PERRIN Baptiste - E3FI Groupe 2

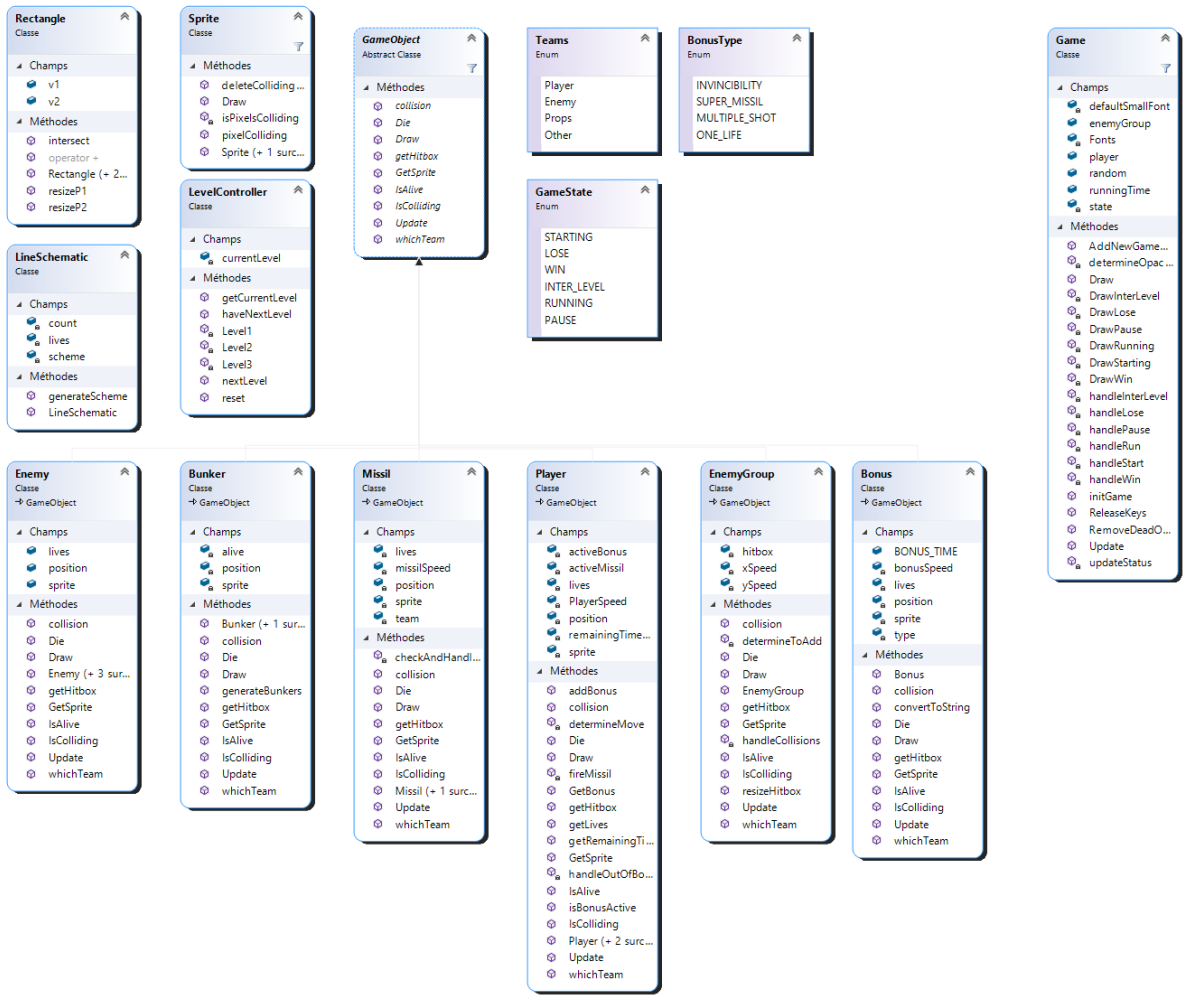
Projet Space Invaders

Projet réalisé par PERRIN Baptiste en 2020 dans le cadre de la matière **Programmation Orientée Objet** de la filière E3FI de l'ESIEE Paris, celui-ci a été effectué en suivant la Piste Bleue.

L'objectif de ce projet était de proposer un jeu vidéo se rapprochant a Space Invaders, en utilisant le langage c# et les principes de base de la programmation orienté objet.

