RÉPUBLIQUE FRANÇAISE
*Liberté*
*Égalité*
*Fraternité*

UNIVERSITÉ CAEN NORMANDIE

Licence 3 Informatique

# FIGHT GAME


*Auteurs :*
DIALLO Mariatou
COTCHO Dewanou
DIALLO Aissatou
MAOUDE Samir

*Chargé du cours :*
MATHET Yann
*Encadrant TP :*
LETELLIER Guillaume


**2 décembre 2024**

# Table des matières

# Introduction

Ce projet s'inscrit dans le cadre du module de Génie Logiciel de la Licence 3 Informatique à l'Université de Caen. L'objectif était de concevoir et développer un jeu de stratégie au tour par tour, mettant en avant des principes essentiels d'ingénierie logicielle tels que l'architecture MVC, l'utilisation de design patterns (Proxy, Factory, Strategy), et la production d'un code modulaire, robuste, et facilement extensible.

Le jeu simule un affrontement entre plusieurs joueurs ou robots sur une grille paramétrable, où stratégie et gestion des ressources (énergie, munitions) jouent un rôle central. Ce rapport présente notre démarche de conception, les choix techniques réalisés, et les outils utilisés pour répondre aux exigences fonctionnelles et non fonctionnelles. Nous détaillons également les aspects algorithmiques et les design patterns employés pour garantir la qualité du code et sa maintenabilité.

# Chapitre 1

# Retour sur le projet et l'organisation

## 1.1 Description du projet

Le projet consiste en la conception d'un jeu de stratégie au tour par tour où des joueurs ou robots s'affrontent sur une grille bidimensionnelle. Chaque joueur contrôle un combattant doté d'une énergie limitée et d'armes variées, telles que des mines, des bombes ou des tirs à portée limitée. L'objectif est de survivre en éliminant les adversaires grâce à des actions stratégiques tout en gérant judicieusement les ressources disponibles.

Le terrain de jeu est une grille paramétrable, composée d'éléments interactifs comme des murs infranchissables, des pastilles d'énergie à collecter et des armes posées ou utilisées par les joueurs. Une particularité du jeu réside dans la visibilité partielle : chaque joueur ne perçoit qu'une portion des informations pertinentes pour ses actions, ce qui est géré via un mécanisme de Proxy.

Pour garantir un développement structuré et évolutif, le jeu repose sur une architecture MVC. Cette approche sépare la logique métier de l'interface utilisateur, permettant de réutiliser le modèle dans différents contextes (interface graphique, terminal, ou application en réseau). Divers design patterns ont été intégrés, notamment Proxy pour la visibilité limitée, Factory pour la création des combattants, et Strategy pour définir les comportements des joueurs et la disposition initiale de la grille.

Le projet met également en avant une configuration facile des paramètres, une simulation avec des joueurs automatisés, et une interface graphique qui intègre des vues multiples pour refléter la perspective de chaque joueur.

## 1.2 Organisation du travail

Le projet a été structuré de façon à garantir une avancée harmonieuse et efficace, avec une répartition des tâches adaptée aux responsabilités de chaque membre de l'équipe. Chaque participant a pris en charge un aspect clé du projet, contribuant activement à son développement global.

### 1.2.1 Répartition des tâches

Pour mener à bien ce projet, nous avons adopté une méthodologie basée sur la collaboration et la spécialisation. Avant de commencer à coder, nous avons pris le temps, en équipe, de réfléchir au sujet, de discuter de la conception globale et de définir les grandes lignes de l'architecture logicielle. Cette étape collective a permis d'assurer une compréhension commune des objectifs et des contraintes du projet, tout en orientant nos choix techniques.

Ensuite, nous avons réparti les tâches en fonction des compétences et des forces de chacun pour maximiser notre efficacité. **Samir** a pris en charge la conception et l'implémentation du modèle du jeu, mettant à profit son expertise en logique métier et en design patterns pour garantir une architecture robuste et respectueuse du paradigme MVC.

**Patrice** s'est concentré sur le développement des classes du package gamePlayer et sur la création de l'interface graphique. Son expérience dans la conception de vues ergonomiques et fonctionnelles a été essentielle pour connecter l'interface utilisateur au modèle du jeu.

