**Rapport du projet de POO**

Sommaire :

1. Présentation du jeu vidéo
2. Hiérarchisation du code
3. Problèmes rencontrés
4. Fonctionnalités avancées

1-Présentation du jeu vidéo

Notre jeu vidéo se nomme « The Legend of Zelda, Tower Defense” et est donc inspiré de l’univers de la saga The Legend of Zelda. Nous avons décider de réaliser un Tower Defense sous la forme d’un chemin sur lequel passe les ennemis, à la manière d’un Bloons TD.

Pour ce faire, nous avons créé plusieurs types d’ennemis ainsi que plusieurs type tours qui possèdent chacun leur attribut spécifique. Chaque ennemi ainsi que chaque tour fonctionnent indépendamment les uns des autres et ne nécessitent aucunes interactions avec le joueur pendant une partie pour que celle-ci puisse se dérouler. Le seul moment où le joueur peut changer le cours du jeu est lorsqu’une wave, c’est-à-dire une vague d’ennemis, se termine. A ce moment-ci, le joueur peut décider de rajouter des tours sur son plateau, les améliorer ou même les supprimer s’il estime que celles-ci ne leur servent à rien et voudrait se faire rembourser la moitié de leur prix. Le plateau est de forme carrée et possède deux types de cases, le chemin et le terrain. Le chemin suit un tracé défini en fonction du niveau choisi et c’est sur ces cases que les ennemis vont avancer jusqu’à arriver devant la princesse Zelda. S’ils arrivent devant elle, ils l’attaquent et s’en vont. La princesse possède un nombre de point de vie limité et si celle-ci subit une attaque qui la fait descendre à 0 points de vie, alors c’est un Game Over et la partie s’arrête. Cependant, cette princesse est défendue par de vaillants héros positionnés par le joueur sur le terrain. Ces héros (ou tours) vont pouvoir attaquer les ennemis grâce à des attaques à distances alors que les ennemis ne peuvent pas les attaquer en retour, ils ne font que de suivre le chemin. Si les héros réussissent à tuer tous les ennemis avant que ceux-ci n’emmènent les points de vie de la princesse à 0, la partie est remporté par le joueur.

Le jeu peut être joué de deux manières par l’utilisateur : soit en mode graphique, soit en mode terminal. Le mode graphique permet d’avoir un affichage dynamique du jeu avec par exemple, des frames différents pendant que les ennemis avancent, des cases de différents aspects en fonction du niveau sélectionné, un panneau latéral permettant de pouvoir choisir les tours que nous voulons poser sur le terrain ou même d’avoir de la musique pendant que notre partie se déroule.

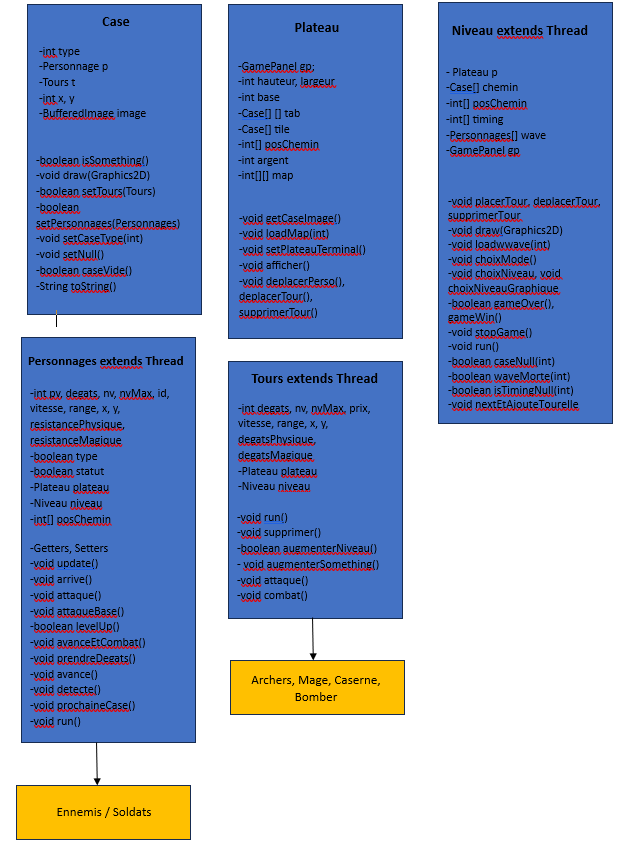
Le mode terminal propose un affichage du tableau chaque seconde dans le terminal à l’aide de System.out.print. Les actions du joueur sont possibles à l’aide de Scanners qui vont lire les données rentrées par l’utilisateur dans le terminal à l’aide de son clavier.

Ces deux modes de jeux sont différents et les classes, les constructeurs ainsi que les fonctions utilisées pour faire tourner chacun de ces modes ne sont pas toujours les mêmes.

Au début de chaque partie le joueur commence avec un certain montant de rubis qu’il devra dépenser en positionnant des tours sur le terrain. Le solde en rubis du joueur va augmenter au fur et à mesure que les ennemis sont tués ce qui va permettre au joueur de pouvoir établir une stratégie soit en rajoutant de nouvelles tours, ou en améliorant celles déjà sur le plateau, soit en économisant un maximum son argent ce qui va lui permettre de gagner un bonus de 10% sur l’argent qu’il à déjà dans son solde à chaque fin de wave.

2-Hiérarchisation du code

Le code de notre jeu est réparti dans plusieurs dossiers et plusieurs classes qui sont liés d’une manière ou d’une autre car ils étendent, implémentent ou possèdent comme attribut d’autres classes. Voici une modélisation graphique de la hiérarchisation de notre code.



