RAPPORT PROJET
PROGRAMMATION ORIENTÉE
OBJET – WARGAME

Période de réalisation : du 26 avril au 19 mai 2025.

Réalisé par :

Marwan BEN MOHAMED
Claudia NANA
Mohamed Rayane RIFFAY
Junior KOUDOGBO
Thed Arthur TOKO DIKONGUE

Encadrement
Mme Dhekra ABOUDA

SOMMAIRE

I - INTRODUCTION

1. Présentation générale du projet

Dans le cadre de notre formation, nous avons été amenés à réaliser un projet de programmation orientée objet en Java. L'objectif était de concevoir et développer une application complète en équipe, en mettant en oeuvre les principes de la modélisation objet, ainsi que des notions de conception logicielle et d'algorithmique que nous avons appris de notre professeur, Madame Dhekra Abouda.

Pour cela, Madame Dhekra a choisi de nous donner comme projet un **Wargame** – un jeu de stratégie au tour par tour opposant deux armées. Ce projet a été réalisé en groupe dans un cadre pédagogique, avec pour ambition de simuler un affrontement militaire sur un plateau de jeu composé de différents types de terrains influençant le comportement des unités.

2. Objectifs du projet

Le projet visait à atteindre plusieurs objectifs :

- Appliquer la programmation orientée objet avec Java (classes, héritage, encapsulation, etc…)
- Utiliser des structures de données adaptées
- Travailler en équipe avec une gestion de projet collaborative
- Concevoir une architecture modulaire
- Mettre en oeuvre des algorithmes de prise de décision automatique à travers le jeu par l'IA

3. Contexte et organisation du projet

La carte de jeu est constituée de différentes cases de terrains (plaine, montagne, forêt, l'eau profonde, village), chacune affectant le comportement des unités présentes sur la carte en termes de mouvement ou de combat.

Les unités aussi ont des caractéristiques spécifiques. Ils ont des potentiels d'attaque, de défense, de vision, de déplacement, et ont aussi des points de vie. L'archer une des unités, a la capacité de d'attaque à distance.
Le détails des spécificités des terrains et unités sera expliqué dans la prochaine partie du rapport.

Ce rapport présente de manière détaillé le projet Wargame. Il inclut :

- Une description fonctionnelle complète des fonctionnalités du jeu
- Des scénarios d'utilisations concrets
- Une présentation de l'interface utilisateur
- Le planning et la répartition des tâches au sein du groupe
- Les choix techniques réalisés, les classes principales et les algorithmes utilisés
- Une rétrospective sur les changements apportés et les axes d'amélioration

II - DESCRIPTION FONCTIONNELLE

1. But du jeu / rôles

Le wargame est un jeu de guerre permettant de simuler des batailles. Le jeu se joue en local entre des humains ou en local contre des IAs.
Le but ici est d'établir une stratégie d'attaque/défense, afin de réduire au silence les unités du camp adverse. Au début, la vision de chaque unité est altérée par le brouillard, qui s'estompe au fur et à mesure que les unités avancent vers la zone adverse.

2. Les unités

Les unités doivent attaquer tout en restant sur leurs gardes et tenir compte de leurs capacités qu'on peut voir ci dessous :

Unité	Attaque	Défense	Mobilité	Vision	PV
Infanterie	5	3	6	4	28
Infanterie lourde	10	10	4	4	38
Cavalerie	8	3	8	6	38
Mage	5	1	5	5	24
Archer	6	2	5	7	33

3. Les terrains

En attaquant, les unités doivent aussi se rendre compte que les terrains sur lesquels elles se trouvent ont une influence sur leurs déplacement, attaque et défense. Plus haut, dans l'introduction, j'avais parlé du fait que l'archer avait la capacité de tir à distance. Cependant, lorsqu'elle a une montagne en face d'elle, ce sera impossible pour elle d'utiliser cette capacité là.
Ci-dessous, les caractéristiques de chaque terrains :

Terrain	Coût de déplacement	Bonus défense	Description
Plaine	1	20 %	Terrain standard
Forêt	2	40 %	Bonne protection
Colline	2	50 %	Excellente défense
Montagne	3	60 %	Défense maximale
Forteresse	1	40 %	Protection moyenne
Village	1	60 %	Meilleure défense
Eau profonde	-	0 %	Infranchissable

