Université Paris Cité

RAPPORT DE PROJET

BD1C

BERNAUD Vinh-San
LUCAS Ivan
MAHANDRY Tania
RIPASARTI Adriano
YANG Jacqueline

I. INTRODUCTION

« Tour de défense » est un jeu de stratégie dans lequel l'objectif est d'empêcher des créatures ennemies d'atteindre une zone d'arrivée. Dans notre version sans labyrinthe, les créatures ennemies apparaissent par troupes un peu partout dans le jeu dans des zones prédéfinies et cela durant plusieurs vagues. Afin de gagner, le joueur devra résister aux différentes vagues ennemies. Pour cela, il pourra placer des tours de défense qui pourront attaquer les ennemis qui passeront à leur portée. Le joueur doit donc essayer de créer des chemins qui forcent les troupes ennemies à rester le plus possible sous les tirs des tours.

Pour déterminer la direction qu'allait prendre notre projet, nous nous sommes inspirés de différents jeux de ce type déjà existant. Ainsi, l'objectif était de créer de toute pièce et le plus fidèlement possible un jeu du type "Tour de défense" en regroupant les fonctionnalités les plus pertinentes des jeux que nous connaissons déjà, en y ajoutant certaines fonctionnalités pouvant apporter une meilleure expérience au joueur. Il s'agit aussi d'utiliser au mieux les outils orientés objet mis à disposition par java. Toutes les fonctionnalités qui ont été implémentées ont pour vocation de représenter au mieux le fonctionnement des jeux de ce type.

Le thème du projet est le système immunitaire : la carte représente une partie du corps, les ennemis sont des agents pathogènes et les tours incarnent le système immunitaire chargé d'éliminer ces dangers. Le choix de ce thème s'est fait d'un commun accord entre tous les membres du groupe, amené par le contexte particulier à ce moment dont nous avons souhaité tirer du positif en le rendant ludique. Nos connaissances en biologie sont cependant assez approximatives, ce qui se reflète dans notre jeu, qui n'a donc pas la faculté d'être utilisé à des fins didactiques.

II. DESCRIPTION DU PROJET

Le joueur doit éliminer tous les ennemis le plus rapidement possible dans l'espoir de survivre aux vagues successives qui sont de plus en plus difficiles au fur et à mesure de l'avancée de la partie.

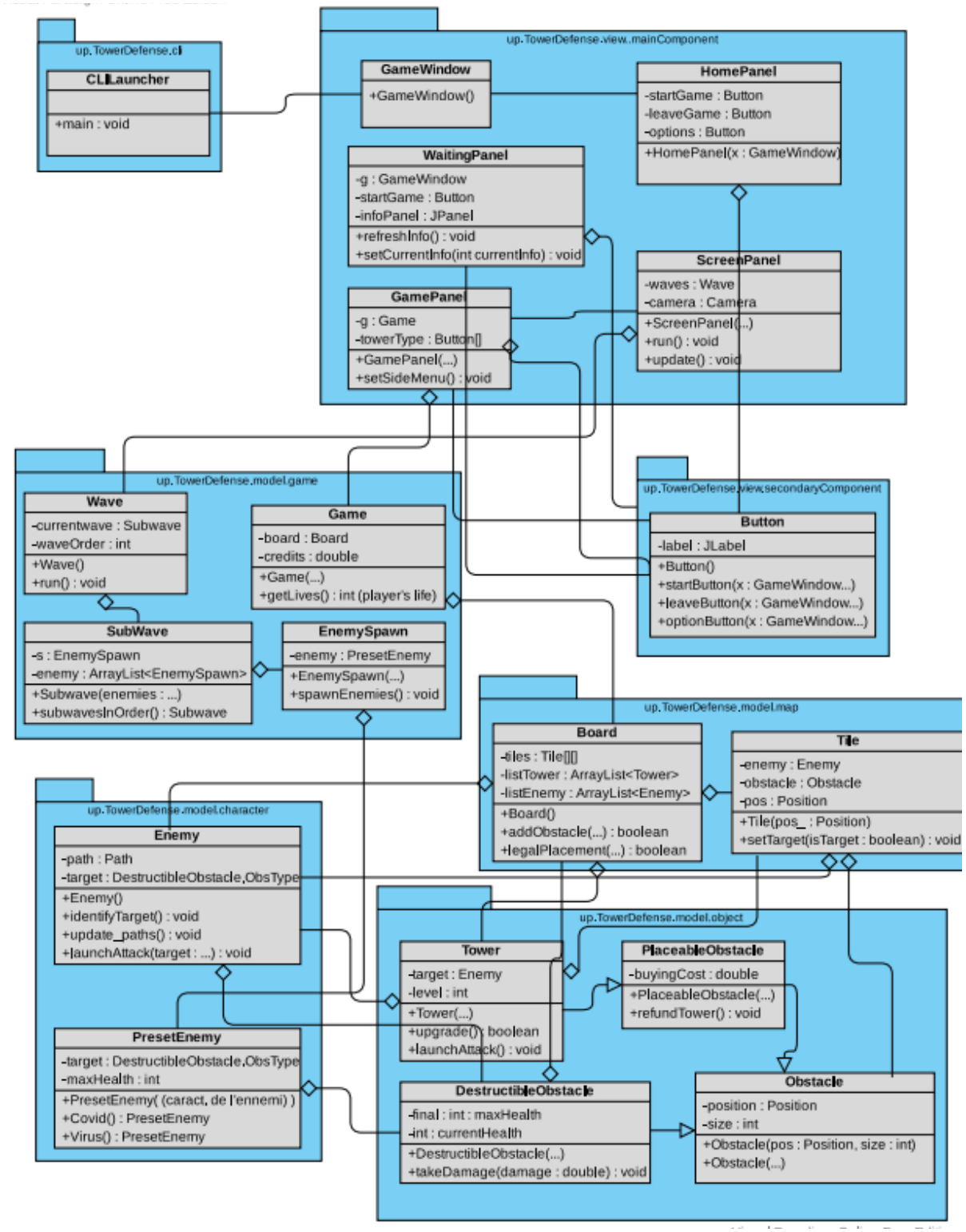
Le projet doit regrouper toutes les fonctionnalités nécessaires au bon fonctionnement du jeu. Le joueur doit pouvoir effectuer des parties avec un nombre de vie initial limité contre plusieurs vagues d'ennemis, interagir avec la carte et effectuer des actions telles que l'achat et le placement des tours, récupérer de l'argent comme récompense en tuant des ennemis, ainsi que d'autres fonctionnalités techniques inhérentes à un jeu de ce type. Mais, il devra aussi répondre à des propriétés non fonctionnelles par sa qualité et ses performances.

