Michel CHEN / Valeria ASMAT

Projet Space Invaders

Piste Verte

https://github.com/ESIEECourses/projet-spaceinvaders2023-michel-chen-valeria-asmat

Introduction

Présentation du projet et objectifs du jeu

Le projet "Space Invaders" constitue une réinterprétation académique du célèbre jeu d'arcade classique. Dans cette version développée en C#, l'objectif demeure inchangé : anéantir un bloc d'ennemis à l'aide d'un vaisseau spatial. L'interface graphique, élaborée à partir de sprites et de bitmaps, se compose du vaisseau du joueur, d'une rangée de bunkers, des vaisseaux ennemis, des missiles, et affiche le nombre de vies restantes du joueur.

Le déplacement du vaisseau du joueur est limité à l'horizontal, contrôlé par les touches fléchées gauche et droite, garantissant qu'il reste dans les limites de l'écran. La menace se matérialise à travers un bloc d'ennemis organisé en lignes, se déplaçant alternativement de gauche à droite et de droite à gauche. Lorsque ce bloc change de direction, il effectue un décalage vers le bas, intensifiant sa vitesse de déplacement à chaque inversion.

Les interactions incluent des tirs de missiles, à la fois émanant des ennemis de manière aléatoire et du joueur par pression de la touche espace. Les collisions entre missiles, vaisseaux ennemis, bunkers et vaisseau du joueur sont définies de manière précise, en tenant compte des pixels des sprites en collision. La vie et la mort dans le jeu sont régies par des mécanismes de décrémentation du nombre de vies associées à chaque élément du jeu.

La victoire ou la défaite surviennent selon des conditions spécifiques : le joueur triomphe s'il éradique tous les vaisseaux ennemis, mais échoue s'il succombe face aux attaques ennemies ou si le bloc d'ennemis atteint la hauteur du joueur. En fin de partie, le joueur peut amorcer une nouvelle session en appuyant sur la touche Espace ou aller à la page de menu en appuyant sur P.

Le jeu permet également une pause, où tous les déplacements et la dynamique du jeu sont suspendus, activée par la touche P. Cette fonctionnalité offre au joueur la possibilité de reprendre la partie en appuyant à nouveau sur la touche P.

Dans ce rapport, nous explorerons la structure du programme, les défis rencontrés lors des tests, ainsi que les solutions apportées pour garantir les fonctionnalités du jeu.

Description succincte du problème

Utilisation de la programmation orientée objet (POO)

L'enjeu majeur du projet "Space Invaders" réside dans la création d'un jeu interactif mettant en œuvre les principes fondamentaux de la programmation orientée objet (POO). L'objectif

principal consiste à développer un programme offrant une expérience de jeu où le joueur manœuvre un vaisseau spatial pour éliminer un bloc d'ennemis. Le jeu est construit autour des concepts clés de la POO, une approche de modélisation logicielle qui utilise des objets pour structurer le code de manière modulaire et réutilisable.

La nature modulaire de la POO s'avère particulièrement appropriée pour représenter les éléments du jeu en tant qu'objets distincts interagissant entre eux. Le vaisseau du joueur, les vaisseaux ennemis, les bunkers, les missiles sont encapsulés dans des classes, facilitant la gestion des comportements spécifiques et des interactions.

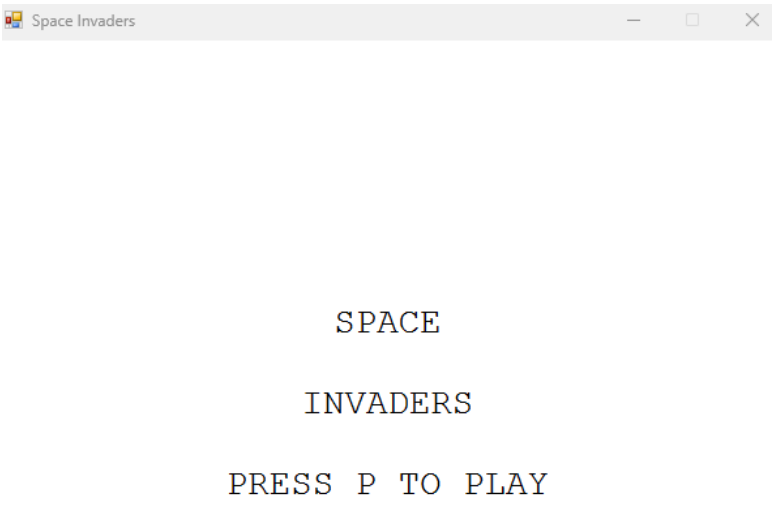
La programmation orientée objet offre une abstraction naturelle des entités du jeu, rendant leur modélisation intuitive. Par exemple, chaque vaisseau peut être considéré comme une instance de la classe "GameObject", héritant des caractéristiques communes tout en pouvant étendre ses fonctionnalités spécifiques. Cela favorise la compréhension du code et permet des modifications ciblées, améliorant la maintenabilité du programme.

