

Sommaire

Table des matières

[Introduction 3](#_Toc155468045)

[En quoi consiste le projet 3](#_Toc155468046)

[Comment se joue le Tower Defense 3](#_Toc155468047)

[Outils utilisés 3](#_Toc155468048)

[Plateforme collaborative 3](#_Toc155468049)

[L’utilisation de l’IA 3](#_Toc155468050)

[Choix de l’IDE 4](#_Toc155468051)

[Contraintes 4](#_Toc155468052)

[Utilisation de Java SwingFX 4](#_Toc155468053)

[Architecture en MV 4](#_Toc155468054)

[Conception 4](#_Toc155468055)

[Utilisation du MVC et explication de son principe 4](#_Toc155468056)

[Structure du projet (+ diagramme en annexe) 5](#_Toc155468057)

[Encapsulation 5](#_Toc155468058)

[Implémentations 6](#_Toc155468059)

[Gestion de la map 6](#_Toc155468060)

[Gestion des ennemis et des projectiles 6](#_Toc155468061)

[Gestion des vagues 7](#_Toc155468062)

[Test 7](#_Toc155468063)

[Observation et Améliorations possibles 7](#_Toc155468064)

[Objectifs initiaux 7](#_Toc155468065)

[Améliorations possibles 7](#_Toc155468066)

[Conclusion 7](#_Toc155468067)

[Annexe 7](#_Toc155468068)

# Introduction

## En quoi consiste le projet

L’objectif de ce projet est de créer un Tower Defense avec comme seule base, les connaissances acquises lors du premier semestre de la deuxième année de licence informatique en Programmation Orientée Objet. L’état final de ce projet témoignera ainsi de la compréhension des principes fondamentaux (l’héritage, le MVC, …) des membres du groupes.

## Comment se joue le Tower Defense

Dans ce projet, le joueur pourra éditer sa propre map et ainsi son propre chemin\* et placer le point d’apparition des monstres (le Spawn) et la tour (la Target). Une fois la map finie, les monstres suivront le chemin dessiné dans le but d’atteindre la tour. L’objectif du joueur sera de neutraliser les monstres en achetant des tours de défense qu’il devra placer dans des endroits stratégiques.

\*Le chemin ne doit pas être coupé entre le Spawn et la Target, le jeu perd son intérêt si les monstres ne peuvent pas atteindre leur cible.

# Outils utilisés

## Plateforme collaborative

Github a été utilisé comme plateforme de codage collaborative, discord a été utilisé comme moyen de communication.

Parsec a été également utilisé occasionnellement pour des sessions de codage en simultané.

## L’utilisation de l’IA

L’aide de l’Intelligence Artificielle a été sollicitée durant ce projet, notamment le plus connu actuellement, ChatGPT qui donne la plupart du temps de bonnes suggestions mais ses réponses deviennent génériques et inutiles lorsque les questions deviennent plus spécifiques, cela s’explique par son fonctionnement qui dépend de la quantité de données analysées pour fournir les réponses. C’est d’ailleurs pour cela qu’il a tendance à mieux nous satisfaire lorsqu’il s’agit de web design et surtout le css, car le principe reste plus ou moins la même peu importe le projet.

Une image contenant texte, capture d’écran, Police

Description générée automatiquementGithub Copilot a également été utilisé, majoritairement pour sa fonctionnalité d’autocomplétions qui analyse le code / le projet et propose automatiquement une suite du code que le programmeur a entamé en fonction de ce qu’il vient de faire où du commentaire/nom de la fonction qu’il vient de mettre. Permettant ainsi de gagner du temps.

Github Copilot générant la suite du code

## Une image contenant capture d’écran, printemps, nature, typographie Description générée automatiquement Choix de l’IDE

Le projet a été codé avec l’IDE Microsoft Visual Studio Code avec le pack d’extensions Java et Git lens/graph\* pour pouvoir commit/push, pull et merge directement depuis l’IDE, le rendant ainsi plus polyvalent.

\*L’application Github Desktop a également été utilisé pour push/pull pour son interface graphique le rendant simple et intuitif.

Contraintes (demandées par le sujet)

## Utilisation de Java SwingFX

L’utilisation autre que Swing est prohibé, seule l’utilisation de Swing et d’autres librairies vues en cours est autorisée. L’utilisation des librairies vues dans les autres UE informatiques est interdite (comme Gradle)

## Architecture en MV

Le projet devra être structuré en MV (modèle et la vue), cette fragmentation rend une éventuelle évolution du projet beaucoup plus simple à faire.

# Conception

## Utilisation du MVC et explication de son principe

Afin de répondre aux contraintes structurelles, nous avons décidé d’utiliser le MVC (Modèle – Vue – Contrôleur) qui est un design pattern (patron de conception) souvent utilisé pour le web. La vue ne pouvant pas directement interagir avec la base de données, elle se charge des formulaires remplis par l’utilisateur, qui sont récupérés par le contrôleur. Le contrôleur va ainsi interroger le modèle avec les données du formulaire qui va à son tour interroger la BDD (base de données). Les données renvoyées par la BDD suivent le chemin inverse.

En JAVA, tout est sous forme d’objets qui sont alors accessibles partout (sans prendre en compte public, private, protected, …), le contrôleur n’est donc plus essentiel, cependant, le MVC n’est pas pour autant inutile en JAVA. En effet le code MVC permet de :

* Séparer le traitement des données et l’affichage
* Prévoir le futur et favoriser l’évolutivité
* Mieux diviser le code pour faciliter la compréhension et l’édition en mode projet
* D’alléger certains fichiers (car le projet a d’abord été en MV)

En outre, le MVC/MV rend le code plus lisible, plus compréhensible, plus évolutive, mais quant à son implémentation, elle n’en est pas moins compliquée. En effet, savoir comment structurer son code en amont est une difficulté majeure lorsque l’on crée un projet en partant de zéro. De plus, certaines sessions de codages, notamment aux premières semaines, ont été inutiles et ont dû être supprimés à la suite d’une restructuration complète du code.