3-Prolèmes rencontrés

Nous avons naturellement rencontré certains problèmes durant la création du jeu, certains que nous avons réussis à résoudre et d’autres qui se révèlent plus compliqués que prévus. Nous avons vu très grand avant de commencer écrire le code du jeu car tout nous semblait logique et faisable, nous avons donc eu pleins d’idées originales pour essayer d’améliorer le gameplay de notre jeu. Nous sommes étudiants en double licence de biologie/informatique, ce projet était donc notre premier jeu vidéo car nous n’avons pas eu à faire le projet Pac-Man. Nous avions donc pensé trop grand et trop compliqué ce qui nous à beaucoup handicapé car nous passions des journées entières sur des éléments que nous avons finalement abandonnés sur le projet final.

L’une de ces idées que nous avons abandonné juste avant la date limite du projet est la caserne. La caserne est une tour qui ne peut uniquement être posée à côté d’un chemin et qui va faire apparaître des soldats à intervalles réguliers sur le chemin. Ces soldats vont marcher dans la direction opposée aux ennemis et lorsqu’un soldat et un ennemi sont amenés à se rencontrer, ceux-ci vont combattre jusqu’à la mort d’un des deux. Nous trouvions cette idée originale et nous avons passé un nombre énorme d’heure sur cette caserne et surtout sur ces soldats. Le problème est que l’arrivée de ces soldats sur le chemin nécessite l’aménagement de tout le code car les ennemis doivent maintenant pouvoir détecter des soldats devant eux, s’arrêter, les attaquer etc. Si vous regardez le code dans la classe Personnages et la classe Niveau, vous pouvez voir que de nombreuses fonctions sont présentes uniquement car nous avions à traiter des interactions entre les soldats et les ennemis. Et nous avons réussi, enfin presque. Les soldats et les ennemis se rencontraient bien, ils combattaient et rendaient les parties encore plus intéressantes. Malheureusement, quelques erreurs d’exécution apparaissaient quelque fois, pas toujours, ce qui était encore plus compliqué à corriger car nous avions encore plus de mal à identifier d’où venaient ces erreurs, même avec du débogage et des tests, les casernes étaient souvent la source de la plupart de nos problèmes. Nous avons arrêté de nous battre contre ces casernes ainsi que contre notre fierté, et nous avons décidé, à contrecœur, d’arrêter de travailler sur ces casernes et d’empêcher à l’utilisateur de pouvoir poser des casernes sur le terrain. Nous avons malheureusement pris cette décision trop tard car nous avions passé tellement de temps dessus que nous voulions vraiment tout faire pour les garder et les corriger complètement. Ceci nous à fait perdre beaucoup trop de temps ce qui explique en partie pourquoi la partie graphique n’est pas complètement fonctionnelle. Le code des casernes ainsi que de tous les autres essais et idées sont encore disponible dans les dossiers, la plupart des choses non abouties sont commentées, ce sont des idées que nous n’avons pas eu le temps de réalisé ou abandonné en cours de réalisation.

Concernant les threads, c’était la première fois que nous utilisions des threads dans du code et nous avons trouvé cela génial. Pouvoir lancer plusieurs fonctions et avoirs plusieurs objets qui tournent en même temps est quelque chose auquel nous n’avions pas pensé avant de commencer à travailler sur le projet. J’aborde le thème des threads dans cette partie car, malgré le fait que nous avons apprécié travailler avec cet outil, nous débutions et nous nous sommes un peu perdus entre toutes les manières de gérer ces threads. Devions nous implémenter Runnable, étendre Thread, les deux ? Comment faire pour démarrer, mettre en pause, arrêter les threads ? Comment faire des fonctions runs qui fonctionnent et qui s’arrêtent ? Nous avons dû faire face à toutes ces questions durant la création de notre jeu c’est pourquoi nous exprimons ceci comme une difficulté rencontrée même si nous sommes maintenant contents d’avoir appris à utiliser les threads.

4-Fonctionnalités avancées

Nous avons réalisé plusieurs fonctionnalités supplémentaires de celles demandés afin de rendre le jeu plus agréable ou plus complet.

Tout d’abord, nous avons souhaiter rendre les tours plus modulables en permettant à l’utilisateur de les déplacer, de les améliorer ou encore de se les faire rembourser. En effet, nous trouvions dommage que les tours, une fois qu’elles sont placées, soient impossible à modifier. Ces pourquoi, au moment d’acheter des nouvelles tours nous avons permis à l’utilisateur de réaliser une ou des actions énoncées précédemment. L’économie est une notion importante dans un Tower Defense c’est pourquoi nous permettons aux joueurs de supprimer leurs tours contre la moitié de leur valeur actuelle. De plus, un joueur qui saura placer juste suffisamment de tours pour survivre sans dépenser tout son argent, se verra reversé 10% de son solde bonus avant le début de la prochaine wave d’ennemis.

Nous vous conseillons également de jouer en mode marathon, car nous avons beaucoup travaillé sur l’ajustement des tours et des ennemis afin de créer un jeu équilibré. Les ennemis peuvent être de plus en plus nombreux et de plus en plus puissant plus la partie dure longtemps. Cependant, ce mode marathon est aléatoire et infini c’est pourquoi 2 parties ne seront jamais similaires et jamais exactement de la même difficulté.

Pour finir, nous avons implémenté des musiques dans la version graphique du jeu, permettant au joueur d’avoir une meilleure expérience lors de ses parties.