Sommaire

Structure du programme



On peut basiquement les différentes classes du projet en plusieurs catégories:

- **Classes Utilitaires :** *Gère un aspect spécifique du jeu*
    - **Rectangle.cs :** Représente un rectangle, utile pour représenter et gérer les "hitboxs"
    - **LineSchematic.cs :** Représente une ligne d'ennemi, utile afin de générer les niveaux
    - **LevelController.cs :** Permet de gérer les niveaux, les méthodes sont statiques
    - **Sprite.cs :** Représente un "sprite", gère les différentes images d'un élément ainsi que les tests de collisions par pixels
- **Les Enums :** *Servant à n'accepter que quelques valeurs*
    - **Teams.cs :** Représente les différentes équipes auquel un GameObject peut appartenir
    - **GameState.cs :** Représente les différents états du jeu
    - **BonusType.cs :** Représente les différents bonus possibles
- **Les GameObjects :** *Les différents objets avec lesquels on peut interagir*
    - **GameObject.cs :** Classe abstraite, représente une entité en jeu, et les méthodes qu'elle doit avoir
        - **Enemy.cs :** Représente un ennemi dans le jeu
        - **Bunker.cs :** Représente un bunker dans le jeu, sert a intercepter les dégats
        - **Missil.cs :** Représente un Missile dans le jeu (Ennemi ou Allié), pouvant faire des dégâts aux autres entités
        - **Player.cs :** Représente le joueur, celui-ci est contrôlable par l'utilisateur
        - **EnemyGroup.cs :** Représente le bloc d'ennemis du niveau, gérant le comportement de celui-ci
        - **Bonus.cs :** Représente un bonus, donnant un avantage temporaire au joueur si ramassé

Et le fichier principal **Game.cs** contenant plusieurs choses:

- La gestion des GameObjects
- La gestion de l'affichage
- La boucle de jeu principale
- La gestion d'éléments commun au jeu (Random, Temps depuis le lancement, etc.)

Addons

J'ai également ajouté quelques Addons au jeu original.

Animations

Afin de proposer une expérience de jeu un peu plus agréable, j'ai ajouté les animations des ennemis, même si celle-ci pourrait être généralisé aux différents GameObjects (puisque utilisant tous la classe Sprite afin de gérer les images).

Celle-ci marche au rythme de une image différente par seconde (On peut gérer ça grâce a un champ dans la classe Game).

Bonus

Un système de bonus a été ajouté au jeu:

A sa mort, un ennemi a une chance sur dix de laisser tomber un bonus, celui-ci tombe doucement vers le joueur, mais cependant, le joueur doit faire attention, celui-ci est sensibles aux dégâts, par conséquent un missile peut le détruire.

Si il atteint le joueur, celui-ci donne un effet aléatoire au joueur (Celui-ci ne peut pas savoir ce qu'il contient avant de le récolter), celui-ci peut être de plusieurs types :

- **Invincibilité**, pendant 6 secondes, le joueur ne peut plus subir de dégâts de la part des missiles ennemis (cependant, celui-ci ne protège pas des collisions directs ennemis-joueur)
- **Multi Missile**, pendant 6 secondes, le joueur tire plusieurs missiles (placé côtes a côtes) quand il tire
- **Super Missile**, pendant 6 secondes, le joueur tire des missiles ayant un plus grand pouvoir de perforation (Faisant plus de dégâts aux ennemis, et potentiellement leur passant au travers après les avoir tués)
- **+1 up**, qui rajoute une vie au joueur

Au futur, j'aimerais bien ajouter plus de Bonus (Tel que des missiles téléguidés, un allié temporaire, ou un bouclier renvoyant le missile a leur envoyeur)

Son

Le son a été ajouté, cependant, celui-ci a une implémentation très basique.

Aujourd'hui, les seuls évènements faisant un effet sonore sont le tir et la mort d'un ennemi.

Système de niveau

J'ai ajouter un système de niveaux, celui-ci est basique et pour l'instant, non influençable par l'extérieur.

Aujourd'hui le jeu ne compte que 3 niveaux différents, ayant chacun un nombre et une disposition d'ennemis différentes.

A terme, j'aimerais bien ajouter un système "personnalisés" de niveau, permettant au joueur de décrire le niveau dans des fichiers, JSON par exemple.

