Java Defense

Adrien Gougeon, Julien Delaunay

November 2020

1 Introduction

L'objectif de ce projet est de concevoir et implémenter une version simplifiée d'un jeu "Tower defense". Dans ce jeu, le joueur doit protéger son château contre des vagues de monstres qui partent d'un spawn (point de départ) et doivent arriver au château en suivant une route. Le joueur possède un certain nombre de points de vie correspondant au nombre de monstres pouvant arriver au château avant la défaite.

Pour bloquer l'avancée des monstres, le joueur construit des tours de défense, dont certaines attaquent les monstres aériens et d'autres les monstres terrestres, avec une vitesse d'attaque et une puissance dépendants de la tour. Lorsque le joueur détruit un monstre il récupère de l'or lui permettant de construire de nouvelles tours.

La partie se termine lorsque le joueur n'a plus de points de vie (i.e., trop de monstres sont arrivés au château du joueur) ou lorsque le joueur à survécu à toutes les vagues de monstres.

De nombreuses versions gratuites de ce jeu existent, que ce soit dans votre navigateur internet ou sur votre appareil Android/IOS.

Figure 1: Capture d'écran d'une réalisation possible de ce jeu (en gris: les monstres terrestres)

2 Règles détaillées

2.1 Zone de jeu :

Le "plateau de jeu" est une grille dont la taille est définie par le programme. Le "spawn" représente le point par lequel les monstres arrivent. Le "chemin" est une route continue entre le spawn et le château du joueur (voir Figure 1). C'est le seul endroit de la grille de jeu où peuvent se déplacer les monstres. Afin de le rendre bien visible, il doit être dans un style graphique différent du reste de la grille de jeu. Par exemple dans la Figure 1 la grille de jeu est remplie avec des carrés verts (pour représenter une prairie), tandis que le chemin est de couleur ocre (pour représenter un chemin de terre, avec des petits buissons). Pour rendre le jeu intéressant, le chemin ne doit pas être une simple ligne droite entre le spawn et le château du joueur: il doit faire des tours et détours, afin de ralentir le parcours des monstres et d'offrir de nombreux endroits au joueur où placer ses tours. Le joueur peut placer des tours sur n'importe quelle case de cette grille sauf sur le chemin ou sur une case sur laquelle il y a déjà une tour. Le joueur démarre la partie avec 100 pièces d'or dans sa réserve. Lorsqu'il détruit un monstre il gagne de l'or.

2.2 Tours :

Le joueur peut construire deux types de tours : les tour d'archers (A) et les tours de bombes (B). Il appuie sur la touche correspondante (A ou B) pour sélectionner un type de tour. La tour est construite à l'emplacement où le joueur clique ensuite avec sa souris, à condition que le joueur possède une quantité suffisante d'or. Chaque type de tour à une certaine vitesse d'attaque, une certaine portée ainsi qu'une certaine puissance d'attaque.

La **vitesse d'attaque** correspond au temps de rechargement de la tour: un temps de rechargement de X signifie que la tour tirera une fois tous les X cycles du jeu, où un cycle correspond à un appel à la méthode **update()** (voir plus loin). La **portée** correspond à la distance à laquelle peut tirer la tour. On se base sur les coordonnées de la libraire *StdDraw*, pour rappel avec cette libraire le coin inférieur gauche de la fenêtre est en $(0,0)$ et le coin supérieur droit est en $(1,1)$.

Tour d'archers :

- La tour d'archers coûte 50 pièces d'or.
- La tour d'archers attaque :
  - à une vitesse (temps de rechargement) de 15;
  - à une portée de 0.3.
- La tour d'archers peut attaquer les cibles aériennes **et** terrestres.

Tour de bombes :

- La tour de bombes coûte 60 pièces d'or.
- La tour de bombes attaque
  - à une vitesse (temps de rechargement) de 20;
  - à une portée de 0.2.
- La tour de bombes **ne peut pas** attaquer les cibles aériennes.

Amélioration des tours :

- Le joueur peut améliorer chacun des deux types de tours : les tour d'archers (A) et les tours de bombes (B).

- Il appuie sur la touche E pour améliorer une tour.
- La tour est ensuite améliorée en cliquant dessus, à condition que :
  - le joueur possède suffisamment d'or.
  - la tour ne soit pas déjà améliorée.
- L'amélioration d'une tour modifie la vitesse d'attaque, la portée ainsi que la puissance d'attaque du projectile.

