2021

Dillinger

FRANCELET SAMY

1 TABLE DES MATIERES

2 Introduction 2
2.1 Contexte 2
2.2 Cahier des charges 2
3 Problèmes rencontrés 3
3.1 Moteur de jeu 3
3.1.1 Modèle MVC 3
3.1.2 Objets du jeu 3
3.2 Système de champ de vision 4
3.2.1 Première idée : Un triangle de détection 4
3.2.2 Deuxième idée : Des rayons de vue 4
3.3 Course poursuite 5
3.3.1 Algorithme de pathfinding A* 5
4 Implémentation 7
4.1 Lancement du jeu 7
4.2 Implémentation des classes 7
4.3 Physique et mouvement 7
5 Bilan 7
5.1 Comment jouer 7
6 Annexe 1 – Diagramme de classe complet 0

2 INTRODUCTION

2.1 CONTEXTE

Dillinger est un petit jeu d'infiltration, réaliser entière en C++ avec les librairies Qt. Ce jeu a été réalisé lors du cours INF2 de la HEI de Sion.

L'idée du jeu est d'incarner un braqueur de banque, du nom de Dillinger, qui doit voler sans se faire attraper un paquet d'argent. Le paquet d'argent se trouve en plein milieu d'un bâtiment rempli de gardes. Les gardes, possédant un œil extrêmement affuté, sont capable de détecter la présence du joueur par la vue, puis de le pourchasser inlassablement.

Le jeu se jouera en vue du dessus, et aura des graphismes très minimaliste par soucis de temps.

2.2 CAHIER DES CHARGES

Les objectifs à atteindre ont été fixé par moi-même, ils sont comme suit :

- V0.0 :
  - Déplacement du joueur
  - Placement des murs et fenêtres
  - Implémentation de gardes et attribution d'un parcours à suivre
- V0.1 :
  - Implémentation du système de champ de vision
  - Implémentation de la poursuite des gardes
- V0.2 :
  - Implémentation de l'objectif à récupérer et à fuir avec
  - Implémentation des caméras de surveillances
  - Menu principal
- V1.0 :
  - Ajout des policiers qui viennent après une capture ratée
  - Ajout de plusieurs niveaux
  - Ajout d'un score

3 PROBLÈMES RENCONTRÉS

3.1 MOTEUR DE JEU

N'ayant comme seule librairies disponibles les librairies Qt, il a fallu créer un petit moteur de jeu et séparer les objets peuplant le jeu intelligemment.

3.1.1 Modèle MVC

Pour la gestion de l'affichage et des contrôles du jeu, le modèle MVC a été choisi

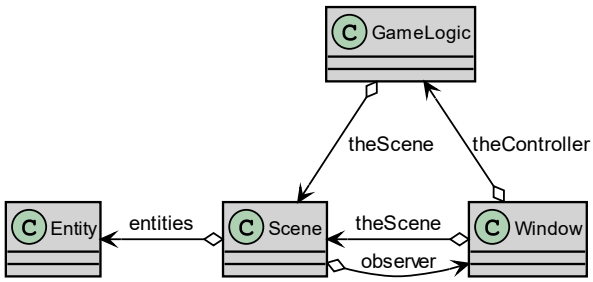

Figure 1 - Modèle MVC implémenté (format simplifié)

On se retrouve donc avec une scène de jeu, peuplée d'entité (Murs comme joueur), contrôlée par le GameLogic qui reçoit des commandes du clavier/souris par la Window.
La scène est capable d'informé la Window qu'elle a été modifiée, pour que la Window puisse rafraichir l'affichage.

