la majorité la programmation orientée objet, c’est-à-dire la création de nouvelles classes dans python, avec des attributs (des variables propres à une instance de cette classe) et des méthodes (des fonctions applicables à des instances de cette classe et permettant par exemple de modifier les attributs de cette classe (on peut penser par exemple à la méthode .append() de la classe list).

Si nous parlons de cela dans ce rapport, c’est parce que la programmation orientée objet est au centre de notre projet. Elle permet de faciliter la gestion de variables, par exemple.

Un exemple concret de notre utilisation des classes est la classe Tile.

Cette classe permet de décrire le comportement d’une « tuile » du plateau. Les différentes instances de classe tuile sont contenues dans la liste de liste décrivant la grille (qui est elle-même contenue dans une classe, on le verra plus tard). Chaque instance (ou objet) de tuile contient l’information de s’il y a un trou à cet emplacement, dans des attributs (l’un pour la tirette verticale à cette emplacement et un autre pour la tirette horizontale) et contient également possiblement un instance de classe Ball (bille) qui est une classe décrivant les billes du jeu. Dans la classe, il y a une méthode (donc applicable à toutes les instances de cette classe) qui va analyser s’il y a un trou dans la tuile, et s’il y a un trou et une bille au même endroit, qui va « tuer » la bille, c’est-à-dire la faire disparaître du jeu.

Sans programmation orientée objet, la solution aurait été de stocker dans la liste de liste une liste contenant les informations de s’il y a un trou (avec un booléen par exemple) et les informations d’une bille (s’il y a une bille, et si oui, son propriétaire), non contenue dans une instance de classe

Cela rendrait le code beaucoup plus lourd, et donc inutilement plus complexe.

## **- Exploration avancée du code**

3.1  : moteur de jeu

Le moteur de jeu est la partie principale du projet, en effet c’est dans cette partie que nous obtenons le système de tirettes, de grilles, et de balles. C’est également la partie la plus simple car elle ne requiert aucune intéraction avec la fenêtre graphique. Ce que tout les autres méchanisme auront. Dans cette partie on y retrouve la classe gérant la grille, les balles, les tirettes, et les cases

Description simple de chaque classes :

Classe grille :

Une classe qui s’occupe de la grille, elle permettra de placer une balle a une case x,y de modifier les attributs de la case x,y, mais également de stocker en mémoire la grille de jeu principal (ici en 7\*7). Ainsi on pourra utiliser chaque autres classes pour affecter la grille

Classe balle :

Une classe simple qui permet de gérer une bille d’un jouer (représenté sous forme d’entier dans le code)  
  
on pourra par exemple initialiser une balle du joueur 1  
  
cette classe permet de déterminer et de changer l’état d’une balle, tombé ou non

Classe tirette :

Cette classe va directement interagir avec la grille pour pouvoir pousser et modifier les trous. Tout d’abord en faisant le décalage intérieurement et ensuite en l’affectant à la grille, ce qui fera que la grille actualisera ce qui se passe a chaque case (i, y) ou (x, i) de la grille, ou i représente l’index a laquelle la tirette est présente et x,y les variables sur lesquels nous itérerons

Classe tile (ou case) :

Cette classe contient plusieurs info, la balle actuelle présente à la case, si il y a un trou horizontal, et s’il y a un trou vertical. On pourra a partir de la classe grille affecter chaque cases une par une pour y mettre les informations nécessaires

Nous avons fait une brève exploration de la partie non graphique du jeu. Penchons nous maintenant sur la partie graphique et le système de scène qui est un peu plus complexe

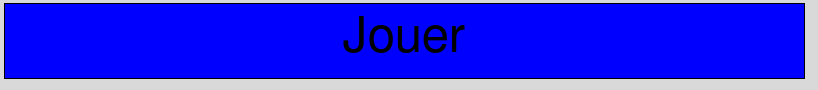
3.2 : système de scène

Tout comme un film, nous pouvons décomposer le projet en différentes scène.  
  
Exemple :

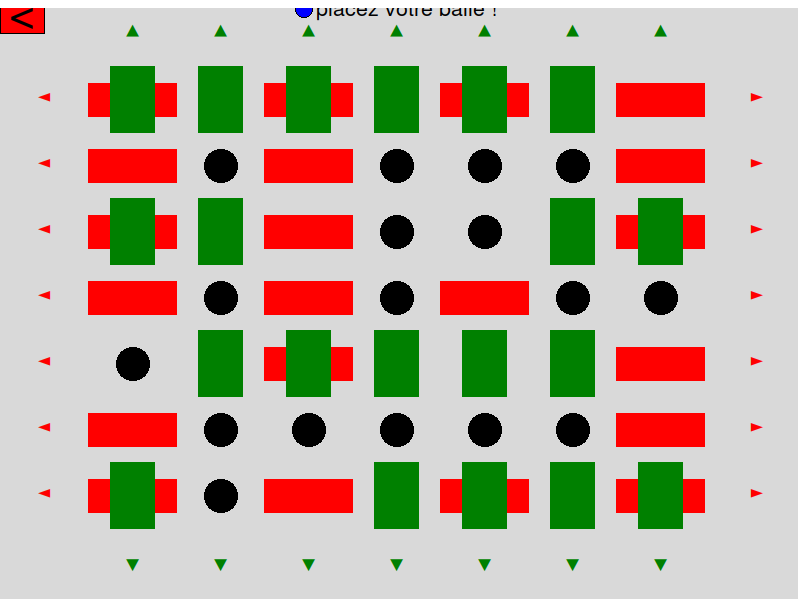
Supposons notre 1ère scène comme étant la scène de menu



Lorsque nous appuyons sur le bouton jouer par exemple, nous changerons de scène pour passer sur la scène de jeu voici un exemple dans le graphique ci dessous



La scène actuelle devient alors la scène de jeu



En mémoire on représente cela depuis une classe nommé Renderer.

Cette classe possède un attribut scene, qui est modifié lorsque la scène actuelle propose un changement de scène

On supposera que la nouvelle scene possède les méthodes update() et draw() dans lesquels nous pourrons mettre à jour les évènement qui se produisent. Et aussi de s’occuper du déroulement de la scène. Par exemple dans notre scène jeu nous nous occuperons de tout ce qui se produit lorsque l’utilisateur fait un clic en fonction de l’état actuel de la scène et de l’endroit du clic

Dans notre programme, le jeu est munis de 4 scène différentes  
  
Scène de menu  
  
Scène de paramètres  
  
Scène de changement de couleurs des balles

Scène de jeu

La scène de menu permet de contrôler le passage entre chaque scènes a l’aide de boutons. C’est dans cette scène que nous pourrons la commande pour revenir au menu afin d’éviter un import dit circulaire. par exemple depuis le menu nous importons la scène de jeu pour pouvoir donc l’appliquer lorsque l’on appuie sur le bouton, et comme nous devons faire un bouton pour revenir au menu, si l’on importe menu depuis le fichier de la scène de jeu nous aurons un import dit circulaire qui va coincer le programme.

Ainsi chaque scène sera dépendante de la scène de menu