Le Mans Université

CSX-325

Tower Defense

Remis à :
Py Dominique
Professeur d'algorithmique
de programmation

Remis par :
Dorso-Martin Antoine
Durand Pierre
Edel Thomas
Gonçalves Quentin

Sommaire

1 Introduction **2**

2 Répartition des tâches **2**

3 Analyse et conception **3**
3.1 Règles du jeu 3
3.2 Choix de conception 4

4 Codage **4**
4.1 Structure de données 4
4.2 SDL 2 5
4.3 Problèmes rencontrés 6

5 Résultats **6**
5.1 Réussite des objectifs 6
5.2 Fonctionnalités du programme final 6
5.3 Bugs éventuels 8

6 Conclusion **8**
6.1 Améliorations 8

1 Introduction

Un Tower Defense est un jeu où l'objectif est de défendre une zone contre des vagues d'ennemis successives. Le but étant d'empêcher les ennemis de détruire la zone à protéger. Pour cela nous pouvons construire progressivement des tours défensives qui par la suite pourront être améliorer. Chaque ennemi tué nous permettra de remporter de l'argent que nous utiliserons pour poser ou améliorer une tour.

Une tour est caractérisée par son coût, le nombre de dégâts qu'elle inflige, sa vitesse d'attaque, sa portée d'attaque et un type d'attaque.

Un ennemi, lui, sera caractérisé par ses points de vie, sa vitesse de déplacement, le nombre de dégâts qu'il inflige et sa vitesse d'attaque.

Il existe deux types de Tower Defense un avec un itinéraire définit, l'autre sans. Dans ce projet nous avons décider de réaliser un Tower Defense où les monstres se déplaceront en suivant un chemin prédéfinit.

2 Répartition des tâches

La répartition des tâches s'est d'abord appuyée sur notre analyse des règles. Nous trouver trois grandes parties que sont :

Thomas	Programmation des monstres
Pierre	Programmation des tours
Antoine	Interface
Quentin	Interface

Cependant, ce planning a été partiellement modifié en raison de nouveaux besoins. Par exemple, le déplacement des monstres n'avait pas été abordé ce qui a amené à l'apparition de nouvelles fonctions et structures. De plus, après réflexion nous avons décidé de répartir la SDL sur deux personnes.

La répartition des tâche s'est donc fait de la sorte :

Thomas	Programmation des monstres et du chemin
Pierre	Programmation des tours et implémentation du déplacement en SDL2
Antoine	interface de jeu et menu en SDL2
Quentin	Programmation du déplacement et du chemin

3 Analyse et conception

3.1 Règles du jeu

Le but du jeu est de survivre à des vagues consécutives d'ennemis appelés monstres. Pour cela, le joueur possède 2 armes à sa dispositions, que sont les tours.

L'une d'elle, la tour "AOE", attaque dans en zone adjacente et inflige peu de dégâts, tandis que l'autre est mono-cible, elle est dit "MONO", et inflige plus de dégâts à un monstre à la fois.

Le joueur doit alors adapté sa stratégie pour empêcher les monstres d'atteindre le monument, qui est le troisième type de tours. Il est identique aux tours AOE, inflique très peu de dégats aux monstres, mais a la particularité d'avoir un nombre de points de vie élevé.

De leur côté, les monstres se déplacent sur un chemin prédéfinit. Le monument se trouve a la fin de celui-ci. Les monstres ne peuvent pas attaquer les tours du joueurs mais seulement le monument. Chaque monstres abattus rapportent des golds.

Les monstres sont divisés en 4 catégories :
- Les normaux : monstres aux caractéristiques basiques, c'est-à-dire qu'elle servent de références à celles des autres monstres.
- Les tanks : monstres aux nombres de points de vie améliorés, ses dégâts et sa vitesse de déplacement sont réduits.
- Les dégâts : monstres aux dégâts amélioré, à la vitesse de déplacement amélioré mais avec un nombre de points de vie réduit.
- Les Boss : monstres aux nombres de points de vie et aux dégâts améliorés mais avec une vitesse de déplacement réduite.

Chaque tour possède un coût en golds pour pouvoir être placée sur la carte. Elles peuvent évoluer pour être plus puissantes.

Chaque évolutions coûte des golds et le prix augmente à chaque fois qu'une tour est améliorée. Le joueur peut placer ses tours n'importe où sur la carte sauf sur le chemin emprunté par les monstres.

Le joueur gagne une fois que son monument a survécut à 9 vagues de monstres. Si son monument ne survit pas jusque là, alors le joueur perd la partie.

3.2 Choix de conception

Les objectifs de notre projet étaient d'avoir des mécaniques de jeu fonctionnelles dans le terminal, mais nous voulions avoir une interface graphique car le Tower Defense se prête difficilement au terminal.

