Arthur Parmentier **M1 Informatique - Université Lyon 2**
Cyrielle Barailler
Victor Sigogneau

Conception agile de projets informatique et génie logiciel

Introduction :

Dans le cadre du cours de Conception agile de projets informatique et génie logiciel, nous devions produire un jeu simulant une partie de balle au prisonnier. Nous avions de base un projet existant nous ajouter et gérer différentes actions comme :

- Les déplacements de 3 joueurs (1 humain, 2 ordinateurs) face à face.
- Les tirs où un système de projectile a été créé avec une vitesse et une direction qui sera donc la balle.
- Le système de collisions : en touchant les autres joueurs ou un rebond avec les murs si collision avec un joueur il y a alors il disparaîtra de la partie.
- L'intelligence des joueurs ordinateurs : un déplacement opposé au tir, des tirs vers l'endroit où il y a le plus de joueurs.

Une fois toutes ces options disponibles, nous devions effectuer une réingénierie du code en mettant en œuvre les patrons de conception vus en cours. C'est-à-dire de réorganiser les éléments de l'application en appliquant les patrons de conception adéquats.

Nous avions un patron de conception imposé le modèle *MVC* et nous avons choisi deux modèles qui nous semblaient les plus adaptés à notre problématique le *Singleton* et le *Prototype*.

Notre projet se trouve sur notre GitHub ou l'on retrouve :

- Un fichier README.md à la racine du projet
- Un fichier maven pour le build du projet
- Les sources (fichiers Java)

Dans ce rapport nous développerons la réingénierie du code en expliquant les différents patterns utilisés leur définition, leur structure, leur fonctionnement, les avantages et enfin l'application à notre jeu de la balle au prisonnier.

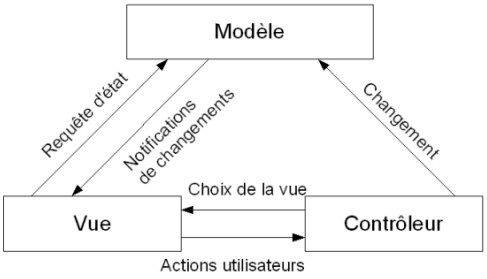
Pattern MVC

Définition du pattern :

Le Modèle-Vue-Contrôleur est un patron d'architecture. C'est-à-dire qu'il se réfère à l'organisation structurelle de l'application. Il est d'un niveau d'abstraction supérieur au patron de conception.

Explication de la structure :

Diagramme de l'architecture MVC :



Source : developpez.com

L'architecture MVC va scinder les responsabilités du système en trois parties distinctes :

- Modèle (logique métier) : Le modèle contient les informations utilisées par le système. Il va les organiser pour qu'elles puissent ensuite être traitées par le contrôleur.
- Vue (interface) : La vue est la façon dont ces informations sont affichées pour l'utilisateur. Elle présente les données en cohérence avec l'état du modèle et capture et transmet les actions de l'utilisateur.
- Contrôleur : Le contrôleur veille à ce que les demandes de l'utilisateur soient correctement exécutées, en modifiant les objets du modèle et en mettant à jour la vue. Il est l'intermédiaire entre le modèle et la vue.

Explication du fonctionnement :

Une fois que le code de notre programme est scindé dans chacune de ses parties, le fonctionnement du MVC est le suivant :

1. L'utilisateur émet une requête
2. Le contrôleur intercepte la requête de l'utilisateur
3. Le contrôleur détermine quelle partie du modèle est concernée et quelle vue y est associée
4. Le modèle traite les interactions avec les données, applique les règles métier et renvoie les données au contrôleur
5. Le contrôleur sélectionne la vue et lui renseigne les données

6. La vue présente les données à l'utilisateur

Avantages du MVC :

Dans ce projet, l'architecture MVC nous a été imposée, mais elle reste un très bon choix pour gérer l'organisation de notre code. Elle va permettre de diminuer la complexité de la conception et de la rendre plus claire et efficace. En suivant cette approche, on peut concevoir un système dans lequel les responsabilités sont clairement définies.

