UNIVERSITE PARIS-SUD

Master Informatique 1ere Année

Année 2016-2017

# Le labyrinthe : C++ et OpenGL

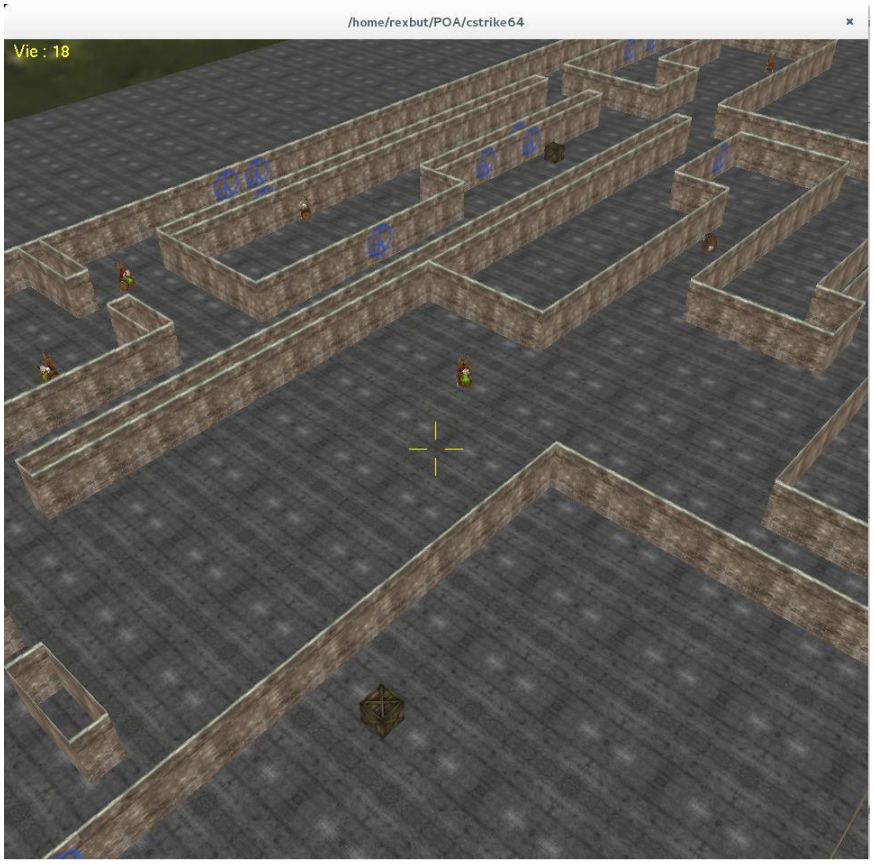
**Rapport du projet de Programmation Objet Avancée**


Par

**BUTET Rémi**
**JAA Ayoub**

**Enseignant :** AMAR Patrick, *Université Paris-Sud*

# Table des matières




## Introduction

Dans le cadre de l'unité d'enseignement de programmation objet avancée de département informatique de l'université Paris Sud, il a été demandé aux étudiants de Master 1, de créer un jeu 3D de type Kill'emAll. Le projet était à réaliser en binôme.

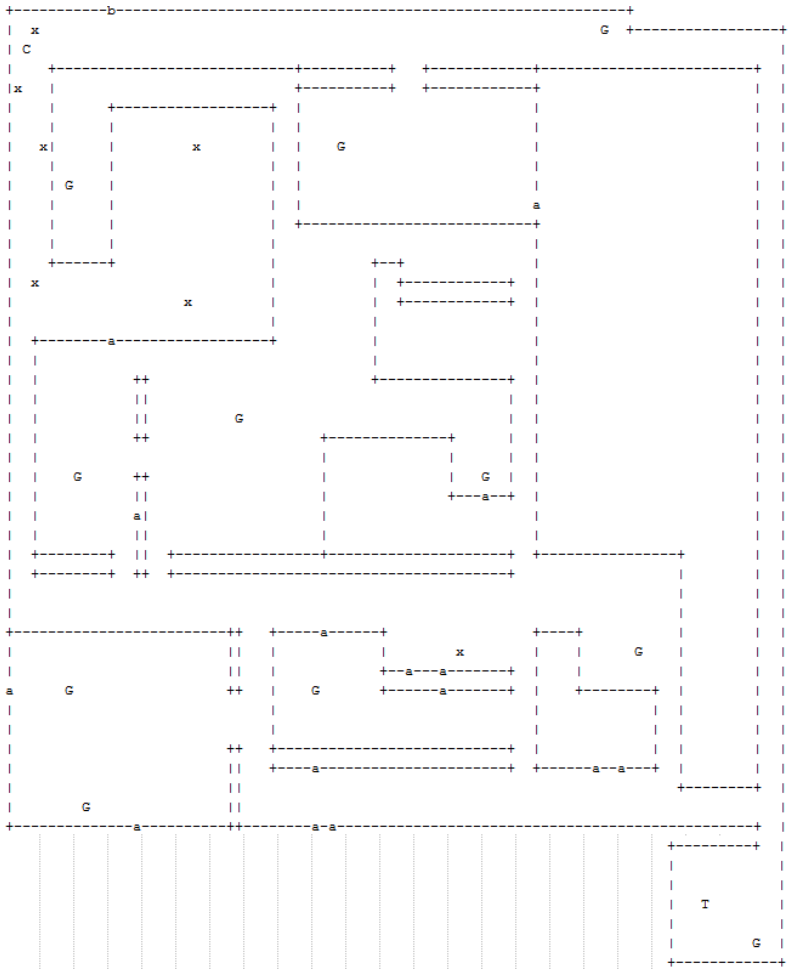
L'objectif du projet était de réaliser un jeu contenant des gardiens et un chasseur. Le principe du jeu est que le chasseur arrive au trésor sans se faire tuer par les gardiens.

