Rapport

Jeu de stratégie au tour par tour

Contributeurs :

Nom et prénom	Numéro d'étudiant
Salah Eddine ELOUARDI	22110344
Aimad LAHBIB	22202135
Yassine EL-AASMI	22011193

Université de Caen Normandie
UFR des Sciences
Département Informatique
2ème année de licence d'informatique

Table des matières

1. Introduction

Ce projet consiste en le développement d'un jeu de stratégie au tour par tour, où plusieurs joueurs s'affrontent sur une grille. Chaque joueur contrôle un ou plusieurs combattants qui peuvent se déplacer, utiliser des armes ou poser des explosifs pour éliminer leurs adversaires. Le but du jeu est simple : être le dernier combattant encore en vie sur la grille.

1.1. Objectifs du Projet

Les principaux objectifs incluent :

— Créer une application paramétrable utilisant un fichier `.xml`.
— Implémenter un modèle basé sur le design pattern MVC.
— Intégrer des fonctionnalités de jeu comme le déplacement, les attaques, les boucliers.
— Développer une interface console et graphique.
— Rendre l'application fluide et modulaire en introduisant différentes fonctionnalités.

2. Architecture Logicielle

2.1. Architecture MVC et Utilisation des Patterns

L'application suit l'architecture MVC (Modèle-Vue-Contrôleur), qui divise le projet en trois composants principaux :

- **Modèle** : Gestion de la logique métier, des données, et des règles du jeu (combattants, armes, grille, etc.).
- **Vue** : Interface utilisateur qui affiche les informations et transmet les actions de l'utilisateur au contrôleur.
- **Contrôleur** : Intermédiaire entre le modèle et la vue, coordonnant leurs interactions et actualisant les données affichées.

3. Utilisation des Patterns dans le Projet

3.0.1. Pattern Factory

Ce pattern est utilisé pour la création des objets dynamiques dans le projet, comme les combattants ou les armes. Il centralise la logique de génération pour simplifier les ajouts ou modifications.

3.0.2. Pattern Strategy

Il permet de définir plusieurs comportements possibles pour les joueurs (humain, IA ou aléatoire). Chaque type de joueur utilise une stratégie différente pour interagir avec le jeu.

3.0.3. Pattern Proxy

Ce pattern contrôle l'accès aux informations sur la grille en limitant ce qu'un joueur peut voir en fonction de sa position et de ses actions.

4. Diagrammes et Patterns de Conception

4.1. Diagramme des Combattants (Factory Pattern)

Explication : Ce diagramme montre l'implémentation du pattern *Factory*, utilisé pour centraliser la création des objets liés aux combattants.

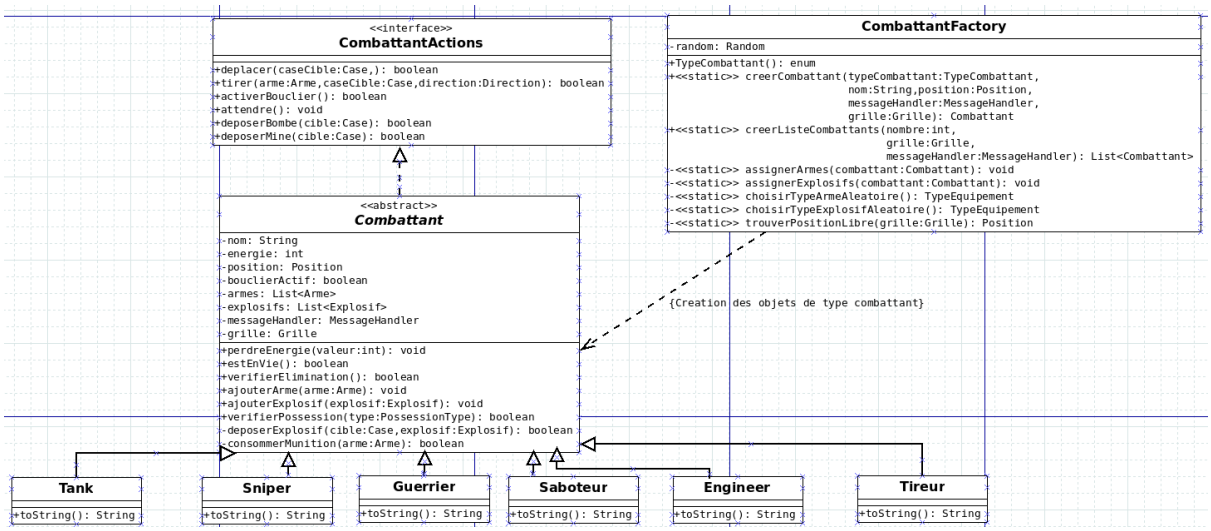

FIGURE 1 – Diagramme UML des Combattants et du Pattern Factory.

