

Synthèse du jeu Space Invaders

- I. Introduction
- II. SpaceInvaders()
- III. Game()
- IV. Defender()
- V. Fleet()
- VI. Alien()
- VII. Bullet()
- VIII. Shelter()
- IX. Conclusion

Introduction :

Un projet qui a pour but de développer un Space Invaders en python. Le projet s'est bien déroulé dans l'ensemble. Nous avons eu un peu de mal à démarrer mais une fois les différentes classes bien comprises, tout s'est bien déroulé. On a terminé les trois étapes et on a rajouté quelques fonctionnalités :

- Les scores du menu affiche seulement les huit meilleurs scores et sont triés dans l'ordre croissant.
- Une image de laser pour les tirs du defender.
- On peut redémarrer une partie sans quitter le jeu.
- Le jeu refuse un pseudo vide.
- Les tirs des aliens changent de couleur quand ils changent d'état.
- Les aliens tirent plus au fur et à mesure de la partie.
- Des animations d'explosion pour les protections et le defender.

Class SpaceInvaders() :

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

def __init__() :

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

def play() :

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

def menu() :

Crée le menu

- 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 trouvée, c'est lors de l'ajout d'un profil dans "resultat.json", il faut l'ajouter dans la liste en fonction de son score. Puis afficher les huit premiers éléments de la liste.
- Input pour le nombre.

- Un bouton start qui pointe vers `scoreGame()` .

def scoreGame() :

Insère les informations du joueur dans "point.json".

- 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 classe 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__() :

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

def keypressed() :

Gère les touches du clavier.

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

def start_animation() :

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

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 fini grâce à la fonction `endGame()` . Si on est en jeu, la fonction `animation()` est appelée récursivement. Sinon on recommence la partie avec `restart()` .

def animation_fire_fleet() :

Gères l'animation du tire des aliens.

- Vérifie si on est en jeu.
- Lance les tirs de la flotte.
- Réduit le temps de lancement à chaque itération, quand `speedFireFleet` descend à 300ms, on le bloque pour éviter d'avoir trop de tirs d'aliens et on appelle `fleet.maj_photo_alien()` pour changer l'état des aliens.
- Rappel la fonction `animation_fire_fleet()` récursivement.

def endGame() :

Vérifie si la partie est terminée ou non.

- Vérifie si le pseudo du joueur n'est pas vide.
- S'il n'y a plus d'aliens, on retourne "gagne".
- Si les aliens sont arrivés 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 son score est meilleur que celui enregistré, on le met à jour.
 - Sinon on l'ajoute dans "resultat.json".

def restart() :

Gère l'affichage du menu pour recommencer une partie.

- 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() :

Relance une partie.

- 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 une partie et ses animations.

Class Defender() :

C'est la classe qui gère le defender, ses tirs, ses vies...

def __init__() :

Définit le defender : taille, nombre de tirs autorisé, nombre de vies...

def install_in() :

Crée le defender.

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

def move_in() :

Gère le déplacement du defender.

- On récupère la taille du canvas et les coordonnées du defender. Comme le defender est une image, on n'a pas trouvé d'autre moyen que de récupérer ligne par ligne ses coordonnées. Ce n'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() :

Gère les tirs du defender.

- Si la taille de la liste `fired_bullets` qui stocke les projectiles du defender est inférieure au tir maximum autorisé, on crée un nouveau projectile en appelant la classe `Bullet()` .

def animation_projectil() :

- Pour chaque projectile du defender, on appelle la fonction `move_in()` de la classe `Bullet()` .

def rm_bullet() :

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

def defender_touche() :

Gère la collision entre un projectile et le defender.

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

def kill_defender() :

Gère l'animation du defender touché.

- On récupère les coordonnées 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 classe qui gère la flotte d'aliens.

def __init__() :

Définit la flotte : taille, le nombre d'aliens par lignes/colonnes, le nombre de tirs autorisés, si les aliens changent d'état...

def install_in() :

- On se déplace comme dans un tableau pour créer un alien dans chaque "case".
- On appelle la fonction `img_alien_vivant()` de la classe `Alien()` pour afficher la photo de alien.
- On ajoute l'alien de la liste `aliens_fleet` qui gère les aliens.
- On met à jour les coordonnées en x et en y.

def move_in() :

Gère le déplacement de la flotte.

- On récupère la taille du canvas et 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 appelle l'animation de chaque alien.

def coord() :

Retourne la position d'un alien aléatoirement (utile pour gérer le tir 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 coordonnées qu'on retourne.

def maj_photo_alien() :

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

def alien_touche() :

Gère la collision entre un projectile et un alien.

- On récupère les coordonnées des projectiles et des aliens.
- Si les coordonnées d'un alien et d'un projectile sont les mêmes :
 - On appelle la fonction `kill_alien()` de la classe `Alien()` pour supprimer l'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 le score.

def fire() :

Gère les tirs de la flotte.

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

def animation_projectil() :

- Pour chaque projectile de la flotte on appelle la fonction `move_in()` de la classe `Bullet()` pour bouger le projectile vers le bas.

def rm_bullet() :

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

def get_width() :

Retourne la taille de la flotte.

Classe Alien() :

C'est la classe qui gère les aliens.

def __init__() :

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

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

Différentes images des aliens.

def install_in() :

Crée un alien avec les paramètres envoyés par la fonction `img_alien_vivant()` ou `img_alien_dead()` .

def img_alien_vivant() :

Crée un alien en appelant la fonction `install_in()` .

def img_alien_dead() :

Crée une animation lorsque l'alien est touché.

def move_in() :

Gère le déplacement de l'alien.

def kill_alien() :

Supprime un alien.

- On récupère les coordonnées de l'alien.
- On supprime l'alien du canvas.
- On appelle l'image de l'animation d'un alien touché avec `img_alien_dead()` .

Class Bullet() :

Crée et gère le déplacement des différents projectiles.

def __init__():

Définit les projectiles : taille, vitesse, qui est le tireur...

def install_in():

Crée soit un cercle ou un laser selon le tireur.

- Si le tireur est un alien :
 - On récupère les coordonnées du tireur
 - On crée un cercle de couleur donnée 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 coordonnées du projectile.
- S'il est dans le canvas, on met à jour son ordonnée (négative ou positive selon le tireur).
- Sinon on supprime le projectile du canvas et on appelle `rm_bullet()` du tireur pour supprimer le projectile de sa liste.

Class Shelter()

C'est la classe qui crée et gère les protections du defender.

def __init__():

Définit les protections : le nombre, combien de tirs elles supportent, leurs tailles...

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

Différentes images des protections.

def install_in():

Crée 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 coordonnées du projectile.
- Si un projectile à les mêmes coordonnées qu'une protection :
 - On supprime le projectile du canvas et de sa liste.
 - On enlève une vie à la protection.
 - S'il n'a plus de vie, on appelle une image d'explosion pour l'animation et on le supprime du canvas.
 - Sinon on appelle `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 classes interagissent entre elles.

La partie sur l'affichage dans l'ordre croissant des points était assez compliqué, on a du chercher un peu pour trouver la [solution](#). On a également eu du mal à comprendre comment récupérer la position d'un alien depuis la flotte avec la fonction `x1, y1, x2, y2 = canvas.coords(...)`

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

A l'avenir on pourrait rajouter grâce à une intelligence artificielle une flotte qui créer des aliens différents et qui descend en continu. Rajouter des projectiles pour le défendre s'il n'a pas été touché depuis longtemps ou encore rajouter du son.

Vous pouvez voir les différentes versions et l'avancé du jeu sur mon [Github](#).