4. Mécaniques principales

Le jeu repose sur les trois (03) mécanismes principaux qui sont : le déplacement des unités, l'attaque/défense, et la fin de partie. Nous allons à présent décrire brièvement ces mécanismes, qui seront expliqués en détails dans la suite du rapport.
Pour commencer, on a vu que chaque unité dispose d'un champ de vision (v) ainsi que d'une capacité de déplacement (d) définie. De plus, chaque type de

terrain possède un coût de déplacement (c), à l'exception de l'eau profonde, qui est infranchissable. Lors du tour d'une unité, le système analyse les terrains situés autour d'elle afin de déterminer toutes les combinaisons de déplacements possibles, en veillant à ce que le coût cumulé n'excède pas la capacité (d) de l'unité. Les cases accessibles dans ces conditions sont alors mises en évidence à l'écran.
Cas concret : Si une une unité possède une capacité de déplacement d = 6 et se déplace sur une case dont le coût est c = 3, il lui restera d = 3 pour le reste du tour. Cette capacité est réinitialisée à sa valeur maximale au début du tour suivant.

En ce qui concerne le mécanisme d'attaque/défense, chaque unité possède des statistiques de base en attaque et en défense. Lors d'une attaque, les dégâts sont calculés en soustrayant la défense de la cible (modifiée par le bonus de terrain) de l'attaque de l'unité attaquante. Les terrains jouent un rôle crucial dans ce calcul, offrant des bonus de défense variables : de +20% pour les plaines à +60% pour les forteresses et les montagnes. De plus, des facteurs supplémentaires comme l'état de l'unité (points de vie restants) et le positionnement (attaque de flanc ou par derrière) peuvent modifier les dégâts infligés.

Enfin, la partie se termine lorsque tous les unités du camp adverse sont éliminées. Aussi, une condition de victoire supplémentaire nous a été proposée comme optionnelle : à partir d'un certain nombre de tours n, une équipe peut remporter la partie si une de ses unité conserve tous ses points de vie jusqu'à ce seuil, visant à encourager des stratégies défensives.

III - SCÉNARIOS D'UTILISATION

1. Démarrage du jeu

Au lancement de la partie, le joueur initialise la session de jeu via la méthode *initialiser()*. Celle-ci déclenche ensuite la création de la partie (*creer()*) et la génération de la carte de jeu. Une fois la carte établie, les unités sont placées sur le plateau via *placerUnites()*.

2. Placement initial des unités

La méthode *placerUnites()* permet d'implémenter la logique de positionnement de départ des unités sur la carte. Ce placement peut être aléatoire, prédéfini ou interactif selon les règles du jeu. Il constitue la fin de la phase d'initialisation.

3. Tour d'un joueur

Chaque tour suit une structure qui se répète, représentée par la boucle *loop [pour chaque joueur]*. Le joueur actif commence par appeler la méthode *jouerTour()*, qui déclenche les actions possibles durant son tour.
L'une de ces actions est le déplacement d'unités via la méthode *deplacer()*. Celle-ci appelle ensuite *verifierDeplacement()* pour s'assurer que le déplacement est valide.

