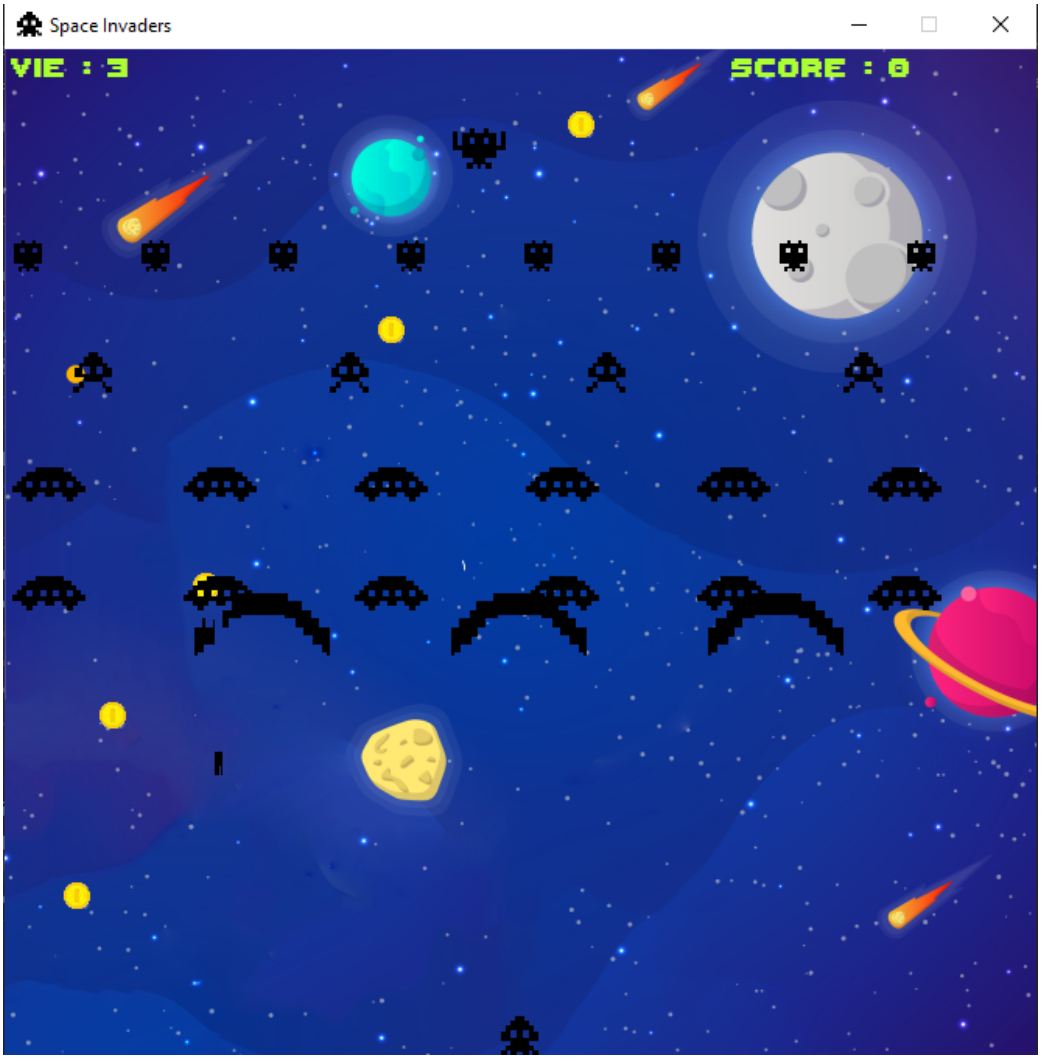
Rapport Space Invaders



GitHub:
https://github.com/ESIEECourses/projet-spaceinvaders2023-mostefa-sba-ismael-abdallah-m-chiri

Binôme: Mostefa-sba Ismaël - M’Chiri Abdallah

ESIEE PARIS

Table des matières

Introduction

Dans le cadre de notre projet C#, nous avons dû réaliser un jeu Space Invaders. Plusieurs pistes s'offraient à nous et nous avons décidé de choisir la piste verte.

Description succincte du problème :

Base du jeu

La partie commence, les vaisseaux ennemis sont générés aléatoirement. Les ennemis se déplacent horizontalement et verticalement jusqu'à atteindre le joueur. Les ennemis peuvent tirer des missiles ce qui leur permettent de blesser le joueur et aussi détruire les bunkers. Le joueur, quant à lui, peut se déplacer avec les flèches directionnelles gauche et droite. Il peut tirer un missile en appuyant sur la touche espace et donc tuer les ennemis ainsi que détruire ses propres bunkers. Si le joueur tue tous les ennemis alors il gagne la partie, sinon il perd la partie.

DLC ajoutés

Amélioration visuelle : Nous avons ajouté des images de fond différentes pour le menu principal et le jeu.

