PROJET EN PROCESSING JAVA

CREATION DU JEU SPACE INVADERS

REALISE PAR :

DIALLO Fatimatou

DIALLO Mamadou Talibe

Le, 02/01/2025

ENCADRE PAR :

Mr. Maxime MARIA,

Faculté des Sciences et Techniques de Limoges

TABLE DES MATIERES

I.INTRODUCTION

Nous avons conçu une version moderne et interactive du célèbre jeu d'arcade **Space Invaders** dans le cadre de ce projet. Ce jeu met en scène un joueur contrôlant un vaisseau spatial pour repousser des vagues d'envahisseurs extraterrestres tout en évitant des obstacles et en accumulant des points.

Notre version de **Space Invaders** se distingue par plusieurs fonctionnalités innovantes :

- **Un menu principal**, permettant de commencer une nouvelle partie ou de quitter le jeu.
- **Un menu pop-up**, accessible via la touche ESC, offrant diverses options : reprendre la partie, sauvegarder la partie en cours, charger une partie sauvegardée, consulter les meilleurs scores ou quitter le jeu.
- **Une gestion intuitive des fins de partie**, avec un menu affichant le score final et proposant de recommencer une nouvelle partie ou de quitter le jeu.
- **Une fonctionnalité de sauvegarde et de chargement**, permettant aux joueurs de reprendre leur partie là où ils l'avaient laissée.
- **Une interface de consultation des meilleurs scores**, conçue pour inciter les joueurs à améliorer leurs performances.

Le jeu a été entièrement développé en **Processing**, en adoptant une approche orientée objet afin d'assurer la modularité et la clarté du code. Les interactions via le clavier, la transition fluide entre les différents états du jeu, et la prise en charge de scénarios variés (victoire, défaite, pause) garantissent une expérience utilisateur immersive et agréable.

Ce rapport explore en détail les différentes composantes du jeu, de la conception générale aux fonctions spécifiques, afin de mettre en lumière les mécanismes qui rendent cette version de **Space Invaders** à la fois captivante et accessible.

II. Conception du Jeu

Dans la conception de notre version de Space Invaders, nous avons opté pour une approche modulaire afin de garantir une clarté dans le code, une meilleure maintenance, et une possibilité d'évolution future. Nous avons structuré notre jeu autour de plusieurs classes principales, en exploitant pleinement les principes de la programmation orientée objet.

1. Fichier Principal - Space Invaders :

Le fichier principal du projet contient le cœur du programme, à savoir l'initialisation du jeu, la gestion de la boucle principale et les interactions entre les objets. Il s'agit de la base qui contrôle l'exécution et la logique du jeu.

1.1 Initialisation et Configuration :

Dans ce fichier, nous initialisons les composants du jeu tels que la taille de la fenêtre et les objets principaux (comme le jeu et le menu).

1.2 Boucle Principale :

La boucle principale du jeu assure l'actualisation du jeu à chaque frame, gère les entrées utilisateur (mouvement du vaisseau, tir), détecte les collisions et met à jour l'affichage. Elle comprend plusieurs étapes :

• **Affichage du jeu** : La méthode **draw()** est responsable de l'affichage de tous les objets à chaque frame, en s'assurant que les éléments visuels sont correctement rendus.

• **Mise à jour des objets** : Les positions du vaisseau, des missiles et des envahisseurs sont recalculées en fonction de leur direction et de leur vitesse. Les missiles sortant de l'écran sont également supprimés pour améliorer les performances.

• **Gestion des entrées** : Le vaisseau se déplace horizontalement avec les touches fléchées ou les touches Q (q) et D (d), et les tirs sont déclenchés par la touche espace.

• **Vérification des collisions** : Le programme détecte si un missile touche un envahisseur, si un missile ennemi atteint le vaisseau du joueur, ou si un missile heurte un obstacle.

Ces éléments clés du fichier principal permettent d'assurer une expérience de jeu fluide et réactive.

2. La Classe Game

La classe Game encapsule la logique du jeu, en particulier la gestion des différents objets et de la boucle principale. Elle se charge de l'initialisation des entités, du lancement du jeu et de la gestion des états (comme la fin du jeu).

Le plateau (Board) est créé pour accueillir les différentes entités et gérer les cellules sur lesquelles se déplacent les objets. Chaque objet (missile, vaisseau, envahisseur) est ensuite créé à une position de départ spécifique.