De plus, la POO facilite la gestion des interactions entre les objets du jeu. Les collisions entre les missiles, vaisseaux, bunkers, etc., peuvent être gérées de manière élégante en exploitant les relations objet-objet. Les méthodes définies dans les classes encapsulent les règles spécifiques à chaque interaction, simplifiant ainsi la logique du jeu.

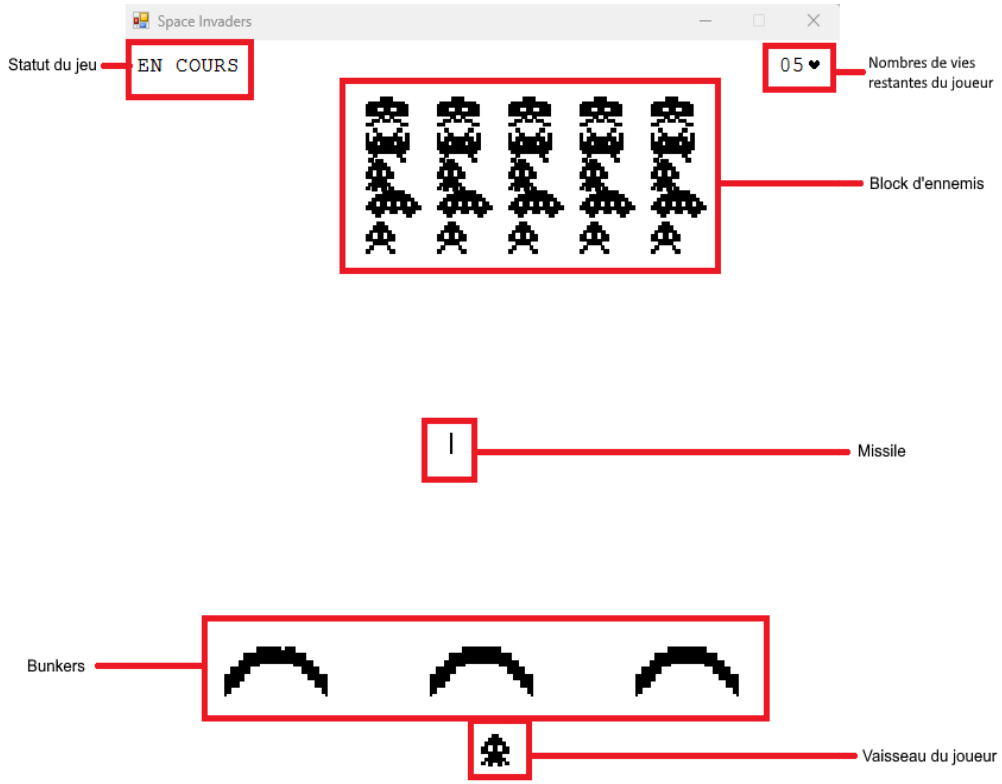
Structure du programme

Présentation de la page de jeu

Sur le menu :



En jeu :



En cas de victoire :

YOU WON

PRESS SPACEBAR TO RESTART THE GAME

PRESS P TO GO TO MENU

En cas de défaite :