Menu Interactif : Nous avons créé un menu navigable avec les flèches directionnelles, permettant de choisir le mode de jeu, contrôler le volume et consulter le classement.

Police de Caractères Personnalisée : Intégration d'un font style personnalisé pour rendre le jeu plus réaliste que nous avons trouvé dans une bibliothèque open source.

Immersion Sonore : Ajout de sons pour enrichir l'expérience de jeu et augmenter l'immersion. Il y a une musique de fond différente pour le menu et le jeu, un son de tire différent pour les ennemies ou le joueur, un son pour les collisions et un son pour le bonus.

Bonus Aléatoire : Nous avons ajouté un bonus qui, lorsqu'il est touché par un joueur, procure l'un des bonus aléatoires suivants : "Big Missile", un missile plus gros ; "Root Missile", le prochain missile immobilise les ennemis ; "Double Speed", qui double la vitesse des missiles.

Score : Nous avons ajouté un système de score dans le jeu. Lorsque l'on touche une ennemie pendant la partie, cela nous donne 10 points de score. Ce score est affiché en haut à droite pendant la partie de jeu et sera utilisé par la suite pour le classement.

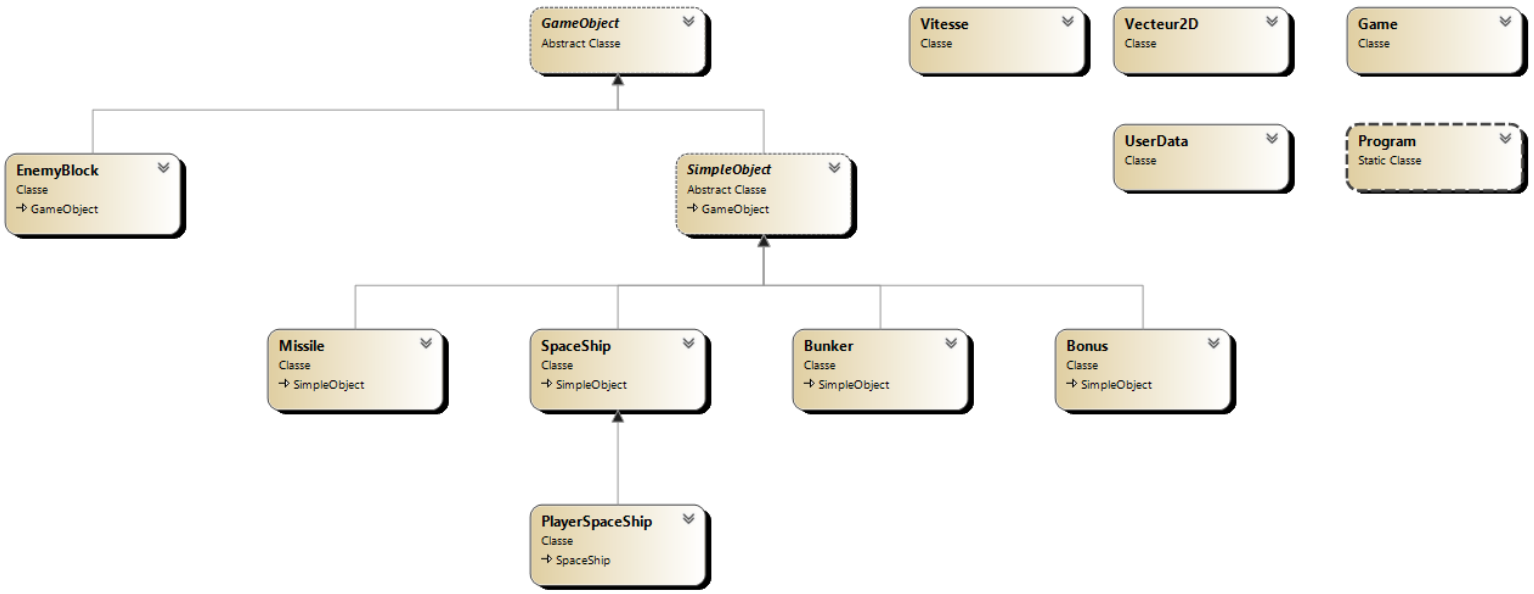
Classements : Mise en place d'un système de Leaderboards en deux étapes. Premièrement, lorsque le joueur joue dans une partie de jeu, il a un score pour chaque ennemi tué. Une fois la partie terminée, que ce soit gagné ou perdu, le joueur doit entrer un pseudonyme et appuyer sur la touche « espace » pour valider. En deuxième temps, le score est enregistré dans un fichier et sera affiché dans la page Leaderboards qui se trouve dans le menu.

Réglage du Volume : Ajout d'une barre de volume dans le menu ajustable avec les flèches directionnelles gauche et droite afin d'augmenter ou baisser le volume globale.

Mode de Jeu 'OneVsOne' : Création d'un nouveau mode de jeu local où 2 joueurs s'affrontent en 1 contre 1 tentant de réduire à zéro les points de vie de l'adversaire, avec des tirs de missiles triplés et des bonus à collecter.

