Rapport du projet « Space Invaders »

Description du projet

Le jeu « Space Invaders » est un jeu de combat de vaisseaux. L'utilisateur dirige un vaisseau, positionné en bas de l'écran. Il peut lui faire se déplacer horizontalement avec les flèches « gauche » / « droite » de son clavier. L'objectif est de détruire l'ensemble des vaisseaux ennemis de l'écran en tirant via la touche « espace ».

Ces derniers apparaissent depuis le haut de l'écran en début de partie, puis se déplacent à droite. Une fois qu'un vaisseau a atteint l'une des limites de la fenêtre, l'ensemble des vaisseaux se rapprochent de l'utilisateur, augmentant légèrement leur vitesse de déplacement et changent de direction. Les ennemis tirent sur l'utilisateur de manière aléatoire. Certains vaisseaux sont plus résistants que d'autres ou se déplacent plus vite.

Entre les deux, des bunkers jouent un rôle de protection. Ils absorbent les projectiles jusqu'à un certain point, car ils disparaissent au fur et à mesure d'entrer en collision. Ils sont à la fois sensibles aux projectiles des vaisseaux ennemis qu'à ceux du joueur. On notera que deux projectiles provenant de deux camps différents se touchent, ceux-ci s'annulent et disparaissent en conséquence.

Le jeu dispose d'un système de sons. Tout au long du programme, on diffuse une playlist de musiques d'arcade en guise de fond. Les actions des vaisseaux (tirs, dégâts, destructions) génèrent aussi des effets sonores éphémères.

D'un point de vue graphique, on affiche les éléments du jeu au-dessus d'une image de fond d'écran. Ceux-ci peuvent être statiques ou bien animés. Les éléments graphiques utilisés reprennent l'univers du célèbre jeu « Pacman ». L'utilisateur est ainsi « Pacman », les vaisseaux ennemis sont des fantômes colorés et les fonds d'écrans sont reprennent le thème sombre de « Pacman ».

Le jeu peut être mis en pause puis reprendre via l'appui sur la touche « P ».

La structure du programme

Structure globale

Le projet est organisé sous la forme MVC :

- Modèle : La logique du jeu (les vaisseaux, les missiles, les bunkers, les projectiles, etc.) ;
- Vue : Relatif à l'affichage (les images, les animations et les sons) ;
- Controller : Lance le jeu et contrôle son état (la boucle principale du jeu) ;

La boucle principale

La gestion des états

La boucle principale du projet se situe dans la classe « Game ». On gère en fait les différents états du jeu (menu, jeu, pause, victoire, défaite), via la classe « GameState ». En clair, « GameState » permet de récupérer l'état actif et de le changer si besoin.

Par exemple, si le vaisseau de l'utilisateur a été détruit, la classe User avertit « GameState », qui se met à jour, puis appelle la méthode appropriée de Game afin de réagir au changement d'état. Dans ce cas, « GameState » lance « SwitchToEnd », qui supprime tous « GameObjects » et affiche le fond d'écran associé.

La gestion du jeu

Chaque objet que l'on doit dessiner et mettre à jour dans le jeu doit hériter de la classe abstraite « GameObject ». L'ensemble des éléments du jeu à prendre en compte sont en fait contenus dans la liste « GameObjects » de la classe « Game ». C'est sur cette classe générique que repose toute la logique du jeu, et en particulier sur les 3 méthodes suivantes :

- « Update » : Mets à jour un objet (ex : réagir à un appui sur une touche du clavier, se déplacer, effectuer une action, etc.) ;
- « IsAlive » : Retourne « True » si l'objet est en vie. Sinon, on supprime cet objet de la liste des « GameObjects » et celui-ci ne sera plus pris en compte ;
- « Draw » : Permet de dessiner l'objet sur la fenêtre du jeu « Space Invaders » ;

En clair, la boucle du jeu revient à parcourir tous les objets de « GameObjects » en continu, d'appeler « Update » pour mettre en action la logique du jeu, « Draw », pour dessiner les objets, puis supprimer ceux qui ne doivent plus être pris en compte via l'appel de « IsAlive ».