YOU LOST

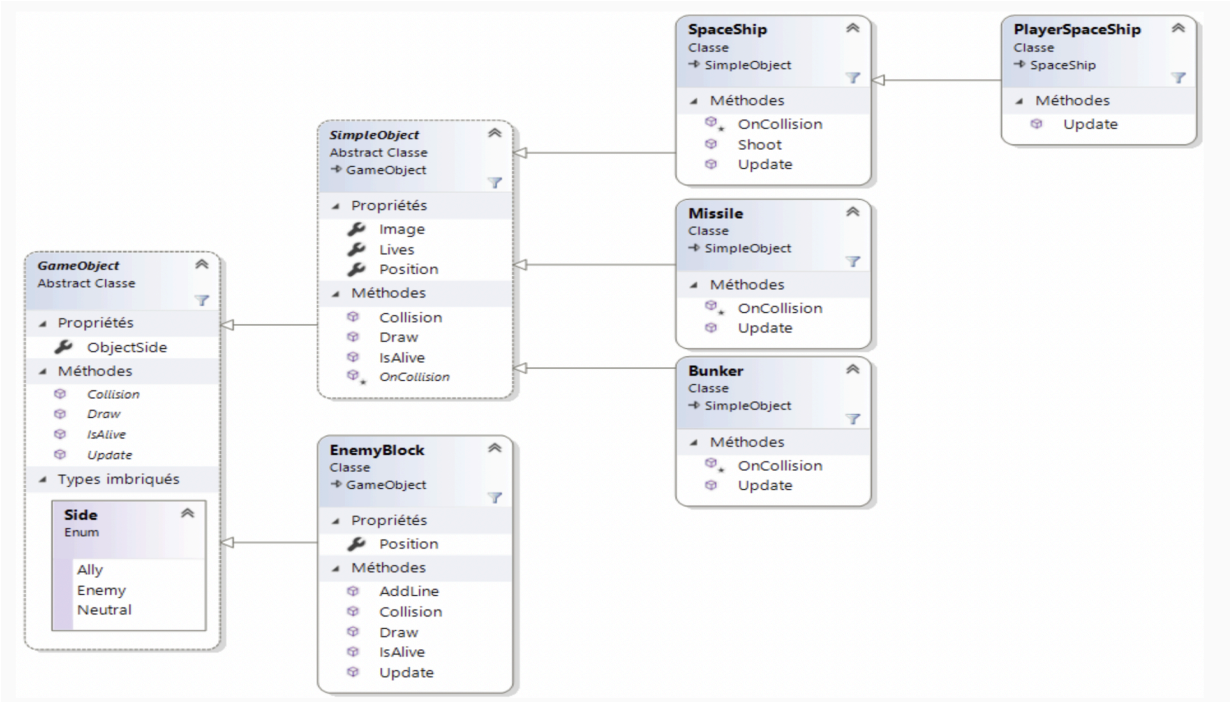
PRESS SPACEBAR TO RESTART THE GAME

PRESS P TO GO TO MENU

Michel CHEN / Valeria ASMAT

Architecture du programme

Pour la réalisation du projet nous avons suivi la piste verte et l'architecture s'avère la suivante:



Source: Piste Verte. Cours: POO. Url: https://perso.esiee.fr/~perretb/I3FM/POO1/projet/pisteVerte.html#friendly-fire

La classe mère GameObject est la classe qui va contenir toutes les méthodes abstraites afin de prendre SimpleObject et EnemyBlock :

- Collision : Cette méthode sera utilisée vérifier si l'Objet entre en collision avec un missile et si oui, cela retournera le nombre de pixels qui sont entrés en collision afin de soustraire le nombre de pixels a la vie du missile et de l'objet concerné.
- Draw : Cette méthode sera utilisée pour chaque objet pour les afficher sur l'écran utilisateur.
- IsAlive : Cette méthode permettra de vérifier si EnemyBlock et SimpleObjet ne sont pas mort (redéfinition de IsAlive dans ceux- ci).
- Update : Cette méthode permet de vérifier le statut de la partie, des touches, la position de l'objet, de créer des actions aléatoires pour les ennemis. Les vérifications mettront à jour les valeurs de la classe concernée.

La propriété ObjectSide permet de définir le type d'Objet créé. (L'appartenance à un des camps des objets créés. Cela permettra de pouvoir détecter la provenance de chaque objet).