Structure du programme

Diagramme de classe



Le diagramme illustre la structure hiérarchique et les relations entre les classes.

La classe **GameObject** est au sommet de la hiérarchie en tant que classe abstraite de base. Elle est classe mère de la classe abstraite **SimpleObject**, qui est à son tour la classe mère pour plusieurs objets spécifiques dans le jeu : Missile, SpaceShip, Bunker, et Bonus. Cela implique que tous ces objets partagent un ensemble commun de caractéristiques et de comportements définis dans GameObject et SimpleObject.

SpaceShip est spécialisée en PlayerSpaceShip, ce qui signifie que le vaisseau du joueur est un type particulier de vaisseau spatial avec des attributs et des comportements supplémentaires spécifiques au joueur. EnemyBlock quant à lui est directement fille de GameObject.

Vitesse et Vector2D sont des classes qui servent à des fins utilitaires comme la gestion de la vitesse et des positions ou des directions dans l'espace 2D du jeu. UserData est isolée de la hiérarchie des objets car elle sert à gérer le leaderboards et n'est donc pas un objet du jeu.

La classe Game est responsable de la logique globale du jeu. Elle implémente le paterne Singleton car il n'y a qu'une seule partie qui peut être lancée en même temps.

La classe Program est une classe statique, elle contient le point d'entrée du jeu pour démarrer l'application.

La structure globale du jeu est une hiérarchie avec héritage pour réutiliser et spécialiser les comportements. Les classes utilitaires sont utilisées pour supporter la logique du jeu, tandis que la classe Game agit comme un coordinateur central car elle y détient toute la logique, et Program est le lanceur du jeu.

Problèmes soulevés :

Un premier problème rencontré était la mise en place de son dans le jeu. Nous avions commencé par utiliser SoundPlayer pour implémenter le son, cependant, avec cette classe nous ne pouvions pas jouer 2 pistes audios en même temps. Nous avons donc eu à utiliser MediaPlayer qui permet de jouer plusieurs pistes audios différentes.

Lorsque nous avions voulu implémenter la barre de son il nous a fallu utiliser 2 images : une image représentant une barre de son vide et une image représentant une barre de son pleine. Ce que nous avons fait, c'est dessiner la barre de son pleine au-dessus de la barre de son vide à chaque fois qu'on augmente le volume. Cependant, avec cette approche, nous n'avons pas réussi à rendre cette barre de son parfaitement fonctionnelle. En effet, la barre de son pleine ne se dessine pas exactement au-dessus de la barre de son vide. Nous avons donc ajusté la position à la main pour que cela coïncide mais si on regarde bien elles ne sont pas parfaitement superposées.

L'implémentation des bonus dans le jeu présentait un défi technique. Initialement, chaque bonus devait être attribué directement aux vaisseaux. Cependant, la gestion des collisions se fait via les missiles. Lorsqu'un bonus est touché, bien que nous puissions déterminer si le vaisseau touché est un allié, attribuer le bonus au vaisseau correspondant n'était pas optimal. Pour résoudre cela, nous avons créé une classe "Vitesse". Chaque vaisseau possède maintenant une variable de type "Vitesse". Lorsqu'un missile est tiré, il copie la variable "Vitesse" du vaisseau qui le lance. Ainsi, chaque fois qu'un bonus est touché, il modifie directement la variable "Vitesse" du vaisseau concerné, rendant le système plus élégant et efficace.

La mise en place du tableau des scores (leaderboard) dans le jeu posait un problème, notamment en ce qui concerne la sauvegarde et l'affichage des noms et des scores des joueurs. Pour résoudre cela, nous avons opté pour une solution où les noms et scores sont enregistrés dans un fichier JSON. Côté programmation, nous avons créé la classe "UserData" qui gère ces données. Cette classe permet d'ajouter de nouvelles entrées ou de récupérer la liste actuelle des noms et scores enregistrés. En ce qui concerne l'affichage du leaderboard, nous avons conçu un système qui affiche cinq noms par page. Pour naviguer entre les pages, nous avons implémenté une fonctionnalité utilisant des opérations de modulo dans le code.

Un autre défi rencontré lors du développement du jeu était la segmentation du code pour améliorer sa lisibilité, particulièrement dans la classe "Game", qui contenait le plus grand nombre de lignes de code. Pour résoudre ce problème, nous avons opté pour une approche modulaire, en créant plusieurs fonctions afin de fractionner le code en sections plus gérables. De plus, nous avons introduit deux régions principales dans la classe : "UpdateRegion" et "DrawRegion". Dans "UpdateRegion", nous avons regroupé toutes les fonctions qui sont appelées dans la méthode "update" de la classe "Game". Cette région gère la logique et la mise à jour des états du jeu. D'autre part, "DrawRegion" contient toutes les fonctions utilisées dans la méthode "draw" de la classe. Cette région est dédiée à la gestion de l'affichage et du rendu graphique du jeu.