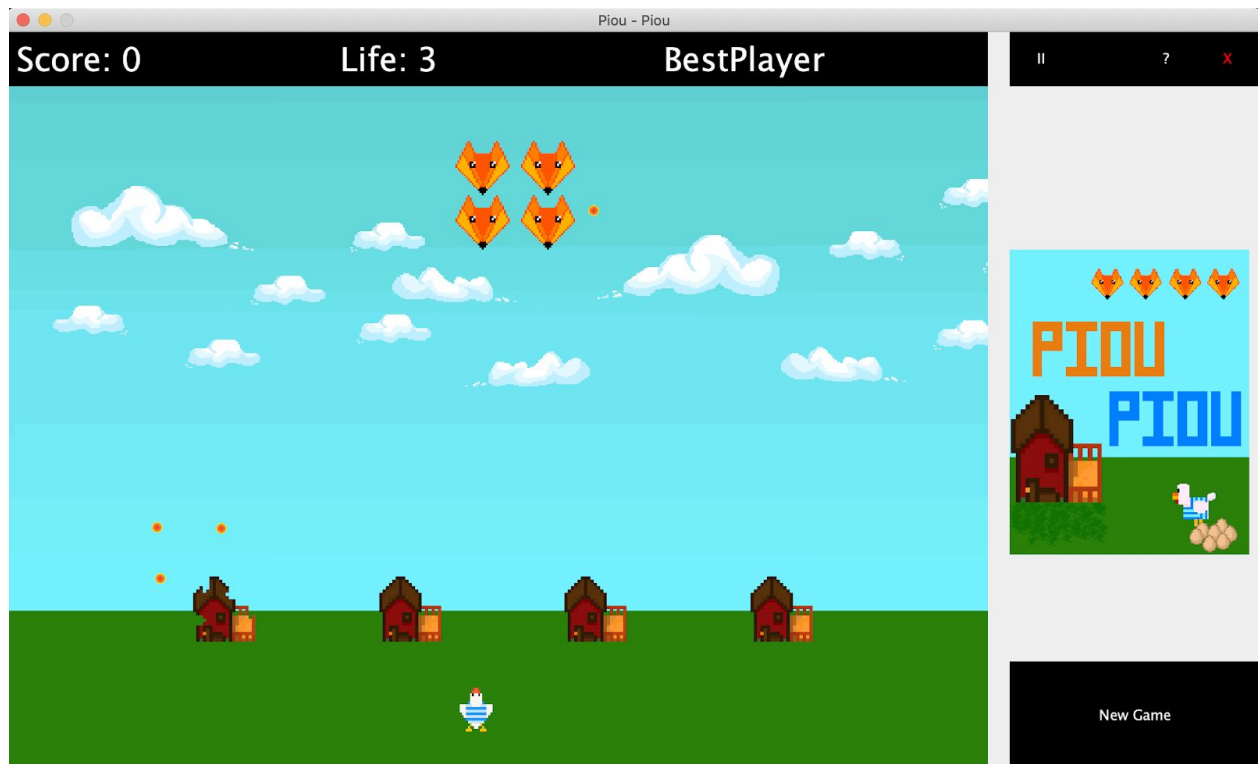


JAVA - IMR1

Rapport du projet Space Invaders

Maëlle Heyrendt - Nicolas Sueur



| | |
|--|-----------|
| Introduction | 3 |
| 1. Diagramme de classe | 4 |
| 2. Gestion de l'architecture logicielle MVC | 8 |
| 2.1. Le modèle | 8 |
| 2.2. La vue | 8 |
| 2.3. Le contrôleur | 9 |
| 3. Le patron de conception Observateur | 10 |
| 3.1. L'observable | 10 |
| 3.2. Les observateurs | 10 |
| 4. Les règles et choix du jeu | 12 |
| 4.1. Gestion des aliens | 12 |
| 4.2. Augmentation des niveaux | 12 |
| 4.3. Conditions de fin de partie | 12 |
| Conclusion | 13 |

