Mardi 16 Janvier 2024
Université de Paris-cité
L2 - informatique générale

# Projet Tower Defense

Rapport de Projet - POOIG

**Projet réalisé par:**
BEALES Archie
JIN Christophe

INFORMATIQUE
Sciences
Université Paris Cité

# Sommaire:

# I. Introduction

Le cours de Programmation Orientée Objet et Interface Graphique (POOIG) nous a proposé un projet de 1 mois et demi pour appliquer les connaissances vues en cours. Nous devions réaliser un jeu de Tower Defense indépendamment des cours et des travaux pratiques. Étant la première fois que nous réalisons un projet sans supervision d'un enseignant durant le développement, la bonne planification et l'organisation étaient essentielles pour mener à bien le projet.

# II. Objectifs

## A. Attentes du projet

Le sujet du projet nous donne déjà quelques contraintes pour débuter notre développement. Notre jeu devait contenir obligatoirement:

- Une version "graphique" dotée d'une interface codée avec java.awt et javax.swing, une fenêtre à part entière où l'utilisateur peut interagir avec sa souris.
- Un version "terminal" qui affiche toutes les informations dans le terminal à la place de la version "graphique".
- Plusieurs niveaux à créer.
- Plusieurs unités avec des caractéristiques différentes.
- Plusieurs types d'ennemis différents.
- Deux modes de jeu différents : un mode marathon et un mode vague par vague.
- Un système d'argent pour acheter les tours.

Nous avions aussi envisagé des objectifs pour le projet en plus des contraintes déjà énoncés:

- Un thème "minecraft"
- Une utilisation de ressources optimisée.
- Une interface moderne et lisible
- Du code lisible et bien organisé
- Un développement organisé et bien répartie en utilisant un Version Control System (git)

## B. Contraintes

Avec les objectifs fixés nous pouvons regarder les contraintes du projet, Le plus important c'est un temps limite de un mois et demi pour le tour réalisé, un cadre

horaire assez compressé. Nous devons aussi utiliser que des packages qui viennent d'une installation classique de java. De plus, on est contraint d' utiliser Java 11.

# III. Gestion du projet

## A. Cahier des charges

Interface Graphique :

- Fenêtre de 1280x800 pixels
- Menu principale avec fond d'écran minecraft
- Les paramètres avec options pour debug String
- Bouton exit pour quitter le jeu
- Sélection des niveaux
- Chaque niveau doit avoir ses informations de vagues écrites de façon lisible
- Compteur de vague sur l'écran du gameplay.
- Une map mise à jour continuellement.
- Un menu Popup pour poser les tours
- Une méthode pour informer le joueur s'il a assez d'argent pour acheter un tour.
- Une barre de vie pour le joueur.
- Un menu si le joueur gagne ou perd le jeu.

Version Terminal :

- Menu principale avec un ASCII art
- Plusieurs options possibles que l'on peut choisir
- Sélection des niveaux
- Plusieurs actions possibles : placer/détruire une unité, lancer la partie.

Logique de Jeu :

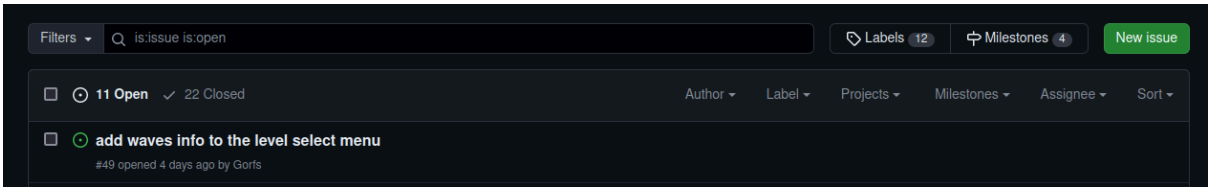
- 5 tours tous différentes
- 3 types de monstres
- Système de vagues
- Des niveaux différents
- Impact minimal sur la mémoire
- Affichage lisible

Development :

- Tout sauvegarder sur github pour ne jamais perdre d'ajout/de changement
- S'organiser avec des tickets/issues.
- Du code commenté pour aider la lisibilité.
- Des tests unitaires.

## B. Répartition des tâches

Nous avons utilisé un backlog de tickets sur github pour déterminer les tâches prioritaires, regarder lesquelles était en cours de travail, celles qui etait fini ainsi que leurs branches.




