PROJET SPACE INVADER

MESTRE PIERRE

2019

ESIEE
PARIS

PROJET SPACE INVADER

Semestre 1	**PIERRE MESTRE**	2019

Table des matières

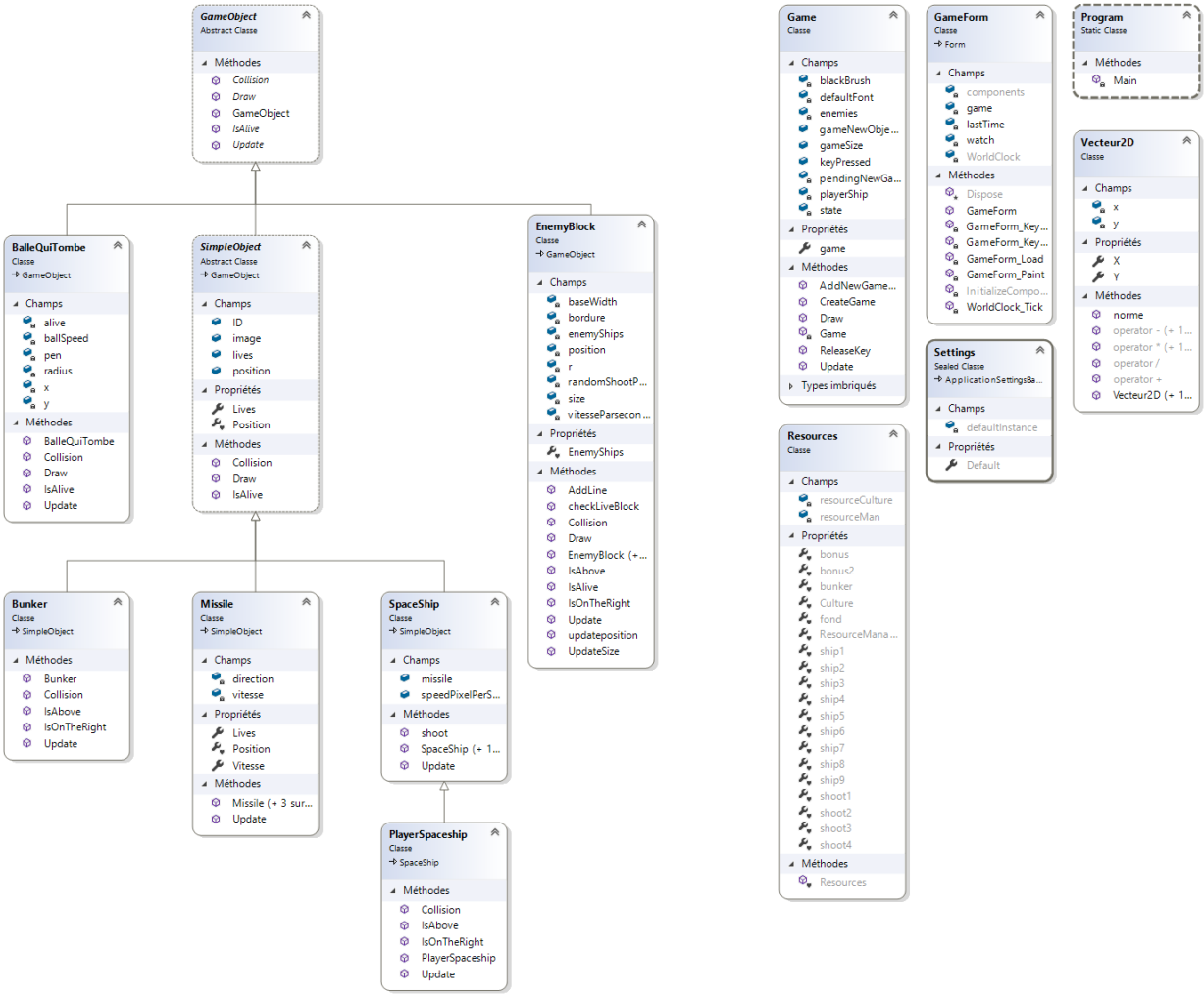
PROJET SPACE INVADER		
Semestre 1	**PIERRE MESTRE**	2019

I. INTODUCTION :

L'objectif de ce premier projet en C# est de reproduire un jeu vidéo, le Space Invader. Le principe de ce jeu est de détruire un block d'ennemie qui se déplace horizontalement sur l'écran, tout en ce rapprochant du joueur.
Une partie est considérée comme gagnée, quand le joueur a détruit tous les ennemis, et qu'il est toujours en vie.
Dans le cas contraire, une partie est considérée comme perdu si le joueur n'a plus de vie, ou si la vague d'ennemis atteint le bas de l'écran.