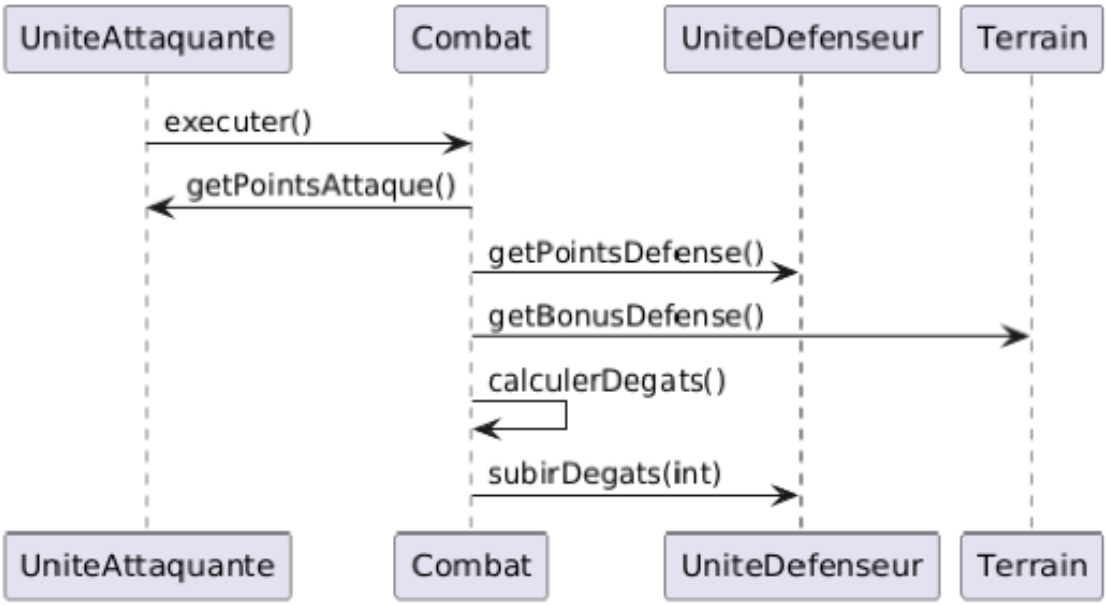
4. Tour de l'IA

Le comportement de l'IA peut suivre une logique similaire, implémentée à travers une instance de *jouerTour()* automatisée, avec ses propres décisions de *deplacer()* et d'engagement en combat.

5. Combat

Un combat commence. L'unité attaquante déclenche l'action en appelant le système de combat. Ce dernier lui demande sa puissance d'attaque. Ensuite, le système interroge l'unité défenseur pour connaître sa capacité de défense, puis consulte le terrain pour savoir s'il apporte un bonus défensif.
Avec toutes ces informations, le système calcule les dégâts que l'attaque va infliger. Une fois les dégâts déterminés, ils sont appliqués à l'unité défenseur, qui les subit. Le tour de combat est terminé.

Cette partie est représentée par le diagramme de cas d'utilisation ci-dessous :



IV - INTERFACE UTILISATEUR

Notre projet contient 2 fenêtres principales : le menu et le plateau de jeu. Des images des différentes fenêtres seront présentes comme guide visuel.

1. Menu

Le menu est composé de 4 boutons. « Nouvelle Partie », « Charger une partie », « Règles », « Quitter ». Nous avons un dossier « Saves » dans notre projet qui permet de stocker les parties sauvegardées, pour pouvoir les charger par la suite.

2. Plateau de jeu

Comme expliqué plus haut dans le document, le plateau de jeu est composé de terrains de forme hexagonales. Les unités sont placées sur les terrains avec leurs points de vie de départ. Afin de différencier les unités de chaque équipe, un voyant bleu ou rouge est placé au dessus de l'unité en fonction de son équipe pour une meilleure visibilité.

Lorsqu'une attaque est effectuée, un voyant est déclenché indiquant que l'attaque a eu lieu, suivi d'un retrait de points de vie de la cible. Si une unité est complètement anéantie, elle disparait alors du plateau de jeu.

V - PLANNING DU PROJET

Ayant reçu le sujet du projet le 26 Avril , nous avons eu jusqu'au 30 Avril pour composer les groupes et l'envoyer à Madame Dhekra. Notre groupe est donc composé de 5 personnes, avec leurs noms mentionnés dans la première page de ce rapport.

Notre objectif était d'être capable de rendre le rapport final, le code source et le powerpoint le 22 Mai. Pour cela, une répartition des tâches et un suivi de l'évolution de projet à travers des réunions étaient nécessaires afin d'atteindre cet objectif. La répartition des tâches a été faite le 2 Mai, ensuite, nous avons commencé le projet. Le 9 Mai, à la fin de la semaine, nous avons eu une première réunion de suivi visant à connaitre l'évolution de nos travaux respectifs (mise en place de l'environnement, interface, composants de base, tests). Le 13 Mai, suite à une réunion, suivi d'un travail d'équipe, nous avons pu mettre en place la première version du jeu fonctionnel et l'avons déposé sur le répo Git dédié, afin de continuer à améliorer le projet et fixer les différents bugs qu'il pouvait rencontrer.

Nous avons rencontrés plusieurs difficultés, notamment l'affichage des hexagones sur le plateau de jeu. Évidemment, l'algorithme permettant de placer correctement les hexagones sur le plateau de jeu n'était pas intuitif et facile à implémenter. Au début, on a voulu placer les hexagones, coté plat vers le haut, mais nous nous sommes rendus compte que le décalage entre le hexagones était difficile à gérer et avons opté pour le coté Peak vers le haut.

Une autre difficulté était de penser à la structure du projet qui n'était pas évidente non plus. Pour cela, nous avons utilisé ChatGPT, qui nous a fourni une structure de projet afin qu'on puisse commencer. Pour finir, la logique du jeu, notamment les conditions d'attaque/défense, ainsi que la logique IA étaient difficiles à implémenter et donc nous a fait perdre énormément de temps, nous poussant à s'engager à travailler pendant les week-ends, en dehors des heures de cours.