Application à notre jeu de la balle au prisonnier :

Dans le contexte du jeu, notre pattern sera composé de la manière suivante :

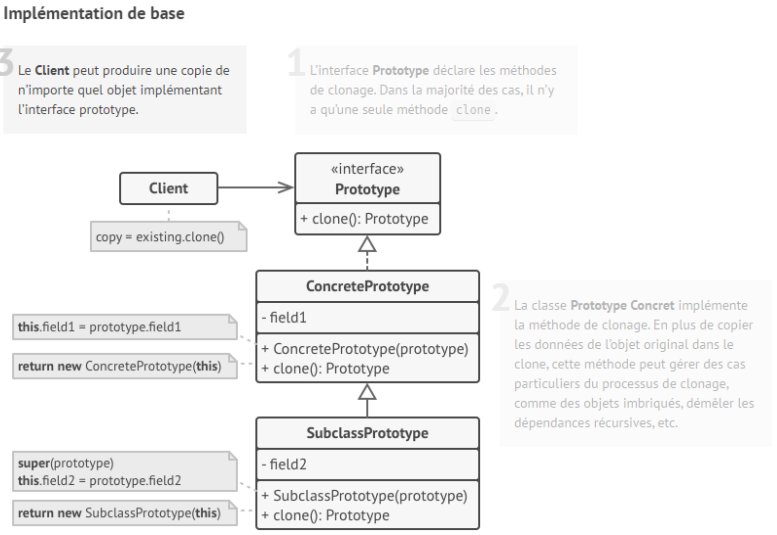
- Le modèle va contenir les attributs des joueurs, soit leur position, leur couleur et leur vitesse. Mais également les attributs des projectiles, leur vitesse et leur direction.
- La vue va permettre l'affichage graphique des joueurs et des projectiles. Elle va aussi permettre de générer les différents affichages du terrain, du score ainsi que les contrôles jeu.
- Le contrôleur va gérer toutes les interactions de l'utilisateur avec le clavier. Il va donc traiter les déplacements des joueurs, mais aussi le lancement et le déplacement des projectiles.

Le Prototype

Définition du pattern :

Le prototype fait partie de la famille des patrons de création, il est donc utilisé pour créer des objets. C'est un patron de conception qui crée de nouveaux objets à partir d'objets existants sans rendre le code dépendant de leur classe.

Explication de la structure :



Comme on peut le voir sur le schéma ci-dessus, le type Client crée un nouvel objet en demandant à un prototype de se cloner. Pour se faire on créé une interface Prototype qui ne contient que la méthode **clone()**. Puis on crée une classe ConcretePrototype qui implémente la méthode **clone()** elle devra récupérer les données de l'objet cloné pour les copier dans l'objet créé. On obtient alors des sous-classes.

Explication du fonctionnement :

Le patron de conception prototype délègue le processus de clonage aux objets qui vont être copiés. Il recopie une instance existante et reporte les valeurs des attributs de l'objet d'origine dans le nouveau grâce à la méthode clone() comme vu au dessus. Un objet qui peut être cloné est appelé un prototype. Il permet également l'ajout de comportements spécifiques ce qui revient à du « polymorphisme pas cher ».

Avantages du prototype :

Le prototype dispose de différents avantages qui sont les suivants :

- On peut cloner les objets sans les coupler avec leurs classes concrètes.
- On clone des prototypes préconstruits et vous pouvez vous débarrasser du code d'initialisation redondant. C'est une économie de ressource.
- On peut créer des objets complexes plus facilement.
- On obtient une alternative à l'héritage pour gérer les modèles de configuration d'objets complexes. On réutilise un comportement sans recréer une instance.

Application à notre jeu de la balle au prisonnier :

Dans le cas de notre jeu de balle au prisonnier, nous aurions utilisé le prototype pour cloner le Player et son IA. En effet, on demande que chaque joueur artificielle possède une tactique propre à lui même : se déplacer aléatoirement ou suivre le joueur contrôlé par un humain. Pour se faire on ne va pas créer une classe héritant de Player à chaque fois mais on va plutôt utiliser le pattern prototype. Ainsi nous pourrons profiter du polymorphisme pour modifier la façon du bot de se déplacer sans recréer une nouvelle instance.
Mise en place :

