LE MAGNY Thomas
IRLES Antonin
Groupe 2 E3FI

ESIEE
PARIS

Thomas LE MAGNY - Antonin IRLÈS - G2I_3I-PR3

Tables des matières :

- ➢ Description du problème
- ➢ Description de la structure du programme
- ➢ Présentation des problèmes soulevés par les tests

Description du problème :

Dans le cadre de l'unité 3I-PR3, nous avons eu pour projet de reproduire un des grands classiques du jeu vidéo : Space Invaders ! Space Invaders est un shoot'em up" où le principe est de détruire des vagues d'aliens au moyen d'un canon en se déplaçant horizontalement sur l'écran. Pour ce faire nous avons utilisé le langage de programmation orientée objet, C#. Nous avons implémenté notre code selon le cahier des charges de la piste verte.

Description de la structure du programme :

Nous avons développé des fonctionnalités de sorte à ce que nous ayons toujours un programme fonctionnel. Au lieu de chercher immédiatement la structure idéale de notre programme, nous avons réécrit le code au besoin.

Nous revenons donc plusieurs fois sur le code généré pour l'améliorer. Construire un modèle qui évolue dans le temps est généralement moins risqué que d'essayer d'imaginer le modèle idéal dès le départ.


Nous avons commencé par développer la classe spaceship qui affiche un vaisseau pour le joueur à l'écran. Cette classe hérite de la classe GameObject, elle est la classe mère de tous les objets que nous allons utiliser dans ce projet. Toute la gestion des déplacements du joueur sera faite dans la méthode Update.

Nous avons ensuite créé la classe missile, qui permet au joueur de tirer. Elle implémente les méthodes abstraites de la classe GameObject Update, Draw, IsAlive. Ces méthodes sont situées dans cette classe car elles sont communes à toutes les classes filles.

Suite au constat que de nombreux éléments se répétaient dans les classes Spaceship et Missile, nous avons dû les refactoriser. Nous avons pour cela extrait une classe mère commune que nous avons appelée SimpleObject.

Nous permettons au joueur de se mettre à couvert grâce à des Bunker mais ces derniers peuvent être détruits par les missiles. Afin que cela soit possible nous avons créé la classe Bunker.

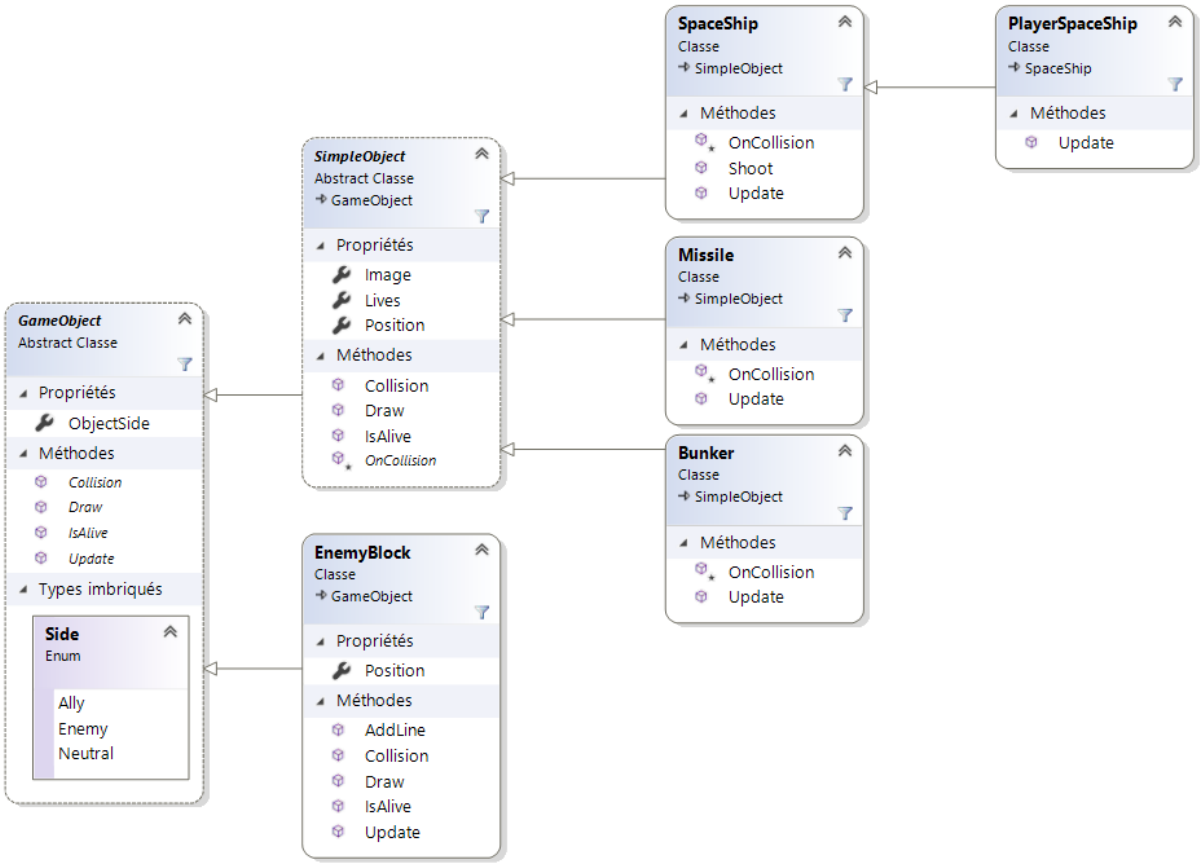
Nous pouvions lors de la création de la classe Bunker tirer des missiles, mais ceux-ci traversaient les bunkers. Nous avons donc dû gérer les collisions. Pour cela, nous avons ajouté la nouvelle méthode abstraite dans la classe GameObject. De sorte que chaque missile test si il serait en collision avec les autres objets du jeu. La redéfinition de cette méthode dans la classe Bunker permet d'effacer les pixel en collision et de retirer autant de nombre de vie au missile.