Méthodes de Gestion des Entités

Cette classe contient plusieurs méthodes dédiées à l'ajout et à la gestion des entités, y compris les envahisseurs, les missiles et le vaisseau. Chaque entité est définie par une classe distincte.

- **Constructeur Game()** : Crée le plateau de jeu, le vaisseau, les envahisseurs et les obstacles.
- **Méthode drawIt()** : Gère le rendu visuel du jeu, incluant le plateau, le vaisseau (et ses missiles), les envahisseurs (et leurs missiles) ainsi que les obstacles. Elle vérifie également les états du jeu (en cours, en pause, victoire ou défaite) via des drapeaux tels que _gameOver et _gameWon.
- **Méthode update()** : Met à jour la logique du jeu à chaque frame, en recalculant les positions des entités (vaisseau, envahisseurs, missiles) et en gérant les collisions (tirs, obstacles, etc.).
- **Méthode loadBoard()** : Charge le plateau de jeu depuis un fichier de configuration (levels/level.txt).
- **Méthode checkWinCondition()** : Vérifie si tous les envahisseurs ont été éliminés.
- **Méthode ModifierMeilleursScores()** : Met à jour le fichier des meilleurs scores à la fin d'une partie si le score actuel dépasse le minimum des cinq meilleurs scores.

Gestion du Vaisseau (Spaceship) : Elle inclut des méthodes permettant de déplacer le vaisseau et de tirer des missiles.

  - **createVaisseau()** : Cette methode permet de créer le vaisseau.
  - **handleKey(int k)** : Gère les déplacements et les tirs du vaisseau en fonction des entrées clavier.

Gestion des Envahisseurs (Invader) : Les envahisseurs sont générés sur le plateau et se déplacent de manière automatique. Lorsqu'un envahisseur atteint le bas du plateau ou touche un obstacle, cela déclenche une condition de fin du jeu.

  - **createInvaders()** : Cette fonction permet de créer les envahisseurs.
  - **updateInvaders()** : Met à jour les positions des envahisseurs. Ces derniers se déplacent horizontalement (gauche-droite ou droite-gauche) et descendent d'une case lorsqu'ils atteignent les bords gauche ou droit du plateau.
  - **checkObstacleCollisionWithInvaders()** : Vérifie si un envahisseur touche un obstacle, si oui termine le jeu.

Gestion des Missiles (Missile) : Chaque missile a une position, une vitesse et une taille. Il se déplace vers le haut ou vers le bas en fonction de son origine (vaisseau ou envahisseur).

  - **Méthode handleInvaderMissiles()** : Elle permet de créer les missiles des envahisseurs.
  - **Méthode updateMissilesInvaders() :** Elle gère les missiles tirés par les envahisseurs et de leurs déplacements.

Gestion des Collisions : Pour gérer les différentes collisions, nous avons utilisés différents types de fonction, comme :

  - **handleMissileSpaceshipCollisionWithInvaders()** : Cette fonction gère les collisions entre les missiles du vaisseau et les envahisseurs en vérifiant leurs positions.
  - **handleMissileSpaceshipObstacleCollisions()** : Elle permet de gérer les collisions entre les missiles du vaisseau et les obstacles.
  - **handleInvaderMissileCollisionWithSpaceship()** : Cette fonction permet de gérer les collisions entre les missiles d'envahisseurs et le vaisseau.
  - **handleInvaderMissileObstacleCollisions()** : Cette fonction gère les collisions entre les missiles d'envahisseurs et les obstacles.
  - **handleTirInvaderTirVaisseauCollision()** : Elle gère les collisions entre les missiles du vaisseau et les missiles des envahisseurs.

3. La Classe Board :

La classe Board représente le plateau de jeu où se déroulent toutes les actions. Elle contient la logique de création du plateau et de gestion des cellules.

Gestion des Cellules

Chaque entité (envahisseur, vaisseau, missile) est assignée à une cellule spécifique sur le plateau. La classe Board gère les coordonnées des cellules et la taille du plateau.

- **getCellCenter()** : Calcule la position centrale d'une cellule en fonction des coordonnées du plateau, utilisée pour placer les objets.
- **Dimensions du Plateau** : Le plateau est défini par un certain nombre de cellules en largeur et en hauteur, stockées dans les variables _nbCellsX et _nbCellsY.
- **drawIt()** : Dessine les cellules du plateau sous forme de rectangles visibles, facilitant le placement et le déplacement des entités.

4. La Classe Invader

La classe Invader représente les ennemis du jeu, c'est-à-dire les envahisseurs. Chaque envahisseur possède une position, une taille et une logique de mouvement propre.

