ESIEE PARIS

3I-PR3 - Programmation Orientée Objet

Rapport de projet :
Space Invaders

Zackary Saada
Antonin Mansour

TABLE DES MATIÈRES

Zackary Saada
Antonin Mansour

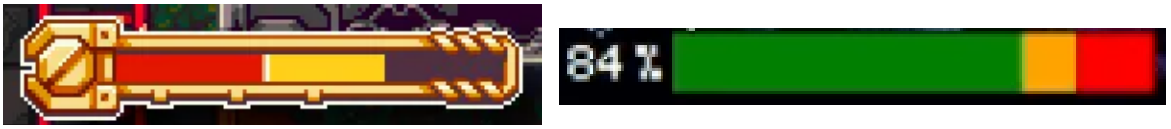
I. INTRODUCTION

Le but du projet est de reproduire un jeu culte : Space Invaders. C'est un jeu de type shoot 'em up ou le but est d'éliminer des vaisseaux aliens qui arrivent du haut de l'écran, notre vaisseau se déplaçant en même temps horizontalement en bas de l'écran.

Le projet est réalisé entièrement en C#. Parmi les propositions de parcours disponibles, notre choix s'est porté sur la piste verte. Cela nous à permis d'être guidé sur la structure du programme, afin de pouvoir imaginer davantage d'extensions.

Voici les extensions que nous avons implémentées au delà de la piste verte suivie :

- Le fond d'écran personnalisé, ainsi que les objets et entitées colorés.
- Le score gagné par le joueur quand il élimine un ennemi, qui correspond à ses points de vie initiaux.
- Une musique de menu, de jeu, et en cas de victoire ou défaite.
- Une difficulté qui incrémente au fil des vagues : Lorsqu'on gagne, on a la possibilité de rejouer ou de retourner au menu. Quoi qu'il en soit, lors de la prochaine vague d'ennemis, les vaisseaux ennemis sont différents, tirent plus fréquemment et ont un nombre de points de vie légèrement plus grand.
- Les bonus : quand un ennemi meurt, il y a 15% de probabilité qu'un bonus apparaisse. Celui-ci peut être :
    - Un bonus de vie : s'il touche le joueur ou que le joueur tire dessus, il gagne 50 points de vie, jusqu'à un maximum de 200.
    - Un bonus missile : s'il touche le joueur ou que le joueur tire dessus, le compteur de super missile augmente de 1 (ce compteur est représenté en haut à gauche). Si le joueur a des super missiles en stock, il peut appuyer sur <flèche du bas> pour appeler un missile en haut de l'écran, qui descend jusqu'en bas en détruisant les ennemis au passage.
- Le calcul des dégâts d'un missile est désormais effectué au prorata des entités rencontrées sur la route. En effet, un missile peut traverser le bunker ou un vaisseau s'il possède assez de points de vie. Le missile perd un nombre de points de vie équivalent aux pixels détruits sur le bunker, ou au nombre de points de vie du vaisseau, et continue sa route pour continuer à blesser d'autres entités.
- Une barre de vie, au lieu d'un simple nombre. Celle du joueur a une valeur de 200 mais est affichée en pourcentage. Lorsque le joueur se fait toucher, ses points de vie sont réduits du nombre de points de vie du missile et une animation a lieu avec les points de vie qui viennent d'être perdus en orange, comme dans certains jeux d'action. Cela crée un visuel plus agréable au lieu d'avoir un changement instantané.



A gauche un exemple de barre de vie, à droite notre implémentation

- Un menu : il est navigable via les flèches <up> et <down> et l'élément sélectionné est affiché dans une couleur différente. Il possède trois sections
    - Une section Play, qui envoie sur le jeu.
    - Une section Controls, qui affiche les contrôles du jeu.
    - Une section High Score, qui affiche le record de la session. Celui-ci est réinitialisé à la fermeture du jeu.

