RAPPORT DU PROJET – SPACE INVADERS

1. Introduction

Le projet Space Invaders avait pour objectif de programmer une version simplifiée puis améliorée du célèbre jeu dans un environnement Tkinter. Le jeu final inclut un défenseur contrôlé par l'utilisateur, une flotte d'aliens qui se déplace, des collisions, des tirs ennemis, un système de vies et un fichier de sauvegarde des meilleurs scores.

2. Explication des classes et lien direct avec le code

ScoreBoard :

Cette classe gère les meilleurs scores grâce au module CSV.
Les scores sont lus depuis un fichier via **csv.reader**, puis stockés dans une liste Python contenant des couples (nom, score).
La fonction **ajout_ou_meilleur()** compare un score existant avec un nouveau score et garde le meilleur.
La sauvegarde utilise **csv.writer** pour écrire les lignes dans le fichier.
Elle contient donc toutes les informations nécessaires pour afficher le classement au lancement du jeu.

Alien :

Chaque Alien utilise une image chargée avec **tk.PhotoImage** puis réduite via **subsample(2,2)**.
Il possède plusieurs attributs importants :

- id : identifiant graphique renvoyé par **canvas.create_image()**
- alive : indique si l'alien est encore vivant
- image : stocke l'image de l'alien

L'alien apparaît dans le canvas grâce à **create_image()**.
Le mouvement utilise **canvas.move()**.
La détection d'un tir se fait grâce à **canvas.bbox()**, qui renvoie un rectangle autour de l'alien et de la balle.
Si les rectangles se chevauchent, l'alien est supprimé par **canvas.delete()** et son attribut **alive** passe à **False**.

Fleet :

Cette classe gère toute la flotte d'aliens.
Elle contient une liste d'objets Alien (**self.aliens_fleet**) ainsi que la direction de déplacement.
Son déplacement utilise **canvas.bbox("alien")** qui permet de récupérer la position globale de la flotte.
Quand elle touche un bord, la direction s'inverse et la flotte descend.

Les tirs ennemis sont réalisés grâce à **random.random()** :
si la valeur est inférieure à 0.05, un alien vivant est choisi au hasard pour tirer.
Cela crée une balle violette **(AlienBullet)** qui descend vers le défenseur.
La flotte gère aussi une liste **alien_bullets** contenant toutes les balles ennemies actives.

Defender :

Le défenseur est un rectangle bleu affiché en bas du canvas.
Il possède plusieurs attributs :

- id : identifiant graphique
- move_delta : distance de déplacement
- fired_bullets : liste contenant les balles tirées
- max_fired_bullets = 8 : limite de tirs simultanés
- lives = 3 : nombre de vies

Il se déplace avec **canvas.move()** lorsque l'utilisateur appuie sur les flèches gauche ou droite.
Le tir se fait via la méthode **fire()**, qui crée un Bullet si le joueur n'a pas dépassé la limite de 8 tirs simultanés.
Les vies restantes sont affichées dans le HUD.

Bullet :

Les balles du joueur sont dessinées sous forme d'ovales rouges.
Elles possèdent :

- id : identifiant dans le canvas
- speed = 8
- active = True : indique si la balle est encore dans le jeu

Elles montent verticalement grâce à **canvas.move()** avec une vitesse négative.
Lorsqu'elles sortent du canvas, elles deviennent inactives (active = False) et sont supprimées.
Chaque balle vérifie si elle touche un alien grâce à **canvas.bbox()**.

AlienBullet :

Les balles ennemies descendent vers le joueur.
Elles possèdent :

- un identifiant id
- un rayon
- une vitesse positive
- un attribut active

Elles utilisent le même principe que Bullet, mais descendent au lieu de monter.
Si elles touchent le défenseur, celui-ci perd une vie.
La collision est vérifiée avec **canvas.bbox()**.

Game :

La classe Game gère la logique principale : score, vies, animation, déplacements, collisions, fin de partie.
Elle crée le canvas, installe la flotte, le défenseur, et l'interface (score, vies).

La méthode **animation()** est rappelée toutes les 60 millisecondes via **canvas.after(60,...)**,
ce qui crée une vraie boucle de jeu.
Elle déplace les balles, la flotte, les tirs ennemis, détecte les collisions et met à jour le HUD.
Elle vérifie aussi si tous les aliens sont morts ou si le défenseur n'a plus de vies,
et déclenche **finish_game()** en cas de victoire ou défaite.

SpaceInvaders :

Cette classe affiche l'écran d'accueil avec les meilleurs scores, lit le nom du joueur,
et lance la partie via la classe Game.
Elle gère la fenêtre principale Tkinter, l'écran des scores et le bouton "Jouer".
C'est le contrôleur principal qui gère toute l'interface du jeu.

3. Conclusion

Le projet nous a permis de comprendre Tkinter, les bases de l'animation, la gestion d'événements clavier, la POO et la manipulation de fichiers CSV.

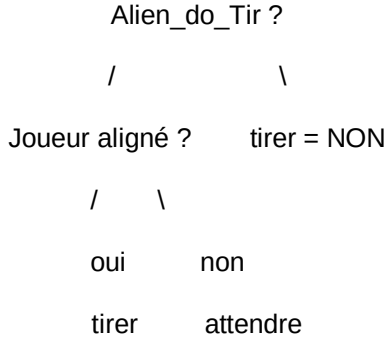
Une amélioration future du jeu pourrait consister à intégrer une forme d'intelligence artificielle inspirée du CM8, qui présente :

- les arbres de décision,
- l'algorithme minimax (jeu du morpion),
- la logique d'IA qui "imite des décisions humaines",
- et la sélection d'actions possibles dans une situation donnée.

Dans le CM8, un arbre de décision sert à représenter les choix possibles d'un joueur.
On peut appliquer exactement la même idée aux aliens.

Exemple 1 : Tirs ennemis intelligents :

Au lieu de tirer au hasard comme dans notre code avec **if random.random() < 0.05**, on peut remplacer ça par une **décision conditionnelle**, comme dans un arbre du CM8 :



Exemple 2 : Adaptation dynamique de la difficulté :

Dans le CM8 (jeu du morpion), l'algorithme Minimax **évalue les situations** :

- Situation favorable = score positif
- Situation défavorable = score négatif

On pourrait reprendre le même principe pour notre projet :

On évalue d'abord le score avec la fonction suivante :

score_situation = nombre_aliens_restants * 2 – vies_du_joueur * 5

Puis :

- Si **score_situation** est élevé (situation favorable pour les aliens) : aliens tirent moins.
- Si **score_situation** est faible (situation favorable pour le joueur) : aliens accélèrent et tirent plus.