La gestion du graphisme

La classe abstraite « DrawableObject » représente un « GameObject » qui doit être affiché par le biais d'un « Drawable ». Cette classe retourne en fait une image à dessiner sur un canvas. Il peut s'agir d'une image statique (figée dans le temps), représentée ici par un « Frame » ou encore d'une animation, gérée par la casse « Animation ». Cette dernière découpe en fait un fichier image (initialement sous forme de matrice N * M) en plusieurs images, et permet d'afficher une animation.

La gestion des objets du jeu

Comme dit précédemment, les objets du jeu héritent tous de la classe abstraite « GameObjects ». Ceux-ci sont gérés via le principe d'héritage et de classes abstraites.

On distingue plusieurs niveaux d'héritage :

- « GameObject » : tous les objets du jeu ;
- « DrawableObject » : les objets à afficher via des images (ou animations) ;
- « AliveObject » : les objets qui ont une vie ;
- « MovingObject » : les objets qui se déplacent (horizontalement et / ou verticalement) ;
- « ShooterObject » : les objets qui sont apte à tirer ;

Le joueur

Le vaisseau du joueur est géré par la classe « User ». Celle-ci hérite directement de la classe « ShooterObject ». Il n'y a rien de spécifique à redéfinir à part les actions à réaliser en fonction des touches (tirs et déplacements).

Le vaisseau des ennemis

Les vaisseaux des ennemis héritent de la classe abstraite « EnnemyObject ». Celle-ci gère les tirs des ennemis et également quelques spécificités liées à leur déplacement par système de bloc. Afin de factoriser quelques lignes de code, il est possible d'utiliser des classes abstraites qui héritent de « EnnemyObjects ». C'est notamment le cas de « BasicEnnemy », qui regroupe les vaisseaux ennemis dont seul quelques propriétés d'affichage diffèrent.

Les bunkers

Le bunker est un cas particulier dans la gestion des éléments du jeu. En effet, celui-ci ne se déplace pas, ne peut pas tirer, et n'a pas de vie. Il hérite donc directement de la classe « DrawingObject ».

Les projectiles

Les projectiles sont gérés par la classe « ProjectileObject », qui hérite de « MovingObject ». On gère dans cette classe abstraite les caractéristiques communes des projectiles comme les comportements lors d'une collision ou encore leur mouvement vertical. Cette dernière donne les deux types de projectiles : « UserProjectile » et « EnnemyProjectile ».

Les composants graphiques

Certains « GameObjects » n'ont pas d'influence sur la logique du jeu. C'est notamment le cas de « BackgroundImage » ou encore de « UserLife », qui ont un rôle purement graphique. Ceux-ci héritent donc directement de la classe « DrawableObject ».

La gestion des sons

On utilise des « MediaPlayer » afin de gérer une playlist de morceaux ainsi que des effets sonores. Un « MediaPlayer » est spécifique à chaque son. On utilise donc une classe spécifique « SongMap », qui contient toutes ces instances et permet de les jouer facilement.

Les effets sonores des « GameObject » sont gérés via la classe « SoundHandler ». En clair, on associe un « SoundHandler » à chaque « GameObject ». Celui-ci possède des méthodes utilitaires de lancement de son, qui sont appelées aux bons moments dans les classes « AliveObject » et « ShooterObject ».