VI - RÉPARTITION DES TÂCHES

Nous avons fait la répartition en 5 grandes parties. Logique de jeu, Interface graphique, Gestion des données, IA et Évènements, Extensions.

Voici une vue d'ensemble des différentes tâches par partie :

1. Logique de jeu (Core)

- Implémentation des règles de jeu
- Gestion des tours de jeu
- Système de combat
- Calcul des déplacements
- Classes de base
- Tests pour la logique de base

2. Interface graphique (GUI)

- Création de la fenêtre principale
- Affichage du plateau hexagonal
- Rendu des unités et terrains
- Gestion des interactions utilisateur

- Menu du jeu

3. Gestion des données (Data)

- Système de sauvegarde/chargement
- Gestion des ressources (images)
- Base de données des unités et terrains
- Configuration du jeu

4. IA et Évènements

- Implémentation de l'IA des adversaires
- Actions d'opportunité
- Pathfinding
- Stratégies de l'IA
- Tests comportements de l'IA

5. Extensions

- Système de tir
- Gestion du brouillard de guerre
- Coordination de l'intégration des composants
- Test d'intégration

Les rôles ont été attribués comme suit :

```
src/
├── main/
│   ├── java/
│   │   ├── core/          # Junior
│   │   ├── gui/           # Rayane
│   │   ├── data/          # Claudia
│   │   ├── ai/            # Thed
│   │   └── extensions/    # Marwan
│   └── resources/
└── test/
    └── java/
```

VII - MODÉLISATION ORIENTÉE OBJET

Description des classes principales

Notre projet est constitué de 8 classes principales décrites ci-dessous, suivi d'un diagramme de classes représentatif.

a. Partie

C'est la classe centrale qui gère l'état d'une partie de jeu. Elle a les attributs principaux :

- carte : la carte de jeu
- unitesJoueur1 et unitesJoueur2 : listes des unités de chaque joueur
- joueurCourant : indique quel joueur joue actuellement
- tour : numéro du tour en cours
- zoneVisibilite : gère la visibilité des unités

Elle gère les méthodes de gestion des tours et des joueurs, ajout et gestion des unités, vérification de la fin de partie, récupération des points de vie.

b. Unite

Représente une unité militaire dans le jeu. Ses attributs principaux :

- type : type d'unité
- position : position sur la carte
- pointsDeVie : santé de l'unité
- pointsDeplacementRestants : points de mouvement restants
- joueur : propriétaire de l'unité (1 ou 2)
- porteeVision : distance de vision de l'unité

Elle gère les méthodes de gestion de déplacements, gestion des points de vie, vérification de l'état de l'unité.

c. Carte

Représente la carte de jeu, avec les attributs principaux suivants :

- largeur et hauteur : dimensions de la carte
- terrains : map des terrains par position
- unites : map des unités par position

Cette classe a les méthodes de gestion du placement des unités, gestion des déplacements, vérification de la validité des positions, accès aux informations de terrain et d'unités.

d. Position

Classe utilitaire pour gérer les coordonnées sur la carte, contient les coordonnées x et y.

e. Terrain

Énumération des différents types de terrains possibles. Chaque type de terrain peut avoir des effets sur le gameplay.

f. TypeUnite

Énumération des différents types d'unités, définit les caractéristiques de base de chaque type d'unité.