**Mariatou** a pris en charge la rédaction du rapport, en synthétisant les aspects techniques et conceptuels du projet dans un document clair et structuré. De son côté, **Aissatou** s'est concentrée sur la production des diagrammes de classe, contribuant ainsi à une meilleure visualisation de l'architecture du projet. Leur travail complémentaire a permis de produire des ressources précises et bien organisées pour expliquer en detail le projet.

En parallèle de nos tâches respectives, nous avons collaboré pour écrire et exécuter les tests unitaires du projet. Ce travail collectif nous a permis de garantir la robustesse de notre code, en vérifiant son bon fonctionnement à chaque étape et en identifiant rapidement les éventuels problèmes.

Cette organisation, alliant réflexion collective, spécialisation ciblée et travail d'équipe, nous a permis d'avancer efficacement tout en maintenant une qualité optimale dans chaque aspect du projet.

## 1.3 Technologies et outils utilisés

Pour réaliser ce projet, nous avons mobilisé plusieurs technologies et outils adaptés aux exigences fonctionnelles et techniques. Ces choix ont permis de garantir une implémentation robuste, modulaire et maintenable.

Nous avons utilisé **Java** comme langage principal pour son adéquation avec la programmation orientée objet et sa richesse en bibliothèques. Pour l'interface graphique, nous avons opté pour les bibliothèques Swing et AWT, qui offrent des outils flexibles pour concevoir des interfaces ergonomiques et personnalisées. Ces technologies ont facilité la création de la vue principale et des vues spécifiques à chaque joueur.

Au niveau de la conception, nous avons intégré plusieurs design patterns :

- **Factory** : pour gérer la création des units, permettant une instanciation flexible et évolutive.
- **Strategy** : pour définir les comportements des joueurs robots et le placement initial des éléments sur la grille.
- **State** : pour définir des joueurs utilisant des stratégies différentes en fonction de leur niveau de vie.
- **Adapter** : pour faire la liaison entre notre model et la vue JTable.
- **Observer** : le modèle notifie ses observateurs, qui SE mettent à jour en conséquence.

Pour assurer la qualité du code, nous avons utilisé JUnit pour écrire et exécuter des tests unitaires. Cette approche a permis de vérifier le bon fonctionnement des différentes composantes du jeu et d'assurer une robustesse accrue.

Enfin, pour la gestion du projet et la collaboration en équipe, nous avons utilisé Git comme système de versionnement. Cela nous a permis de travailler efficacement en parallèle, de suivre l'évolution du code et de résoudre les éventuels conflits de manière structurée.

# Chapitre 2

# Au coeur du code

## 2.1 Configuration du jeu

Ce jeu est totalement configurable à travers le fichier gameSetting.xml se trouvant dans le dossier livraison/src/fightGame.

## 2.2 Packages et classeS

Deux packages principaux ont été développés dans ce projet : le package gamePlayer et le package fightGame. Étant donné le nombre élevé de classes créées, un fichier présentant les diagrammes des classes du projet a été fourni. Ce fichier est disponible dans le dossier livraison/rapport/ressources/.

### 2.2.1 Package gamePlayer

Ce package contient les entités de base qu'on peut retrouver dans un jeu de combat. Nous laissons la possibilité à tout éventuel ajout dans le cas où ce package sera utilisé comme base dans la réalisation d'un autre jeu de combat. Dedans, vous y retrouverez les objets tels que montre l'image 2.1.

