Compte rendu méthode de conception: Mystic Robot

Aubry Nicolas, Leblond Valentin, Mori Baptiste, Chagneux Dimitri

Octobre - Novembre 2018

Table des matières

1 Introduction

Voici notre jeu, pour la méthode de conception. Nous avons voulu un jeu, avec des règles simples, une approche pour l'utilisateur assez intuitive. Mais en ayant un jeu respectant les différentes règles imposées par l'énoncé.

2 Le jeu

2.1 Les règles du jeu :

À chaque tour de jeu, les joueurs jouent l'un après l'autre selon un ordre initialement défini, ou selon un ordre tiré aléatoirement à chaque tour.

2.1.1 Une action possible est :

- Déplacement d'une case (4 directions possibles seulement).
- Dépôt d'une mine ou d'une bombe sur l'une des 8 cases voisines.
- Utilisation d'un tir horizontal ou d'un tir vertical.
- Déclenchement du bouclier, qui protège des tirs et bombes lors du prochain tour.
- Ne rien faire pour économiser son énergie.

2.1.2 La grille :

La grille peut contenir, lors de sa création, des murs (cases non-utilisables par les combattants et infranchissables par les tirs), et des pastilles d'énergie que les combattants peuvent récupérer en se plaçant dessus.

2.1.3 Les bombes, mines, tirs et shield :

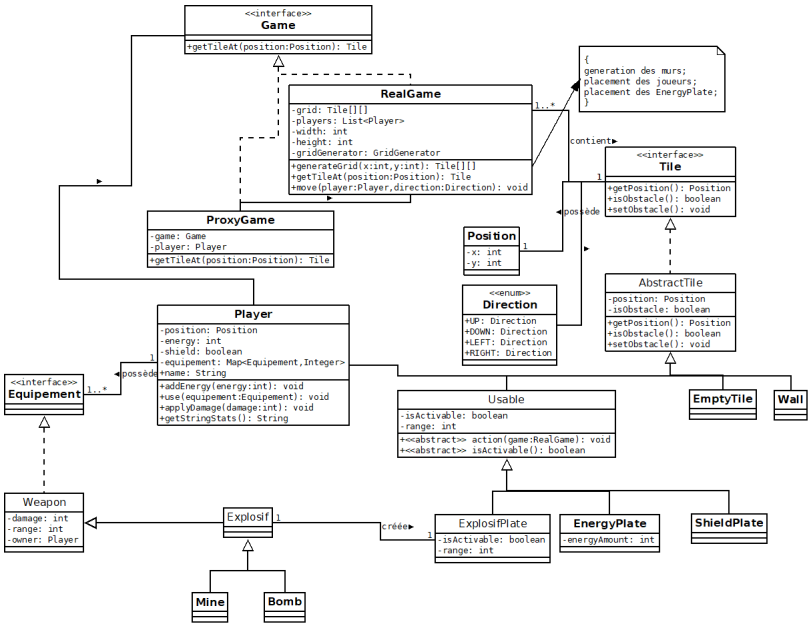
- Une mine explose lorsqu'un combattant se place sur la case qu'elle occupe.
- Une bombe explose au bout d'un certain délai t (compte à rebours en nombre de tours de jeu), et impacte les combattants se trouvant sur l'une des 8 cases voisines, ou , comme une mine, si l'on se place dessus.
- Les bombes et les mines peuvent avoir deux types de visibilité : visible de tous les combattants, ou visible seulement du combattant qui l'a déposée.
- Un tir horizontal impacte les combattants se trouvant sur la même ligne horizontale, à un nombre de cases inférieur à la distance (portée limitée). Idem pour un tir vertical (i.e sur la même colonne).

— Une explosion ou un tir impactant un combattant fait baisser son énergie d'une certaine valeur qui fera partie des paramètres du jeu.
— Un déplacement et l'utilisation du bouclier ont eux aussi leurs coûts respectifs.
— Un combattant perd le jeu lorsque son énergie est négative ou nulle. Il disparaît alors du jeu.
— Le gagnant est le dernier survivant.

3 Diagramme UML du jeu :

3.1 Le diagramme :

FIGURE 1 – Voici notre Diagramme UML



3.2 Explications :

Nous avons voulu, un diagramme simple, mais en même temps assez complet afin de pouvoir s'y retrouver facilement et que toutes les personnes du groupe

puissent le comprendre assez aisément. Ensuite, nous avons voulu le maximiser un maximum, c'est-à-dire éviter les redondances, les classes inutiles et la répétition de code inutile. Vous pouvez retrouver le diagramme au format PNG dans le rapport.

Donc après plusieurs relectures du diagramme et du code, nous avons pu ainsi enlever des parties du code qui devenaient à nos yeux inutiles.

4 Les patterns et modèles utilisés :

4.1 Le pattern Strategy :

Le pattern Strategy est un pattern qui permet d'externaliser le contenu de certaines méthodes pour pouvoir modifier les comportements soit dynamiquement, soit en utilisant de nouvelles implémentations.