Projectiles :

Chaque tour à un type spécifique de projectiles: ce sont des bombes ou des flèches en fonction de tour. Les flèches peuvent attaquer les cibles aériennes et terrestres tandis que les bombes ne peuvent attaquer que les cibles terrestres. Les flèches font 2 points de dégâts et se déplacent de 0.04 (par appel à **update()**), tandis que les bombes font 8 points de dégâts pour une vitesse de déplacement de 0.02.

2.3 Monstres :

- Les monstres apparaissent sur une zone précise du plateau de jeu fixée et représentée par une image. C'est ce qu'on appelle le spawn.
  - Quand la partie commence, aucun monstre n'apparaît pendant 20 secondes.
  - Ensuite, des monstres apparaissent à intervalles réguliers, en étant de plus en plus nombreux au fur et à mesure que la partie avance. C'est à vous de régler la fréquence d'apparition des monstres pour que le jeu pose un défi tout en restant amusant.
- Ils avancent en suivant le chemin avec une vitesse variant selon le type de monstres.
- Lorsqu'un monstre arrive au bout du chemin et atteint le château du joueur, celui-ci perd un point de vie.

Monstres de base :

- Les monstres de base de la première vague possèdent 5 points de vie.
- Ils se déplacent à une vitesse de 0.01.
- La récompense lorsqu'un monstre de base est détruit est de 5 pièces d'or.
- Ce sont des monstres terrestres.

Monstres volants :

- Les monstres volants de la première vague possèdent 3 points de vie.
- Ils se déplacent à une vitesse de 0.02.
- La récompense lorsqu'un monstre volant est détruit est de 8 pièces d'or.
- Ce sont des monstres aériens.

⚠Les projectiles sont tirés en ligne droite (normalement) et les monstres se déplacent: il y a donc des chances que les projectiles soit ratent les monstres, soit ne tombent pas pile sur son centre. Vous devez donc implémenter un petit moteur de collisions, avec une "hitbox" pour les monstres, c'est à dire un carré ou un cercle autour du monstre qui définit la zone où le projectile touche le monstre et lui fait des dégâts.

2.4 Les vagues :

Une partie dure plusieurs vagues, la difficulté augmente en même temps que le nombre de vagues. Les monstres peuvent avoir plus de points de vie, être plus nombreux, ou apparaître dans un intervalle de temps plus court. Lors d'une vague le type des monstres reste le même (*monstre de bases* **ou** *monstres volant*). Lorsqu'une vague se termine, le joueur gagne un bonus d'or et dispose de quelques secondes avant que la vague suivante commence. Il peut en profiter pour construire de nouvelles tours ou améliorer les tours existantes qui n'ont pas été améliorées.

3 Implémentation

Nous vous fournissons un squelette de code qui gère toutes les mécaniques de base d'un jeu en Java. Les classes qui vous sont fournies vous donnent une base solide sur laquelle construire votre code. Nous les expliquons un peu ici, elles sont également commentées en Javadoc. Ce code s'appuie sur la bibliothèque **StdDraw**, que vous connaissez déjà, et qui à été choisie pour sa simplicité d'utilisation. Le squelette vous fournit une classe **World** qui implémente le "monde" du jeu avec toutes ses entités, et un **Main** qui gère la boucle principale du jeu. Dans le code fournit cette boucle est exécutée toutes les 20 millisecondes. À chaque cycle toutes les entités du jeu doivent exécuter leur méthode **update()** qui gère les déplacements et actions, et leur méthode **draw()** qui gère l'affichage.

La création d'un nouveau type de monstre demande de sous-classer la classe **Monster**. Une entité de démonstration, **BaseMonster**, vous est fournie. C'est un cercle bleu qui se déplace indéfiniment du haut en bas de l'écran. Cela permet d'utiliser la classe **Position** (qui sera utile pour placer les tours et déplacer les monstres par exemple) tout en vous montrant l'utilisation de base des méthodes **update()** et **draw()**.

Nous ne vous avons fourni que la classe BaseMonster héritant de la classe Monster. C'est à vous de concevoir le reste de la hiérarchie de classes permettant de réaliser ce jeu. Vous pouvez toutefois remarquer que les règles fournies ci-dessus peuvent vous donner une bonne intuition d'une partie des classes dont vous avez besoin ! Les valeurs qui sont données dans ces règles pour les points de dégats, les points de vie ou la vitesse des projectiles par exemple ne sont que des valeurs pour vous guider dans l'élaboration du jeu, rien ne vous empêche de les modifier pour rendre le jeu plus intéressant.