4.1 Dessin et Position des Envahisseurs

- **drawIt()** : Charge l'image du sprite d'un envahisseur et l'affiche à la position correcte sur le plateau.

4.2 Gestion des Mouvements

Les envahisseurs se déplacent automatiquement selon une logique définie. Cette classe s'assure également de détecter si un envahisseur atteint le bas du plateau, ce qui entraîne une fin de jeu immédiate.

- **checkBottomCollisionInvader()** : Vérifie si un envahisseur a atteint la ligne du bas du plateau.

5. La Classe Missile

La classe Missile représente les projectiles utilisés dans le jeu, qu'ils soient tirés par le vaisseau ou par les envahisseurs. Elle assure la gestion des caractéristiques principales des missiles : position, vitesse, et interactions avec les autres objets (collisions).

5.1 Dessin des Missiles

La classe inclut une méthode pour dessiner chaque missile sur le plateau :

- **drawMissile()** : Charge et affiche une image représentant un missile à sa position actuelle sur le plateau.

5.2 Mouvements des Missiles

Les missiles peuvent être tirés par le vaisseau ou par un envahisseur. Leurs directions est déterminée par l'origine du tir (haut pour le vaisseau, bas pour l'envahisseur).

- **Missiles du Vaisseau** : Les missiles tirés par le vaisseau se déplacent vers le haut.
    - **updateTirVaisseau()** : Mise à jour de la position des missiles tirés par le vaisseau, qui se déplacent vers le haut.
- **Missiles des Envahisseurs** : Les missiles tirés par les envahisseurs se déplacent vers le bas.
    - **updateTirInvader()** : Mise à jour de la position des missiles tirés par les envahisseurs, qui se déplacent vers le bas.

5.3 Détection des Collisions

Des méthodes spécifiques permettent de vérifier si un missile touche un envahisseur, un obstacle ou un autre missile.

- Collisions entre les missiles du Vaisseau et les Envahisseurs **:**
    - **checkCollisionTirSpaceshipWithInvader()** : Vérifie si le missile tiré par le vaisseau touche un envahisseur.
- Collisions entre le Vaisseau et les Missiles des Envahisseurs **:**
    - **checkCollisionTirInvaderWithSpaceship() :** vérifie si un missile tiré par un envahisseur touche le vaisseau.
- Collisions entre Missiles **:**
    - **checkCollisionTirVaisseauAndTirInvader()** : vérifie si un missile tiré par le vaisseau touche un missile tiré par un envahisseur.
- Collisions entre les missiles du Vaisseau et les Obstacles **:**
    - **checkCollisionTirSpaceshipWithObstacle()** : vérifie si un missile tiré par le vaisseau touche un obstacle.

La classe Missile constitue un élément central de la dynamique du jeu. En gérant le dessin, les mouvements, et les collisions, elle assure une expérience fluide et immersive, tout en permettant une interaction riche entre les différents éléments du jeu.

6. La Classe Spaceship

La classe Spaceship représente le vaisseau contrôlé par le joueur. Elle gère les déplacements, les tirs, et la gestion de l'état du vaisseau (dégâts).

6.1 Déplacement et Tir

- **Méthode moveSpaceship()** : Permet de déplacer le vaisseau à gauche ou à droite en fonction des entrées du joueur.
- **Méthode createMissileSpaceship()** : Crée un missile à partir du vaisseau lorsque l'utilisateur appuie sur la touche de tir.

6.2 Gestion des Dégâts

- **isDamaged** : Une variable booléenne qui indique si le vaisseau a été endommagé. Lorsqu'il est touché, l'image du vaisseau change pour une image endommagée.
- **Méthode updateSpaceshipAndTir()** : Met à jour l'état du vaisseau et de ses missiles. Elle s'assure également que le vaisseau endommagé revient à son état normal après un certain délai.

7. La Classe Obstacle

La classe Obstacle représente des obstacles statiques sur le plateau qui peuvent bloquer les missiles.

7.1 Dessin des Obstacles

Les obstacles sont affichés à une position définie, et leur taille est déterminée par la taille de la cellule. Ils servent de barrières contre les missiles des envahisseurs ou du vaisseau.

- **drawObstacle()** : Cette méthode permet de charger et afficher une image répresentant un obstacle à une position spécifiée sur le plateau.

