Projet 2 NSI:

Tower Defense

Année 2020-2021

1 Introduction

Le but de ce TP est de faire un jeu basique de type "Tower defense". Il s'agit d'un type de jeu ou l'on construit des tours pour arrêter des créatures qui essayent de traverser une zone.

2 Évaluation

Le rendu attendu est le jeu ainsi qu'un rapport court décrivant les difficultées rencontrées et les fonctionnalités implémentées.
Le code sera évalué sur les critères suivants:

- Lisibilité
- Présence de commentaires
- Réponds aux attentes du projet

3 Mise en place

- Récuperer l'archive "jeu" dans les documents en consultation.
- Lancer un éditeur de code.

Figure 1: "Jeu" de départ

4 Un début de jeu.

Dans cette partie, nous allons continuer à nous familiariser avec le fonctionnement de Pygame Zero, soit:

- L'utilisation des fonctions draw() et update() qui sont exécutés a chaque fois qu'une image est affichée.
- Le fonctionnement des acteurs que l'on défini avec Actor('nom_image') qui permettent de définir l'image utilisée mais aussi les coordonnées et la fonction qui affiche le monstre.
- Tester la fonction on_mouse_down() qui détecte les clics de souris.
- Découvrir l'utilisation permettant d'afficher des images et du texte a l'écran:
  - screen.blit('image', (coord1,coord2)) pour afficher des images.
  - screen.draw.text('mon_texte',(coord1,coord2)) pour afficher du texte.

Les objectifs de cette partie sont les suivants:

- Utiliser l'acteur pour afficher la position du monstre. Initialiser les coordonnées du monstre au centre du chemin a gauche.

- Afficher l'image de fond sous le monstre et l'image interface au dessus (utilisez les coordonnées 0,0 pour ces images).
- Utiliser la fonction update pour déplacer le monstre vers la droite (en modifiant ses coordonnées).
- Faire en sorte qu'un clique de la souris réinitialise la position du monstre sur la gauche de l'écran.

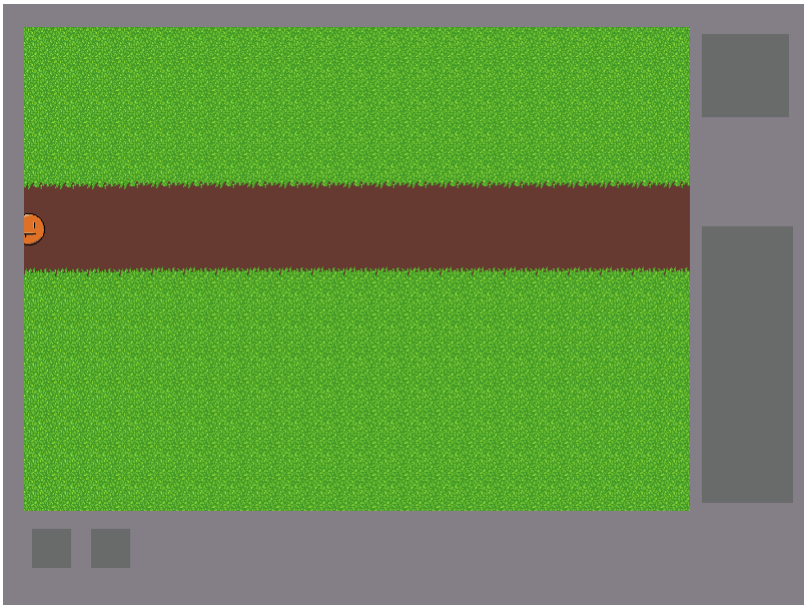

Figure 2: Une capture d'écran du jeu à ce stade.

5 Un de chaque

Nous allons maintenant modifier notre programme pour avoir les mécaniques de base du jeu.
Nous allons mettre en place les fonctionnalités suivantes:

- Un compteur de vie.
- Une tour.
- Une balle tirée par la tour.

Les attendus sont les suivants:

- Donner au joueur des vies (2 par exemple). Les afficher dans le cadre en haut a droite.

- Quand le monstre arrive à droite de l'écran, le joueur perds une vie et le monstre est placé a droite de l'écran.
- Si le monstre atteint la droite de l'écran alors que le nombre de vie est à 0: Le jeu s'arrête et on affiche "PERDU" au centre de l'écran.
- Creer une tour (un acteur avec l'image canon') qui est par défaut en haut a gauche de l'écran.
- Mettre à jour la fonction de clique pour qu'elle ne réinitialise plus la position du monstre mais que la tour soit déplacée à la position du curseur au moment du clic.
  - Pour cela il faudra rajouter un argument a la fonction utilisée précédemment: on_mouse_down(pos)
  - l'argument pos contiendra les coordonnées de votre souris sous forme d'un tuple (x,y). Vous pouvez récupérer les composantes x et y par exemple en faisant `x,y = pos`

Une fois ces fonctionnalités implémentées, il ne reste plus qu'a définir le fonctionnement des balles (un acteur avec l'image 'balle'):

- Pour cette première version, les balles apparaissent au niveau du canon et avance vers la droite (plus vite que le monstre).
- Si une balle atteint le bord droit de l'écran, elle réapparaît à la position du canon.