Lorsque le joueur lance la partie, il peut utiliser les flèches de <gauche> et de <droite> pour se déplacer et la <flèche du haut> pour tirer un missile. S'il a récupéré un bonus super missile, il peut appuyer sur <flèche du bas> pour le déployer.

En bas à droite, se trouve la barre de vie, et en bas à droite le nombre de points. Les points sont calculés en fonction du nombre de points de vie des vaisseaux ennemis, et des points de vie du joueur à la fin d'une vague.

Si le joueur remporte une vague en éliminant tous les ennemis, son score s'affiche et il peut appuyer sur <espace> pour lancer une nouvelle vague ou <entrée> pour faire une pause et retourner au menu principal. Dans tous les cas, il regagne 50 points de vie, et son stock de super missiles est conservé et les ennemis deviennent plus forts.

Si il perd, son score s'affiche aussi, et il peut appuyer sur <entrée> pour retourner au menu principal. Tout est alors réinitialisé : vie, score, stock de missile, numéro de vague, etc…

II. STRUCTURE DU PROJET

Le projet s'articule autour de plusieurs classes / fichiers. Voici la présentation des principales.

- Une classe Game, partie centrale du projet. C'est ici que toutes les autres classes seront instanciés. Elle contient également la méthode Update qui s'occupe des interactions avec le joueur et qui appelle les méthode update des autres classes, et de la classe Draw, qui s'occupe d'afficher tous les éléments à l'écran.

- Une classe Menu, qui s'occupe de la gestion des états du jeu (Pause, Menu, Play…) ainsi que de gérer la navigation dans les menus visuel de l'application.

- Une classe abstraite GameObject, qui est la classe mère de toutes les entités du jeu (projectile, vaisseau, bunker…). C'est ici que les méthodes importantes sont définies de manière abstraite : la méthode Update, qui s'occupe des changements d'états de l'entité, la méthode Draw, qui affiche l'entité, la méthode Collision, la méthode IsAlive (leurs noms sont assez explicites) et le camp de l'entité (neutre, ennemi ou allié).

- Une classe SimpleObject, qui implémente les méthodes communes des entités, notamment la méthode Collision, IsAlive et Draw.

- Une classe SpaceShip, qui représente le vaisseaux, qu'ils soient ennemis ou alliés (vaisseau du joueur). Elle va gérer le tir de missiles, ainsi que la gestion une fois que la collision à été repérée.

- Une classe EnemyBlock qui réunit tous les vaisseaux ennemis de la partie et gère leurs déplacements conformément aux règles de Space Invaders

- Une classe PlayerSpaceShip, qui va gérer le déplacement du joueur, ainsi que l'affichage de la vie du joueur.

- Une classe Projectile, qui va gérer le déplacement du projectile, la gestion si le missile sort de l'écran, ainsi que le test de collision avec les autres entités. cette classe est la classe mère des autres projectiles, qui sont au nombre de 3 :
    - Missile, tiré par le joueur ou par les ennemis, il appartient au camp allié ou ennemis, il entre en collision avec le camp adverse.
    - BonusLife, qui apparaît aléatoirement lorsqu'un ennemi meurt. Il descend, rentre en collision uniquement avec le joueur.
    - Bonus Missile, qui apparaît aléatoirement lorsqu'un ennemi meurt. Il descend, rentre en collision uniquement avec le joueur.

- Une classe Bunker, qui affiche des pixels qui bloquent les tirs, ainsi qu'une classe Trigger qui permet de désactiver ces derniers

III. PROBLÈMES RENCONTRÉS

Au cours du développement de notre projet, les tests nous ont permis de soulever plusieurs choses. D'abord, nous avons dû régler les problèmes d'équilibrage (vie du joueurs, vie des ennemis, dégâts du joueur, dégâts des ennemis ou encore bonus en partie et bonus en fin de partie) rendant le jeu trop facile ou trop difficile. De plus, ces tests nous ont permis de soulever plusieurs bugs.