# I. Le labyrinthe

Avant de commencer à mettre en place les différents acteurs du jeu, il faut commencer par créer le terrain de jeu.

## I.1 Le fichier texte

Dans un premier temps, nous allons analyser le fichier texte qui représentera le terrain de jeu. Ci-contre, voici un aperçu de ce fichier texte. On distingue plusieurs symboles :



« G » pour gardien, « C » pour chasseur, « - » pour un mur, « + » marquant le début ou la fin d'un mur, « || » comme double mur, « T » pour trésor, « x » pour une boîte, et enfin « a » et « b » pour des affiches (ou autre, pour cela on vérifie si la lettre est comprise entre a et z avec **x exclus**).

Une fois que l'on a compris comment fonctionne le labyrinthe, le but du jeu sera de « parser » le fichier afin de permettre l'affichage des éléments. Le moteur graphique sera OpenGL.

## I.2 « Parser » le fichier

En passant le fichier en paramètre, le but final est d'obtenir l'affichage du labyrinthe ainsi que les positions des différents gardiens, du chasseur (que vous incarnerez) ainsi que les caisses et le trésor. On parcourt notre fichier et à chaque on vérifie quel objet de l'environnement on a lu. Une fois l'élément reconnu, à l'aide d'un compteur ligne et colonne et des coordonnées x et y, on place le nième élément d'un type particulier (boîte, garde, trésor…). On vérifie également lorsque que l'on lit le début ou la fin d'un mur délimité par le « + », ou si un mur est vertical ou horizontal. Puis on met en place les valeurs des différentes cases qui vont permettre le déplacement des gardiens, en effet on part du trésor avec sa case à 0, puis les cases à coté à 1, et ainsi de suite.

# II. Les différents modes

Nous allons maintenant nous intéresser aux différents modes des gardiens, car ces dernières auront un comportement qui évoluera en fur et à mesure du jeu.

## II.1 Mode Patrouille

Le comportement « globale » des gardiens. Ces derniers se déplacent sur la carte sans aucune consigne particulière, cependant, ils pourront à tout moment passer en mode attaque ou défense selon l'évolution de la partie. Ils s'orientent comme bon leur semblent et couvrent toute la carte s'ils le veulent.

## II.2 Mode Attaque

S'active lors que le gardien a réussi à voir le chasseur. Dès qu'ils le voient, ils tirent en s'approchant de lui. Etant donné que cela rend le jeu injouable, on a dû mettre en place un délai entre chaque tir et une probabilité de rater sa cible, pour permettre au chasseur de prendre une décision : fuir ou lui tirer dessus.

## II.3 Mode Défense

Pour ce mode, nous avons mis en place un système de point. On vérifie le statut des gardiens et s'ils sont toujours vivant peuvent rapporter jusqu'à 256 points selon leur distance au trésor. Puis on additionne tous les points des gardiens et on vérifie qu'il y est un total d'environ 2048 points à plus ou moins 256 points. Si le nombre de point n'est pas assez élevé, des gardiens en mode Patrouille passeront en mode Défense et s'il y en a trop certain repasseront en mode Patrouille.

Lorsque qu'un gardien en mode patrouille se dépasse, il aura une plus grande probabilité d'aller dans la direction du trésor. Pour cela nous avons sauvegarde pour chaque case leur distance au trésor, nous avons réalisé ce calcule grâce à l'algorithme Dijkstra que nous avions vu en cours.

### II.4 Particularités des modes et évolutions de la partie

Dans les différents modes, les gardiens peuvent voir le chasseur. Pour ce faire, nous avons simulé le déplacement d'une boule de feu et nous vérifions qu'elle peut atteindre le joueur sans aucune collision. De plus, nous avons mis en place un champ de vision (entre 10 et 80 blocs) qui se réduit selon la vie du gardien car les gardiens ne pourront pas voir le chasseur de très loin même si aucun obstacle ne gêne (boite ou mur).

Autre particularité concernant la boule de feu, l'impact et la précision, en effet, pour le chasseur on a mis en place un impact. Etant donné que toucher précisément un gardien est difficile, nous avons mis un impact qui couvre le bloc touché ainsi que ceux aux alentours (9 cases, 3*3). Par contre les gardiens doivent toucher le chasseur précisément. Ensuite nous avons mis en place un système de point de vie (à 20) ainsi qu'un système de régénération au bout de 2 secondes sans impact (pour tout le monde). Comme autre fonctionnalité, les cases (box ou « x » sur la carte) sont déplaçables.

## Conclusion

Le but de ce projet était dans un premier temps de savoir coder à partir d'un travail déjà fournit car en entreprise ou dans d'autres projets, il se peut qu'une partie du code soit déjà implémenter et donc il faudra comprendre la pensée du développeur et utiliser les outils mis en place afin de compléter le travail. De plus cela nous a permis d'avoir un affichage en OpenGL et donc d'avoir une autre expérience dans le domaine du jeu vidéo en 3D. Enfin, nous avons pu mettre en pratique notre savoir en C++ et donc de pouvoir implémenter un maximum de fonctionnalité.

FIN