3.1.2 Objets du jeu

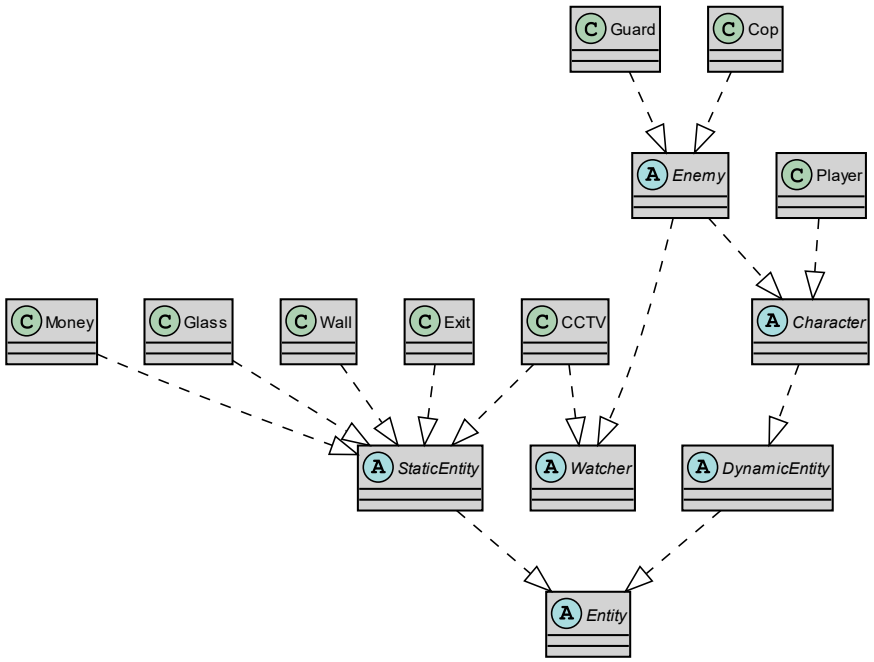

Figure 2 - Structure des GameObjects

Les objets sont donc séparés ainsi : Tous sont des entités, soit dynamique, soit statique.
Seules les entités dynamiques pourront se déplacer dans la scène. Il y a également certains objets, statiques et dynamiques qui sont des « Watcher ». Il s'agit des gardes et caméras qui auront la faculté de détecter le joueur.

3.2 Système de champ de vision

Le moteur de jeu étant prêt, il faut ajouter un système de détection.

3.2.1 Première idée : Un triangle de détection

Au premier abord plus simple, léger en calcul. Le problème arrivant quand il faut bloquer le champ de vision car traversant un mur. Le triangle n'en sera plus un et le calcul des nouveaux coins du Polygone semble hors de portée.

3.2.2 Deuxième idée : Des rayons de vue

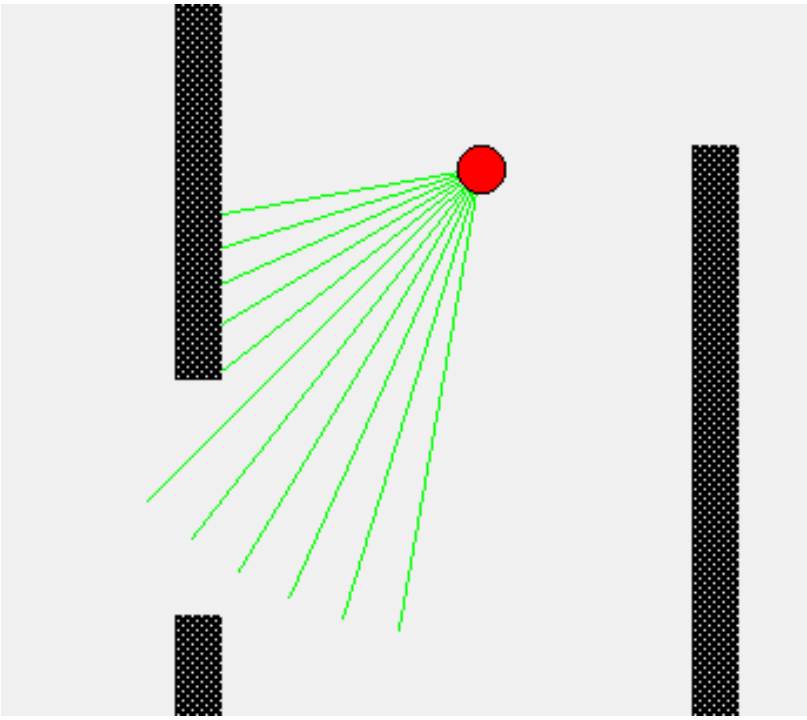

Figure 3 - Gardes avec ses rayons de vue

Cette solution paraît, au premier abord, plutôt lourde, mais fonctionne finalement à merveille.
La méthode est simple : Générer plusieurs lignes de « rayons de vue », calculer les intersections avec les murs, et raccourcir la ligne. Actuellement chaque « Watcher » possède plus de 100 « rayons de vue », et cela ne pose pas de problème de performances.

