

Rapport Technique de La Conception de *GridWar*

Encadré par :

- *ABDELLAHOU姆 Hamza*

Réalisé par :

- *CHAAB Malak*
- *MALLEK Nesrine*
- *TOUADI Nour el Houda*

Le sommaire :

1. **INTRODUCTION**
2. **PRESENTATION DE PROJET**
 - *contexte du projet*
 - *objectifs du projet*
3. **ANALYSE DU PROJET**
 - *architecture de classes*
 - *description des packages*
 - *description des classes*
 - *relation entre les classes*
4. **FONCTIONNALITES DU JEU**
 - *carte de jeu*
 - *ressources*
 - *bâtiments*
 - *unités*
 - *système de combat*
5. **INTERFACE UTILISATEUR**
6. **DIFFICULTES RECONTREES**
7. **CONCLUSION ET PERSPECTIVES**

1.INTRODUCTION :

Dans le cadre du module Programmation Orientée Objet (POO), ce rapport présente la conception et le développement de GridWar, un jeu vidéo de stratégie. Ce projet consiste à concevoir un univers interactif où le joueur doit gérer une faction sur une carte divisée en cases, impliquant la gestion de ressources, la construction de bâtiments et le déploiement d'unités de combat. L'objectif principal de ce travail est de mettre en pratique les concepts fondamentaux de la POO, notamment l'encapsulation, l'héritage et le polymorphisme.

2.PRESENTATION DE PROJET :

2.1 *contexte du projet* :

Le défi principal de ce projet est de traduire des mécaniques de jeu complexes en une architecture logicielle robuste en utilisant le langage Java. Le développement doit mettre en avant l'utilisation des concepts fondamentaux de la POO, tels que :

- L'encapsulation pour la protection des données des unités et des bâtiments.
- L'héritage et les classes abstraites, notamment pour la gestion des différents types d'unités (Soldat, Archer, Cavalier) à partir d'une base commune.
- Les interfaces pour définir des comportements modulaires.
- Les collections Java (comme HashMap ou ArrayList) pour assurer une gestion dynamique et efficace des ressources et des entités présentes sur la carte.

2.2 *objectifs du projet* :

- **Maîtrise de la POO** : Appliquer concrètement les piliers de la programmation orientée objet, à savoir l'héritage, le polymorphisme et l'encapsulation.
- **Architecture Modulaire** : Concevoir une structure logicielle flexible qui permet d'ajouter facilement de nouveaux éléments (unités, cartes, niveaux) sans modifier le code de base.
- **Gestion de Systèmes Complexes** : Implémenter des algorithmes pour la gestion des ressources, le déplacement sur grille et la résolution de combats avec une part d'aléatoire.
- **Qualité logicielle** : Garantir un code propre, documenté, organisé en packages et respectant les principes SOLID pour assurer l'extensibilité du projet

3.ANALYSE DU PROJET :

3.1 architecture de classes :

GridWar/src/com/strategicgame/

|__ StrategicGame.java (main class)

|__ core/

 |__AiController.java

 |__GameJava.java

 |__GameManager.java

 |__GameState.java

|__combat/

 |__ CombatObserver.java

 |__ CombatResolver.java

|__buildings/

 |__Building.java

 |__BuildingFactory.java

 |__BuildingType.java

 |__CommandCentre.java

 |__ResourceFarm.java

 |__ResourceMine.java

 |__ResourceSawmill.java

 |__TrainingCamp.java

|__map/

 |__GameMap.java

 |__Position.java

 |__Tile.java

```
|__TileType.java
```

```
|__units/
```

```
    |__Archer.java
```

```
    |__Cavalry.java
```

```
    |__Soldier.java
```

```
    |__Unit.java
```

```
    |__UnitFactory.java
```

```
    |__UnitType.java
```

```
|__player/
```

```
    |__Faction.java
```

```
    |__Player.java
```

```
|__resources/
```

```
    |__ResourceManager.java
```

```
    |__ResourceType.java
```

```
|__ui/
```

```
    |__GameUI.java
```

```
|__util/
```

```
    |__Constants.java
```

```
    |__GameUtils.java
```

3.2 description des classes :

Le projet est organisé en plusieurs packages :

- **core** pour la gestion de la partie et du tour par tour .
- **map** pour la carte et les cases .
- **player** pour les joueurs .
- **units** pour les unités de combat .
- **buildings** pour les bâtiments .
- **resources** pour les ressources .

