PEZENNEC Océane
PHILIPPE Noé
Groupe 5

Synthèse du jeu Space Invaders

Class SpaceInvaders() :

> C'est la class qui lance le jeu, créer une frame, créer le menu, et enregistre le joueur dans le fichier "point.json".

def __init__() :

- Créer la frame, sans oublier `self.root.resizable(0, 0)` pour éviter qu'on puisse agrandir la fenêtre.
- Créer un canvas pour le menu

def play() :

- Lance le menu et `self.root.mainloop()` pour que la fenêtre reste ouverte

def menu() :

- Créer le menu avec :
  - Titre
  - Les huit meilleurs scores rangés dans l'ordre croissant. Il est compliqué de trier, en fonction des points, la liste de dictionnaire du fichier "resultat.json". L'astuce qu'on a trouvé, c'est lors de l'ajout d'un profil dans "resultat.json", il faut l'ajouter dans la liste en fonction de ses scores. Puis afficher les huit premiers éléments de la liste.
  - Input pour le nombre
  - Un bouton start qui pointe vers `scoreGame()`

def scoreGame() :

- Supprime le canvas 2
- Enregistre le joueur dans "point.json" en récupérant son pseudo
- Lance `start()`

def start() :

- Lance le jeu
- Lance les animations du jeu

Class Game() :

> C'est la class qui commande le jeu, elle gère les animations, l'affichage des scores, la fin du jeu, l'enregistrement du joueur dans "resultat.json" et le restart d'une partie.

def __init__() :

- Créer la flotte d'aliens, le defender, le shelter (protection)
- Créer le Canvas du jeu
- Créer l'affichage des scores :
  - Affiche le meilleur score du joueur s'il a déjà joué
  - Affiche le score de la partie
- Un variable `speedFireFleet` pour gérer la vitesse de tire des aliens

def keypressed() :

> Gère les touches du clavier

- Appel la fonction `move_in()` du defender avec la taille du déplacement (positive ou négative) comme paramètre

def start_animation() :

- Lance les animations du jeu : `animation()`
- Lance l'animation du tire des aliens : `animation_fire_fleet()`

def animation() :

- Lance les animations du defender, de la flotte et du shelter
- Met à jour les points
- Vérifie si le jeu n'est pas finit grâce à la fonction `endGame()`. Si on est en jeu, la fonction animation est appelé récursivement. Sinon on restart le jeu avec `restart()`

def animation_fire_fleet() :

- Vérifie si on est en jeu
- Lance les les tires de la flotte
- Réduit le temps de lancement récursif à chaque itération, quand `speedFireFleet` descend à 300ms, on le bloque pour éviter d'avoir trop de tires d'aliens.

def endGame() :

- Vérifie si le pseudo du joueur n'est pas vide
- Si il n'y a plus d'aliens on retourne "gagne"
- Si les aliens sont arrivé en bas ou si le defender n'a plus de vie on retourne "perdu"
- Si le jeu est fini, si le joueur existe déjà et que sont score est meilleur que celui enregistré, on le met à jour. Si c'est un nouveau joueur on l'ajoute dans "resultat.json".

def restart() :

- On supprime les widgets du canvas
- On affiche le titre en fonction du retour de `endGame()`
- Même affichage que pour le menu mais avec un bouton restart qui pointe vers `newGame()`

def newGame() :

- Supprime le canvas pour laisser la place à celui de la nouvelle partie
- Met à 0 les points du fichier "point.json" et récupère le nom du joueur
- Relance le jeu et les animations

Class Defender() :

> C'est la class qui gère le defender, ses tires, ses vies…

def __init__() :

> définie le defender : taille, nombre de tires autorisé, nombre de vie…

def install_in() :

- Créer le defender avec une image depuis la fonction `ship_image()`
- Affiche les vies du defender et les ajoutes dans la liste pour pouvoir les manipuler

def move_in() :

> Gère le déplacement du defender

- On récupère la taille du canvas et les coordonnés du defender. Comme le defender est une image, on a pas trouvé d'autre moyen que de récupérer ligne par ligne ses coordonnés. C'est pas vraiment optimisé mais ça marche.
- si on se déplace vers la gauche, on vérifie que le defender ne dépasse pas le canvas, idem pour la droite.

def fire() :

- Si la taille de la liste `fired_bullets` qui stockes les projectiles du defender est inférieur au tire maximum autorisé, on créer un nouveau projectile en appelant la class `Bullet`

def animation_projectil() :

- Pour chaque projectiles du defender on appel la fonction `move_in()` de la class `Bullet()`

def rm_bullet() :

> Enlève le projectile donné en paramètre de la liste `fired_bullets`

def defender_touche() :

> Gère la colision entre un projectile et le defender

- On se déplace dans la liste des projectile de la flotte passé en paramètre. On récupère les coordonné de chaque projectiles, si le defender à les même coordonné qu'un projectile alors :
  - On supprime le projectile du canvas et sa liste
  - On supprime le projectile du canvas et sa liste
  - On appel `kill_defender()` pour gérer l'animation du defender touché.

def kill_defender() :

- On récupère les coordonnés du defender
- On affiche l'image d'explosion depuis `ship_touched_image()`
- On supprime l'image au bout de 300ms

Class Fleet() :

> C'est la class qui gère la flotte d'aliens.

def __init__() :

> Définie la flotte : taille, le nombre d'aliens par lignes/colonnes, le nombre de tires autorisé, si les aliens changes d'état…

def install_in() :

- On se déplace comme un tableau pour créer un alien dans chaque "case".
- On appel la fonction `img_alien_vivant()` de la class `Alien()` pour affiché la photo de alien.
- On ajoute l'alien de la liste `aliens-fleet` qui gère les aliens.
- On met à jour les coordonnés en x et en y.

def move_in() :

> Gère le déplacement de la flotte.