Des problèmes sont apparus quand nous avons créé les bonus. Fonctionnellement parlant, les bonus sont des projectiles qui partent du haut de l'écran, et qui ne rentrent en collision qu'avec le camp allié. La solution trouvée à été de créer un camp Bonus, qui ne rentrerais en collision qu'avec les alliés. Une limitation est apparue : si le joueur tirait sur le bonus, ce dernier mourrait et rien ne se passait. Cela vient du fonctionnement des collisions dans notre code. À ce moment, la collision entre le bonus et le joueur était vérifiée dans la classe bonus, au sein de la méthode Collision, la condition pour entrer en collision était : le camp de l'entité est allié, et alors on augmente sa vie. Cependant, si l'entité touchée était bien un allié, mais qu'il s'agissait d'un missile allié, c'était ce dernier qui voyait sa vie augmenter. La solution à alors été de gérer ce comportement dans une méthode à part, appelé au moment de la mort des entités (handleEnnemieDie() pour la génération des bonus, et handleBonus pour l'application des bonus).

Nous avons voulu ajouter des sons pour les tirs et leur impact sur un vaisseau. Nous nous sommes rendus compte qu'il était impossible pour le module System.Media de jouer plusieurs sons en même temps, ce qui causait de nombreux bugs audio. Nous avons décidé de supprimer cette fonctionnalité pour ajouter des musiques d'ambiance à la place, qui ne seraient jouées qu'une seule à la fois.

Lors de la création du menu, nous nous sommes rendus compte que notre classe Game n'était pas très lisible. Nous avons donc décidé de déplacer beaucoup de fonctionnalités de navigation et de gestion des états du jeu dans la classe Menu, laissant la classe Game gérer le côté fonctionnel du jeu. Nous avons également dû se décider sur la logique des menus, si on continuait de jouer après avoir gagné ou pas, ce qui a causé quelques conflits à résoudre lors de la fusion de branches entre elles.

Un autre problème à été pour les fin de partie, quand le bloc ennemi arrive au niveau des bunkers. Initialement, nous avions prévu d'activer les collisions entre les ennemis et les bunkers, ce qui aurait produit une destruction progressive des bunkers. Malheureusement, ce fonctionnement causait trop de ralentissements. Nous avons donc opté pour une autre solution : ajouter une ligne (trigger dans le code), qui une fois franchie par le bloc, fait disparaître les bunkers.

IV. ÉVOLUTIONS POSSIBLES

Quelques fonctionnalités que nous aurions voulu implémenter et quelques bugs connus subsistent dans notre jeu, qui malheureusement ne peuvent pas être réglés dans le temps imparti. Il est important pour nous de noter que nous avons passé de très bons moments à imaginer des extensions toujours plus ambitieuses les unes des autres, et que si nous ne nous étions pas imposés des limites nous serions toujours à développer ce projet bien plus tard. Parmi les fonctionnalités que nous voulions ajouter :

- Beaucoup de bonus supplémentaires, par exemple la possibilité de tirer plusieurs missiles avant que le premier ne sorte de l'écran ou ne touche un ennemi, un bonus de points de vie permanents en tant que "bouclier", des missiles explosifs, etc.
- Des bonus permanents à choisir lors de la complétion d'une vague, qui resteraient d'une vague à une autre et qui seraient tous effacés lors d'une défaite. C'est un système qui peut faire penser au genre de jeu vidéo rogue-like.
- Un menu paramètres ("Settings") avec des préréglages pour la difficulté, et la possibilité d'activer ou désactiver certains bonus. Le score serait multiplié en fonction de la difficulté choisie.
- Des améliorations d'interface et de beauté du jeu.

Voici les bugs connus que nous n'avons pas eu le temps de régler :

- Les bonus n'apparaissent parfois pas au bon endroit lors de l'élimination d'un vaisseau ennemi.
- Il est possible d'envoyer des super missiles dans le menu Pause ou dans les menus Gagné/Perdu. Ceux-ci sont envoyés à la sortie du menu Pause, ou perdus dans le deuxième cas.

Voici le diagramme de classe de notre application, dans le cas où nous voudrions greffer de nouvelles classes et objets :