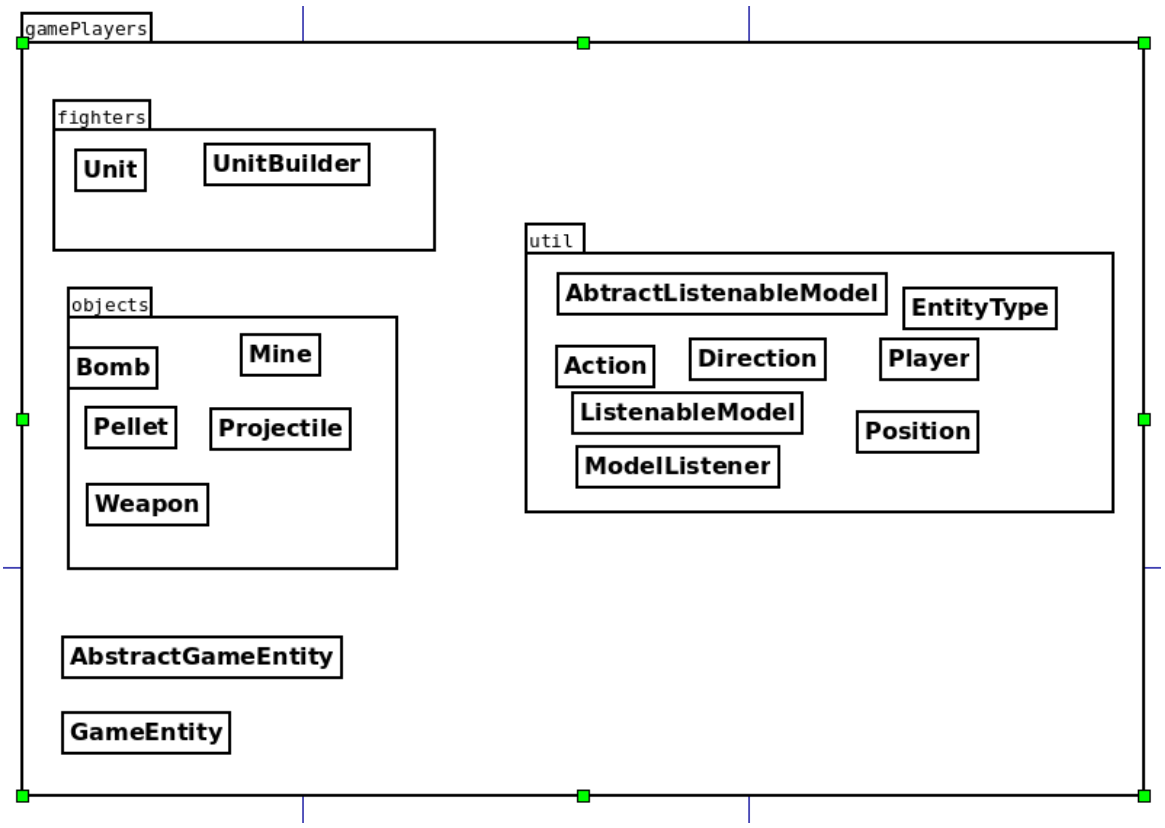


FIGURE 2.1 – gamePlayerPackage

### 2.2.2 Package fightGame

Ce package constitue le cœur du jeu, regroupant le modèle, la vue et les contrôleurs, qui forment ensemble l'architecture MVC.

— Modèle (Model) : Le modèle comprend le gameboard, qui est représenté par un tableau associatif où chaque position contient une liste d'entités. Il intègre également les stratégies du jeu ainsi que les classes responsables de la gestion des entrées et sorties.

— Vue (View) : La vue regroupe les classes dédiées à la création des interfaces graphiques. Elle constitue une partie essentielle du projet en gérant les interactions entre le modèle et l'utilisateur. Chaque composant graphique est développé séparément, puis intégré pour former une interface cohérente et fonctionnelle.

— Contrôleur (Controller) : Le contrôleur inclut un adaptateur permettant de convertir le modèle gameboard en un format compatible avec un JTable. Il contient également le gameManager, chargé de superviser le déroulement du jeu à l'aide d'un second thread, garantissant ainsi une exécution fluide et synchrone.

2.2.