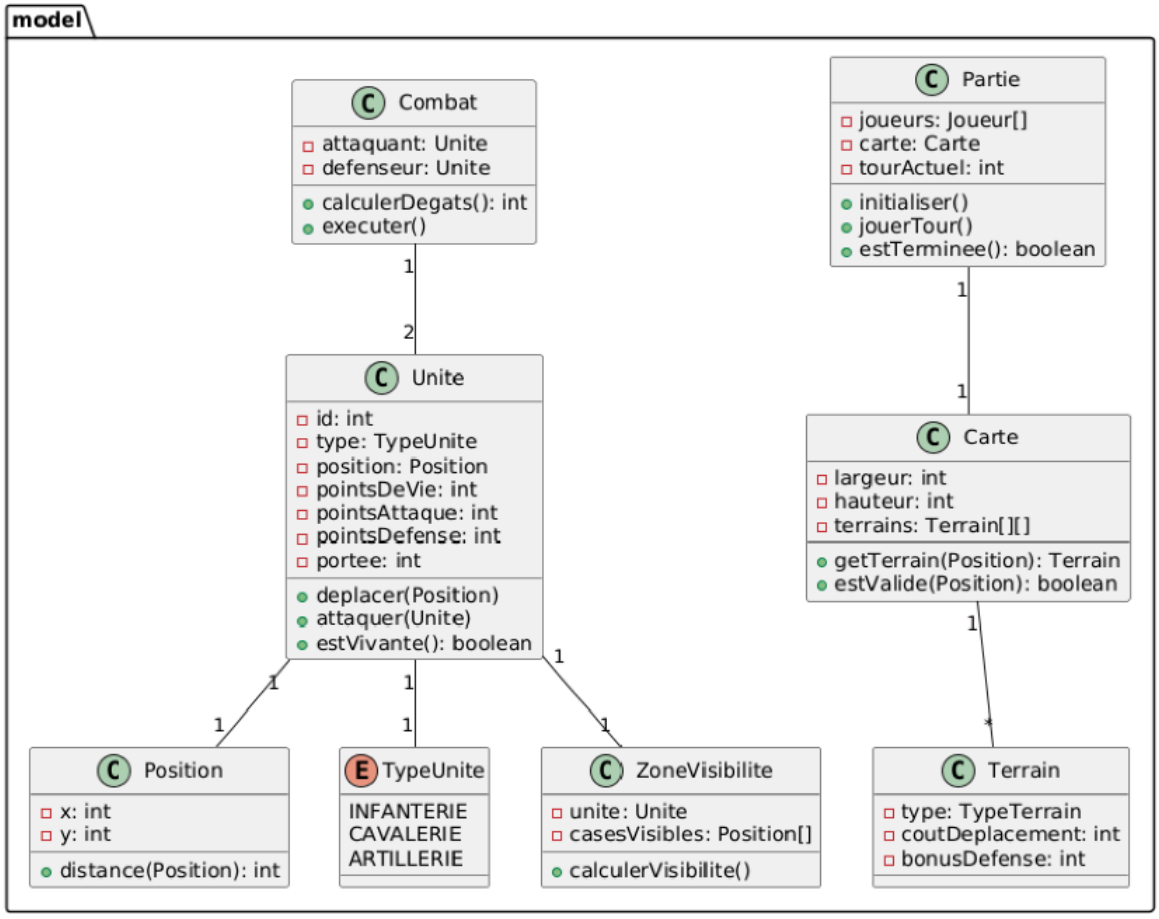
g. ZoneVisibilite

Gère la visibilité des unités de la carte, calcule quelles cases sont visibles pour chaque joueur.

h. Combat

Gère les mécaniques de combat entre les unités, calcule les dégâts et les résultats des affrontements.

Cette architecture suit un modèle orienté objet avec un séparation claire des responsabilités entre les différentes classes. Nous l'avons représenté avec un diagramme des classes UML que vous pouvez voir dans la figure ci-contre :



VIII - ALGORITHMES IMPORTANTS

A. Algorithme de l'IA

L'algorithme suit une approche gloutonne. L'IA récupère la liste de ses unités et celle des unités ennemies. Le choix dépend du numéro du joueur IA (1 ou 2). Ensuite, il parcourt toutes les unités de l'IA en ignorant les unités mortes. Elle va donc rechercher une cible : pour chaque unité, trouve l'ennemi le plus proche, calcule la distance entre l'unité et chaque ennemi, la garde en mémoire.

Recherche de cible:

```
// Chercher l'ennemi le plus proche
Unite ciblePlusProche = null;
double distanceMin = Double.MAX_VALUE;

for (Unite ennemi : unitesEnnemi) {
    if (!ennemi.estVivant()) continue;

    double distance = unite.getPosition().distance(ennemi.getPosition());
    if (distance < distanceMin) {
        distanceMin = distance;
        ciblePlusProche = ennemi;
    }
}
```

Si un ennemi est trouvé, si l'ennemi est à la portée (distance <= 1.5) alors il y a attaque, sinon, si l'unité peut se déplacer, il se rapproche de l'ennemi.

Prise de décision :

```
if (ciblePlusProche != null) {
    // Si l'ennemi est proche, l'attaquer
    if (unite.getPosition().distance(ciblePlusProche.getPosition()) <= 1.5) {
        Combat.resoudre(unite, ciblePlusProche,
            partie.getCarte().getTerrain(ciblePlusProche.getPosition()));
        unite.consommerPointsDeplacement(points:1);
    }
    // Sinon, essayer de se rapprocher
    else if (unite.peutSeDeplacer()) {
        deplacerVersPosition(unite, ciblePlusProche.getPosition());
    }
}
```

Pour se déplacer, l'IA évalue les 8 cases adjacentes possibles, vérifie si le déplacement est valide et choisi la case qui minimise la distance vers la cible.