Voici donc l'implémentation dans notre code, du pattern Strategy, pour la création d'un Parser permettant de gérer des fichiers externes. Dans la première capture on peut voir l'interface.

```
public interface Parser {

  public Map<String,String> executeTexture();

  public ArrayList<Map<String,Map<String,String>>> executeConfig();

}
```

Donc, après notre interface, nous avons la classe ParserCrochet.java qui implémente les méthodes de l'interface.

```
public class ParserCrochet implements Parser {

  private String texturePath;
  private String configPath;

  public ParserCrochet(String texturePath, String configPath) {
    this.texturePath = texturePath;
    this.configPath = configPath;
  }

  @Override
  public Map<String,String> executeTexture() {
```

```
public interface Parser {

  public Map<String,String> executeTexture();

  public ArrayList<Map<String,Map<String,String>>> executeConfig();

}
```

Ici, on réécrit la méthode executeTexture().

4.2 Le pattern Factory :

Le pattern Factory permet de générer dynamiquement des types d'objets.
Voici un exemple de code dans lequel nous utilisons le pattern Factory (Dans le Main.java) :

```
RobotFactory factory = new RobotFactory(parser.executeConfig(), 3);
```

Et dans RobotFactory.java :

```
   * Constructeur de la classe qui fait appel aux m thodes createStuffs, create
   * @param config la configuration de la partie (les armes et les robots)
   * @param playerNumber le nombre de joueurs pour la partie
   */
  public RobotFactory(ArrayList<Map<String,Map<String,String>>> config, int playe
    this.weaponConfig = config.get(0);
    this.robotConfig = config.get(1);

    this.createStuffs();
    this.createRobots();
    this.initPlayerList(playerNumber);
  }
```

Dans notre projet, on se sert du pattern afin de créer des objets Player qui ont des caractéristiques prédéfinies (énergie, stuff, rôle, etc.). On l'utilise dans la classe RobotFactory, c'est grâce à cela que l'on arrive à dire si on a des joueurs de types sniper, tank ou autres.

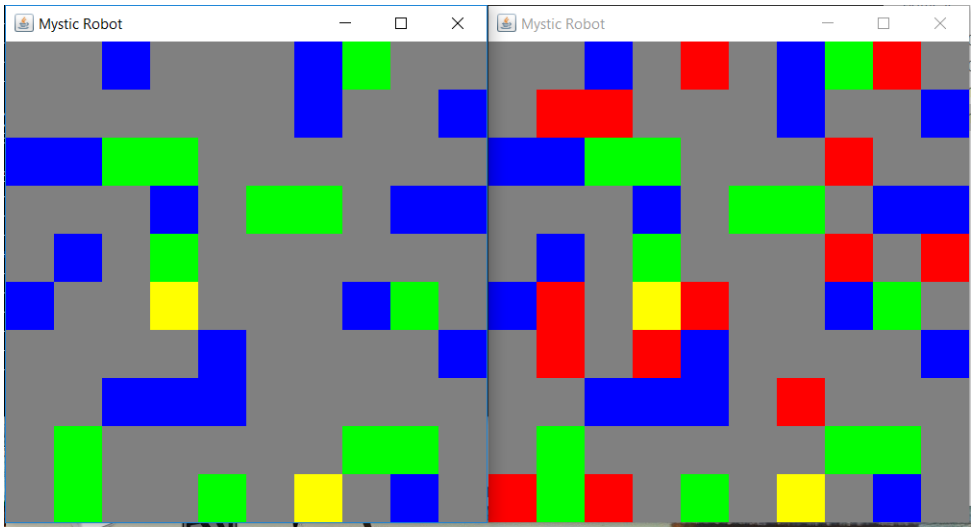
4.3 Le pattern Proxy :

Le pattern Proxy permet de servir d'intermédiaire entre l'objet à contrôler et celui qui veut y accéder.

Dans notre jeu, nous avons implémenté le pattern Proxy, afin de permettre aux différents joueurs de ne pas voir les mines disposées par les ennemis. Mais de voir seulement les mines ou bombes que le joueur a lui-même poser.

Voici un exemple de ce que cela peut donner graphiquement (cf :Figure 2) :

FIGURE 2 – Utilisation du pattern Proxy :



Cette figure nous montre de manière graphique comment fonctionne notre pattern Proxy. Tout d'abord, on peut voir que nous avons deux interfaces. Il y a une interface "joueur", ou c'est la vue que le joueur obtient. Et il y a une interface "Master" qui elle voit toute la grille en entière avec les bombes qui sont posées. L'interface "Master" est celle que l'on voit à droite. Les carrés rouges représentent les bombes posées (le pattern Proxy à été l'un des premiers à avoir été implémenté, donc le nombre de bombes est assez élevé, car c'était au début du développement du jeu.) dans cette version, les murs n'étaient pas encore présents. Les carrés jaunes représentent les joueurs, les bleus du shield, les rouges des bombes et les verts l'énergie (la vie).