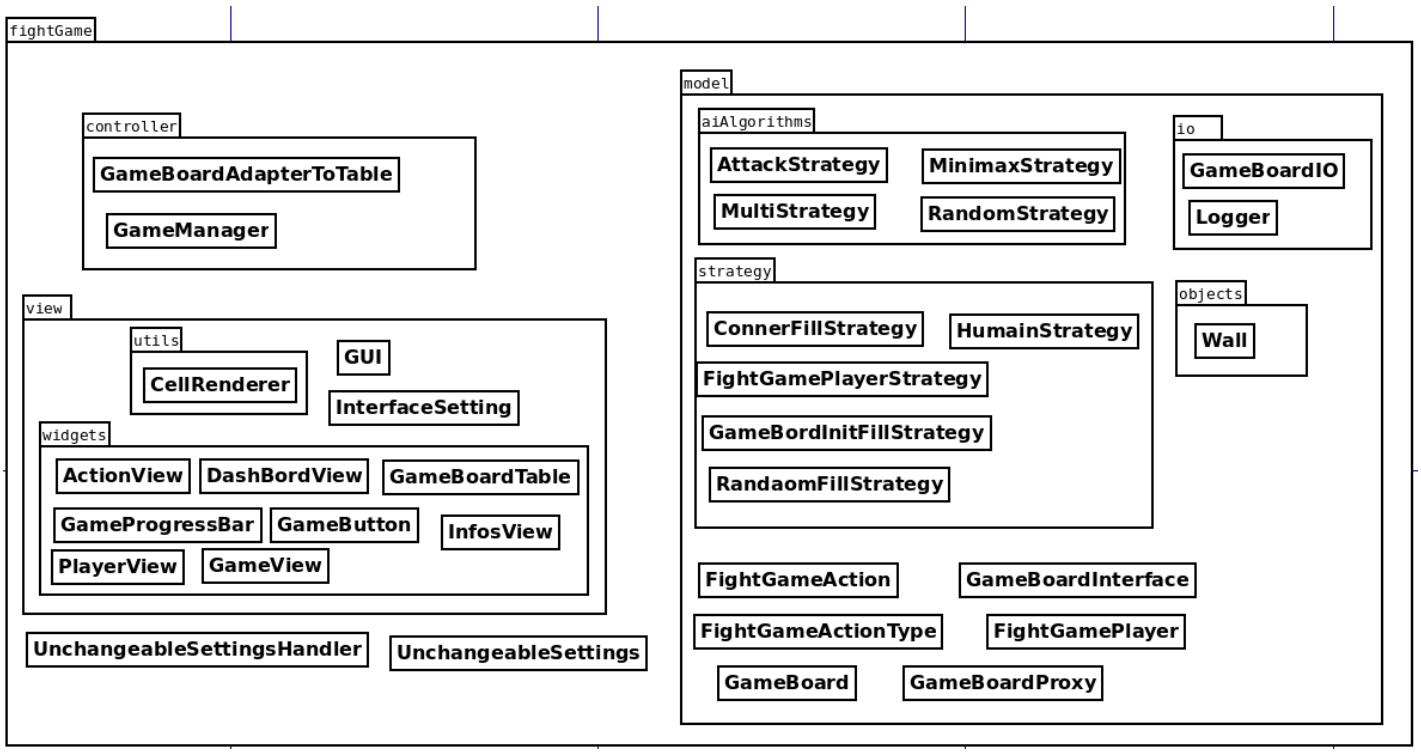


FIGURE 2.2 – Package fightGame

### 2.2.3 Strategy

Dans le cadre de ce projet, nous avons développé deux catégories de stratégies : celles utilisées pour le remplissage initial de la grille et celles adoptées par les joueurs pour leurs actions au cours du jeu.

#### Stratégies de remplissage de la grille

Deux stratégies principales ont été conçues pour organiser la disposition initiale des éléments sur la grille :

1. *RandomFillStrategy* : Cette stratégie remplit la grille en plaçant les joueurs, les pastilles d'énergie et les murs de manière entièrement aléatoire. Cela permet de créer des configurations variées et imprévisibles à chaque partie.
2. *CornerFillStrategy* : Cette stratégie place les joueurs dans les quatre coins de la grille, garantissant une répartition initiale équilibrée. Les autres éléments, tels que les pastilles d'énergie et les murs, sont ensuite disposés de manière aléatoire sur le reste de la grille.

Ces deux approches offrent une flexibilité dans le paramétrage des parties et permettent de tester différents scénarios.

#### Stratégies des joueurs

Quatre stratégies distinctes ont été développées pour définir les comportements des joueurs :

1. RandomStrategy : Cette stratégie choisit une action de manière totalement aléatoire parmi celles disponibles. Elle simule un joueur imprévisible, idéal pour des tests de base.
2. AttackStrategy : Cette stratégie privilégie les actions offensives, incitant le joueur à attaquer agressivement ses adversaires en utilisant des tirs, des bombes ou des mines.

3. MinimaxStrategy : Basée sur l'algorithme Minimax, cette stratégie calcule la meilleure action possible en évaluant les gains et les pertes potentiels à partir de l'état actuel du jeu. Elle simule un joueur réfléchi et stratégique.
4. MultiStrategy : Cette stratégie combine les trois précédentes. Le joueur adapte son comportement en fonction de son niveau de vie.