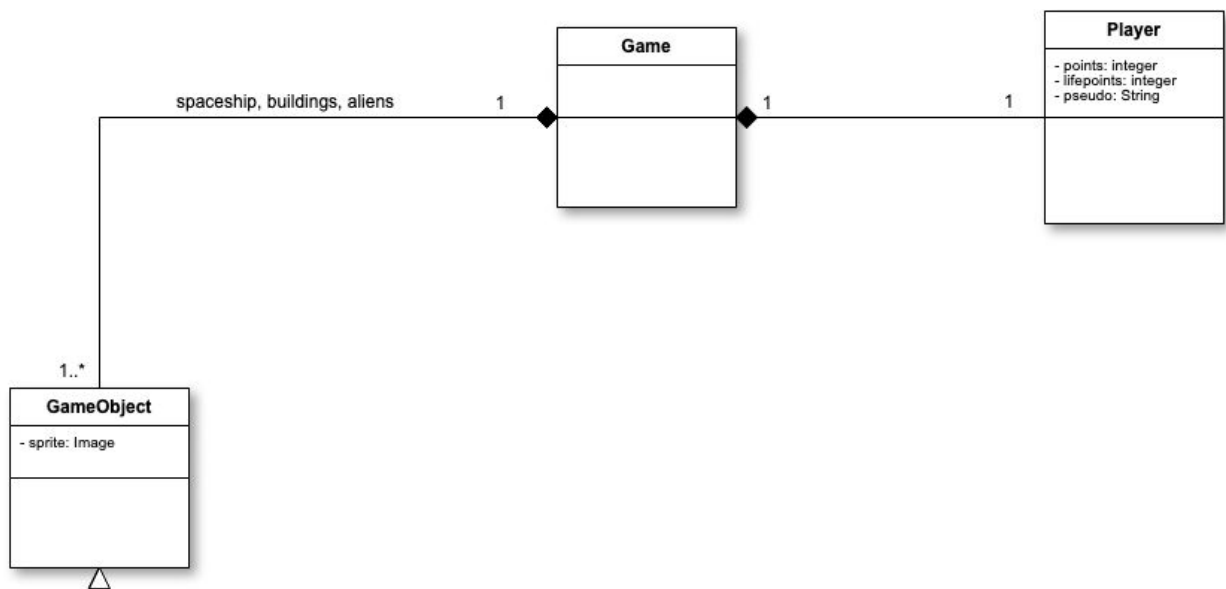
Introduction

Lors de ce projet, nous avons réalisé un Space Invaders à partir d'un cahier des charges. Cela nous a permis de réaliser un diagramme de classe pour mettre en forme notre façon de voir le jeu. De plus, lors de ce projet, nous avons pu mettre en place l'architecture logicielle MVC ainsi que le patron de conception Observateur. Enfin, nous avons utilisé Swing pour notre interface utilisateur.