3.3 description des classes :

GameManager orchestre la partie (création des joueurs, tours de jeu, fin de partie), tandis que **StrategicGame** pilote la boucle principale et les actions possibles (attaquer, déplacer, construire, entraîner). Les classes **Unit** et ses sous-classes représentent les différentes unités avec leurs points de vie, attaque et défense, alors que **Building** et **CommandCenter** modélisent les bâtiments avec des points de vie et une fonction **takeDamage**.

3.4 relation entre les classes :

Chaque Player possède une collection d'unités, de bâtiments et de ressources, et interagit avec la **GameMap** via le **GameManager**. L'**AiController** utilise ces mêmes classes pour jouer automatiquement le tour du joueur machine, en choisissant des cibles dans les unités et bâtiments adverses.

4.FONCTIONNALITES DU JEU :

4.1 carte de jeu :

La carte est représentée par une grille de cases (Tile) contenant des informations sur le type de terrain et l'occupation (unité ou bâtiment), ce qui permet de vérifier les déplacements et la validité des positions.

4.2 ressources :

Les ressources (par exemple or, bois, nourriture) sont gérées par des classes dédiées et sont consommées lors de la construction de bâtiments ou de l'entraînement de nouvelles unités.

4.3 bâtiments :

Les bâtiments incluent notamment le **CommandCenter**, qui sert de base principale du joueur et qui peut être attaqué et détruit comme les autres bâtiments grâce à la méthode **takeDamage** et au test **isDestroyed**.

4.4 unités :

Les unités de type **soldat** ou **archer** possèdent des statistiques d'attaque, de défense et de points de vie, et peuvent être entraînées, déplacées sur la carte et engagées dans des combats contre les unités ennemis.

4.5 système de combat :

Le système de combat, centralisé dans **CombatResolver**, calcule les dégâts subis en fonction de l'attaque de l'attaquant, de la défense de la cible et d'une part aléatoire, puis met à jour les points de vie et supprime l'unité ou le bâtiment détruit de la liste du joueur.

5.INTERFACE UTILISATEUR :

L'interface actuelle est une interface console gérée par la classe **GameUI**, qui affiche des menus textes pour voir la carte, lister les unités et bâtiments, entraîner, construire, déplacer et attaquer, ainsi que des messages d'erreur ou de confirmation. Une version graphique **JavaFX** a été esquissée mais n'a pas été finalisée, le projet se concentre donc sur une version console fonctionnelle et stable.

6.DIFFICULTES RECONTREES :

La réalisation du projet **GridWar** a été marquée par plusieurs défis, principalement liés à l'organisation et aux contraintes temporelles :

- **Gestion du temps** : Le volume important des fonctionnalités à implémenter (système de combat, gestion des ressources, architecture modulaire) par rapport au délai imparti a nécessité une hiérarchisation stricte des tâches.
- **Coordination à distance** : En raison de la période des vacances et de l'éloignement géographique des membres de l'équipe, nous n'avons pas pu travailler physiquement ensemble. Cela a rendu la communication synchrone plus complexe pour la prise de décisions rapides.
- **Travail collaboratif sur le code** : L'intégration des différentes parties du projet (classes d'unités, gestion de la carte, interface console) a demandé une attention particulière pour éviter les conflits de versions et assurer la cohérence de l'architecture logicielle globale.
- **Cotés technique** : Les principales difficultés techniques ont concerné la gestion cohérente du tour par tour entre le joueur humain et l'IA, ainsi que la séparation claire des responsabilités entre **StrategicGame**, **GameManager** et **AiController**. La mise en place du déplacement sur la carte et l'attaque des bâtiments a également nécessité plusieurs ajustements pour synchroniser les listes d'unités/bâtiments des joueurs avec l'affichage console.

7.CONCLUSION ET PERSPECTIVES :

Le développement de **GridWar** a été un moment fort de notre apprentissage, nous amenant à construire une architecture fidèle aux principes de la

programmation orientée objet tout en répondant à un cahier des charges exigeant. Le jeu final, utilisable en console contre une IA simple, illustre cette organisation structurée en plusieurs ensembles spécialisés et offre de nombreuses pistes d'évolution, notamment une IA plus performante, une interface graphique JavaFX et davantage d'unités, de terrains et de bâtiments.