KOURBANHOUSSEN Idriss – M1 RSA

PROGRAMMATION AVANCEE : Rapport de projet

I. Introduction

Ce projet a pour but de développé un jeu **d'Escrime** avec un langage de programmation orienté objet. J'ai utilisé pour ma part le langage **Python** afin de développer ce jeu. L'Escrime est un sport de combat qui consiste à toucher un adversaire avec la pointe ou le tranchant (estoc et taille) d'une arme blanche sur les parties valables sans être touché. Ce programme permet de faire jouer deux joueurs locaux à un jeu avec un affichage dans le Terminal, les deux joueurs peuvent réaliser plusieurs types de mouvements (déplacements, attaquer, bloquer) à partir d'un seul clavier. Le jeu se déroule dans une scène qui peut être modifié et choisi parmi plusieurs (à travers des fichiers de type x.ffscene), ces scènes comportent les emplacements des joueurs 1 et 2 sur la scène ainsi que l'emplacement des différents obstacles. Je vais donc dans ce rapport expliquer globalement les différentes fonctionnalités qui ont été implémentés dans mon programme.

II. Implémentation similaire aux deux versions

Toutes les fonctionnalités « required » ont été implémenté et marche, ainsi que toutes les fonctionnalités de la partie « improvments ».

```
C:\Users\ikour\Desktop\prog_av\Projet\projet>python3 main.py

Bienvenue sur Jeux d'Escrime !

Voulez-vous utiliser jouer sur la version Terminal ou Graphique (T/G)? T
Combiens d'images par secondes voulez-vous (20 conseillé)? 20
Voulez-vous reprendre la Partie enregistrée (Y/N)? N
0 : ____1_____________________x_____________________x_____________________2_____
1 : ____1_____________________x_____________________________________________2_____
2 : ____1_____________________x_____________x_____x_____x_____________2_____

Choisissez le numéro de la scène : 3
Choisissez à nouveaux : 1
```

J'ai réalisé deux versions de ce jeu qui sont la **version terminale** ainsi qu'une **version graphique** qui peuvent tous les deux être lancé via un seul fichier main.py. À l'intérieur de ce fichier, je demande tout d'abord les différentes préférences de l'utilisateur comme la version sur laquelle il veut jouer le jeu, le nombre de rafraichissements par seconde ou encore s'il veut modifier les différents attributs des joueurs. Je pourrais donc, avec

ces informations, modifier les paramètres du jeu et des personnages en fonction de ces attentes et lancer la version souhaitée.

Pour afficher et donner le choix à l'utilisateur de choisir entre les **différentes scènes**, j'ai lu le tous les fichiers présents dans le dossier scènes et je les affiche à l'écran en numérotant les scènes. Je demande ensuite à l'utilisateur choisir une de ces scènes en précisant le numéro en question, je peux grâce à ce numéro enregistrer la scène dans la variable globale « stage » ce qui permettra de savoir plus tard l'emplacement des joueurs et des obstacles.

Pour éviter d'avoir des dizaines de variables globales différentes dans mon programme, j'ai préféré créer deux classes **joueur_1** et **joueur_2** qui héritent tous les deux d'une classe joueur. Cette dernière contient tous les attributs des joueurs tels que leurs vitesses de déplacements, leurs vitesses d'attaques, ces attributs qui sont mis à jour à la création des joueurs ne serons plus modifiés, mais cette classe contient aussi d'autres attributs. En effet, elle contient le score du joueur, ses coordonnées ou encore son état courant (attack, block ou rest), ces attributs changerons tout au long du Jeu et des actions réalisées par l'utilisateur. La seule différence entre les classes des 2 différentes versions et que la version graphique contient un attribut en plus qui est l'image actuelle du joueur.

Attention : dans la version terminale avec « **curses** », les coordonnées sont de type (y, x) tandis que dans la version graphique avec « **tkinter** » elles sont de type (x, y).

J'ai aussi réalisé une fonctionnalité qui permet d'enregistrer une partie jouée par un utilisateur et pouvoir la rejouer plus tard. Pour sauvegarder la partie, j'ai enregistré l'ensemble des données nécessaires pour que je puisse recréer la partie comme l'état de la scène avec l'emplacement des joueurs 1 et 2 ainsi que les emplacements des obstacles. J'ai en plus enregistré les attributs des joueurs comme leurs vitesses de déplacements, la vitesse d'attaque, en plus des scores bien sûr. Toutes ces données me permettent de pouvoir mettre à jour les valeurs des attributs de joueurs dans leurs classes respectives. À chaque fois qu'un utilisateur veut enregistrer sa partie, j'écrase les données qui sont présentes dans le fichier « **partie_save.txt** » et réécrit les nouvelles données listées précédemment.