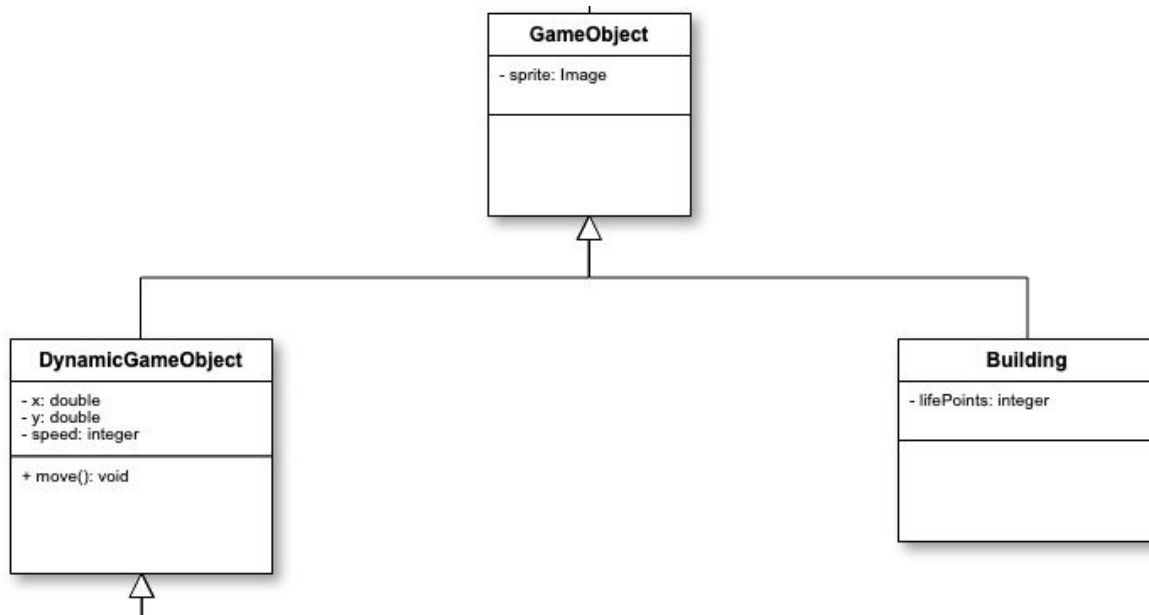
Vous trouverez dans ce rapport les informations nécessaires à la compréhension de notre développement ainsi que quelques spécificités de notre code.

1. Diagramme de classe

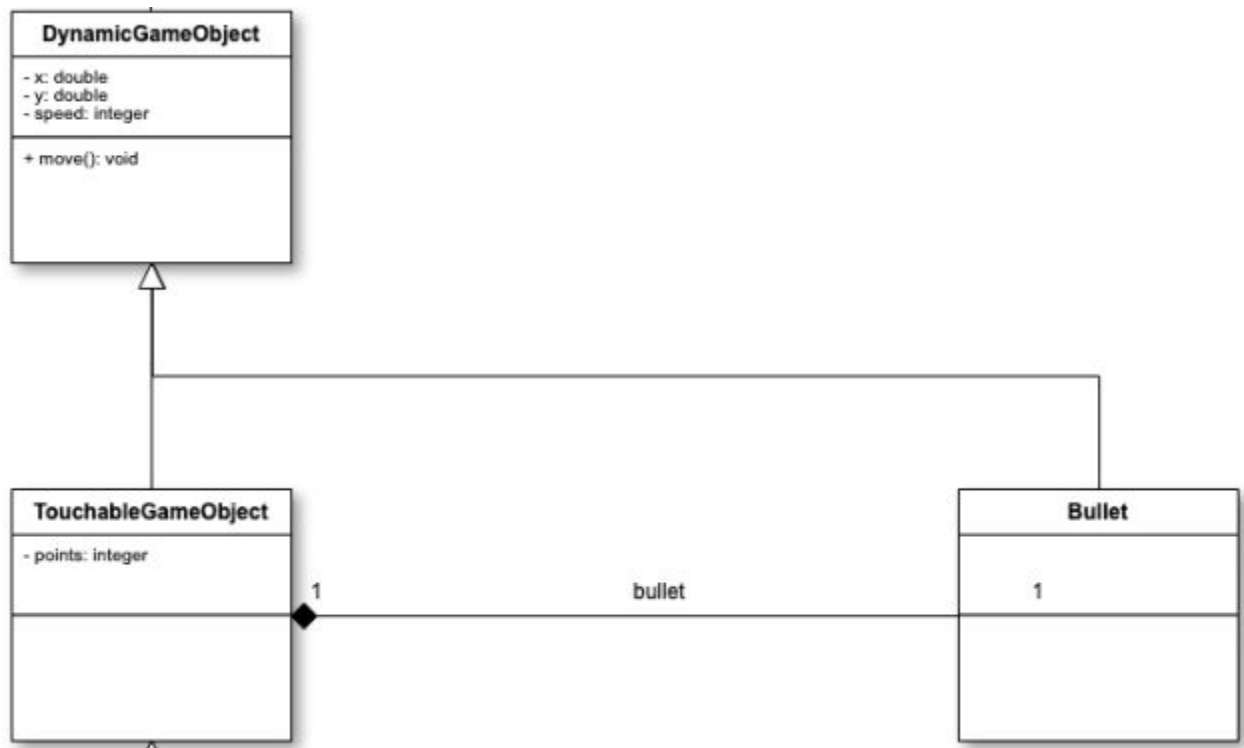
La première étape dans la construction du jeu est la définition du modèle du jeu. Pour cela, nous avons réalisé un diagramme de classe permettant de représenter l'ensemble de nos classes avec leurs attributs principaux ainsi que les différentes liaisons entre celles-ci.