A. Conception du Projet.

Le code source est situé dans le dossier : **view/src/main/java/up/TowerDefense/**

Le projet a été construit de sorte que lors de l'exécution de la classe principale CLILauncher, l'interface graphique soit exécutée, ouvrant la fenêtre principale du jeu qui permet au joueur de sélectionner les paramètres de la partie et de la lancer.

Cette interface échangera des informations avec le modèle, la vue et le contrôleur tel que décrit ci-dessous :


up.TowerDefense.cli
CLILauncher
+main : void
up.TowerDefense.view.mainComponent
GameWindow
+GameWindow()
HomePanel
-startGame : Button
-leaveGame : Button
-options : Button
+HomePanel(x : GameWindow)
WaitingPanel
-g : GameWindow
-startGame : Button
-infoPanel : JPanel
+refreshInfo() : void
+setCurrentInfo(int currentInfo) : void
ScreenPanel
-waves : Wave
-camera : Camera
+ScreenPanel(...)
+run() : void
+update() : void
GamePanel
-g : Game
-towerType : Button[]
+GamePanel(...)
+setSideMenu() : void
up.TowerDefense.model.game
Wave
-currentwave : Subwave
-waveOrder : int
+Wave()
+run() : void
Game
-board : Board
-credits : double
+Game(...)
+getLives() : int (player's life)
SubWave
-s : EnemySpawn
-enemy : ArrayList<EnemySpawn>
+Subwave(enemies : ...)
+subwavesInOrder() : Subwave
EnemySpawn
-enemy : PresetEnemy
+EnemySpawn(...)
+spawnEnemies() : void
up.TowerDefense.view.secondaryComponent
Button
-label : JLabel
+Button()
+startButton(x : GameWindow...)
+leaveButton(x : GameWindow...)
+optionButton(x : GameWindow...)
up.TowerDefense.model.map
Board
-tiles : Tile[][]
-listTower : ArrayList<Tower>
-listEnemy : ArrayList<Enemy>
+Board()
+addObstacle(...) : boolean
+legalPlacement(...) : boolean
Tile
-enemy : Enemy
-obstacle : Obstacle
-pos : Position
+Tile(pos_ : Position)
+setTarget(isTarget : boolean) : void
up.TowerDefense.model.character
Enemy
-path : Path
-target : DestructibleObstacle,ObsType
+Enemy()
+identifyTarget() : void
+update_paths() : void
+launchAttack(target : ...) : void
PresetEnemy
-target : DestructibleObstacle,ObsType
-maxHealth : int
+PresetEnemy((caract. de l'ennemi))
+Covid() : PresetEnemy
+Virus() : PresetEnemy
up.TowerDefense.model.object
Tower
-target : Enemy
-level : int
+Tower(...)
+upgrade() : boolean
+launchAttack() : void
PlaceableObstacle
-buyingCost : double
+PlaceableObstacle(...)
+refundTower() : void
DestructibleObstacle
-final : int : maxHealth
-int : currentHealth
+DestructibleObstacle(...)
+takeDamage(damage : double) : void
Obstacle
-position : Position
-size : int
+Obstacle(pos : Position, size : int)
+Obstacle(...)