Au niveau des rafraichissements, j'ai utilisé des threads, ce qui me permettent de réaliser des actions sans interrompre ou mettre en pause le fil d'exécution. Tout au début, lors de l'initialisation des fenêtres, j'ai appelé la fonction thread « **frames refresh terminal** » qui contient une boucle while qui tourne en continu. Dans cette boucle, j'attends « x = **1/number_refresh** » secondes avant de rafraîchir toutes les fenêtres (la

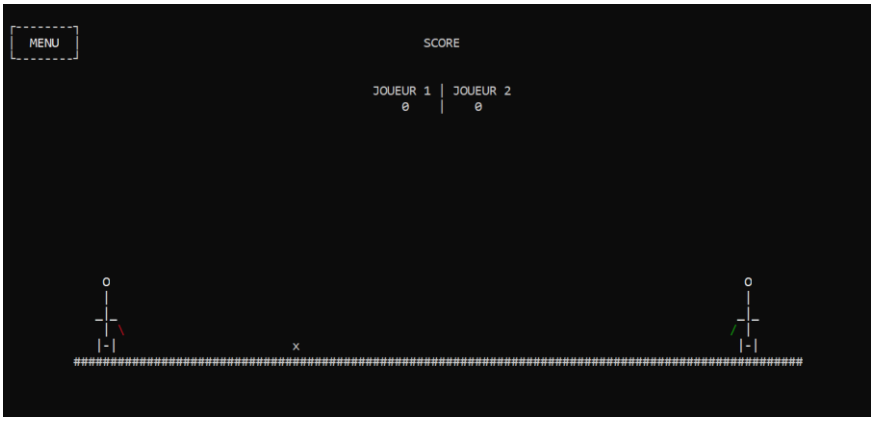
fonction refresh pour la version terminale). De ce fait, le programme affichera la nouvelle position du joueur que tous les x secondes. De plus, à chaque fois qu'il y a un rafraichissement, je mets une variable globale rafraichissement à True ce qui permet à l'ensemble des autres fonctions mon programme de savoir précisément ce qui se passe. Cette variable rafraichissement me permet aussi de calculer mouvement des joueurs. En effet, si par exemple le joueur à une vitesse de 2, donc je vais attendre que la variable globale passe 2 fois à True avant de déplacer mon joueur.

Pour le mouvement **d'attaque**, je mets d'abord mon épée à la position d'attaque puis je rentre dans un thread qui attend que la variable rafraichissement passe attacking_speed fois à True avant d'attaquer. J'ai mis cette attente dans un thread pour mon programme puisse continuer à récupérer les déplacements de l'autre joueur et ainsi esquiver l'attaque. J'ai mis en place un système de verrou sur cette variable afin toutes les fonctions n'accède pas à cette dernière en même temps, ce qui pourrait poser des problèmes.

Pour le mouvement de **bloc**, je mets d'abord mon épée à la position d'attaque puis je rentre dans un thread qui attend que la variable rafraichissement passe bloking_time fois à True avant de remettre le joueur en état rest par défaut.

Pour les sauts voir dans la partie III.

III. Version Terminal



Pour réaliser l'affichage de l'interface utilisateur de la version terminale, j'ai utilisé la bibliothèque « **curses** ». La bibliothèque curses propose des facilités pour la lecture du clavier et l'affichage écran sur les terminaux texte telle que les consoles linux. J'ai créé au sein plusieurs sous fenêtre au sein de ma fenêtre principale afin de séparer leurs affichages séparément. Nous avons la première fenêtre **menue** qui permet de mettre en pause la partie, la fenêtre **score** qui affiche les scores de joueurs et enfin la fenêtre **scène** qui affiche bien entendu la scène et les joueurs qui se déplacent.

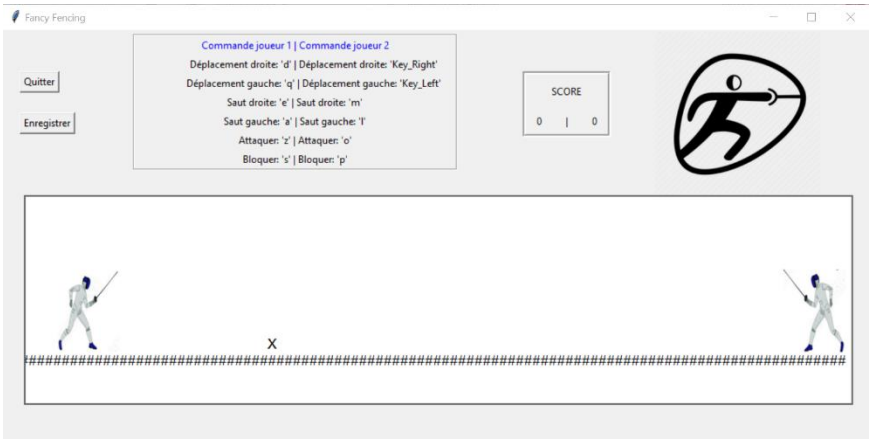
J’ai mis en place une boucle while qui attend continuellement les touches claviers de l’utilisateur ainsi que les cliques de la souris. Je regarde donc à chaque boucle le type de touche qui a été tapé comme la touche « d » qui permet de déplacer le joueur 1 vers la droite. J’appelle donc à ce moment-là la fonction « **mouvement_joueur_terminal** » qui va faire l’action.

