

**RAPPORT**

**LABYRINTHE HANTE**

**Clément SAUVARD**

**&**

**Joé DE OLIVEIRA**

Fonctionnalités implémentées et fonctionnelles

* Mode autonome
* Chargement dynamique du labyrinthe
* Positionnement correct des éléments selon le fichier .txt (Chasseur, gardiens, affiches, trésor, murs)
* Gestion des collisions (ex : les murs ne peuvent pas être traversés, que ce soit pour le chasseur ou les gardiens, la boule de feu est arrêté par les obsctacles).
* Algorithme de Dikjstra pour le plus court chemin
* Mode attaque et mode défense pour les gardiens
* Gestion de la vie (Un chasseur a un capital de vie, le gardien aussi).
* Le gardien perd de la précision de tir quand il est blessé
* Le chasseur peut tuer les gardiens
* Gestion des collisions entre la boule et un élément (on sait lorsque la boule touche un mur, un gardien, une caisse, le trésor, ou est dans le vide)
* Fin du jeu lorsque le touche le trésor

Fonctionnalités implémentées mais pas totalement fonctionnelles

* Les gardiens tirent sur le chasseur lorsque le chasseur et le gardien en question sont sur une même ligne et qu’il n’y a pas de mur les séparant (le gardien « voit » le chasseur) mais la collision entre la boule et le chasseur n’est pas fonctionnelle car nous ne mettons pas à jour la position du chasseur dans le tableau \_data. Ainsi le chasseur ne perd jamais de vie.
* Précision du tir : le calcul de la précision du tir du chasseur selon sa vie est bon (comme celui du gardien), mais comme expliqué dessus le chasseur ne perd jamais de vie donc on ne peut pas tester dans le jeu.

Fonctionnalités non implémentées

* Régénération de la vie au bout d’un certain temps pour chasseur et gardien
* Jeu en réseau

Fonctionnement du mode attaque défense :

Voici comment nous avons implémenté le mode attaque : on crée un tableau résultant de notre algorithme de dikjstra, (qui contient donc une numérotation représentative du plus cours chemin) qui fabrique un chemin plus court non pas vers le trésor mais vers un point aléatoire du labyrinthe (en vérifiant que ce point ne soit pas un obstacle), le gardien va donc s’y diriger et une fois qu'il dépassera un certain seuil (c’est-à-dire, si la valeur de la case où il se trouve dans le tableau de dikjstra est grande en l’occurrence) il revient en mode défense et se rapproche du trésor, en utilisant cette fois ci le tableau de dikjstra concernant le trésor. Ainsi, un fois qu’il se rapproche trop du trésor, il repasse en mode attaque et un nouveau point aléatoire dans le labyrinthe est désigné et il doit s’y diriger.

A savoir : le code utilise c++ 14 notamment pour la fonctionnalité « auto » (déduction de type)