Projet de POOIG

I) Introduction

Le projet de POOIG est un exercice dans lequel vous devez faire la démonstration que vous avez passé le cap de la simple programmation telle qu'elle était faite en L1. Les éléments *Orientés Objet* couverts au cours du semestre (l'héritage, les interfaces, la généricité, les MVC, les exceptions, les énumérations...) sont autant d'éléments conceptuels qui doivent enrichir votre style et vous permettre d'écrire des programmes plus clairs, avec une décomposition qui invite à une réutilisation simple lorsqu'il vous faut résoudre des problèmes conceptuellement proches.

L'objectif de ce projet est de programmer un jeu de type Tower Defense [1] dans votre propre style. Il s'agit d'un type de jeu dans lequel le joueur doit défendre sa base contre une armée d'ennemis en construisant des tours. Les sections suivantes décrivent plus en détail les contraintes que votre implémentation devra satisfaire. Avant de commencer, lisez bien les règles ainsi que les différentes contraintes d'ordre technique, en particulier celles liées au paradigme Orienté Objet.

II) Généralités

— Le projet est à faire en binôme. Les monômes ne seront pas acceptés sauf pour des questions de parité. Vous pouvez vous associer à quelqu'un qui n'est pas dans votre groupe de TD (utilisez le forum dédié sur Moodle pour entrer en contact). Il n'y aura absolument pas de trinômes, et il est entendu que si des travaux se ressemblent trop nous prendrons les sanctions qui s'imposent.

— Il vous faut déclarer la constitution de votre binôme sur Moodle, section Projet, jusqu'au 10 novembre minuit.

— La note de soutenance pourra être individualisée. Chacun doit donc maîtriser l'ensemble du travail présenté, y compris la partie développée par son camarade.

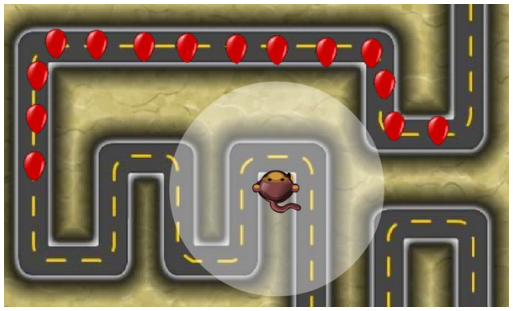
— La soutenance aura lieu durant la période des examens et votre travail sera à rendre quelques jours avant. Nous vous donnerons les dates exactes lorsque les réservations seront confirmées.

III) Présentation superficielle du projet

Dans un jeu de Tower Defense, le joueur doit défendre une base contre des ennemis. Ceux-ci suivent un chemin déterminé et le joueur doit construire suffisamment de tours

1. `https://fr.wikipedia.org/wiki/Tower_defense`

TYPE 1. Plants vs. Zombies



TYPE 2. Bloons TD 2

FIGURE 1 – Exemple des deux types de Tower Defense qui vous sont proposés.

pour les détruire avant qu'ils n'atteignent la base. Nous vous proposons de choisir entre les deux styles de Tower Defense illustrés à la figure 1.

1. Les ennemis avancent sur plusieurs lignes droites et les tours sont construites directement sur leur chemin. Les monstres attaquent les tours et doivent détruire celles-ci avant de pouvoir continuer à avancer.
2. Les ennemis avancent le long d'un chemin sinueux qui peut se croiser lui-même et éventuellement avoir des embranchements. Les tours sont construites hors du chemin et sont indestructibles ; en contre-partie, le trajet ennemi ne peut jamais être obstrué.

Dans chaque cas, il faudra :

1. un ou plusieurs point de départ d'où les ennemis apparaissent ;
2. un point d'arrivé (votre base) vers lequel les ennemis cheminent ;
3. plusieurs types d'ennemis, chacun ayant ses propres caractéristiques (barres de vie, vitesse, etc.) ;
4. plusieurs types de tours (attaque rapide ou lente, portée longue ou courte, etc.) ;

Vous devrez aussi créer une monnaie qui permettra au joueur d'acheter les tours. Chaque tour aura un prix et l'argent sera récolté ou augmentera au fil du temps à un rythme déterminé. Nous vous invitons à faire preuve de créativité : n'hésitez à inventer des type de tours ou d'ennemis avec des caractéristiques différentes de ce qui est suggéré !

A titre indicatif, les caractéristiques qui suivent peuvent (et doivent) être personnalisés pour differencier les travaux par binômes :

— *Diversité des Tours* : Les jeux de Tower Defense proposent souvent une variété de tours ayant des caractéristiques et des capacités différentes. Certaines tours peuvent ralentir les ennemis, d'autres infligent des dégâts à distance, et d'autres encore sont efficaces contre des types d'ennemis spécifiques.