Ces stratégies permettent de simuler une diversité de comportements, allant du plus simple au plus complexe, offrant ainsi un champ d'expérimentation riche et varié pour le jeu.

**Model**

## 2.3 Gestion de la sauvegarde et du chargement

Le jeu peut être sauvegardé dans les modes Watch, New ou H vs R (Humain contre Robot). Les fichiers de sauvegarde ont l'extension .fightGame. Lors du chargement depuis un fichier de sauvegarde, seuls les modes New et H vs R sont jouables. Si la sauvegarde concerne une partie où l'humain jouait, le jeu continue là où il s'était arrêté. Dans les autres cas, le jeu ne peut être repris qu'en mode New, où il est contrôlé via le bouton Next.

## 2.4 Explication de code non trivial

La méthode `minimax` est une fonction récursive qui explore les différentes actions possibles pour chaque joueur, évalue les scores associés et applique l'élagage alpha-bêta pour optimiser la recherche.

**Étapes de fonctionnement :**

1. **Condition de sortie :** Si la profondeur maximale est atteinte, si le joueur actuel est mort, ou si le jeu est terminé, la méthode retourne le score de l'état courant.

Listing 2.1 – Condition de sortie

```
if (gameBoard.isGameOver() || depth == 0 || !currUnit.isAlive()) {
    double score = this.evaluate(gameBoard.getPlayers());
    return new Result(score, null);
}
```

2. **Récupération des actions possibles :** Le jeu génère les actions possibles pour le joueur en fonction de sa position et de celle de ses adversaires.
3. **Exploration des actions possibles :** Pour chaque action possible :
   — Le plateau de jeu est cloné.
   — L'action est simulée sur le plateau cloné.
   — L'algorithme est appelé récursivement.
4. **Maximisation et minimisation :** L'algorithme alterne entre :
   — Maximisation du score pour le joueur principal.
   — Minimisation du score pour les adversaires.
5. **Élagage alpha-bêta :** L'élagage permet d'abandonner l'exploration d'une branche si elle ne peut pas fournir un meilleur résultat que les scores déjà trouvés.

Listing 2.2 – Elagage alpha-bêta

```
if (beta <= alpha) {
    break;  // Elagage alpha-beta
}
```

**Evaluation**

La méthode `evaluate` calcule le score d'un état de jeu en fonction de plusieurs critères :
— L'énergie restante du joueur (plus elle est élevée, mieux c'est).
— La distance moyenne entre le joueur et ses adversaires. Une distance plus courte favorise un comportement agressif.

# Chapitre 3

# Expérimentation

## 3.1 Lancement du projet

Le projet se lance avec ant. Dans le dossier livraison, il suffit de lancer la commande : **ant run**. L'interface d'accueil se présente alors à l'utilisateur.

## 3.2 Interface d'accueil

Nous avons conçu trois options principales pour expérimenter le jeu :

1. Observer un combat entre robots (**Watch**) : Cette option permet de regarder une partie se dérouler automatiquement, sans intervention humaine. Les joueurs robots agissent en suivant leurs stratégies définies, offrant une simulation autonome du jeu.
2. Contrôler le déroulement du jeu entre robots (**New**) : Ici, les joueurs robots s'affrontent, mais le déroulement est manuel. L'utilisateur doit appuyer sur le bouton "Next" après chaque tour pour avancer dans la partie.
3. Jouer contre des robots (**H vs R**) : Cette option permet à l'utilisateur de participer activement en affrontant les joueurs robots sur la grille.

Ces options sont accessibles via l'interface d'accueil du jeu, offrant une flexibilité à l'utilisateur selon ses préférences. Pour exécuter les deux premières options (combat entre robots), il est impératif de régler le nombre de joueurs humains à zéro dans les paramètres avant de lancer la partie. Cette configuration garantit que seuls les robots participent à la partie. L'image 3.1 nous présente l'interface en question.


