UNIVERSITE PARIS-SUD

Master Informatique 1ere Année

Année 2016-2017

Rapport de Programmation Objet Avancée

par

Tong XUE

Dimitrios CHRISTARAS PAPAGEORGIOU

Labyrinthe Hanté

Enseignants : *Patrick Amar, Université Paris-Sud*

Sommaire

I. Introduction

Lors de ce projet on a été initié à la programmation c++ avancée, avec comme but le développement d'un jeu FPS, en ayant déjà l'interface graphique. Cela nous permet de bien distinguer les taches entre interface et comportement. On a réussi à développer un jeu qui marche assez bien en ajoutant une fonctionnalité de téléportation en plus pour rendre le jeu plus intéressant. Pour tester notre jeu, il faut utiliser le fichier lab1 fournit dans l'archive. Sinon pour tester avec d'autres fichiers il faut changer manuellement les variables LAB_WIDTH et LAB_HEIGHT du fichier Labyrinthe.h. Dans ce rapport on fournit une explication générale de notre programme. Pour des détails des algorithmes, voir les commentaires dans le code.

II. Labyrinthe

Premièrement, le system lit un fichier txt et crée le labyrinthe. Ce fichier est créé avec les caractères suivants :

- : murs horizontaux
| : murs verticaux
+ : intersections de murs
C : chasseur
G : gardiens
x : caisses
T : trésor
a/b : affiches

Initialisation

Pour initialiser le labyrinthe on lit ce fichier et par rapport au caractère trouve on effectue différentes choses, qu'on peut voir en commentaires dans la fonction init_lab du fichier Labyrinthe.cpp.

III. Les trois modes du gardien

On a utilisé une stratégie assez similaire pour chaque mode, qui essaie d'imiter le mouvement naturel.

3.1 Patrouille

Les gardiens bougent de façon aléatoire, mais sélectionnent une direction pas très différente de celle qu'ils ont. De plus, ils peuvent choisir exactement le mouvement d'avant, ou de changer un peu l'angle pour qu'ils ne bougent pas tout le temps en ligne droite. Ceci est effectue avec une probabilité plus forte de garder le même mouvement et une probabilité plus faible de changer un peu l'angle. Pour éviter les murs on a implémenté une méthode très naturelle. Si le gardien voit un mur il change un peu l'angle (angle qui permet de garder la même direction) pour éviter le mur.

3.2 Défense

Pour cette mode le mouvement est exactement le même sauf qu'on ajoute une condition que chaque fois le gardien est plus proche du trésor. Pour cela on utilise le tableau qui contient pour chaque case du labyrinthe la distance minimale au trésor.

3.3 Attaque

De même pour cette mode la seule différence est qu'on se dirige vers le chasseur. Pour cela un mouvement pour lequel la distance entre le gardien et le chasseur est inferieur qu'avant. Pour qu'il n'est pas trop facile de tuer un gardien, le gardien augmente se vitesse et change parfois la direction de son mouvement, comme pour le mode patrouille.

Chasseur visible

Un gardien va en mode attaque s'il voit le chasseur. Pour cela on trace la droite entre le chasseur et le gardien et on parcourt tous les points de cette droite en utilisant l'algorithme de tracé de segment de Bresenham. Si tous les points du labyrinthe sont vides et si la distance entre le chasseur et le gardien est supérieure au champ de vision du gardien, alors le chasseur est visible. A cause de problèmes d'arrondis, parfois il y a des gardiens de l'autre côté du mur trouvent le chasseur visible.

Boule de feu

Le gardien tire la boule vers le chasseur, même s'il ne voit pas exactement vers la direction du chasseur. Si le gardien a moins de points de vie, il y a 50% de chance qu'il rate sa cible (On change l'angle de la boule de feu de 10 degrés). De plus le gardien s'il voit le chasseur il tire chaque seconde. « friendly fire » n'existe pas dans notre jeu, c'est a dire si un gardien tire sur un autre gardien rien se passe.

3.4 Changement des modes

Protection d'un gardien

Pour chaque gardien n, le potentiel de protection P(n) est le rapport entre la distance maximale D_{max} au trésor et sa distance au trésor.

$$P(n) = \frac{D_{max}}{distance(n, Trésor)}$$

Distances au trésor

Dans le même fichier on calcule la distance minimale de chaque case au trésor. Pour cela on utilise l'algorithme de Dijkstra en partant de la case du trésor.

Somme des potentiels de protection

Pour trouver la protection totale, on ajoute les potentiels de protection des tous les gardiens en mode défense. Si cette somme est inferieure a un seuil, alors le premier gardien patrouilleur du tableau des gardiens devient un gardien défenseur. Sinon si cette somme est supérieure au seuil, alors le premier gardien défenseur devient patrouilleur.

Seuil

Le seuil est la somme des potentiels de protection lors du lancement du programme.

IV. Particularités

4.1 Teleporte

Si on tire vers une affiche, alors on se téléporte vers une autre affiche d'un mur du labyrinthe.

4.2 Points de vie

Le gardien a moins des points de vie que le chasseur. Sa vie est de 100 est il est mort à deux touches contrairement au chasseur qui est à 5.
En haut à gauche on affiche les points de vie restant au chasseur. Le chasseur est mort à 4 touches.

4.3 Sons

Lors on blesse un gardien, celui-ci produit un son avec un volume par rapport à la distance au chasseur. Ce son est différent si le gardien meurt. Quand le gardien meurt on utilise la fonction rester_au_sol fournit pour le laisser sur le sol.

4.4 La fin du jeu

You win

On gagne lorsque le chasseur se trouve dans une case de distance 1 au trésor. Quand on gagne tous les gardiens s'immobilisent et le chasseur peut seulement bouger dans le labyrinthe. Le message « You win » s'affiche.

You lose

On pert quand on a été touché 5 fois par des boules de feu des gardiens. Le message « You lose » s'apparait. Les gardiens s'immobilisent et le chasseur peut seulement changer l'angle de vue.

On a choisi d'effectuer se comportement pour que quand le joueur perd, il ne peut pas explorer le labyrinthe. Cela lui permet de rejouer sans savoir la partie du labyrinthe qu'il n'a pas déjà explorer. Par contre s'il gagne on lui permet de se déplacer partout dans le labyrinthe.

ANNEXE

Manuel d'utilisation

Gestion du clavier :

- " w " : aller en avant
- " a " : marcher à gauche
- " s " : aller en arrière
- " d " : marcher à droite
- " w+s " : courir

Gestion de la souris :

- Un clic gauche : tirer

Lancer du programme

- Compiler sous Linux 64 bits : make -f Makefile-Proto cstrike64
- Compiler sous MacOSX : make -f Makefile-Proto macstrike
- Ouvrir le jeu : ./cstrike64 –l labs/lab1 -f

Bibliographie

Amar, P. (2017). Programmation objet avancée en C++. [online] Lri.fr. Available at: https://www.lri.fr/~pa/progcxx.html [Accessed 8 May 2017].