Une session Parsec a été initiée lors d'une seconde phase de restructuration du code, visant à améliorer la structure du projet sans perturber excessivement le code préexistant et à éviter les défis rencontrés lors de la première restructuration. Cette démarche vise également à faciliter la resynchronisation des membres du groupe.

Une fois le code structuré, le code devient plus explicit et l’implémentation de nouvelles fonctionnalités devient plus triviale.

## Structure du projet (+ diagramme en annexe)

### Encapsulation

Chaque champ dans notre projet est soit privé, soit protégé (sauf pour les champs primitifs statiques finaux). Leur accès par les classes clientes est contrôlé à l'aide de getters, setters, itérateurs, etc. afin de s’assurer qu’il n’y a aucune violation de l’encapsulation.

### Principe de Responsabilité Unique

Chaque classe a une responsabilité clairement identifiable et une nomenclature explicite, par exemple :

- MessagesView : gère l'affichage des messages à l'écran (alertes, messages informatifs...).

- HealthBar : gère l'affichage des barres de santé, réutilisable pour d'autres classes (EnemiesView, BaseView...).

- StringHelper : gère l'affichage des chaînes de caractères sur l'interface utilisateur. Le respect de ce principe a rationalisé le débogage et facilité la détection des erreurs, rendant le code plus utilisable.

### Efficacité

Les HashMaps ont été utilisées pour charger des images et les réutiliser. Les floats ont été utilisés à la place des doubles lorsque la précision n'était pas nécessaire.

### Utilisation de Design Pattern

#### Factory

Le modèle de conception de la méthode d'usine a été utilisé pour la création d'ennemis, de tours et de types de tuiles (voir les figures 1 à 3). Cela a rendu le code réutilisable et facile à comprendre. L'encapsulation de la logique de création dans une classe différente facilite l'extension.

#### State

Le projet a mis en œuvre ce modèle à la fois dans la vue et dans le modèle. L'éditeur de carte utilise ce modèle en raison de la nature mutable de ses états. Les états de l'éditeur de carte implémentent une interface fonctionnelle appelée MapEditorState, puis substituent la fonction handleState. La classe de l'éditeur de carte a une référence à un objet implémentant l'interface MapEditorState, utilisé pour implémenter la logique de l'éditeur. (fig.4)

L’utilisation de state a également été utile pour les états de vagues. (fig.5)

#### Observer

Les tuiles représentent l'observable, et dès qu'un ennemi est sur une tuile, il notifie tous ses observateurs (qui sont des tours). De plus, lorsque qu'un ennemi quitte une tuile, il notifie également les observateurs de la tuile sur laquelle l'ennemi était avant de se déplacer.

# Implémentations

## Gestion de la map

## Gestion des ennemis et de leurs projectiles

### Les déplacements ennemis

L'intelligence artificielle (IA) mise en œuvre suit des règles simples. Chaque ennemi est équipé d'un tableau bidimensionnel de booléens, ayant les mêmes dimensions que la carte du jeu. À chaque fois qu'un ennemi traverse une tuile (atteignant son centre), le booléen correspondant à sa position est défini sur True. Cette information est cruciale pour la logique de l'IA.

Logique de l'IA :

- Si une tuile a déjà été traversée, l'ennemi ne change pas de direction.

- Si la tuile n'a pas été traversée et que la tuile suivante n'est pas praticable, l'ennemi ajuste sa direction vers la première tuile praticable détectée, puis marque la tuile comme traversée.

- En l'absence de tuiles praticables, la direction est simplement définie sur AUCUNE.

### Les projectiles

#### Implémentation

Une image contenant texte, Police, capture d’écran, graphisme

Description générée automatiquementL'objectif était de diriger les projectiles dans une direction spécifique. Initialement, trouver une solution à ce problème était complexe. Cependant, après avoir réalisé que l'utilisation de vecteurs était appropriée dans ce contexte, les attributs xVelocity et yVelocity ont été introduits.

#### Une image contenant texte, capture d’écran, Police Description générée automatiquement Explications mathématiques

En considérant A comme la position de l'ennemi et B comme la position initiale du projectile, le vecteur AB a été examiné :

- Obtenir la direction de AB équivaut à trouver la direction du projectile.

- Puisque AB = ||AB|| \* u, où u est le vecteur de direction, calculer AB/||AB|| fournit une direction.

- En multipliant ||AB|| par la taille unitaire de notre jeu, nous obtenons le vecteur de direction en pixels.

- En multipliant AB/||AB|| \* taille\_en\_pixel par une valeur "vitesse", la vitesse du projectile est définie en pixels par image.

Ainsi, cette approche fournit une méthode efficace pour cibler les ennemis.

## Gestion des vagues

# Test

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement À la différence du projet de préprofessionnalisation, les tests unitaires et d'intégration n'ont pas été employés dans cette phase. (A noter qu’une fois la structuration faite, repérer les erreurs devient bien plus rapide.)

Des déclarations d'impression (print) ont été utilisées comme moyen de vérification du bon fonctionnement des fonctionnalités. Lorsqu'une erreur n'était pas immédiatement identifiable, des points d'arrêt (break points) ont été utilisés pour isoler le problème.

# Observation et Améliorations possibles

## Objectifs initiaux

## Améliorations possibles

# Conclusion

# 

# Annexe

(les 7 photos à ajouter)