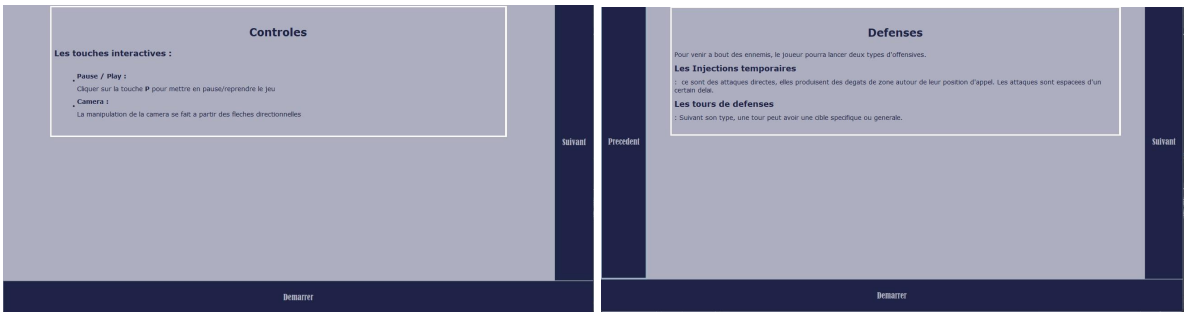
L'intégralité des objets utilisés dans le projet n'a pas été représentée ici, seules les classes principales sont présentes.

B. Implémentation du projet, propriétés fonctionnelles :

- Menu d'accueil

Sur le menu d'accueil, le joueur a accès à différents boutons cliquables. Un pour lancer la partie, un autre pour quitter l'interface et un dernier pour ouvrir un menu permettant de configurer différents paramètres de la partie telle que son nombre de niveaux.

- Menu principal



Dans le menu principal, le joueur a accès aux différentes composantes du jeu, en particulier aux boutons permettant d'interagir avec le jeu.

- Environnement de jeu

En lançant la partie, la carte est générée et le joueur peut placer les différents types de tours présents dans le jeu ainsi que des murs pour forcer les ennemis à dévier de leur trajectoire. Il dispose d'une certaine quantité d'argent au départ puis à chaque ennemi tué le joueur gagne une certaine somme d'argent dépendante du type d'ennemi tué. Il peut aussi modifier différentes options de jeu comme la vitesse de jeu.

- Ennemis

Le jeu comporte différents types d'ennemis, chacun avec des caractéristiques différentes. Chaque ennemi possède une résistance, puissance, une portée d'attaque, une vitesse de déplacement et une agressivité.

Les statistiques de chaque ennemi sont données dans les annexes.

Chaque type d'ennemi a une image et une taille qui permettent de l'afficher sur la carte. Le déplacement des ennemis est déterminé par l'algorithme A* que nous détaillons un peu plus loin.

Bacterium

Champignon

Virus

Parasite

Covid

- Tour

Le jeu comporte différents types de Tour, chacune ayant des caractéristiques différentes. Chacune à un prix que le joueur doit payer avec ses gains pour pouvoir la placer.