II. STRUCTURATION DU PROGRAMME :

Voici le diagramme de classes de mon programme montrant les dépendances, les différentes méthodes, et propriétés des classes de mon programme :



Pour coder mon programme je me suis aider de la piste verte du sujet, mais j'ai fait pas mal de hors-piste afin d'adapter la structure de mon programme a ma façon de programmer.

a. Classe SpaceShip

Tous les attributs sont des héritages de la classe SimpleObject.

TYPE	ATTRIBUTS	FONCTION
Double	SpeedPixelPerPerson	Donne une vitesse de déplacement
Vecteur2D	Position	Détermine la position
Int	Lives	Nombres de vies
Missile	Missile	Attribut un missile
Image	Image	Donne une apparence
Int	ID	Donne un ID qui permet de différencier si c'est un Player ou un ennemi

Comme on peut le remarquer, la classe SpaceShip contient une méthode Shoot() qui permets de créer un missile. La direction permet d'indiquer la direction que celui-ci va emprunter (vers le haut ou le bas) en fonction de qui tire le missile. Si c'est un joueur le missile ira vers le haut, si c'est un ennemi le missile ira vers le bas.

Un missile est créé que s'il n'y a pas déjà de missile créé et que le missile précédemment créé est hors champs ou s'il est mort.

Le paramètre ID permet de donner un ID au missile. Cet ID est le même que l'ID du lanceur. Je m'explique. L'intérêt d'attribuer un ID au missile permet de mieux gérer le friendly fire.

LANCEUR	ID LANCEUR	ID MISSILE LANCER
Ennemi	5	5
Player	1	1
Bunker	2	-

Ainsi plus tard dans le programme, quand on appellera la méthode Collision() cela ne fera des dégâts que si l'ID du missile et l'ID de l'objet percuter sont différents.

b. Création du vaisseau du joueur

Pour créer un joueur il suffit d'appeler la classe PlayerShip dans le constructeur de la classe Game, qui hérite de la classe SpaceShip.

c. La classe Missile

Voici les différents attributs de la classe missile qui hérite de SimpleObject :

TYPE	ATTRIBUTS	FONCTION
Double	Vitesse	Donne une vitesse de déplacement
Vecteur2D	Position	Détermine la position
Int	Lives	Nombres de vies

Image	Image	Donne une apparence
Int	ID	Donne un ID qui permet de différencier si c'est un Player ou un ennemi
Int	Direction	-1 vers le haut 1 vers le bas

Les constructeurs de cette classe sont appelés dans la classe shoot() dans PlayerShip et dans EnnemyBlock.

Les différentes méthodes de missile sont Update(), IsAlive() et Draw().

Update() permet de déplacer automatiquement le missile dans une direction donnée. En effet cette direction dépend de qui est le lanceur, un ennemi ou le joueur. De plus le missile regarde constamment si il n'est pas rentré en collision avec un objet.

Draw() permets de dessiner graphiquement le missile.

IsAlive() check constamment si le missile est en vie ou pas. Celui-co ne l'est plus quand il est hors champs ou quand il n'a plus de vie.

d. Déplacement du joueur et tir

La gestion du déplacement du joueur se passe dans la méthode Update() de la classe PlayerShip. On vérifie les entrées du clavier, et on associe une action en fonction de la touche pressée.

TOUCHE	ACTION
Flèche droite	Déplace le joueur sur la droite
Flèche gauche	Déplace le joueur sur la gauche
Espace	Tire un missile

e. La classe Bunker

La gestion des bunkers est gérée dans la classe Game. C'est dans le constructeur de game, que l'on créé les bunkers, et que on les places. Pour les places je prends la taille de la fenêtre de jeux que je divise par 3.

La classe en elle-même est juste un héritage de la classe Simple Object et reprend donc ces attributs.

f. Collisions

La gestion de la collision entre les missiles et les objets a été la partie la plus difficile pour moi. En effet, j'avais des problèmes de coordonnées avec mes ennemis du coup j'ai mis beaucoup de temps à résoudre les problèmes que j'ai pu rencontrer.