- On récupère la taille du canvas et de la taille de la flotte.
- Si le déplacement est positif :
  - Si la flotte dépasse le canvas, on change le signe du déplacement, on déplace les aliens vers le bas.
- Idem si le déplacement est négatif.
- On appel l'animation de chaque alien.

def coord() :

> Retourne la position d'un alien aléatoirement (utile pour gérer le tire des aliens).

- On récupère un entier aléatoire entre 0 et le nombre d'aliens.
- On récupère un alien correspondant à l'indice aléatoire.
- On récupère ses coordonnés qu'on retourne.

def maj_photo_alien() :

- Met à jour la photo de chaque aliens si on est dans l'état "Angry".

def alien_touche() :

> Gère la collision entre un projectile et un alien.

- On récupère les coordonnés des projectiles et des aliens
- Si les coordonnés d'un alien et d'un projectile est la même :
  - On appel la fonction `kill_alien()` la classe `Alien()`
  - On supprime le projectile du canvas
  - On supprime le projectile de sa liste
  - On supprime l'alien de sa liste
  - On met à jour les scores

def fire() :

> Gère les tire de la flotte.

- Si on a pas dépassé le nombre de tires autorisé :
  - Si les aliens sont dans l'état "Angry", appel la class `Bullet()` avec une couleur orange.
  - Sinon la couleur est bleu.

def animation_projectil() :

- Pour chaque projectiles du de la flotte on appel la fonction `move_in()` de la class `Bullet()`.

def rm_bullet() :

> Enlève le projectile donné en paramètre de la liste `fired_bullets`.

def get_width() :

> Retourne la taille de la flotte.

Class Alien() :

> C'est la class qui gère les aliens.

def __init__() :

- Comporte seulement l'id des aliens initialisé à "None" car C'est la class `Fleet()` qui gère le déplacement, les tires… des aliens.

def get_image(), def get-deathImage(), def get_imageAngry :

> Différentes images des aliens .

def install_in():

> Créer un alien avec les paramètre envoyé par la fonction `img_alien_vivant()` ou `img_alien_dead()`.

)`
def img_alien_vivant() :

> Créer un alien en appelant la fonction `install_in()`.

def img_alien_dead() :

> Créer une animation lorsque l'alien est touché.

def move_in() :

> Gère le déplacement de l'alien

def kill_alien() :

- On récupère les coordonnés de l'alien.
- On supprime l'alien du Canvas.
- On appel l'image de l'animation d'un alien touché avec `img_alien_dead()`.

Class Bullet() :

> Créer et gère le déplacement des différents projectiles.

def __init__() :

> Définie les projectiles : taille, vitesse, qui est le tireur…

def install_in() :

> Créer soit un cercle ou un laser selon le tireur.

- Si le tireur est un alien :
  - On récupère les coordonnés du tireur
  - On créer un cercle de couleur donné en paramètre

- Si c'est le defender on récupère l'image du laser

def image() :

> Image du laser

def move_in() :

> Gère le déplacement du projectile

- On récupère les coordonnés du projectile.
- S'il est dans le canvas, on met à jour son ordonné (négative ou positive selon le tireur).
- Sinon on supprime le projectile du canvas et on appel `rm_bullet()` du tireur pour supprimer le projectile de sa liste.

Class Shelter()

> C'est la class qui créer et gère les protections du defender.

def __init__() :

> Définie les protections : le nombre, combien de tires ils supportent, leurs tailles…

def image(), imagebreak(), imagebreak2(), shelter_touched_image(), shelter_kill_image() :

> Différentes image des protections.

def install_in() :

> Créer les protections

- On récupère la taille du canvas.
- On place les protections en fonction de la taille du canvas.

def shelter_touche_defender() :

> Gère la collision entre un projectile et une protection.

- On récupère les coordonnés du projectile
- Si un projectile à les même coordonnés qu'une protection :
  - On supprime le projectile du canvas et de sa liste.
  - On enlève une vie à la protection.
  - Si il n'a plus de vie, on appel une image d'explosion pour l'animation et on le supprime du canvas.
  - Sinon on appel `break_shelter()` pour gérer l'animation de la collision.

def break_shelter() :

> Gère l'animation de la collision.

- On récupère la photo de l'explosion.
- On la supprime au bout de 300ms.
- On met à jour l'image la protection en fonction de ses vies.

Conclusion :

Un projet très intéressant, nous n'avions jamais fait de python avant cette année. Le plus difficile est de comprendre comment les class interagisses entre elles.

La partie sur l'affichage dans l'ordre croissant des points était assez compliqué, on a du cherché un peu pour trouver la solution. On a également eu du mal à comprendre comment récupérer la position de d'un alien depuis la flotte.

Il reste seulement un petit beug sur jupytere lors de la fin d'une partie, il ne fait pas crasher le jeu mais il est tout de même présent. On a pas réussis à le résoudre mais on pense que c'est du à certaines animations qui ne sont pas terminées alors que le jeu est fini.

A l'avenir on pourrait rajouter une flotte qui descend en continu, avec des aliens différents. Rajouter des projectiles pour le defender s'il n'a pas été touché depuis longtemps.