8. La classe Menu : Interaction Utilisateur

La classe Menu est responsable de l'affichage des différents menus. Nous avons centralisé toute la gestion des interfaces utilisateur dans cette classe pour simplifier les interactions. Voici les menus implémentés :

1. **Menu principal** : Permet de commencer une nouvelle partie ou de quitter le jeu.
2. **Menu pop-up (pause)** : Accessible avec la touche **ESC**, il met en pause le jeu et offre les options de sauvegarde, chargement, consultation des meilleurs scores, reprise de la partie ou de quitter le jeu.
3. **Menu de fin de partie** : S'affiche lorsque le joueur gagne ou perd. Il montre le score final et propose de reprendre une nouvelle partie ou de quitter le jeu.

Menu Principal

Au lancement, le joueur arrive sur un écran principal avec les options suivantes :

- 1. JOUER
- 2. Quitter le jeu

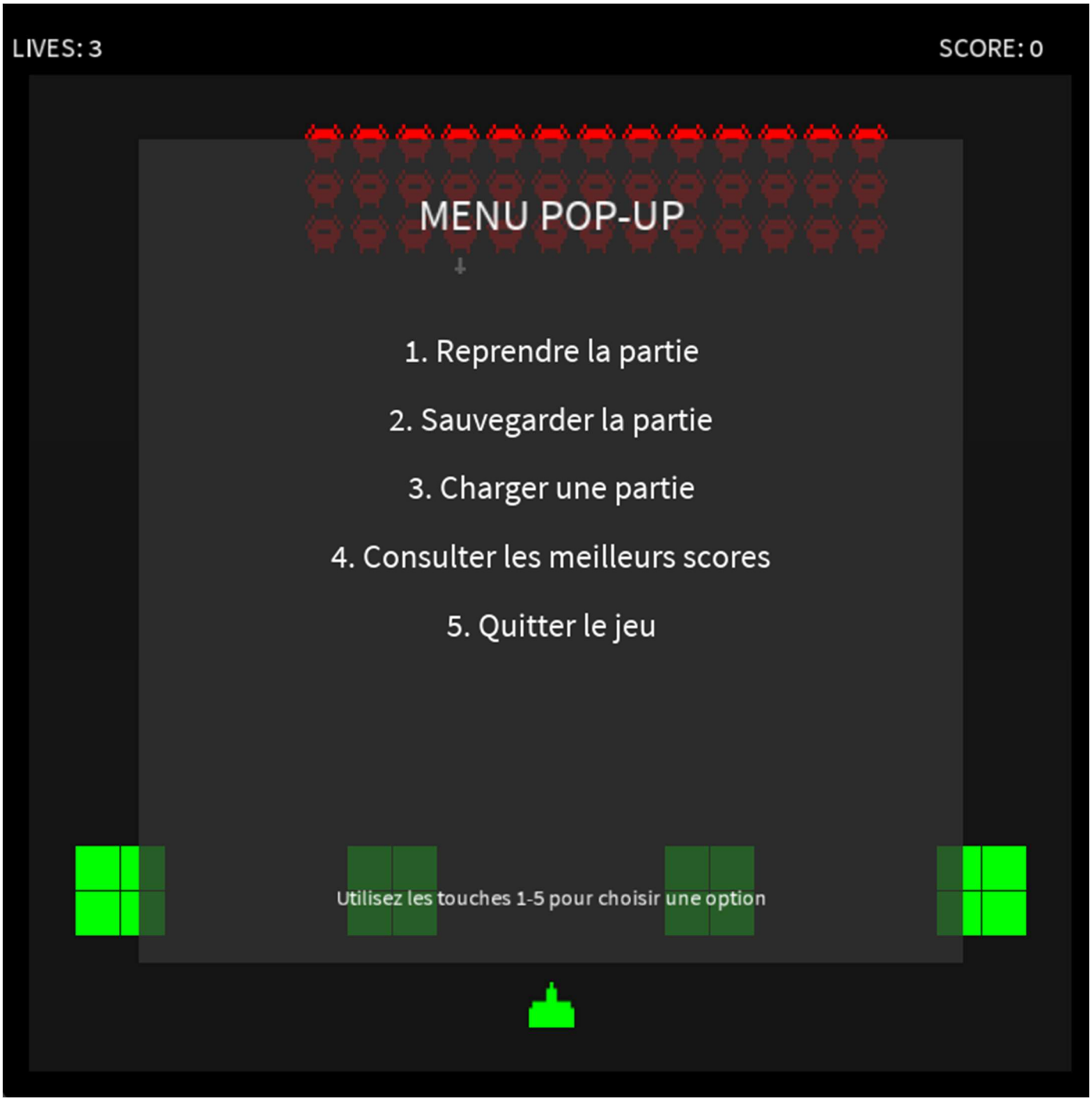
Afin de jouer, le joueur doit cliquer sur la touche 1, et pour quitter le jeu, il doit cliquer sur la touche 2.