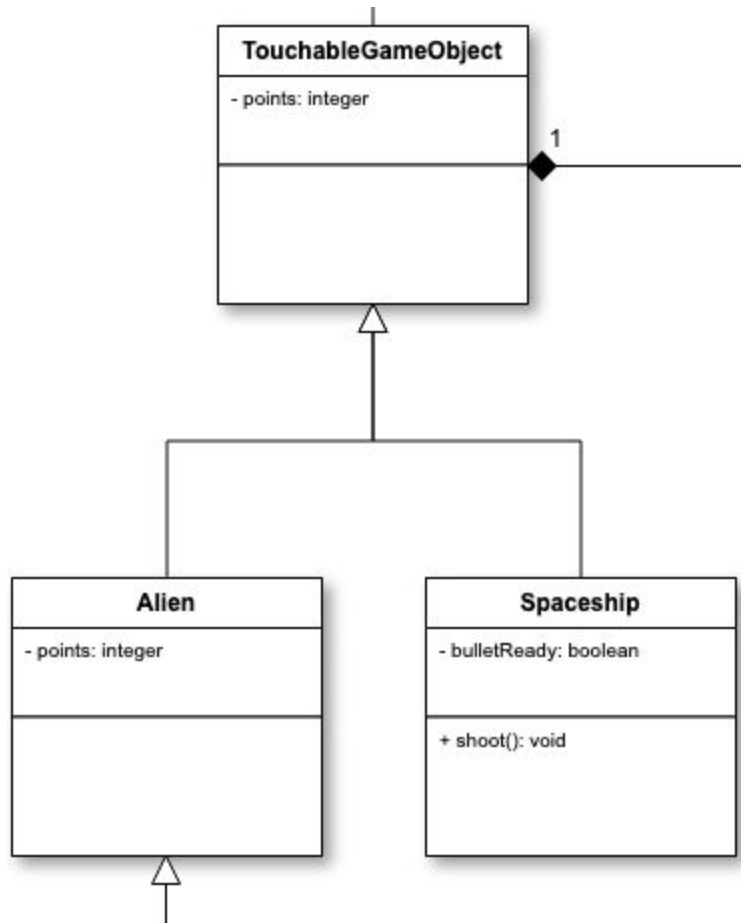
Tout d'abord voici la classe **Game**. Cette classe représente le jeu en lui même. Ses compositions permettent de spécifier qu'un jeu est composé de un ou plusieurs **GameObject** ainsi qu'un **Player**. Un **Player** possède des *points* (qui représentent le score du joueur), des *lifepoints* ainsi qu'un *pseudo*. Un **GameObject** possède un *sprite* qui est simplement défini par une image.



Ensuite, deux classes héritent de **GameObject**. La classe **Building** (bâtiments protégeant le notre vaisseau) possède des *lifePoints* permettant de ne pas détruire directement ces blocs au premier impact d'un tir alien. La classe **DynamicGameObject** représente l'ensemble des objets susceptibles d'avoir un mouvement dans le jeu. Cette classe comporte donc les attributs *x* et *y* afin de pouvoir représenter les différents objets dans un plan 2D. l'attribut *speed* permet de rendre possible la modification de la vitesse d'un objet particulier dans le jeu. Pour finir, elle possède une méthode *move* qui servira à faire déplacer les différents **DynamicGameObject**.