Anti-Champis

Anticorps

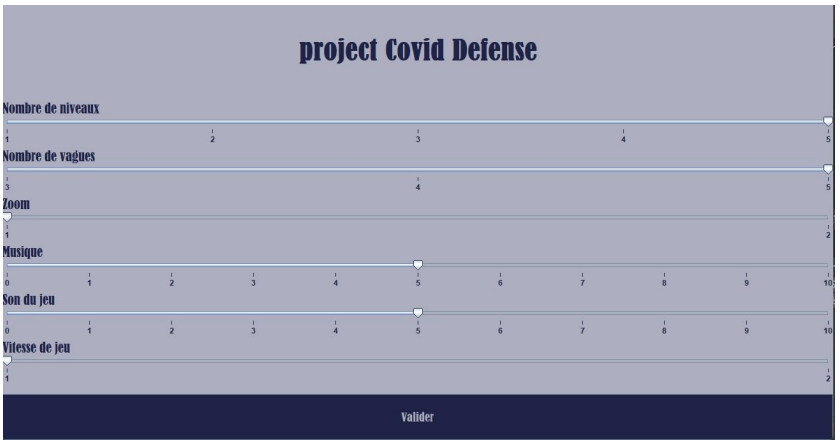
Leucocytes

- Mur

Le joueur peut placer des murs sur la carte et ainsi forcer les ennemis à faire un détour pour rejoindre la zone ciblée.

- Déroulement de la Partie

La partie comportera un certain nombre de niveaux et de vagues déterminés par le joueur dans les paramètres avant le début de la partie.



Les numéros du niveau et de la vague en cours sont indiqués dans l'interface graphique. Lors de chaque vague un certain nombre d'ennemis apparaît et le joueur doit les empêcher par tous les moyens d'arriver à la zone qu'il doit protéger. Le joueur commence la partie avec 20 vies et 1000 unités d'argent et à chaque ennemi atteignant la zone à protéger, le joueur perd une de ses vies.

Le nombre d'ennemis restant à tuer dans la vague en cours est également indiqué dans l'interface graphique.

- Déplacement et autre

Le joueur peut aussi se déplacer sur la carte grâce aux flèches directionnelles. Il a par ailleurs la possibilité de mettre le jeu en pause en appuyant sur la touche « P ». Ces commandes sont précisées avant le début de la partie.

La carte est parsemée de zones où le joueur ne peut pas poser de tour et où les ennemis ne peuvent pas accéder. Il y a une carte différente pour chaque niveau, ce qui donne de la diversité à l'expérience du joueur au cours de sa partie.

Pour l'algorithme de pathfinding, nous avons choisi d'implémenter A*, car ce dernier nous semblait particulièrement bien adapté à notre modélisation de la carte et à nos attentes. Pour "lisser" le trajet des ennemis, nous avons implémenté une classe SubPath qui permet de découper le chemin renvoyé par le pathfinding, et d'en faire des arcs de cercles, ce qui donne au déplacement des ennemis un rendu plus agréable à l'œil.

C. Propriétés non fonctionnelles :

Afin de faciliter au maximum l'utilisation du jeu et de rendre plus ergonomique le partage des informations entre celui-ci et le joueur, la mise en page du jeu doit faciliter au maximum la démarche à l'aide d'une présentation claire et intuitive

III. REALISATION TECHNIQUE

Le langage majoritaire utilisé dans ce projet est Java.

A. Les différents outils utilisés :

- Gradle

Nous avons choisi d'utiliser gradle pour l'organisation de l'arborescence du projet. En définissant la version de Java minimale à utiliser pour le projet (ici Java10) via gradle, cela assure une certaine compatibilité de version entre les utilisateurs.

A l'exécution du projet, gradle est exécuté et l'arborescence est créée. Gradle nous permet alors de compiler par la suite l'intégralité des fichiers de l'arborescence, et ainsi d'exécuter le projet.

Installation :

1. Cloner le répertoire :

Pour cloner le répertoire, l'utilisateur pourra utiliser la commande :

```
git clone https://gaufre.informatique.univ-paris-diderot.fr/ripasart/projet-tour-de-defens e.git
```

ou

```
git clone https://gaufre.informatique.univ-paris-diderot.fr/ripasart/projet-tour-d e-defense.git
```

2. Compiler le jeu :

Pour compiler le jeu, l'utilisateur pourra utiliser la commande:

```
./gradlew build
```

3. Exécuter le projet :

Pour exécuter le jeu et ainsi avoir accès à l'interface graphique du jeu le joueur devra utiliser la commande

```
./gradlew run
```

- Git

L'outil git nous a permis de travailler en équipe grâce à ses différentes fonctionnalités comme le partage de fichiers via le dépôt git. Cela nous a donné la possibilité de travailler sur ce projet ensemble sans avoir à échanger manuellement chaque changement auprès des membres de l'équipe.

https://gaufre.informatique.univ-paris-diderot.fr/ripasart/projet-tour-de-defense

B. Problèmes rencontrés :

Durant ce projet, notre groupe a rencontré un certain nombre de problèmes, tout autant au niveau de l'organisation que de l'implémentation de nouvelles fonctionnalités :

L'architecture du projet :