Pause et Menu Pop-Up

Lorsque le joueur appuie sur la touche **ESC**, le jeu se met en pause et un menu s'affiche. Ce menu offre les options suivantes :

- 1. Reprendre la partie en cours.
- 2. Sauvegarder l'état actuel du jeu.
- 3. Charger une autre partie.
- 4. Consulter les meilleurs scores.
- 5. Quitter le jeu.

Afin de choisir une option, le joueur doit cliquer sur les touches allant de 1 à 5 en fonction de son choix.



Fin de Partie

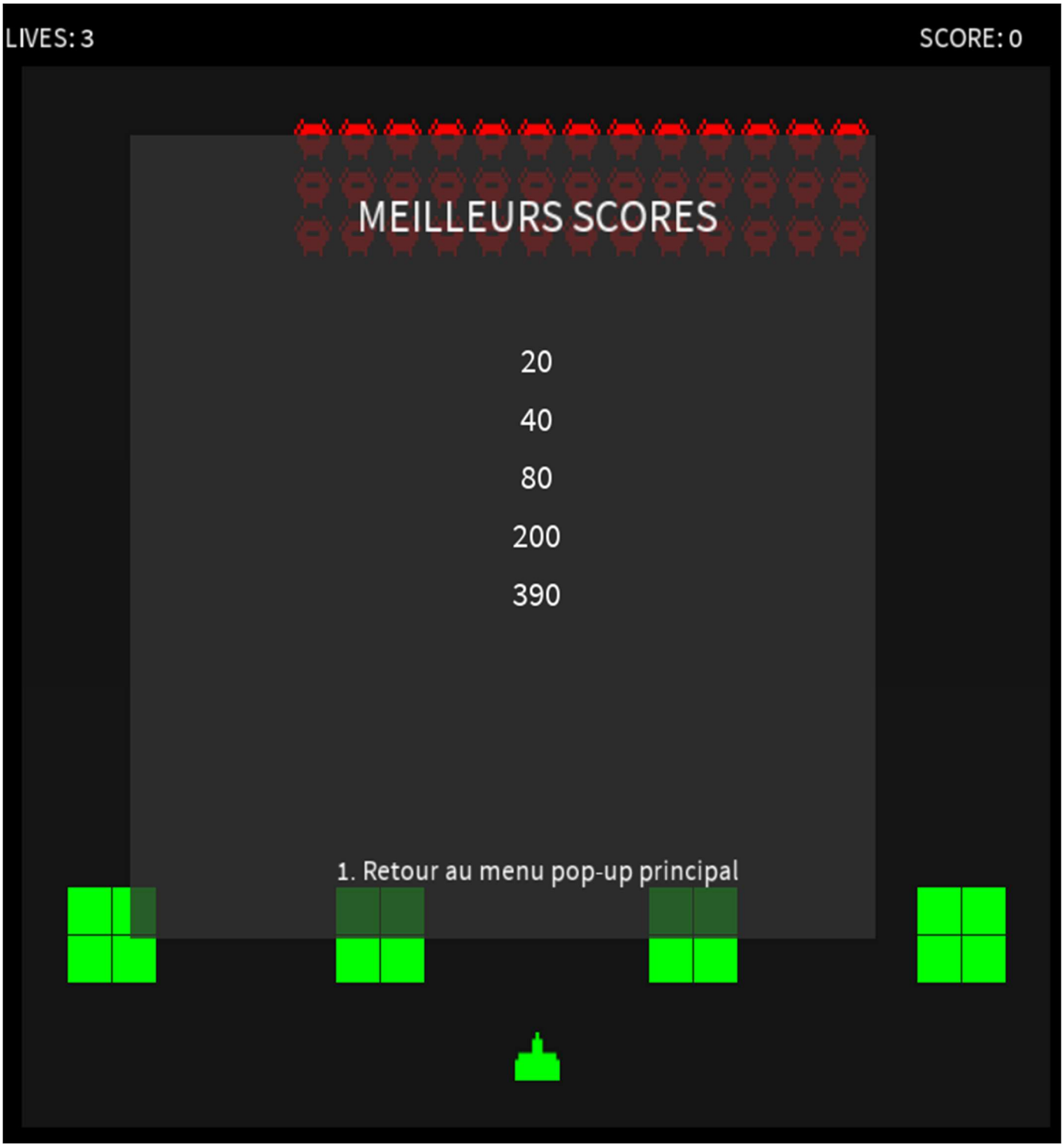
Lorsque le joueur gagne (élimine tous les ennemis) ou perd (le vaisseau est détruit), un menu de fin de partie s'affiche. Ce menu montre :