— *Ressources* : Les joueurs peuvent gagner des ressources (souvent sous forme d'argent virtuel) en éliminant des ennemis. Ces ressources sont utilisées pour construire de nouvelles tours, les améliorer ou acheter des capacités spéciales.

— *Progression* : Au fur et à mesure que les joueurs réussissent à repousser les vagues d'ennemis, le jeu devient de plus en plus difficile. Les ennemis deviennent plus nombreux, plus rapides et plus résistants, ce qui oblige les joueurs à adapter leur stratégie.
— *Xp et héros* : Certains jeux de TD permettent aux joueurs de personnaliser leur expérience en choisissant des héros ou en débloquant des compétences spéciales pour les aider dans la défense.

IV) Cahier des charges

On rappelle que la bonne utilisation des notions de Programmation Orientée Objet constituera une part importante de l'évaluation. Même s'il était fonctionnel, un code qui ne comporterait qu'une seule classe serait un cas extrême fortement pénalisé. Essayez d'illustrer au maximum les aspects vus en cours.

Charges minimales

Au minimum, votre code devra réaliser :

1. Un environnement de jeu, réalisant l'accueil de l'utilisateur, et permettant de procéder au *paramétrage du jeu* :
   — choisir le niveau de difficulté (facile, moyen ou difficile) ;
   — choisir le tableau de jeu (chemins ou décors différents) ;
   — choisir le mode de jeu (normal ou marathon).
2. L'implémentation des règles générales décrites ci-dessus.
3. Un mode de jeu « texte » (pouvant se jouer entièrement dans le terminal). Il n'a pas vocation a être très ergonomique, nous vous le demandons pour être certains que votre travail ne soit pas bloqué par le traitement de problèmes liés à l'interface graphique.
4. Un mode de jeu « graphique » utilisant `awt` et `swing` (et uniquement ces librairies, pas de `gradle` ou `maven`, pas de `javafx`).

Fonctionnalités avancées

Ces fonctionnalités avancées viennent compléter le cahier des charges minimal. Elles sont *indépendantes* les unes des autres et elles ne sont *pas obligatoires*. Néanmoins, elles seront valorisées lors de l'évaluation. Même si vous ne les implémentez pas, il est utile de réfléchir aux questions qu'elles posent, et d'anticiper les modifications qu'elles provoqueraient dans votre code. En général, suivre des ambitions généralistes aide à produire un code « bien conçu », c'est pourquoi nous ouvrons ici votre horizon.

Sauvegardes. Permettre de sauvegarder des parties, et de les charger lors du paramétrage du jeu. (En java les sauvegardes se font assez simplement en faisant en sorte que

vos objets implémentent l'interface `Serialisable`. Référez vous à la documentation de cette interface.)

Améliorations. Permettre au joueur d'améliorer ses tours, soit avec un système d'expérience (une tour évolue en tuant des ennemis) soit en utilisant la monnaie du jeu.

Débloquage de contenu. Créer du contenu supplémentaire (tableaux, modes de jeu, niveaux de difficultés, etc.) que le joueur doit débloquer en accomplissant des tâches particulières.

V) Conseils préalables à l'implémentation

a) Sauvegarde

Sauvegardez régulièrement votre travail. Chaque fois que vous envisagez une modification importante, conservez bien la version antérieure. Ces précautions permettent d'éviter des catastrophes ! Vous pouvez utiliser chez vous `git` et `eclipse`, mais veillez à vous conformer strictement aux contraintes de rendu : **pas de `gradle`, `maven` ou `javafx`**.

b) Décomposition du code

Pour pouvoir maîtriser la complexité de votre travail, il vous faut absolument le découper en objets et méthodes qui joueront des rôles bien délimités. Vous avez une grande liberté dans la façon dont vous ferez votre développement, mais veillez à distinguer les choses conceptuellement. Vous remarquerez qu'alors chaque petite variation ou évolution pourra se faire facilement et localement. De plus, cela est essentiel pour pouvoir vous répartir le travail.

Il est toujours préférable d'avoir plusieurs petites méthodes à écrire plutôt qu'une seule grande méthode qui ferait un travail compliqué à déchiffrer. Si vous avez quelque part dans votre code un bloc qui fait plus d'une vingtaine de lignes, songez à introduire une phase intermédiaire pour en réduire la taille.

c) Développement progressif

« Premature optimization is the root of all evil » [2]. Commencez par élaborer une version jouable *minimale* (mais fonctionnelle) du jeu avant de vous lancer dans les fonctionnalités plus compliquées comme les aspects graphiques. Vous pouvez décrire votre organisation dans le rapport.