Écrans améliorés

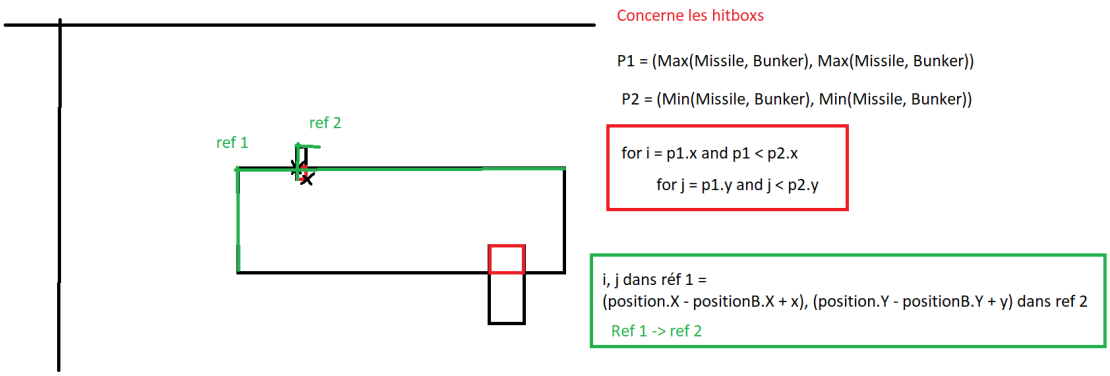
J'ai décidé d'améliorer l'UI du jeu, même si ceci était pas demandé par le cahier des charges initiales.

J'ai donc, comme dit précédemment, ajouté quelques écrans, et ajouté quelques effets visuels (La barre de chargement liés aux bonus, les éléments "clignotants").

Problèmes rencontrés

Lors de la réalisation du projet, j'ai pu rencontrer quelques problèmes

Collision par Pixel



J'ai eu quelques difficultés a mettre en place la collision, cependant, j'ai réussi a trouver une solution en posant sur des schémas mon problème, sa a été facilement la partie la plus difficile à implémenter du projet, celle-ci m'a pris plusieurs heures a implémenter.

```
        public List<Vecteur2D> pixelColliding(Sprite b, Vecteur2D position,
Vecteur2D positionB)
        {
            List<Vecteur2D> listColliding = new List<Vecteur2D>();
            if (b == null) return listColliding;

            //On détermine le rectangle en collision
            Vecteur2D v1Collision = new Vecteur2D(Math.Max(0, positionB.X -
position.X), Math.Max(0, positionB.Y - position.Y));
            Vecteur2D v2Collision = new Vecteur2D(Math.Min(Draw().Width,
positionB.X + b.Draw().Width - position.X), Math.Min(Draw().Height, positionB.Y
+ b.Draw().Height - position.Y));

            for (int y = (int)v1Collision.Y; y < (int)v2Collision.Y; y++)
            {
                for (int x = (int)v1Collision.X; x < (int)v2Collision.X; x++)
                {
                    if (isPixelsColliding(x, y, position, positionB, b))
                    {
                        listColliding.Add(new Vecteur2D(x, y));
                    }
                }

            }

            return listColliding;
        }

        private bool isPixelsColliding(int x, int y, Vecteur2D position,
Vecteur2D positionB, Sprite b)
        {
            return !Draw().GetPixel(x, y).Equals(Color.FromArgb(0, 255, 255,
255))
                        && !b.Draw().GetPixel((int)(position.X - positionB.X +
x), (int)(position.Y - positionB.Y + y)).Equals(Color.FromArgb(0, 255, 255,
255));
```

```
        }
```

Son

Lors de l'ajout de l'addon des Effets sonores, j'ai pu remarquer que lorsque deux sons se jouaient simultanément, le dernier arrivé écrasait le premier son.

Une solution n'a pas encore été trouvé, par faute de temps et d'envie, j'ai cependant trouvé quelques solutions après des recherches sur internet que l'utilisation d'api comme DirectX sont nécessaires.

Habitudes du Java

Étant développeur Java, j'ai quelques habitudes de celui-ci, par conséquent, je n'ai pas utilisé les propriétés du c#, à la place, j'ai mit en place beaucoup de getter et de setter afin d'accéder à mes éléments privés.

On peut donner comme exemple :

```
public override Rectangle getHitbox();
public override Sprite GetSprite();
```