4.4 Le modèle MVC :

Notre modèle MVC on a la table des joueurs qui écoute tous les joueurs de la liste des joueurs. La table des équipements qui écoute le Game, et la grille qui écoute aussi le Game.

4.4.1 Le Modèle :

Le Modèle : notre modèle est composé du package game, qui contient toutes nos classes gérant le modèle.

4.4.2 La Vue :

La Vue : la vue, elle nous sert à afficher graphiquement notre jeu. Nous utilisons les JFrames ainsi que les JPanel. Notre vue est composée du package gui, qui contient toutes les classes permettant de générer graphiquement nos différentes interfaces graphiques.

4.4.3 Le Contrôleur :

Le Contrôleur : le contrôleur est aussi composé du package gui.

5 Utilisation du jeu :

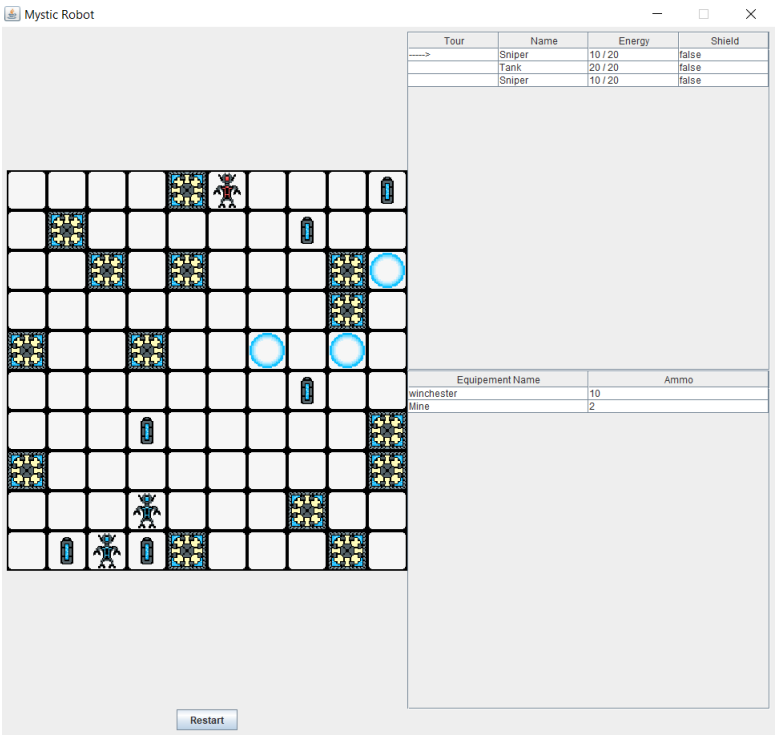
Pour l'utilisation du jeu, nous vous mettons une image qui vous permettra de comprendre le fonctionnement du jeu ainsi que la manière d'y jouer.

Le jeu est contrôlé par des IA, les joueurs n'ont pas la possibilité de jouer. Ce sont les IA qui contrôlent les robots, on peut alors suivre les IA essayées de se battre les unes contre les autres.

Pour repérer notre joueur, il suffit de regarder quel robot est entouré de rouge. Cela signifie que c'est son tour de jouer. De plus, une flèche indique dans la liste des joueurs lequel doit jouer.

Nous avions un autre mode de jeux, ou l'on peut se déplacer soit même. Pour se déplacer, il suffit d'utiliser les flèches du clavier. On peut apercevoir sur le coté droit de l'interface les joueurs présents ainsi que leurs points de vie et s'ils portent un shield.

FIGURE 3 – Utilisation du Jeu :



6 Conclusion

La matière méthode de conception, nous a permis de mieux comprendre certains aspects de programmation (Par exemple l'utilisation des différents patterns vus durant les CM et TP.). De plus, la répartition des taches a été un des piliers centraux, concernant la mise en place du jeu.

Le plus long dans la gestion de ce projet a été de mettre en place les différents Patterns comment les implémenter, ou les mettre, lesquels nous sont utiles et lesquels nous seront inutiles. Le plus long a été la mise en place du jeu en façon "console". Car nous avons d'abord dû tout implémenter avant de commencer à vraiment réaliser l'interface graphique de notre jeu.

Ce projet nous aura permis de mieux comprendre les différentes façons de concevoir la mise en place d'un projet à rendre, du papier à la version finale. Comment bien commencer et les bonnes façons d'y parvenir.

7 Références

7.1 Références bibliographiques :

Référence aux différents CM et TP de Yann Mathet.

https://mathet.users.greyc.fr/

https://ecampus.unicaen.fr/course/view.php?id=15145

7.2 Références externes :

Référence aussi à différentes documentations présentes en ligne :
Concernant la mise en page latex :

http://www.xm1math.net/doculatex/url.html

https://urlz.fr/8eFS

https://docs.oracle.com/javase/7/docs/index.html

https://stackoverflow.com/questions/50439945/strategy-pattern-java