FIGURE 3.1 – Interface d'acceuil du jeu

## 3.3 Quelques précisions sur les règles du jeu

— La mort d'un joueur déclenche l'explosion de toutes ses bombes placées
— Dorénavant une mine explose quand on l'ajoute dans la cellule d'un unit
— Le timer de la bombe se décrémente à chaque cycle complet( c'est à dire à chaque fois que c'est au tout de celui qui l'a placée de jouer)

## 3.4 Interface du déroulement d'une partie

L'image 3.3 présente l'interface de déroulement d'une partie du jeu. Sur cette dernière y figurent trois zones. La zone du dashboard qui présente les informations sur l'évolution de l'énergie et des armes du joueur, la zone de la grille qui montre les joueurs et les effets des actions. Enfin, nous avons la zone de contrôle du jeu où se trouvent les boutons de contrôle.

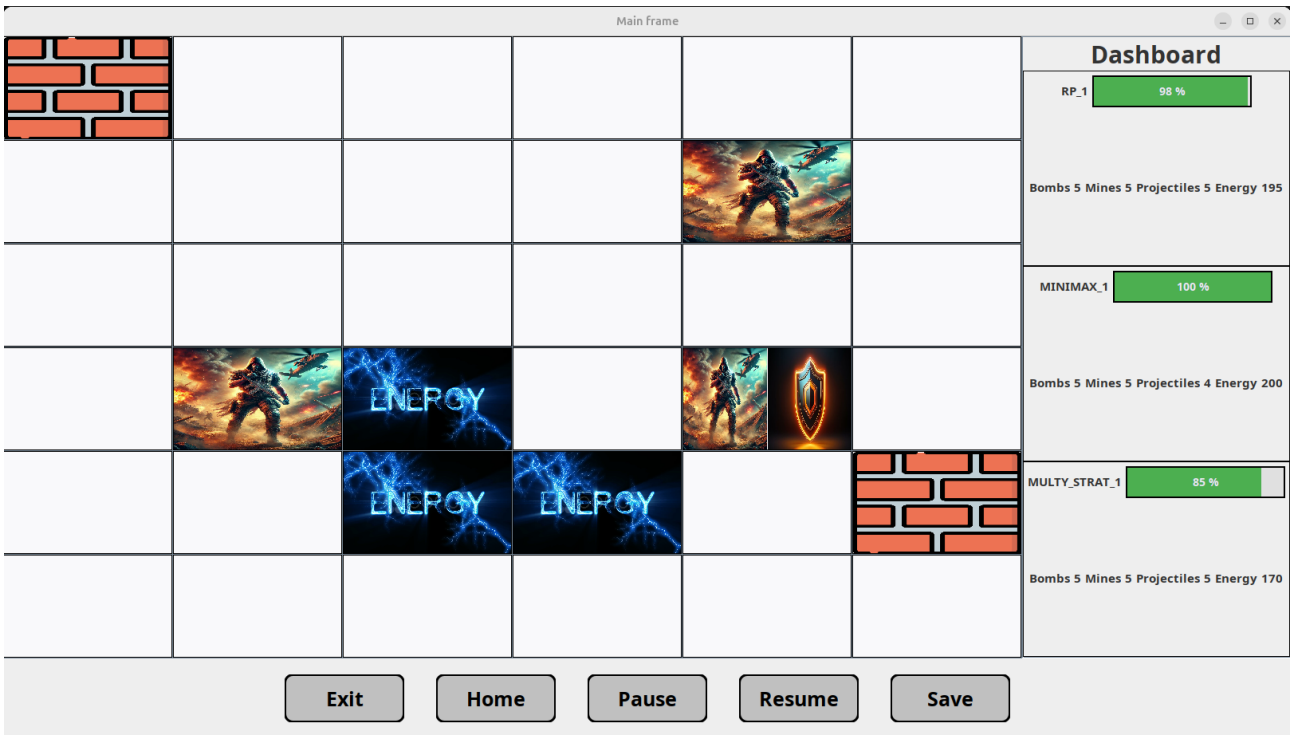


FIGURE 3.2 – Interface du jeu

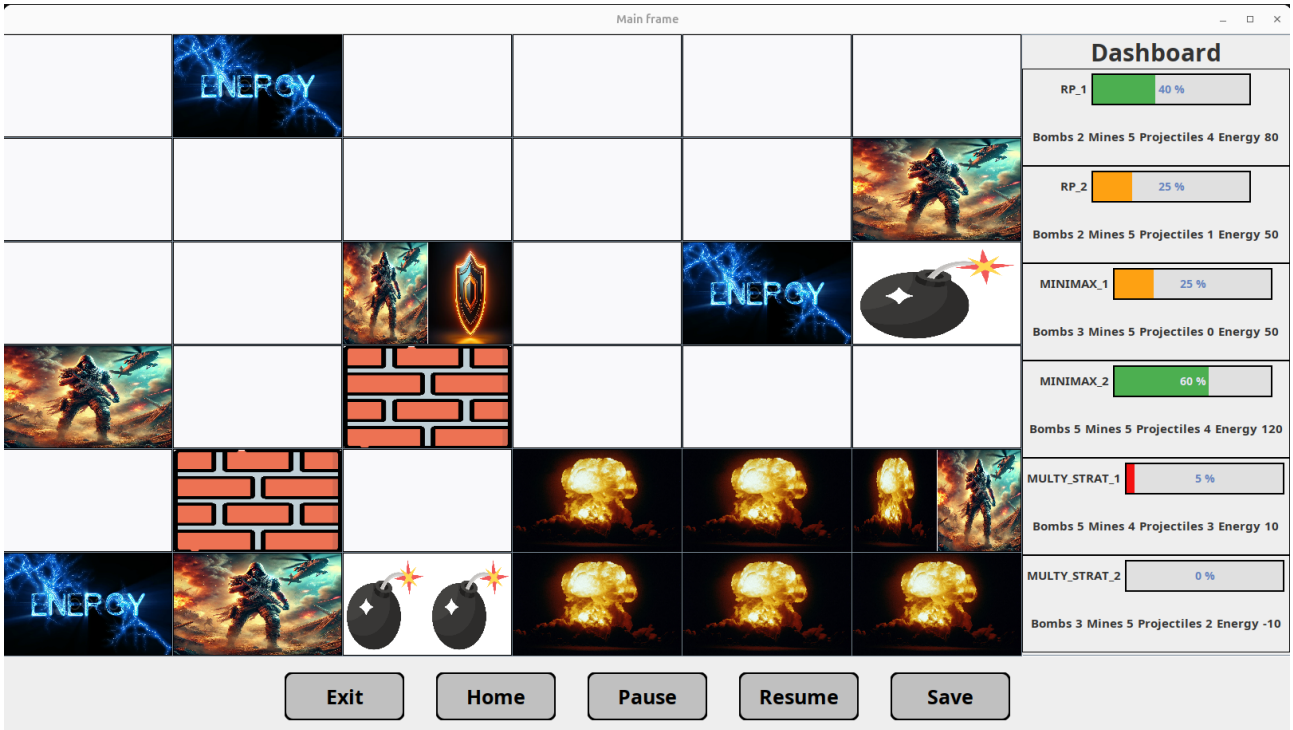


FIGURE 3.3 – Un état du déroulement du jeu

# Conclusion

Ce projet a été une expérience enrichissante qui nous a permis d'appliquer et de consolider nos compétences en conception logicielle et en programmation orientée objet. En développant un jeu de stratégie au tour par tour, nous avons dû relever plusieurs défis techniques, notamment la mise en œuvre d'une architecture modulaire et le respect du paradigme MVC, garantissant la séparation entre le modèle, la vue et le contrôleur.

Grâce à l'utilisation de technologies comme Java, Swing, AWT, et à l'intégration de design patterns tels que Factory, Strategy et State, nous avons pu concevoir un système robuste, évolutif et facile à maintenir. L'écriture de tests avec JUnit et la gestion collaborative via Git ont également joué un rôle clé dans la réussite de ce projet.

En proposant plusieurs options de jeu et en expérimentant différentes stratégies, nous avons pu non seulement rendre l'application interactive et adaptable, mais aussi analyser les comportements des joueurs et la dynamique des parties.

Ce projet a non seulement renforcé notre maîtrise technique, mais aussi notre capacité à travailler efficacement en équipe, de la réflexion initiale à la réalisation finale. Il constitue une base solide pour d'éventuelles améliorations ou extensions futures. Par exemple, l'intégration d'un mode multijoueur en ligne permettrait de rendre le jeu plus interactif et attractif pour les utilisateurs. De plus, le développement de stratégies plus sophistiquées pour les joueurs robots, utilisant par exemple des algorithmes d'apprentissage automatique, pourrait augmenter la complexité et la richesse des parties.