Il devait y avoir plusieurs types de monstres et de tours. Une augmentation de la difficulté croissante. Une gestion de sauvegarde automatique.

Le jeu devait être capable de gérer des vagues de monstres, les attaques des tours et les déplacements des monstres de manières automatique.

De plus un système de monnaie, récupérable sur les monstres, devait permettre d'acheter des tours pour pouvoir les poser ou les améliorer.

Les monstres se déplacent sur un chemin prédéfinit. Le monument se trouve a la fin de celui-ci. Les monstres n'attaquent pas les tours du joueurs, nous avons décidé qu'ils ne pouvaient attaquer que le monument. Chaque monstres abattus rapportent des golds. Ils sont divisés en 4 catégories :

4 Codage

4.1 Structure de données

L'analyse du projet a amené à réfléchir sur les différentes structures de données nécessaire au développement du Tower Defense.

Tout d'abord, nous avons décidé d'utiliser une matrice pour créer la carte du jeu mais aussi pour pouvoir placer les tours. En effet, les tours dépendent d'une position en x et en y. De plus, cette matrice est utilisée pour l'affiche par SDL.

Le chemin, de son côté, n'utilise pas de matrice. Nous avons préférer le faire sous forme d'une liste où chaque élément correspond à des coordonnées dans la matrice. De plus, la liste est faite avec deux structure, une qui contient les variables utilisée pour la position (x et y) et une autre constituée de pointeur sur la structure précédente facilitant le parcours de la liste par les fonction.

Les monstres sont gérés dans un tableau, ce qui nous permet alors de pouvoir poser plusieurs monstres sur une même case de la matrice. Chaque monstre possède un pointeur sur le chemin ce qui définit alors sa position en x et en y.

Ensuite, des structures ont été utilisées pour les objets, la création des tours, des monstres et du monument. Ces objets possédant des caractéristiques, il est donc plus

pratique de les organiser sous forme de structure pour avoir un plus grande facilité d'accès aux variables.

De plus, bien que le monument soit défini comme une tour, ils ne possèdent pas tout à fait les mêmes attributs, le monument à une variable "vie" en plus. Il a donc fallu créer une structure propre celui-ci.

4.2 SDL 2

Le choix du jeu nous imposait d'avoir une interface graphique. En effet le Tower Defense est un jeu principalement visuel. La SDL nous étant imposé nous avons commencé à travailler sur la SDL1. Cela nous a permis de prendre en main les mécaniques de la bibliothèque .

La SDL est une librairie de C permettant de faire de l'affichage graphique. De nouvelles fonctions étaient donc à apprendre. Pour pouvoir affiché les éléments du jeu il nous fallait des images et savoir les gérer. Au départ nous devions avoir un fond prédéfinit et n'afficher que les monstres et les tours par dessus, mais nous avons trouvé un pack d'image nous permettant d'afficher toutes la carte a chaque changement. Nous pouvions ainsi avoir plusieurs possibilités de cartes différentes une fois le programme fonctionnel.

Dans notre cas, la SDL est utilisée de manière à mettre à jour l'écran. Pour expliqué son fonctionnement, elle se compose d'une *window*, qui est la fenêtre. Nous avons ensuite plusieurs objets graphiques, comme le *renderer*, qui s'apparente à un calque. C'est sur celui-ci que nous allons appliqué toutes nos images. À la fin de nos opérations (placement de tours ou déplacement des monstres), il est ensuite recopié sur la *window*.

Chaque image est chargée dans SDL via une *surface*. C'est l'élément de plus bas niveau. Il sert a utiliser toutes les autres fonctions de la SDL. Une fois l'image chargée, il faut l'appliquer à une *texture* pour qu'ensuite elle soit copiée dans le *renderer*. Nous pouvons ainsi choisir la position, la taille et la transparence. La recopie écrase alors les images qui été alors présentent.

Pratiquement tout se passe en arrière plan. A chaque tour de jeu, l'écran est mis à jour, on affiche la base de ce dernier (les menus, le sol, le chemin, le monument et les tours en places) et ensuite on affiche les monstres (seuls éléments mobiles du jeu) et le nombre de golds qui varie en permanence.

4.3 Problèmes rencontrés

Pendant toute la programmation, nous avons pu rencontrer quelques problèmes. Le plus récurent était au niveau des "include". Nous avions décidé au départ de créer un fichier "all.h" qui contiendrait toute les "include" des fichiers nécessaires à l'exécution du programme et qui serait le seul fichier à être inclue dans autres. Cependant, à certain moment ce fichier générait un bug à la compilation : il ne trouvait pas certaines fonctions ou variable.

Étant donné que ce légé problème nous faisait perdre beaucoup de temps, car il fallait vérifier tout les fichiers, nous avons décidé de supprimer les "include" du fichiers all.h pour les mettre dans tous les autres fichier "include_fichiers.h".