La classe **TouchableGameObject** hérite des propriétés et méthodes de la classe **DynamicGameObject**. Elle représente les objets qui, en plus d'être mobile, peuvent être touché par des tirs (comme les aliens par exemple). Cette classe possède l'attribut *points* qui est défini par le nombre de points que vaut un objet de ce type. À chaque fois qu'un missile de notre vaisseau détruira un alien, l'ensemble de ses points sera attribué au score du joueur. De plus, la classe **TouchableGameObject** est composé d'un objet **Bullet**.



Les classes **Alien** et **Spaceship** héritent des propriétés et méthodes de **TouchableGameObject**. La classe **Spaceship** possède un booléen *bulletReady* afin de savoir s'il est possible de tirer. Elle possède aussi une méthode *shoot* qui permet de tirer un missile.

Pour résumer, nous avons essayé de partir du plus général pour ensuite affiner nos classes en fonction de nos besoins notamment grâce à l'héritage et la composition des différentes classes. Ce modèle nous permettra donc d'implémenter le jeu Space Invader.

2. Gestion de l'architecture logicielle MVC

Pour réaliser ce jeu, nous avons utilisé l'architecture logicielle MVC (Modèle-Vue-Contrôleur). Il nous permet de séparer en trois modules notre code.

2.1. Le modèle

Le modèle permet de contenir les données de notre jeu et représente donc l'univers dans lequel s'inscrit l'application.

Si nous reprenons le diagramme de classe présenté dans la partie 1 de ce rapport, notre modèle comporte, en plus des classes génériques, des classes pour stocker les informations d'un :

- vaisseau joueur,
- alien,
- vaisseau alien,
- bâtiment,
- missile.

En plus de ces classes, nous avons aussi créé une classe permettant de contenir les informations d'un joueur (points de vie, score, pseudo).

Ce modèle de données nous permet de donc de pouvoir gérer l'ensemble des données de notre jeu.

2.2. La vue

La vue est la partie visible d'une interface graphique. Elle se sert du modèle et contient des éléments visuels ainsi que la logique nécessaire pour afficher les données provenant du modèle.

Pour notre application, nous avons choisi d'utiliser Swing pour gérer la partie graphique. Dans un premier temps, nous avons créé une classe MainView qui hérite de JFrame et qui nous permet de contenir l'ensemble des JPanel de notre jeu. Dans un second temps nous

avons codé une classe `PlaygroundAreaView.java` qui hérite de `JPanel` et qui représente notre plateau de jeu. Dans un troisième temps, la classe `InformationAreaView` représente la partie droite de la fenêtre de notre jeu. Elle permet de réaliser différentes actions (à l'aide de boutons) comme l'accès aux paramètres du jeu, la possibilité de mettre en pause celui-ci, de recommencer une partie ou bien même de quitter le jeu.

Les autres vues présentes dans notre projet permettent d'afficher le game over, les paramètres du jeu ou encore les informations d'un joueur au dessus du plateau du jeu.

Enfin, les sprites alternatifs présents dans le jeu ont été réalisés par nous même.

2.3. Le contrôleur

Le contrôleur (`GameController.java`) permet de traiter les modifications du modèle et de la vue en fonction des actions de l'utilisateur. Dans notre cas, il s'occupera d'initialiser et de gérer l'ensemble des objets présents dans notre jeu. Enfin, pour animer ce projet, nous avons utilisé un timer avec un délais de 5 millisecondes qui permet de rafraîchir l'écran de notre jeu.

3. Le patron de conception Observateur

Le patron observateur est un patron de la famille des patrons comportementaux. Il est utilisé pour envoyer un signal à des modules qui jouent le rôle d'observateurs. En cas de notification, les observateurs effectuent alors l'action adéquate en fonction des informations qui parviennent depuis les modules qu'ils observent (les observables). Cela permet de limiter le couplage entre les modules aux seuls phénomènes à observer.