Aperçu graphique

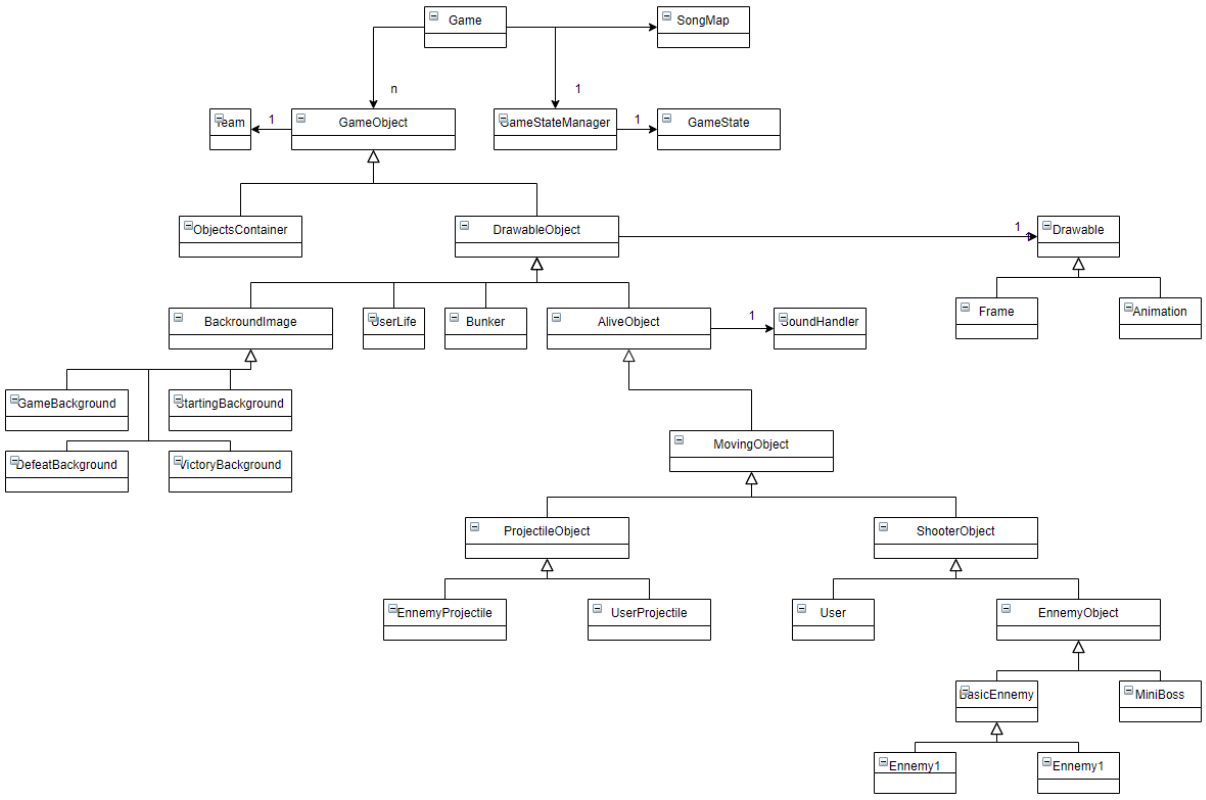

Diagramme permettant de visualiser les relations entre les différentes classes du programme

Les difficultés rencontrées

Dans l'ensemble, le projet s'est très bien passé. Je ne connaissais pas le C#, mais je n'ai pas été dérouté pour autant. Au contraire, ce projet a été l'occasion de bien comprendre son fonctionnement et je pense maintenant maîtriser les concepts fondamentaux du langage. Globalement, j'ai été plutôt à l'aise dans la réalisation du projet. Cependant, j'ai rencontré quelques difficultés.

La gestion des sons

Le principal problème que j'ai rencontré consistait en la gestion du son avec « MediaPlayer ». En effet, dès l'instant ou je jouais un son avec un « MediaPlayer », la fenêtre du jeu se réduisait et le titre de la fenêtre était tronqué. Le programme ne plantait pas, et il n'y avait pas de warnings sur la console. J'ai longuement essayé de débugger le problème en utilisant l'exécution en mode « pas à pas », mais en vain.

En faisait des tests, j'ai réussi à résoudre le problème en créant une nouvelle instance de « MediaPlayer » avant d'afficher graphiquement la fenêtre de jeu. Une simple insertion de « MediaPlayer media = new MediaPlayer() » avant d'appeler « InitializeComponent » (dans « GameForm » de la classe « Form1.cs »), permettait de résoudre ce problème.

Afin d'avoir un code plus « propre », c'est donc précisément à cet endroit que j'appelle la fonction « SongMap.Instance.Load », me permettant de charger en mémoire mes « MediaPlayer » avant l'ouverture de la fenêtre.

Pour résumer, afin de résoudre le problème de redimensionnement de fenêtre lors de la création du premier « MediaPlayer », je charge en mémoire les « MediaPlayer » avant sa création.

Le respect des conventions de code

La seconde difficulté rencontrée a été de respecter la convention de code suivante : « une fonction ne doit pas contenir plus de 10 lignes de code ». En effet, certaines fonctions sont complexes et sont difficilement découplables. Pour résoudre le problème, je découpais les « grosses » fonctions en sous-fonctions privées quand c'était possible. Dans le cas contraire j'utilisais des fonctions lambdas.