Les ennemis dans Space Invaders ont un comportement assez particulier : ils arrivent via une rangée de vaisseaux spatiaux identiques, et ils se déplacent de manière synchronisée. Ainsi, une fois qu'un navire atteint le bord de l'écran, tous les ennemis descendent et inversent leur sens de déplacement horizontal. En fait, nous pouvons observer que ce qui compte, c'est le rectangle englobant de tous les conteneurs.

Pour gérer cette situation, nous allons créer une nouvelle classe EnemyBlock pour gérer un groupe d'ennemis. Cette classe représente un GameObject, mais ne correspond pas à un objet simple (elle n'est pas représentée par un sprite) : elle héritera donc de GameObject.


Dans Space Invaders, les tirs ennemis peuvent endommager un bunker ou le vaisseau du joueur, mais pas un autre vaisseau ennemi : en d'autres termes, pas de tir ami. Nous avons donc créé une énumération Side composée des trois valeurs Ally, Enemy et Neutral.

Voici ci-dessous, la structure du code final demandé que nous avons respecté :



Nous avons ensuite ajouté plusieurs addons, afin de rendre notre jeu plus amusant. Ceci vont modifier la structure général du code avec l'ajout de nouvelle classe tel que Enemy, Bouclier, Coeur, ou encore des windows form tel que StartMenu et Shop.

Nous permettons donc via un menu de démarrage d'accéder au shop où l'utilisateur peut choisir le design de son joueur. L'utilisateur peut également rentrer son pseudonyme dans le champ prévu dans le menu. Cela permet d'alimenter notre classement des meilleurs scores, que nous avons également ajoutés et qui est stocké dans un fichier "HightScore.txt". Le meilleur score est d'ailleurs affiché sur le menu de démarrage. Tous ces menus se font à l'aide de Windows Form. Nous pouvons enfin revenir à l'écran de démarrage lorsque le jeu est en pause ou terminé. Pseudonyme et skin du joueur sont aussi stockés dans un fichier "Skin.txt" afin de faire la transition entre le menu de démarrage et de jeu.

Nous avons ajouté un système de niveau configuré dans un fichier "Level.txt". Pour chaque niveau on peut choisir l'EnnemyBlock, le nombre de bunker et la vitesse du vaisseau. Il y a 5 niveaux au total (valeur modifiable) et à la fin de ces 5 niveaux, un bonus est ajouté au score en fonction du nombre de vies et d'un timer (temps de jeu) puis le score total est sauvegardé dans le fichier des scores. D'ailleurs, si le temps de jeu dépasse les 500 secondes, le joueur a perdu.

Nous avons revu tout le design du jeu afin de le rendre plus esthétique. Les différentes actions de l'utilisateur et du jeu émettent désormais des sons correspondants à l'action même. Les vaisseaux ennemis sont désormais animés à l'aide de sprite que l'on utilise d'ailleurs aussi pour animer une explosion lorsqu'ils meurent. Les vaisseaux ennemis vont aussi se déplacer selon un timer, et un temps max qui diminuera afin qu'ils aillent plus vite.

Pour finir nous avons ajouté différents bonus. Le premier est un bouclier qui protège le vaisseau de l'utilisateur des tirs adverses durant un temps imparti. Le second est un cœur qui permet d'ajouter une vie supplémentaire au joueur dans le cas où il en aurait moins de 5 (nombre de vie de base). Ces bonus apparaissent aléatoirement après la mort d'un vaisseau ennemi.


Présentation des problèmes soulevés par les tests :

Nous avons fait face à quelques difficultés lors de la mise au point de notre projet, nous allons les citer ci-dessous, ainsi que la démarche que nous avons accomplie pour les régler.

Tout d'abord, la superposition de sons superposés était impossible nativement. Pour cela nous avons ajouté une référence à ***Windows Media Player***, en pointant vers le fichier *wmp.dll*.

Ensuite, la détection de la collision, nous a paru complexe. Pour cela à plusieurs reprises améliorer notre fonction et vous déboguer notre code pour trouver nos failles, nous avons ainsi comparé chaque pixel des deux objets passés en paramètres.

Un autre des obstacles que nous avons rencontré a été le déplacement du bloc d'ennemis, notre première difficulté a été la mise à jour de la taille de celui-ci, mais aussi l'action réalisée après la collision avec les limites de l'écran.

Enfin nous avons eu des difficultés pour passer du jeu au menu de démarrage, puisqu'après le jeu restait dans l'état dans lequel on l'avait laissé. On a donc dû mettre une boucle dans la fonction main du programme qui relance le menu de démarrage tant qu'on demande de retourner au menu.