B. Gestion de l'attaque/défense

```
// Calcul du bonus de défense du terrain
int bonusDefense = terrainDefenseur.getBonusDefense();
int defenseEffective = defenseur.getType().getDefense() +
    (defenseur.getType().getDefense() * bonusDefense) / 100;

// Calcul des dégâts bruts
int degatsBruts = attaquant.getType().getAttaque() - defenseEffective;

if (degatsBruts <= 0) {
    degatsBruts = 1; // Dégâts minimum
}

// Application de l'aléatoire
int variationAleatoire = random.nextInt(POURCENTAGE_ALEATOIRE + 1) - (POURCENTAGE_ALEATOIRE / 2);
int degatsFinaux = Math.max(a:1, degatsBruts + (degatsBruts * variationAleatoire) / 100);
```

La défense est calculée : défense de l'unité + bonus du terrain. Les dégâts sont calculées : attaque - défense (minimum 1). Une variation aléatoire de ±25% est appliquée. Les dégâts sont infligés à l'unité défenderesse. L'attaque est une valeur fixe selon le type d'unité.

C. Calculs des effets terrains

Le calcul de la défense effective est basée sur les caractéristiques des terrains et s'effectue comme suit : *Défense effective = Défense de base + (Défense de base * Bonus terrain)*

D. Déroulement d'un tour / Fin de partie :

```java
public void finDeTour() {
    // Récupération des PV pour les unités non déplacées et non attaquées
    List<Unite> unitesJoueurCourant = (joueurCourant == 1) ? unitesJoueur1 : unitesJoueur2;
    for (Unite unite : unitesJoueurCourant) {
        if (unite.estVivant() && !unite.peutSeDeplacer()) {
            unite.recupererPV(TAUX_RECUPERATION);
        }
    }
    // Changement de joueur
    joueurCourant = (joueurCourant == 1) ? 2 : 1;

    // Si le joueur 1 reprend la main, on incrémente le tour
    if (joueurCourant == 1) {
        tour++;
    }
    // Réinitialisation des points de déplacement
    reinitialiserPointsDeplacement();
    // Mise à jour de la visibilité pour le nouveau joueur
    zoneVisibilite.calculerVisibilitePourJoueur(joueurCourant);
}
```

Un tour se déroule en 5 phases.

A) Phase de récupération : les unités qui n'ont pas bougé récupèrent 10% de leurs PV

B) Phase de changement de joueur : le joueur actif change (1 -> 2 ou 2 -> 1)

C) Phase de gestion du tour : Si le joueur 1 reprend la main, le numéro de tour augmente. Maximum 20 tours pour la victoire du défenseur.

D) Phase de réinitialisation : toutes les unités du nouveau joueur actif récupèrent leurs points de déplacement.

E) Phase de la mise à jour : La visibilité est recalculée pour le nouveau joueur actif

La partie se termine si toutes les unités d'un joueur sont détruites, ou si le nombre de tours (20) est atteint et le défenseur a encore son nombre de vies initial.

IX - CONCLUSION

Ce projet de wargame en Java a été une vraie opportunité pour mettre en pratique nos connaissances en programmation orientée objet et pour apprendre à travailler efficacement en équipe sur un projet concret.

Tout au long de la conception du jeu, nous avons appris à structurer notre code, à organiser des classes qui interagissent de manière cohérente, à gérer la complexité croissante du système de jeu, et à prendre en compte les contraintes propres à un tour par tour stratégique. Nous avons particulièrement apprécié de devoir réfléchir à la logique de l'IA, à la gestion des unités et à l'impact des terrains sur les mécaniques de combat.

Même si certaines fonctionnalités prévues au départ ont dû être simplifiées ou mises de côté, nous sommes satisfaits du résultat final. Le jeu est jouable, et équilibré pour permettre de futures évolutions. Nous voyons d'ailleurs plusieurs axes intéressants d'amélioration : enrichir le comportement de l'IA, améliorer le graphisme (user friendly).

Remerciements

Nous tenons à remercier chaleureusement notre professeure **Mme Dhekra ABOUDA** pour son accompagnement tout au long du semestre, ses cours pertinents, et la liberté qu'elle nous a laissée dans la conception du jeu. Son encadrement nous a permis de progresser à la fois techniquement et humainement.