Dans le **Main** nous vous avons fourni les fonctions **keyPress**, **drawMouse** et **mouseClick** permettant d'avoir les interactions avec l'utilisateur (interaction clavier et souris.) La fonction keyPress retourne la touche utilisée sur le clavier par l'utilisateur. La fonction drawMouse récupère la position de la souris et devra effectuer une action dépendante de la touche retournée par l'utilisateur. Et pour finir, la fonction drawMouse qui permet d'afficher une image sur le curseur de la souris correspondant à la touche utilisée (par exemple, une image de tours d'archers) Pour afficher une image nous vous recommandons d'utiliser la fonction : *StdDraw.picture(normalizedX, normalizedY, image, squareWidth, squareHeight);* en remplaçant *image* par le chemin de votre image (présente dans le dossier "images" par exemple.)

4 Comment aborder ce projet ?

Dans un premier temps, nous vous recommandons de bien relire le code fourni et de vous familiariser avec ce dernier. Par exemple comprendre quel fonction est appelée lorsque j'appuie avec ma souris ou quel fonction est appelé lorsque j'appuie sur A sur mon clavier. Comment un monstre se déplace, comment le carré jaune indiquant le point de départ du monstre est crée. Une fois que le code est bien compris, plusieurs choix s'offrent à vous :

- Si vous préférez la conception objet, vous pouvez commencer par déterminer les classes qui manquent pour couvrir toutes les règles du jeu, et en implémenter de premières versions.

- Si vous préférez travailler sur l'interface graphique, vous pouvez commencer par représenter le chemin et le background du plateau de jeu.

5 Notation

Le projet est à réaliser en binôme. Vous déposerez votre travail sur Moodle début janvier (cf Moodle pour la date exacte). Vous devez déposer un dossier compressé zip avec:

- **le code source de votre jeu** (ainsi que les images éventuelles), **en tant que projet Eclipse**. Vos évaluateurs utiliseront la fonctionnalité *File - Import... - Existing project into workspace* de Eclipse pour charger votre projet. Si ça ne marche pas car vous avez rendu sous le format d'un autre IDE: vos aurez la note de 0/20.
  NB: cela ne vous empêche pas d'utiliser votre IDE préféré qui peut ne pas être Eclipse. Vous devez juste réserver un peu de temps à la fin pour préparer un projet Eclipse pour le rendu.
- un fichier **README.txt**, donnant vos noms et une description de votre jeu: ce que vous avez fait, les commandes de votre jeu, les bonus que vous avez ajoutés. Tout ce à quoi vous voulez que votre évaluateur fasse attention, écrivez le ici!

Si deux (ou plus) binômes rendent des projets très similaires : tous les participants dans la triche auront la note de 0/20 (cette note compte pour 1/4 de votre moyenne totale de PO).

Vous aurez la note de **14/20** si vous implémentez un "jeu de base" correspondant exactement aux règles indiquées ci-dessus:

- Le jeu doit être totalement fonctionnel avec les règles ci-dessus
- Il n'a pas besoin d'être joli, mais doit être jouable et simple à utiliser (testez le vous-même ou faites le tester par vos amis)
- Le code doit présenter une conception objet propre et des commentaires Javadoc pour les classes et méthodes principales
- L'implémentation doit impérativement utiliser des collections Java.

Si vous désirez avoir plus de 14/20, vous pouvez gagner des points bonus en améliorant le "jeu de base". Laissez parler votre créativité! En sachant que n'étant pas une formation artistique, nous récompenserons mieux les améliorations démontrant votre maîtrise de la programmation que votre capacité à coller de jolies images...

Quelques idées d'améliorations possibles du jeu:

- Rajouter une interface graphique (affichage de l'or, des points de vie, des positions au moment de placer les tours, des niveaux d'amélioration des tours...)
- Rajouter des cheatcodes pour aider vos évaluateurs (bonus d'or, bonus de vie, passage rapide à la vague suivante, augmentation considérable des dégâts des tours...ou des monstres)
- Rajouter d'autres types de monstres dont des boss (vous pouvez jouer sur la vie et la vitesse de déplacement a minima)
- Rajouter d'autre types de tours/projectiles (ex: missiles guidés, lasers frappant immédiatement la cible mais à courte portée et très chers,...)
- Changement plus profond: donner des points de vie aux tours, et permettre à certains types de monstres de tirer des projectiles sur les tours lors de leur passage. Eventuellement offrir une possibilité de réparer les tours endommagées moyennant un coût en or.
- Proposer plusieurs plateaux de jeu avec des chemins voire des graphismes différents (ça peut juste être un changement dans les couleurs). Plus difficile: génération aléatoire de chemins (qui doivent toujours être "corrects" et amusants) afin d'avoir un nombre infini de niveaux.
- ...

Nous espérons que vous vous amuserez bien en faisant ce projet, c'est aussi le but!