Pour les identifier l’emplacement des joueurs, j’ai enregistré dans l’attribut **coords_joueur** les coordonnées (y, x) de la tête du joueur, ce qui me permet d’afficher le reste du corps ainsi que l’épée. À chaque fois que je veux déplacer le joueur, je mets à jour les nouvelles coordonnées, je supprime tout ce qu’il y a dans la fenêtre et j’affiche les joueurs par rapport aux coordonnées spécifié.

Pour les **sauts**, c'est assez simple, j’ai juste mis à jour les coordonnées du joueur (y, x) à chaque mouvement du joueur. Tout d’abords le joueur monte => (y-1, x), du coup tout le personnage est décalé vers le haut par rapport à son ancien emplacement, puis le joueur avance (y, x+2) pour le saut à droite ou recule (y, x-2) pour le saut à gauche, et enfin, on le remet au niveau de la scène (y+1, x). Entre chaque changement de coordonnées, si par exemple le joueur à une vitesse de 2, donc je vais attendre que la variable globale passe 2 fois à True avant de déplacer mon joueur.

Pour le **menu pause**, j’ai enregistré les coordonnées de début et de fin de ma phrase « Menu » et donc quand dans ma boucle while le programme remarque une clique de la souris, je regarde si les coordonnées de la souris correspondent à celle où est placée ma fenêtre menue. Si c’est le cas, je vais dans une autre fonction, ce qui me permet de ne plus lire les touches claviers pendant un temps. Dans la fonction **affiche_menu_terminal**, j’agrandis la fenêtre menue et j’affiche les commandes des joueurs ainsi que les mots "enregistrer", "quitter" et "reprendre". J’attends ensuite le clic de la souris, si le clic correspond aux coordonnées où j’ai écrit enregistrer, je vais dans la fonction qui enregistre la partie courante dans un fichier comme expliqué précédemment. Si l’utilisateur clique sur "quitter", je sors de la fonction et je retourne le string « quitter » ce qui me permettra de sortir de la boucle while et arrêter curses. Et enfin si l’utilisateur appuie sur "reprendre", je sors de la fonction et je retourne le string « reprendre » et je reprends ma boucle while pour continuer le jeu. Quand je sors de la fonction, je remets ma fenêtre menue à son état d’origine.

IV. Version Graphique



Pour réaliser l'affichage de l'interface utilisateur de la version graphique, j'ai utilisé la bibliothèque « **tkinter** » qui est une bibliothèque graphique libre d'origine pour le langage Python, permettant la création d'interfaces graphiques. J'ai créé sur mon interface des boutons qui permettent **d'enregistrer** et de **quitter** la partie. Une fenêtre pour les **commandes utilisateur** et une autre pour afficher le **score**. Et enfin la fenêtre **scène** qui est en réalité un **Canvas** qui affiche bien entendu la scène et les joueurs qui se déplacent.

Contrairement à la version terminale, puisque l'interface est beaucoup plus grande, **chaque case va de 10 en 10**. Par exemple, si le personnage est à la position x, je dois faire x+10 pour la déplacer d'une case vers la droite. La deuxième différence est que contrairement à la version terminale, j'utilise des images pour représenter les joueurs et non des caractères (les images utilisées étaient en libre de droit téléchargé sur pixabay). Les coordonnées (x, y) de la position des joueurs dans l'attribut correspondant est le coin gauche de l'image.

La troisième différence par rapport à la version terminale est qu'il n'y a **plus la présence d'une boucle while**. En effet, sur tkinter la fonction **bind** qui s'enclenche automatiquement quand une touche est tapée, et la touche est envoyée à une fonction que j'ai définie. Dans cette dernière, les déplacements se font directement dès que les coordonnées sont modifiées. De ce fait, entre chaque changement de coordonnées, si par exemple le joueur à une vitesse de 2, donc je vais attendre que la variable globale passe 2 fois à True avant de déplacer mon joueur.

Pour le bouton pour quitter le jeu, avec tkinter c'est beaucoup plus simple. Nous pouvons directement demander à la bibliothèque de fermer la fenêtre et en conséquence d'arrêter le Jeu.

Pour l'affichage des joueurs, au lieu de modifier l'orientation de l'épée, je modifie l'image en fonction de l'État dans lequel le joueur se trouve « block », « attack » ou « rest ».