3.3 COURSE POURSUITE

Les gardes peuvent se déplacer et détecter le joueur. Cependant ils doivent pouvoir foncer sur le joueur, sans se coincer dans un mur. Un algorithme de « pathfinding » a donc été utilisé. L'algorithme **A*** a été choisi, car beaucoup de ressource sont disponible sur Internet.

3.3.1 Algorithme de pathfinding A*

Le principe est simple, assigner a chaque case un coût, et chercher le bon chemin en passant par le chemin ayant le moins de coût :

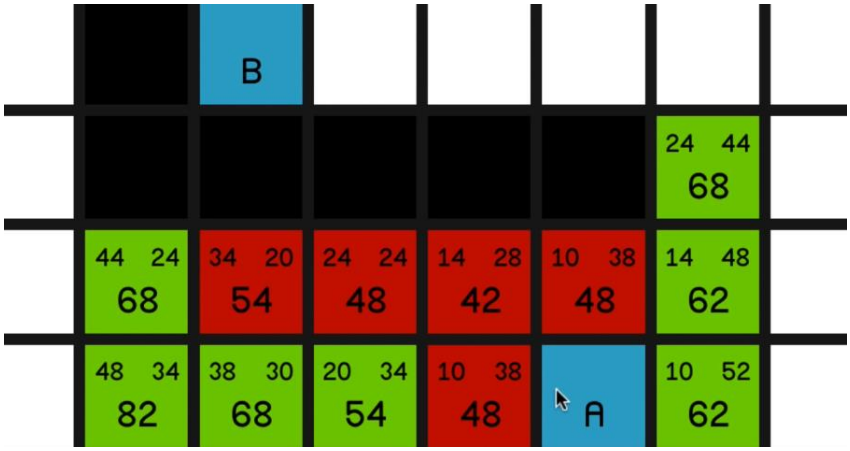

Figure 4 - Exemple de calcul des coûts

Ici on souhaite aller de la case A, à la case B. On va d'abord calculer les coûts de toutes les cases voisines à A. On distingue 2 coûts : La distance à l'origine et la distance à l'objectif. On itère en sélectionnant la case avec le coût le plus bas, on recalcule le coûts de tous les voisins, etc..