4.2. Proxy Pattern pour la Grille

Explication : La classe `ProxyGrille` agit comme un intermédiaire entre les joueurs et la grille réelle (`Grille`).

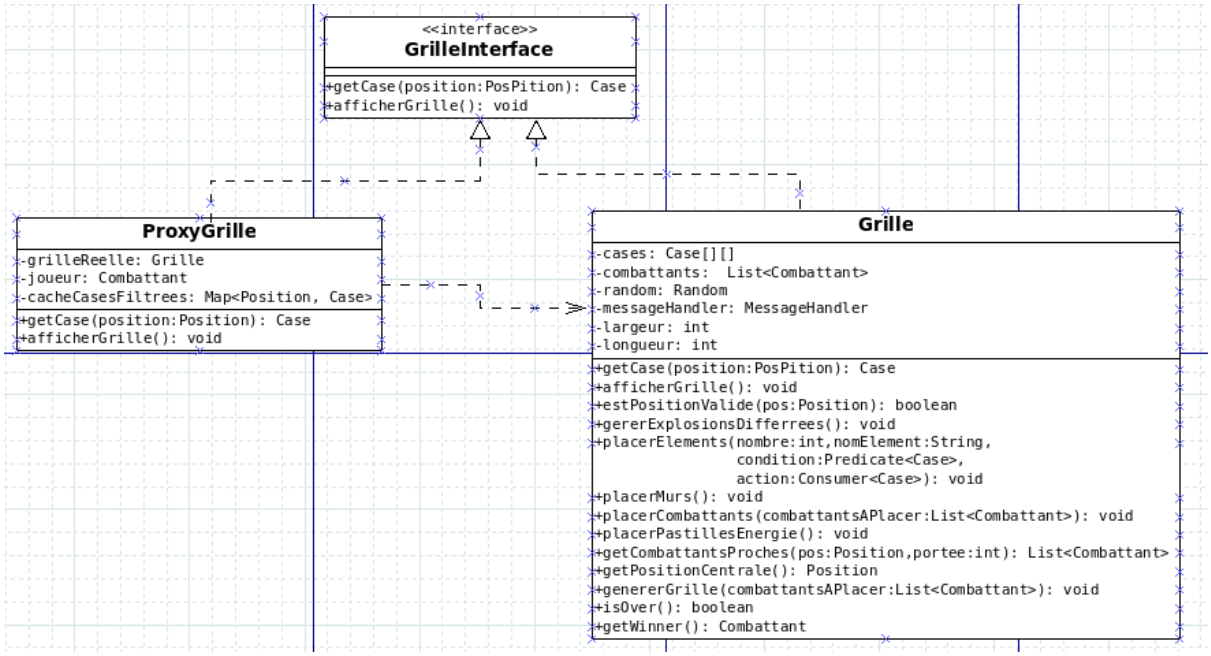

FIGURE 2 – Diagramme UML du Pattern Proxy appliqué à la Grille.

4.3. Strategy Pattern pour l'affichage des Message et la creation des strategies

Utilisation du Pattern Strategy

Le pattern **Strategy** est utilisé dans notre projet sous deux formes :

- **Gestion des messages** : Une interface `MessageHandler` définit deux méthodes principales :
  - `afficherMessage` pour afficher les messages.
  - `afficherErreur` pour afficher les erreurs.

  Ces méthodes sont implémentées par deux classes concrètes : `ConsoleMessageHandler` (pour la console) et `GraphicMessageHandler` (pour l'interface graphique).
- **Stratégies de jeu** : Une interface `Joueur` définit une méthode `strategie` permettant de modéliser différents comportements de joueurs. Cette méthode est implémentée par :
  - `HumainJoueur` : Contrôlé par un joueur humain.
  - `AleatoireJoueur` : Qui joue de manière aléatoire.
  - `AIJoueur` : Une intelligence artificielle prenant des décisions basées sur des heuristiques.