- nous avions commencé à travailler avec la vue, le contrôleur et le modèle dans des modules différents, mais cela a causé des problèmes de dépendances circulaires dans gradle, si bien que nous avons dû les regrouper dans un seul

L'environnement de jeu et SWING:

- notre carte est construite dans le modèle comme un tableau de tableaux de cases, mais les positions sur la carte étaient matérialisées par une classe Position avec des attributs x et y inversés par rapport aux indices du tableau, ce qui a posé de nombreux problèmes lors du passage de l'un à l'autre, notamment lors de l'utilisation de la caméra

Modélisation des composants du jeu :

- pour la conception des vagues, en raison du caractère cyclique des vagues, nous avons dû passer beaucoup de temps afin de trouver quel outil était le mieux adapté entre utiliser un nouveau thread ou Timer en parallèle du thread principal, ou manipuler tout simplement des variables qui stockent le temps d'apparition de chaque vague.

  La première alternative était bonne cependant elle a causé de nombreux problèmes de concurrence étant donné que les deux threads manipulaient plusieurs mêmes objets en même temps. Finalement, nous avons opté pour les variables en guise de chronomètre

- le pathfinding ne renvoyait pas de résultats dans certaines situations (si la case ciblée n'était pas atteignable par exemple), il a donc fallu le modifier un peu pour qu'il change de cible au bout d'un certain temps de recherche pour que le jeu ne se bloque pas, ou lorsqu'il passait durant

ses calculs à côté d'une case objectif (car il est possible que pour rejoindre la case ciblée, il ait besoin de faire un détour l'amenant à frôler d'autres cases objectif sans s'y rendre))

- la récupération de la position d'un ennemi se faisant en fonction du temps qu'il a passé sur le plateau, cela a posé des problèmes en cas de changement de vitesse, de mise en pause du jeu ou si l'ennemi était figé sur place par une attaque d'une tour.
  Pour la mise en pause et le cas où l'ennemi est figé, il a fallu stocker les durées de ces évènements, et les déduire du temps de vie de l'ennemi. Mais pour les changements de vitesses, cela devenait très compliqué si plusieurs changements de vitesse se succédaient. Nous avons donc décidé de recalculer le path de l'ennemi à chaque changement de vitesse. Par ailleurs, il a fallu réinitialiser le temps de vie de l'ennemi et les durées de pause/freeze à chaque calcul du path, car la position de départ du nouveau path correspondait à l'instant 0,
- des ennemis et tours "fantômes" restaient dans le modèle après leur mort et même d'un niveau à l'autre, et ce pour deux raisons : certains attributs static de certaines classes n'étaient pas réinitialisés au passage au niveau suivant, et les suppressions d'ennemis/tours se faisaient parfois lors d'itérations sur les listes desquelles on tentait de les enlever, créant ainsi des ConcurrentModificationException.

IV. BILAN DU PROJET

Au début de notre projet, nous nous sommes fixés les objectifs principaux suivants:

Le programme devait être capable de :

Description	Etat
Supporter un environnement de jeu, réalisant l'accueil de l'utilisateur, et permettant de procéder au paramétrage du jeu	Atteint
Déployer un jeu en mode infini	Nous avons plutôt opté pour le mode fini, qui est composé de plusieurs niveaux
Afficher les cartes correctement suivant un modèle prédéfini	Atteint, mais la solution adoptée consomme énormément en temps
Permettre l'évolution des personnages et des tours	Atteint
Implémenter un algorithme du "plus court chemin" pour les ennemis, notamment A*	Atteint
Implémenter toutes les règles du jeu	Atteint

Bien que nous ayons rencontré quelques problèmes au cours de ce projet, nous avons tout de même pu atteindre la majorité de nos objectifs. Il pourrait néanmoins rester quelques bugs que nous n'avons pas pu résoudre à temps.

V. ANNEXES

1. Les types d'ennemis :

Types	Vitesse*	Dégât d'attaque*	Résistance*	Portée*	Coût	Vie
Bacterium	1	10	1	5	20	100
Virus	2	15	1,25	5	30	100
Champis	0,75	5	2	3	60	100
Parasite	1	50	1,75	5	50	100
Covid	0.5	100	2	5	100	400

Vitesse* : échelle de grandeur

Dégât d'attaque* : vie enlevée par une attaque

Résistance* : coefficient atténuant les attaques

Portée* : en nombre de cases

2. Les types de tours :

Type	Coût	Portée	Puissance	Vie	Freezer
Anti-Champis	150	10	100	300	Non
Leucocyte	100	10	200	200	Non
Anticorps	150	10	50	200	Oui