Algorithme et structure de données - projet

# Introduction

Le but de ce projet est l’intégration d’une IA dans le jeu Bomberman développé lors du semestre d’automne du P2. Ce jeu a été programmé en C++ avec le framework Qt.

Deux joueurs s’affrontent sur une carte. L’idée est l’implémentation et l’utilisation concrète d’un algorithme vu en cours d’algo durant l’année. La thématique la plus approprié à Bomberman, à notre avis, est l’ajout d’une fonctionnalité de « Path finding ».

Le joueur IA doit avoir comme fonctionnalités la recherche du chemin le plus court vers le joueur humain. L’algorithme de Dijkstra est utilisé. Une file de priorité minimum a été spécialement développée pour répondre aux besoins de notre projet. Afin de se frayer un chemin au travers des cartes, le joueur IA a la possibilité de poser des bombes. Le joueur pose une bombe et se déplace à un endroit sûr et le plus proche possible.

# Implémentation

## Carte de jeu et recherche de chemin

La carte du jeu se compose de 30 lignes et de 30 colonnes de blocs, représentés par des objets de type « MapBloc ». C’est la carte qui s’occupe des traitements algorithmiques nécessaires à la création du graphe et de le recherche de chemin. L’IA devra appeler les méthodes de la carte pour obtenir ses itinéraires.

### Construction du graphe

Lors de la création du graphe, chaque bloc est parcouru à la recherche de ses voisins. Seuls les blocs étant de type indestructible ne sont pas considérés comme des voisins car il est impossible de les détruire, et donc de s’y rendre. Le graphe est recréé lorsqu’une bombe explose afin de tenir compte d’éventuels nouveaux chemins.

La méthode « buildGraph () » s’occupe de la construction du graphe.

Nous avons préféré une implémentation du graphe par liste d’adjacence, car le fait d’ajouter les voisins de chaque bloc était très bien compatible avec le projet existant et nécessitait peu d’adaptations.

### Recherche de chemin

La méthode « getShortestPath(depart,destination) » retourne la liste des blocs sur lesquels le joueur IA doit se déplacer pour atteindre le plus rapidement possible sa destination. Cette méthode est appelée :

* Au début de la partie
* Lorsque le joueur humain se déplace
* Quand une bombe explose
* Quand le joueur IA pose une bombe et cherche un chemin vers un bloc « sûr »

Lors du parcours du graphe pour la recherche de chemin, les nœuds sont ajoutés dans la file de priorité. Leur priorité est calculée en fonction de leur type (un bonus de puissance des bombes est prioritaire sur un bloc vide par exemple).

Avec la préoccupation d’améliorer les performances, nous avons mis en place une implémentation multi-threadée : Le joueur IA est géré par un thread, ainsi que la construction du graphe. Une gestion de la concurrence a été mise en place afin d’éviter les problèmes d’accès concurrents.

Le joueur IA se déplace grâce à des simulations de pressions de touches du clavier, afin de faciliter l’intégration de l’IA dans le fonctionnement actuel du projet.

## File de priorité minimum

Une classe implémentant une file de priorité minimum a été développée. Comme elle est implémentée avec des templates, elle est générique et supporte différents types.

Cette file est implémentée grâce à la classe QList, qui est un tableau de taille dynamique. Cette QList est gérée à la manière d’un arbre : un nœud d’indice *i* est le père des nœuds d’indices *2i* et *2i+1*.

Chaque élément de la file est un PriorityQueueNode, classe développée pour ce projet, qui contient un pointeur vers le MapBloc correspondant. Ce choix a été fait afin de séparer la gestion de la priorité des éléments de la file par rapport aux MapBlocs. De plus, cette implémentation permet d’intégrer facilement plusieurs joueurs IA, car la priorité des blocs ne dépendant pas des blocs eux-mêmes, plusieurs joueurs pourraient construire simultanément une file de priorité correspondant à leur propre itinéraire.

De plus, comme la file de priorité minimum est générique, il est plus efficace de travailler avec des éléments indépendants du type des objets que l’on veut prioriser.

# Structure des classes

Comme notre projet s’insère dans un projet plus grand et pour faciliter la lecture du code correspondant, voici les modifications d’intérêt que nous avons réalisées pour ce projet, ainsi que leur emplacement.

A screen shot of a computer

Description generated with very high confidence

Gestion du graphe et recherche de chamins

File de priorité minimum (implémentation dans priorityqueue.h car classes génériques)

Logique de jeu – surtout le constructeur et la méthode startGame()

Comportements de l’IA