- Le score final du joueur.
- Les options pour reprendre une nouvelle partie ou quitter le jeu.
    - 1. Rejouer : (Il doit cliquer sur la touche 1 pour reprendre une nouvelle partie).
    - 2. Quitter le jeu : (Pour quitter le jeu, il clique sur la touche 2).

Affichage des Scores

Les meilleurs scores sont stockés et affichés dans une section dédiée du jeu, permettant au joueur de voir ses performances.

Pour reprendre la partie en cours, il doit cliquer sur le bouton 1 et revenir au menu pop-up principal afin de reprendre la partie.


LIVES: 3
SCORE: 0
MEILLEURS SCORES
20
40
80
200
390
1. Retour au menu pop-up principal

III.DIFFICULTES RENCONTREES ET SOLUTIONS

1. **Gestion des Collisions** : L'une des difficultés majeures a été de bien gérer les collisions entre missiles, vaisseau et envahisseurs. La solution réside dans l'utilisation de la fonction sqrt() pour calculer la distance entre deux objets et vérifier si celle-ci est inférieure à la somme de leurs rayons.
2. **Mouvements des Envahisseurs** : Faire en sorte que les envahisseurs se déplacent correctement sur l'écran sans sortir des limites était un autre défi. Nous avons utilisé un système de coordonnées basé sur des cellules et des méthodes pour inverser le mouvement lorsque les bords sont atteints.
3. **Optimisation des Missiles** : Assurer la suppression des missiles lorsque ceux-ci sortent des limites du plateau était une tâche importante. Nous avons utilisé des méthodes pour vérifier en continu leur position et les supprimer si nécessaire.

IV. CONCLUSION

Le projet **Space Invaders** représente un défi technique intéressant, notamment en ce qui concerne la gestion des mouvements, des collisions, et des interactions entre les objets du jeu. Les classes ont été conçues de manière modulaire, ce qui permet une gestion claire et efficace de chaque aspect du jeu. Cette approche facilite non seulement le développement initial, mais aussi la maintenance et l'extension du jeu. Par exemple, chaque objet du jeu, qu'il s'agisse du vaisseau, des missiles, des envahisseurs ou des obstacles, est géré par une classe dédiée, permettant une grande flexibilité pour l'ajout de nouvelles fonctionnalités ou la modification de comportements spécifiques sans perturber l'ensemble du système.

L'utilisation de la bibliothèque **Processing** a été un atout majeur dans ce projet, facilitant l'intégration d'éléments graphiques pour créer une interface fluide et intuitive. Le rendu visuel du jeu, avec des animations simples mais efficaces, a permis de rendre l'expérience plus immersive, en particulier pour les mouvements des vaisseaux et des missiles ainsi que les collisions. L'interface du jeu est simple, mais fonctionnelle, et le joueur peut facilement comprendre les mécaniques du jeu tout en s'amusant.

Le projet a également mis en évidence l'importance de la gestion des événements, notamment pour le déplacement du vaisseau, le tir de missiles, et la détection des collisions avec les ennemis ou les obstacles. Ces éléments sont cruciaux pour offrir une expérience de jeu fluide et réactive, et ont été gérés avec soin tout au long du développement.

Cependant, il existe encore plusieurs opportunités d'amélioration pour enrichir le jeu, comme l'ajout de nouveaux niveaux, des ennemis variés avec des comportements distincts, un système de score, des bonus, et des animations supplémentaires pour rendre l'expérience encore plus engageante.

En somme, ce projet **Space Invaders** est un excellent exemple de la manière dont un jeu peut être développé en suivant des principes solides de programmation orientée objet et en tirant parti des capacités de bibliothèques comme **Processing**. Il combine une logique de jeu bien pensée avec une interface graphique simple mais efficace, offrant ainsi une expérience ludique agréable et fluide. Bien que la version actuelle soit fonctionnelle, elle offre également une base solide pour des extensions futures et des améliorations, permettant d'enrichir l'expérience et de rendre le jeu encore plus captivant.