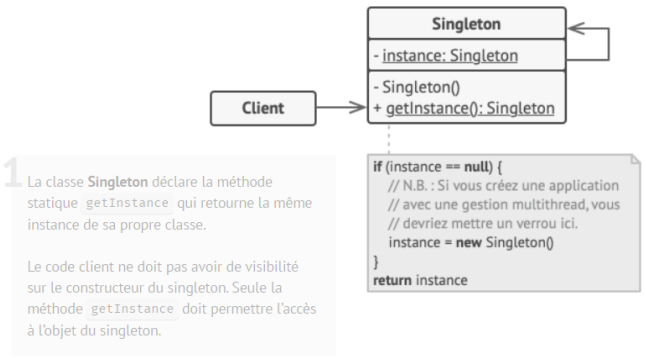
- Prototype → PlayerIAInterface
- ConcretePrototype → PlayerIA (on implémente le clonage)
- Puis on peut le cloner autant de fois qu'on a d'ordinateurs

Le Singleton

Définition du singleton :

Le singleton fait partie de la famille des patrons de création, il est donc utilisé pour créer des objets. Son but est d'éviter qu'une classe ne crée plus d'un objet. Pour ce faire, on crée l'objet souhaité dans une classe propre, puis on le récupère sous forme d'instance statique. Le singleton est l'un des patrons les plus simples, mais les plus puissants dans le développement de logiciels. Nous avons mis en place ce patron de conception.

Explication de la structure :



Comme nous pouvons le voir sur cette représentation, le singleton complet est constitué d'un seul objet car il n'y a qu'une seule et unique instance d'une classe à créer.

Explication du fonctionnement :
Le constructeur de la classe est privé (seules les méthodes de la classe peuvent y accéder). L'instance unique de la classe est donc stockée dans une variable statique privée puis la méthode publique statique de la classe est appelée et grâce au If dans la fonction on crée alors une instance au premier appel ou on retourne l'instance déjà existante. On peut alors ne retourner que l'instance qui existe déjà et ainsi ne pas créer de doublons.

Avantages du prototype :

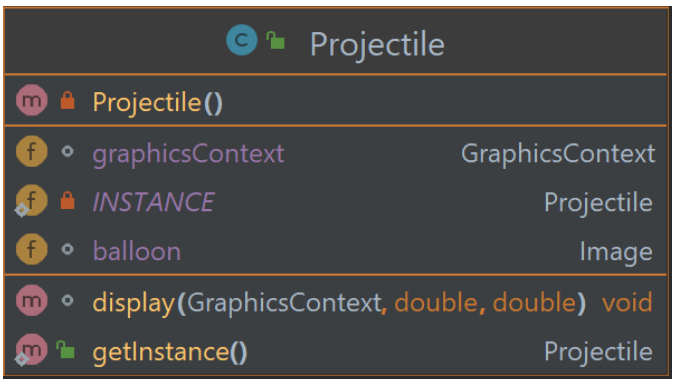
Le pattern singleton a de nombreux avantages :

- Il peut s'écrire rapidement et facilement car il n'est pas peuplé d'innombrables variables globales. En effet, on obtient un point d'accès global à cette instance.
- On peut exercer un contrôle précis sur le moment et la manière dont on y accède. L'objet du singleton est uniquement initialisé la première fois qu'il est appelé.
- Un singleton peut être dérivé en utilisant des sous-classes pour remplir de nouvelles fonctionnalités.
- On garantit l'unicité de l'instance de la classe.

Application à notre jeu de la balle au prisonnier :

Dans le cas de notre jeu de la balle au prisonnier, nous avons utilisé le singleton pour l'objet **Projectile**. En effet, dans notre jeu nous avons besoin d'uniquement un ballon et cela nous permet d'avoir des bus et de se voir avec deux ballons.

Voici le diagramme de notre classe Projectile :



Conclusion

Pour conclure, la conception de ce projet nous a permis de développer et de consolider plusieurs de nos connaissances. Nous avons pu apprendre et revoir les bases du langage de programmation Java ainsi que l'utilisation de Java FX.

Un des objectifs du projet était de mettre en place et de maîtriser des outils de gestion de code. Pour ce faire, nous avons utilisé Maven comme outil de build, Git comme outil de versioning et enfin Github pour la gestion de notre projet.

De plus, nous avons découvert les différents patrons de conception, leurs structures ainsi que leurs avantages. Nous avons donc appris à mieux structurer notre projet pour le rendre plus modulaire.

Pour finir, ce projet a été réalisé en groupe. Il était donc indispensable de se partager les tâches et de bien communiquer afin que tous les membres travaillent de façon équitable sur le projet. Bien que tous les objectifs de ce projet ne soient pas atteints, nous pensons que nous avons réussi à faire de notre mieux pour gérer ce projet avec nos connaissances du langage Java.