La classe SimpleObject définit les variables de base (une image, le nombre de vie et la position de l'objet créé) nécessaires à la création d'un missile, d'un bunker (abris) ou encore d'un Spaceship (vaisseau).
Il est aussi composé de la méthode abstraite OnCollision qui sera adaptée en fonction de la classe Fille liée.

La classe EnemyBlock définit un HashSet (un ensemble de valeurs) qui va contenir tous les ennemis de type Object faisant référence à la classe Spaceship.
Un Spaceship sera par défaut un ennemi.
Les méthodes Collision, Draw, IsAlive, Update seront outrepassé afin de les adapter au cas précis de la classe.

Principaux défis et problèmes rencontrés lors du développement

En développant le jeu, on a rencontré différents problèmes, en l'occurrence:

1. Collision pixel à pixel: S'avère la fonction la plus complexe du programme, cette fonction permet d'évaluer la collision entre deux objets et vérifier si les pixels des images s'intersectent. La difficulté reposait surtout sur la définition du système de coordonnées pour comparer les pixels de chaque image. L'ordre de conditions à établir était primordial et on a dû tester plusieurs cas pour arriver à la solution. D'ailleurs, on a dû comprendre le système d'affichage des images pour arriver à changer la couleur en cas de collision (de noir à blanc).
2. Problèmes de performance liés à la destruction des missiles: Après avoir finalisé la fonctionnalité de tirs de missiles, on a constaté un problème de performance. Après une dizaine de tirs, le jeu s'avérait lent et le jeu commençait à faire des erreurs. Le problème c'était que le missile disparaissait du jeu mais seulement en affichage, pas en tant qu'objet. Ceci signifiait que les missiles restaient en mémoire, et , étant donné qu'un des objectifs était le tir de missiles, la quantité stockée était importante. On a dû éliminer les missiles, pas seulement les faire "disparaître".
3. Tirs aléatoires des missiles: Le block ennemies possède la fonction d' effectuer tirs de missiles de façon aléatoire. Au moment d'implémenter ce point, on a vu qu'à chaque lancement, il n'y avait un seul missile mais une quantité pareille à la quantité d'ennemies. On a dû créer deux objets random pour gérer l'aléatoire de tirs mais aussi l'aléatoire de qui fait le tir. La difficulté reposait sur comment définir l'objet random, car il devait se positionner de manière global et pas à chaque appel d'un numéro aléatoire.
4. Création du bloc d'ennemies: il fallait créer des rangs des ennemies de façon carré, on pourrait améliorer le jeu en créant des rangs supplémentaires chaque fois que le bloc avance vers le joueur. Cela nous pose le défi de pouvoir les générer automatiquement. Pour le développement fait, on est resté sur une mise statique du bloc.

Addons

1. Un son différent a été rajouté lors du tir d'un ennemi ou du joueur:
   Nous avons utilisé la librairie System.Media (librairie du système d'exploitation) afin de jouer des sons.

2. Une image d’explosion est affichée un bref instant, lors de la mort d’un vaisseau ennemi.

3. Menu d’accueil lors du démarrage du jeu.

Possibles améliorations et conclusion

La création du jeu nous a posé des défis qui nous ont permis de mieux comprendre la programmation orientée objet. En faisant le projet on a pu mettre en pratique les concepts théoriques déjà réalisés et évalués lors du QCM. Néanmoins, on trouve qu' il y existe plusieurs possibilités d'amélioration du jeu, en l’occurrence:

1. Mettre le jeu en couleurs: On pourrait changer l'apparence des sprites pour donner une UI plus agréable aux utilisateurs. Les couleurs pourraient attirer beaucoup plus.
2. Mettre des sons supplémentaires: Les sons pourraient rendre le jeu plus interactif et engageant. On pourrait rajouter des sons en cas de défaite ou en cas de réussite par exemple.
3. Création de niveaux: les joueurs pourront s’affronter à des dispositions du jeu plus difficiles a chaque réussite.

Ces exemples sont compatibles avec le développement fait jusqu’à maintenant. En tant que étudiants avec plus d'expérience avec C# depuis le début du semestre, ce serait une continuation intéressante du jeu.