Je me suis fais aider par beaucoup de camarades dans cette partie, notamment Thibault qui ma montrer et expliqué comment il a fait (j'ai donc repris la même structure de code pour cette méthode, ils sont s'semblables, mais ce ne sont pas les mêmes).

En ce qui concerne le fonctionnement, cette méthode est appelée dans l'update de la classe Missile.

Elle check s'il les coordonnées des pixels du missile sont à un moment donnée égales avec un pixel noir d'un objet. Si c'est le cas, le missile perd des points de vies et l'objet également.

Si c'est un bunker, les pixels noirs disparaissent, si c'est le joueur il perd des points de vies.

g. Bloc d'ennemis – Création – Déplacement – Destruction

La gestion des ennemis est très simple. Il s'agit d'ajouter dans un espace rectangulaire dédié (un block) différentes lignes d'ennemis.

Sur chaque ligne, on peut trouver la même forme d'ennemi, mais les formes peuvent varier en fonction des différentes lignes.

Toute cette gestion de création et d'ajout est gérer dans le constructeur de la classe Game qui appel la méthode AddLine() de la classe EnnemyBlock.

Chaque ennemi a un certain nombre de vie (dans mon jeu une vie) et tire de manière aléatoire et automatiquement. Un ennemi est un SpaceShip.

Pour gérer le déplacement, on déplace en X chaque ennemi un a un, qui fait déplacer le block. Si les ennemis arrivent sur une bordure de jeu, ils descendent en Y et changent de direction et accélèrent leur vitesse. C'est déplacements sont gérés dans la méthode Update() de la classe EnnemyBlock.

En ce qui concerne la destruction des ennemis, dès qu'un missile rentre en contact avec un ennemi, alors il y a appel de la méthode Collision() associé qui enlève des points de vies a l'ennemi en question.

Si l'ennemi n'a plus de vie il disparait et il est détruit (supprimer de la liste d'objet). Cela est géré dans la méthode IsAlive().

h. Friendly Fire

Pour la gestion du friendly fire, je n'ai pas fait comme sur la piste verte (bien que la réflexion soit semblable). Chaque ShapceShip a un ID en fonction de son statut :

LANCEUR	ID LANCEUR	ID MISSILE LANCER
Ennemi	5	5
Player	1	1
Bunker	2	-

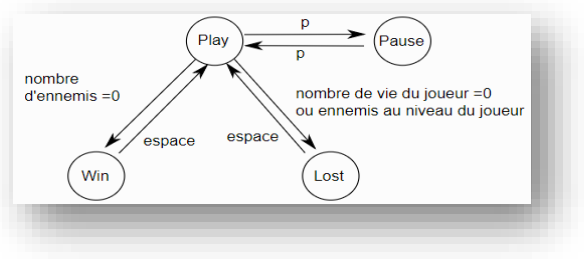
Lorsqu'un missile est créé, il porte le même ID que le SpaceShip qui l'a créé.

Au cours de son déplacement, si le missile porte un ID différent de l'objet avec lequel il est rentré en collision, il y aura des dégâts. A l'inverse si l'ID du missile est le même, il n'y aura pas de dommages.

i. Affichage des vies

L'affichage des vies est très simple, il s'agit d'afficher sur la zone de jeux le contenue de la variable Lives du playerShip.

j. Gestion de l'option Pause et de partie gagnée ou perdue



Le but de cette partie est de pouvoir mettre en pause le jeu à tout moment en appuyant sur la touche P. Aussi le but est d'arrêter le jeu lorsque la partie est perdue ou gagné et d'afficher le texte en conséquence.

Pour cela avec une énumération, le jeu peut se mettre en plusieurs états : PAUSE, PLAY, WIN ; LOSE.

Si le jeu est en pause, win ou lose toutes les actions courantes dans le jeu sont figer ou stopper. Sinon il tourne normalement quand il est en play.

Tous ces changements d'états sont gérés dans le update() de la classe Game.

k. Les plus :

Modification de l'icône du jeu.

III. CONCLUSION :

Développer SpaceInvader n'est pas en soit quelque chose de complexe, mais il faut avoir de bonnes connaissances en C# et être à l'aise avec toutes les notions d'héritage et d'implémentation, ce qui n'étais pas complètement mon cas au début.

Toute fois avec de la persévérance et de l'aide, j'ai réussi a sortir un jeu fonctionnel qui répond au cahier des charges.