Vous pouvez tester la collision entre 2 acteurs avec la fonction actor1.colliderect(actor2)
Il faut réinitialiser la position de la balle et du monstre si la balle touche le monstre.
Ainsi on aura testé la collision, et "tué" le monstre ainsi que "détruit" la balle si la balle le touche.

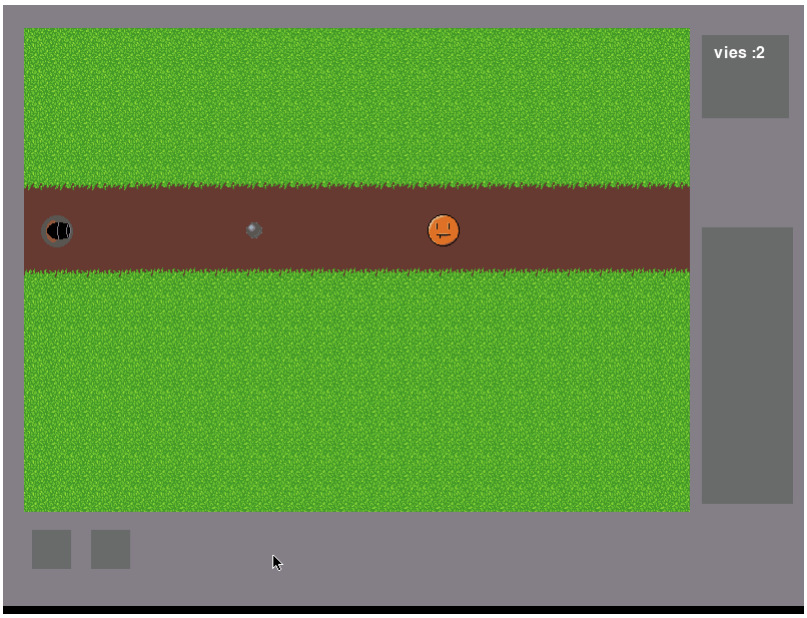

Figure 3: L'avancement actuel du projet

6 Une direction pour les projectiles !

Voici une fonction qui prends en paramètre une source et une destination (un acteur pygame zero) ainsi qu'une vitesse. Et renvoie la vitesse sur chaque coordonnée sous forme d'un tuple (x,y).

```
def vitesse_balle(depuis, vers, vitesse):
    """Cette fonction prends en entrée une position de depart "
    depuis" et d'arrivée "vers" qui sont des tuples (x,y) ainsi qu'
    une vitesse et renvoie un tuple de vitesse (x,y)"""
    vers_x = vers.x - depuis.x
    vers_y = vers.y - depuis.y
    distance = math.sqrt(vers_x**2 + vers_y**2)  # On calcule l'
    hypotenuse
    vers_x = vers_x/distance  # On divise tous les cotés du
    triangle par l'hypotenuse
    vers_y = vers_y/distance  # pour avoir le pas en x et y d'une
    hypotenuse de longueur 1.
    return vers_x*vitesse, vers_y*vitesse  # On multiplie par la
    vitesse
```

- Intégrez la fonction à votre code pour définir la vitesse de la balle à sa création.
- Modifiez votre code pour que la balle retourne à ça position initiale si elle sort du jeu (si elle touche un des 4 bords) OU si elle touche la créature .

7 Des listes ? Des listes !

L'objectif de cette partie est de permettre la création de plusieurs entitées de chaque type.
Nous allons commencer par les créatures, après tout, pour un tower defense il faut plusieurs ennemis.
Plutot que d'avoir un seul actor('creature') nous allons créer un tableau et créer plusieurs actor('creature') puis les mettres dans celui-ci.
Ensuite, il faudra modifier dans votre code toutes les opérations avec des acteurs de type 'creature' par une boucle dans laquelle on fera cette opération sur chaque créature.
Par exemple si on avait le code suivant pour créer la créature:

```
creature = Actor('creature')
```

Et le code suivant pour déplacer la créature:

```
creature.x += 2
```

Les nouveaux codes seraient:

```
creatures = [Actor('creature'), Actor('creature')] # (Si on voulait
     avoir 2 créatures, le code peut être modifier pour un nombre
    différent de créatures.)
```

Et:

```
for c in creatures:
    c.x += 2
```

Une idée d'adaptation du code pour les tours serait de créer un tableau vide au départ, et que chaque clique puisse créer une tour. Une fois le tableau plein, le clique pourrait soit ne plus rien faire, soit modifier la position d'une des tours (la plus vieille ? dans l'ordre ? Au hasard ?).
Il faudra accompagner ce tableau de tour d'un tableau de balles de même taille, pour que chaque tour puisse tirer et avoir une balle.

8 Améliorations diverses

Il s'agit ici de pistes d'améliorations, vous pouvez les suivre ou en proposer d'autres.

- Mettre des points de vie aux créatures (creature.vie = 2 par exemple). Lors d'une collision, détruire la balle et enlever une vie à la créature, jusqu'à ce qu'elle ait 0 vie).
- Introduire de l'argent (pièces d'or ?) dans le jeu, chaque créature tuée rapporte des pièces d'or qui sont nécessaires pour construire des tours.
- Un chemin plus complexe pour les créatures. Il faudra alors programmer leur déplacement à travers ce terrain.
- ...

9 Pour aller encore plus loin

- Des vagues de monstres.
- pouvoir gagner.
- D'autres type de tours.
- D'autres type de monstres (rapide, ...).