3.1. L'observable

Dans notre projet, c'est le contrôleur (`GameControlleur.java`) qui prend le rôle d'observable. On l'utilise notamment pour :

- signaler l'action d'un joueur grâce à la méthode *actionJoueur()*,
- signaler une fin de partie grâce à la méthode *gameOver()*,
- signaler l'évolution du plateau de jeu grâce à la méthode *actionPerformed()*,
- signaler la création d'un missile grâce à la méthode *setBulletSprite()*,
- signaler la création d'aliens grâce à la méthode *setAliensSprite()*,
- signaler la mise en place de l'image de fond du jeu grâce à la méthode *setBackgroundImage()*.

3.2. Les observateurs

Le rôle d'observateur est attribué aux vues qui nécessitent de réaliser des actions en fonction des notifications du contrôleur. L'utilisation d'un tel rôle pour les vues est présenté ci-dessous.

La classe `InformationGameAreaView.java` représente la zone d'informations au dessus du plateau de jeu. Elle observe le contrôleur et s'occupe de mettre à jour les points de vie et le score du joueur à chaque fois qu'une notification est reçue.

La classe `MainView.java` représente la fenêtre de jeu. Elle observe le contrôleur et s'occupe de changer l'affichage du plateau de jeu avec celui de fin de partie à chaque fois qu'un

joueur perd la partie. Les conditions de fin de partie sont présentés dans la partie 4 ci-dessous.

La classe `PlaygroundAreaView.java` représente le plateau de jeu. Elle observe le contrôleur et s'occupe de redessiner la scène à chaque qu'une notification est reçue.

4. Les règles et choix du jeu

4.1. Gestion des aliens

Tout d'abord, comme l'a montré le diagramme de classe présenté en partie 1 de ce rapport, un Alien possède un nombre de points. Ce nombre de points est une sorte de "récompense" qui va être attribué au score du joueur lorsqu'un alien meurt. De plus, la classe alien héritant de la classe TouchableGameObject, elle dispose donc de l'ensemble des méthodes et attributs de celle-ci.

Pour générer les aliens sur le plateau de jeu, nous utilisons une liste de listes d'aliens. Chaque colonne est donc représentée par une liste d'aliens. De plus, nous avons fait en sorte de pouvoir toucher uniquement un alien avec un missile. Enfin, tant que le missile n'a pas touché un alien ou n'est pas sortie du plateau de jeu, il n'est pas possible de retirer un missile.

4.2. Augmentation des niveaux

On caractérise la fin d'un niveau par la mort de tous les aliens sur le plateau. À chaque début de niveau, on redessine un nombre aléatoire d'aliens. Pour réaliser cela, nous avons codé la méthode *changementDeNiveau()* dans la classe GameController.java. Dans cette méthode, nous initialisons un objet Random qui nous sort un nombre aléatoire. Ensuite, grâce à ce nombre, nous générons de nouveaux aliens en fonction du chiffre aléatoire. Enfin, nous réglons aussi la chance qu'un alien tire un missile.

4.3. Conditions de fin de partie

Il existe trois conditions de fin de partie.

La première condition est inscrite dans le code de la classe PlaygroundAreaView.java et plus précisément dans la méthode *checkCollision()* qui permet de vérifier la collision entre deux objets. Si le joueur se fait toucher trois fois par un missile (et donc que ses points de vies sont à zéro) la partie se finit.

La deuxième et troisième condition se trouvent dans le contrôleur (`GameController.java`) et plus précisément dans la méthode `moveAliens()`. Celle-ci permet de déplacer les aliens et le vaisseau alien sur le plateau de jeu. Si les aliens ou le vaisseau alien arrive au niveau de la ligne du vaisseau, alors la partie se finit.

Enfin, pour montrer au joueur que la partie est finie, on affiche à la place du plateau de jeu un écran "Game Over".

Conclusion

Dans ce projet, nous avons éprouvé des difficultés notamment sur la partie focus entre nos `JPanel`. En effet, lorsque nous sélectionnons un bouton du Panel de droite, il n'était plus possible d'interagir avec le plateau de jeu.

Ce projet nous a permis de mettre en pratique les différents éléments vu en cours (MVC, paterne Observateurs, diagramme de classe, Swing...).