  Cela permet de modifier le comportement d'un joueur sans changer le reste de la logique du jeu.

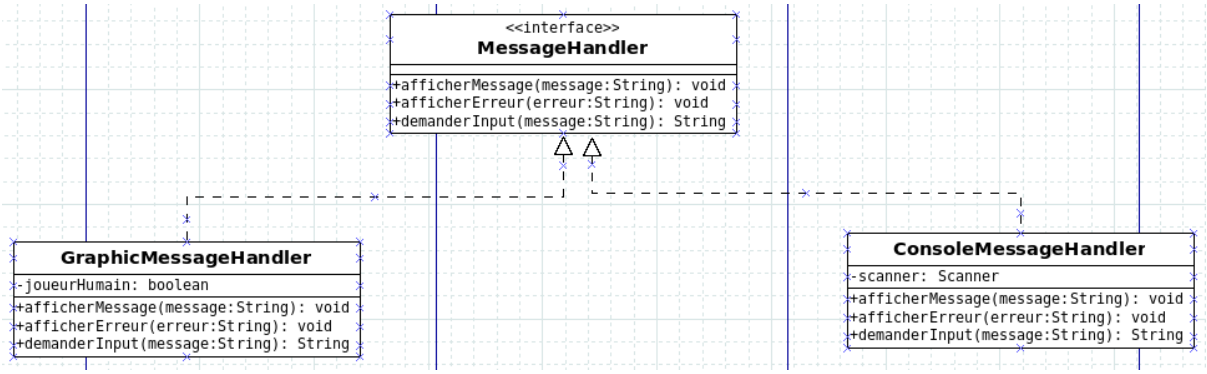

FIGURE 3 – Diagramme UML du Pattern strategy appliqué à l'affichage des Messages.

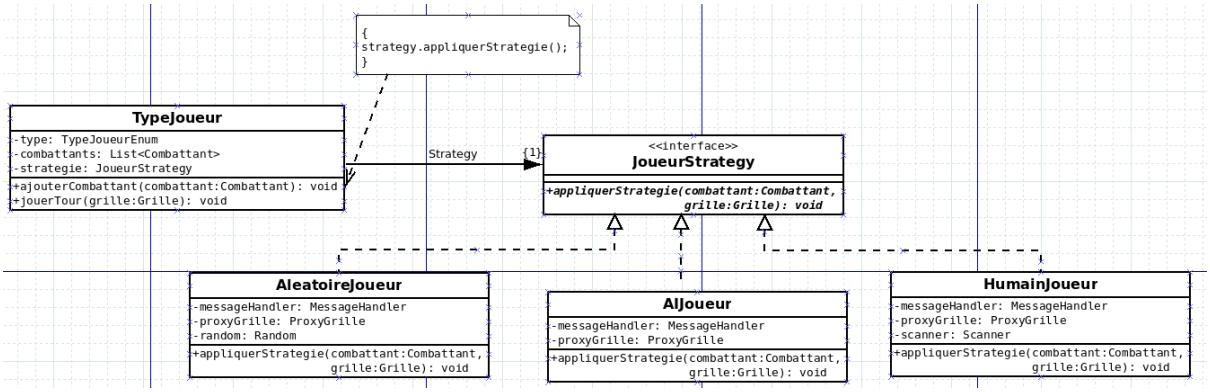

FIGURE 4 – Diagramme UML du Pattern strategy appliqué à la creation des strategie de jeu.

5. Algorithme Heuristique AI

L'algorithme d'heuristique AI est utilisé pour la prise de décisions des joueurs contrôlés par l'intelligence artificielle. Il repose sur :

- **Analyse de l'environnement** : Identification des ennemis proches, obstacles et ressources.
- **Priorisation des actions** : Offensive (tir), défensive (bouclier), tactique (poser des explosifs).
- **Gestion des ressources** : Utilisation optimale des munitions et explosifs.
- **Déplacement stratégique** : Recherche d'une position tactique avantageuse.