5 Résultats

5.1 Réussite des objectifs

Au final nous avons un jeu fonctionnel en interface graphique. Plusieurs types de monstres ont été implémentés ainsi que 2 types de tours. La difficulté est croissante et elle varie en fonction des vagues : plus le joueur avance dans les vagues plus le jeu devient difficile.

Comme voulu, le joueur n'a pas besoin d'intervenir pour que ses tours attaquent ou que les monstres se déplacent, tout est géré automatiquement. Le joueur n'a plus qu'a poser des tours et les améliorer pour défendre au mieux son monument.

Le système de monnaie (golds) est lui aussi fonctionnel, il s'actualise automatiquement lors de l'achat d'une tour ou du gain sur un monstre.

Enfin, le placement des tours sur la carte et leurs évolutions sont sauvegardées au fur et à mesure du déroulement du jeu. Le joueur peut quitter à tout moment une partie et la continuer plus tard avec le même plateau de jeu (c'est-à-dire les tours aux mêmes endroits et aux mêmes niveaux que la partie précédente).

5.2 Fonctionnalités du programme final

— Le joueur a la possibilité de commencer une nouvelle partie ou de charger la partie précédente. C'est-à-dire que les tours qu'il avait posé au jeu précédent

seront présentent aux mêmes positions et avec le même niveau.

— Le joueur peut poser 2 types de tours afin d'éliminer les monstres et protéger le monument. Les tours AOE et MONO

— En cliquant sur les icônes des tours à gauche de l'écran, il peut poser des tours sur le plateau de jeu en glissant-déposant celles-ci.

— Le joueur peut aussi faire évoluer ses tours jusqu'au niveau 5. Lors de l'évolution, elles gagnent en dégâts.

— L'évolution d'une tour se fait toujours en glissant-déposant l'icône d'une tour depuis le menu gauche vers une tour sur le plateau. Le type de la tour sur le plateau doit correspondre avec celui de la tour sélectionnée.

— Le joueur ne peut pas poser de tour sur une case déjà occupée par une autre ni sur le chemin.

— En haut à gauche est indiqué le numéro de la vague actuel.

— En dessous du numéro de vague, est indiqué l'état d'avancement de la vague. À savoir, le nombre de monstres tués et le nombre de monstres de la vague.

— Les vague vont de 1 à 9.

— Le nombre de monstre par vague augmente linéairement de 20 à 100 à la vague 9.

— Le joueur doit faire face à 4 types de monstres :
  — Les monstres normaux, leurs caractéristiques (vie, vitesse, dégâts) sont la référence des 3 autres.
  — Les monstres dit "tank" : ils ont plus de vie, se déplacent à vitesse normale et font peu de dégâts.
  — Les monstres dit de "dégâts" : ils ont peu de vie, se déplacent vite et font plus de dégâts.
  — Les monstres dit "boss" : ils beaucoup plus de vie, se déplacent lentement et font plus de dégâts.

— En appuyant sur la touche "echap" ou en cliquant sur la croix, le joueur revient sur le menu principal (si il était sur le jeu) ou quitte le jeu si il était

déjà sur le menu.

— Le jeu est fini lorsque le monument n'a plus de points de vie ou que le joueur a complété la vague 9.

— À la fin de la partie, le joueur est redirigé vers le menu principal.

— Plus le joueur avance dans le jeu, plus celui-ci devient difficile : les monstres deviennent de plus en plus rapide, apparaissent plus vite et en plus grand nombre.

5.3 Bugs éventuels

Les bugs les plus critiques ont été corrigés lors des phases de tests. Néanmoins le joueur a la possibilité de tricher en créant une nouvelle partie et la relancent. En effet les golds et le niveau de la vague ne sont pas sauvegardés, seuls les tours le sont. Le jeu recommence toujours avec 1000 golds et à la vague 1, peu importe que ce soit une nouvelle partie ou un chargement.

6 Conclusion

Ce projet s'est révélé très enrichissant car il nous a permis d'appliquer nos connaissances en programmation sur un sujet de notre choix. De plus, nous avons pu nous rendre compte comment réussir à travailler ensemble et aussi de comprendre les difficultés que le travaille d'équipe peut soulever.

6.1 Améliorations

L'amélioration la plus importante serait de mieux gérer la sauvegarde, permettre au joueur de recommencer une partie à la vague ou il en était et avec le même nombre de golds.

L'amélioration des graphismes, implémenter des animations (attaque des tours, pas exemple).

Équilibrage du jeu et modifier la gestion de la difficulté croissante : par exemple les monstres pourraient devenir de plus en plus fort au fur et mesure des vagues, actuellement ils sont seulement plus rapide et apparaissent plus vite.

Ajouter un écran de fin lorsque le joueur gagne ou non.

Références