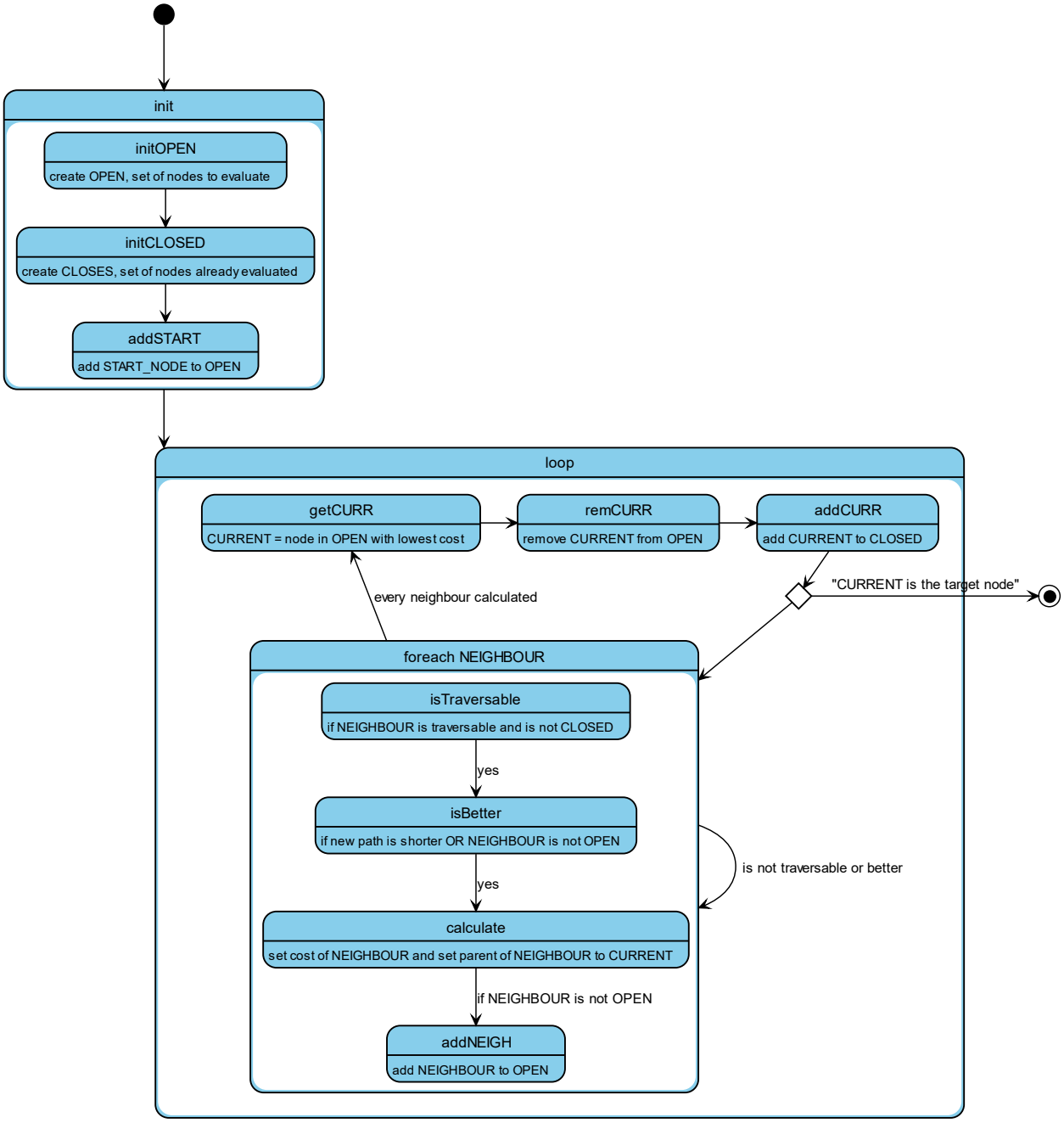

*Figure 5 - Diagramme d'état de l'algorithme A**

La recherche se fait donc en 2 boucles en série :

- La boucle qui cherche la node la moins coûteuse, et qui se termine quand on a trouvé l'objectif
- La boucle qui calcul le coût de chaque voisin de la node actuelle

Si on assigne des « parents » à chaque node, on peut faire le chemin arrière une fois la node finale trouvée.

4 IMPLÉMENTATION

4.1 LANCEMENT DU JEU

Pour faciliter le lancement du jeu, ainsi que le passage du jeu au menu et inversement, une class « Dillinger » a été créée, au format Singleton pour être accessible de partout dans le code.
Une méthode « loadLevel » est disponible pour lancer un niveau. Pour le moment seul un niveau est codé en dur dans le programme.

4.2 IMPLÉMENTATION DES CLASSES

Un digramme détaillé de toutes les classes est disponible en annexe 1

4.3 PHYSIQUE ET MOUVEMENT

La classe GameLogic calcule périodiquement les déplacements de toutes les entités dynamiques en fonction de leur vitesse.

5 BILAN

La V0.1 du cahier des charges est implémentée. La barre a été misée un peu trop haute par rapport aux limitations de temps.
Les fonctionnalités de pathfinding et champs de vision pourraient être encore optimisé.

L'implémentation de nouveaux niveaux se fait très facilement

5.1 COMMENT JOUER

Le jeu se contrôle avec les touches WASD du clavier, le personnage ne se déplace qu'à une seule vitesse. Pour le moment il n'y a pas d'objectif, autre que d'essayer d'éviter les gardes pour le défi

5.2 SIGNATURE

Réchy, le 7 juin 21,

Francelet Samy

6 ANNEXE 1 – DIAGRAMME DE CLASSE COMPLET