Algorithm 1: Algorithme de prise de décision de l'IA

Data: Position des ennemis, munitions disponibles, état du joueur
Result: Action optimale (attaque, défense ou déplacement)

```
Identifier les ennemis proches;
if un ennemi est à portée then
    if les munitions sont suffisantes then
        Attaquer l'ennemi;
    else
        Se replier pour éviter les dégâts;
    end
end
else
    if aucune action immédiate n'est nécessaire then
        Poser une bombe stratégique;
        Se déplacer vers une position défensive;
    else
    end
end
```

6. Tests et Exécution

6.1. Tests effectués

Nous avons réalisé plusieurs tests pour vérifier la robustesse du système :

- **Actions des combattants** : Déplacement, poser une mine, poser une bombe, tir, activation du bouclier.
- ****Pattern Proxy**** : Tests sur la méthode suivante de la classe `Case` :

```
public Case copierAvecFiltrage(Combattant combattant) {
    Case copie = new Case(this.position, this.grille);
    copie.contientPastilleEnergie = this.contientPastilleEnergie;
    if (this.contientPastilleEnergie()) {
        copie.setContientPastilleEnergie(true);
    }
    copie.estMur = this.estMur;
    copie.occupant = this.occupant;

    if (explosif != null && explosif.estVisiblePour(combattant)) {
        copie.explosif = this.explosif;
    }
    return copie;
}
```

- **Mock tests** : Vérification du bon fonctionnement des gestionnaires de messages (`MessageHandler`).

6.2. Exécution

Pour exécuter le projet, consultez le fichier `README.txt` fourni avec le code.

7. Vue dans different mode(Console,Graphique)

```
{Message} :
Tour du joueur de type : ALEATOIRE
{Message} : Vue personnalisée pour ENGINEER1 :

Proxy Grille
 .  #  .  .  .  .  .  .  #  .
 .  .  .  .  .  #  .  .  .  .
 #  .  .  .  .  .  .  .  .  .
 .  .  .  .  #  .  .  .  .  .
 E  #  .  #  .  .  .  #  .  #
 #  .  .  .  .  .  .  .  #  #
 .  #  .  .  #  .  .  .  .  .
 #  .  .  .  .  T  .  .  .  .
 .  .  .  .  .  .  .  #  .  R
 .  .  .  .  .  .  E  .  #  3

{Message} : ENGINEER1 a déposé un mine à la position (0, 3).

Grille Réel
 .  #  .  .  o  .  .  .  #  o
 .  .  .  .  .  .  #  .  .  .
 #  .  .  .  .  .  .  .  o  .
 §  .  o  .  #  .  .  o  .  o
 E  #  .  #  .  .  o  #  .  #
 #  .  .  .  o  .  .  #  #  .
 .  #  .  .  #  .  .  .  .  .
 #  .  .  .  T  o  .  .  o  .
 .  .  .  .  .  .  §  #  .  R
 .  o  o  o  .  .  E  .  #  3

{Message} :
Énergie des combattants :
ENGINEER1 (AI) : 50 Energie
TIREUR1 (AI) : 50 Energie
TANK1 (ALEATOIRE) : 45 Energie
ENGINEER1 (ALEATOIRE) : 50 Energie

{Message} : Appuyez sur Entrée pour passer au tour suivant...
```

FIGURE 5 – Figure du mode console

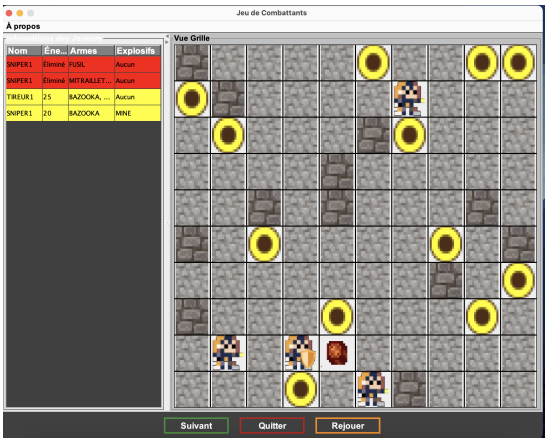

FIGURE 6 – Figure de mode graphique

8. Conclusion et Perspectives

Ce projet montre comment les pactterns de conception et une architecture bien pensée permettent de développer une application robuste et modulaire. Les améliorations futures incluent :

— Une interface graphique plus intuitive.
— Des stratégies IA encore plus avancées.
— Optimisation des performances du moteur de jeu.