Veillez à tester rigoureusement votre code après chaque ajout conséquent, et *avant* de passer à l'étape suivante. Écrivez pour cela toutes les fonctions d'affichage nécessaires. Cela

2. Donald Knuth, *The Art of Computer Programming.*

peut sembler fastidieux au premier abord, mais vous y gagnerez sur le long terme en passant moins de temps à corriger les erreurs. Vous pouvez documenter vos tests dans le rapport.

d) Vue et Modèle

Il est important de bien séparer la vue du modèle. On rappelle que la *vue* désigne l'ensemble des aspects liés au rendu graphique, et le *modèle* regroupe l'ensemble des concepts qui sont sous-jacents, vraiment propres au jeu, et qui sont largement indépendants de la vue.

Ainsi, si vous souhaitez faire évoluer votre programme pour en changer la présentation, par exemple pour l'adapter à l'écran d'un téléphone ou d'une tablette, alors l'essentiel du jeu sera tout de même préservé. Les changements à faire relèvent tous de la *vue*. De la même façon, si on souhaite changer un peu les règles, ajouter des variantes, celles ci concernent essentiellement le *modèle*.

Dans votre cas, vous devrez dans un premier temps présenter les choses en mode « texte », puisque les aspects graphiques seront abordés progressivement en cours de semestre. Lorsque vous serez plus avancé dans votre réflexion et que vous aurez davantage de connaissances sur les interfaces graphiques, il sera temps de redéfinir les vues. Vous pouvez prévoir cela en définissant assez tôt une interface ou une classes abstraites pour les vues attendues. Il suffira ensuite de l'instance par des sous-classes plus ou moins évoluées (textuelle ou graphique). L'association entre les modèles et les vues se fera en introduisant un champs typé `VueGenerale` dans le modèle de votre jeu, et réciproquement si besoin. Pour rendre compte d'une modification, graphique ou textuelle, le jeu s'adressera à sa vue. La vue se chargera aussi de transmettre les actions choisies par le joueur au modèle.

VI) Aspects pratiques de l'évaluation

Rendu

Les travaux sont à rendre sur `Moodle` sous la forme d'une archive nommée *`nom1-nom2`* selon la constitution de votre binôme, et qui s'extraira dans un répertoire *`nom1-nom2`*. Elle devra contenir :

- les sources et ressources de votre programme (fichiers `.java`, images, etc.). Ne polluez pas votre dépôt avec des fichiers `.class` inutiles.
- un fichier nommé `README` qui indique comment on se sert de votre programme (compilation, exécution, utilisation). Faites en sorte que son utilisation soit la plus simple possible. Ne pensez pas que nous utilisons le même environnement de développement que celui que vous avez choisi.
- **Important :** vous n'utiliserez pas javafx mais uniquement swing et les choses vues dans ce cours-ci. Pour simplifier la portabilité vous n'utiliserez pas non plus ni maven

ni gradle, même si la majorité d'entre vous les ont utilisés en conduite de projet. Ici, tout doit être compilable sous java 11 et « à la main ».

— Le correcteur doit povoir tester votre travail [3] en seulement deux étapes :

1. décompresser votre dépôt dans une console unix ;
2. lire votre fichier README et y trouver la ligne de commande qui lance le programme.

— un rapport au format *PDF* d'au moins cinq pages **rédigées** expliquant les parties du cahier des charges qui ont été traitées, les problèmes connus, et les pistes d'extensions que vous n'auriez pas encore implémentées. Il devra contenir impérativement une représentation graphique du modèle des classes (le plus simple est qu'elle soit manuscrite, puis scannée). Le rapport n'est en aucun cas une impression de votre code !

— toute chose utile pour rendre la soutenance fluide.

Soutenance

La soutenance devra pouvoir se dérouler sur une machine du script à partir des sources que vous avez déposées. Si tout se passe bien elle se déroulera devant un ou deux enseignants dans un mélange de questions et de tests. Il pourra vous être demandé de modifier votre code pour répondre à une question spécifique.

Idéalement, vous pouvez proposer des "tutoriaux" (accessibles lors du paramétrage du jeu), ou de parties sauvegardées (si vous avez implémenté cette fonctionnalité) pour mettre le joueur directement dans des situations précises, afin de gagner du temps lors de la soutenance. Cela n'est pas obligatoire.

3. Testez vous même votre rendu sur une autre machine ou pour le moins dans un dossier séparé de celui où vous avez travaillé. Si nous n'arrivons pas à exécuter votre code vous serez évidement fortement pénalisé.