Ainsi avec la séparation des tâches dans des branches différentes on a pu travailler indépendamment et quand il était temps de réunir notre travail, on utilise des pull request pour tout rassembler.


Ce système permet une répartitions des tâches facile, indépendant, et dynamique

# IV. Point Technique

## A. structures des classes

```
Java ->
	config ->
		Cell.java
		Config.java
		Map.java
		Path.java
		Slot.java
	geometry ->
		IntCoordinates.java
		RealCoordinates.java
	gui ->
		gameUI ->
			addTowerPopUp.java
			GamePanel.java
			GamePlayPanel.java
			MapPanel.java
			ShopPanel.java
			Tile.java
			towerAddPanel.java
			UIPanel.java
		menu ->
			GameLostPanel.java
			GameWonPanel.java
			LevelPanel.java
			LevelSelectMenu.java
			MainMenu.java
			SettingsMenu.java
		Game.java
		GameWindow.java
		TermGame.java
		TermMainMenu.java
		TermPrepMenu.java
	main ->
		Main.java
		TermMain.java
	misc ->
		Print.java
		Debug.java
	model ->
		GameState.java
		Monster.java
		Player.java
		TowerAdvanced.java
		TowerBasic.java
		TowerExpert.java
		TowerMaster.java
		Towers.java
		TowerUltimate.java
```

## B. boucle interne de mise à jour

Nous avons plusieurs boucles dans plusieurs classes pour gérer le jeu. La boucle principale se situe dans la class Game.java

Voici la boucle pour le mode graphique:

```java
public static void gameLoop(){
    //!! do not remove thread, the gameloop needs to run on a different thread to the rest of the application
    new Thread(()->{
        long priorTime = System.nanoTime();
        while(running){
            if(System.nanoTime() - priorTime > 16600000*2){
                // 1/60th of a second has passed, updating gameView
                updates++;
                Debug.out("fps -> " + 1.0f/((System.nanoTime() - priorTime)*1E-9));
                priorTime = System.nanoTime();
                GameState.updateGameState(updates);
                updateGUI();
            }
        }
    }).start();
}
```

*(la boucle pour le mode terminale est presque identique sauf la fonctions update QUI() est remplacé par une autre fonction animate())*

Cette boucle permet de manipuler la vitesse du rafraîchissement du jeu facilement. Nous pouvons donc accéder au nombre de mis à jour de l'application grâce au compteur "updates". La fonction qui met à jour la logique interne du jeu se situe dans GameState.java Elle permet de gérer l'état des vagues, les déplacements de zombies, et la vie des zombies.

## C. Intégration des fichiers config

Nous avons décidé d'utiliser des fichiers configs au format txt pour décrire certaines parties "hard coder" de notre projet, notamment les niveaux et les vagues.

src > resources > map > level_0.txt

PerdSonNoeud, 2 months ago | 1 author

```
1  N,N,N,N,N,N,N,N,N,N,N,N,N,N
2  N,P,P,P,N,N,N,N,N,N,N,N,N,N
3  N,P,N,P,N,N,N,N,N,N,N,N,N,N
4  N,P,N,P,P,P,P,N,P,P,P,N,N
5  N,P,N,N,N,N,P,N,P,N,P,N,N
6  N,P,N,N,P,P,P,N,P,N,P,P,B
7  S,P,N,N,P,N,N,N,P,N,N,N,N
8  N,N,N,N,P,N,N,N,P,N,N,N,N
9  N,N,N,N,P,P,P,P,P,N,N,N,N
10 N,N,N,N,N,N,N,N,N,N,N,N,N
```

Voici ci-contre le fichier config pour le premier niveau. Nous traitons les textes dans les codes java pour importer le niveau. Utiliser les fichiers configs permet non-seulement d'avoir une meilleure lisibilité des niveaux, mais aussi de faciliter la créations des nouveaux niveaux ainsi que faciliter la tâche de la compilation du projet.

# V. Bilan du projet

Nous avons beaucoup appris de ce projet, les difficultés rencontrées, l'organisation, les nouvelles erreurs sans réponses malgré les recherches… Globalement le projet était une très bonne expérience notamment pour apprendre à mieux gérer la structure de nos fichiers.

Un des soucis qu'on a croisé à été de faire la version terminal d'abord, cela nous a empêché de faire une version fluide une fois sur la version graphique.