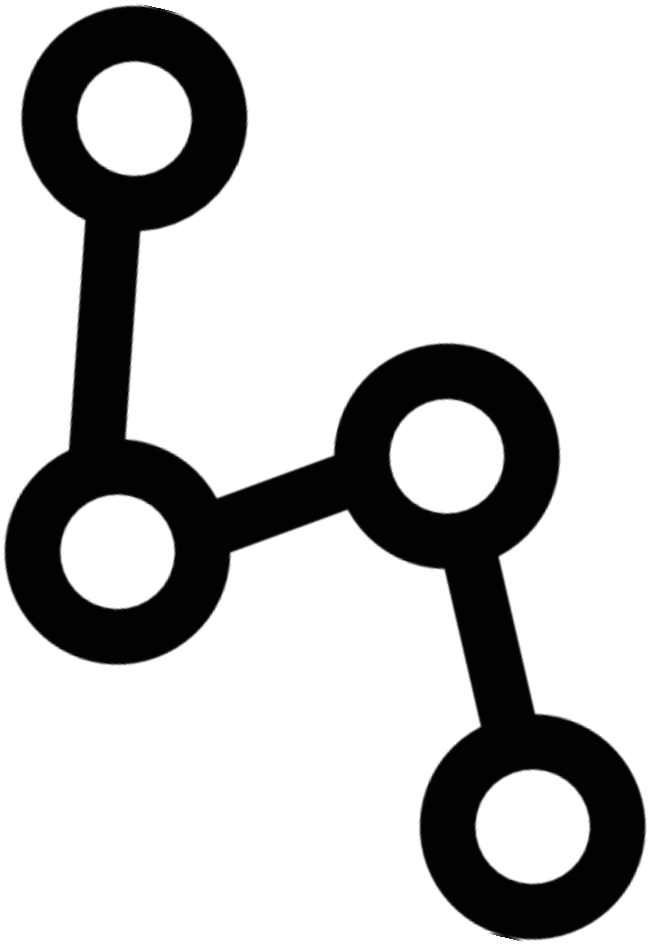
Pathfinding game



Monfroy Wendy

Projet 2018 – ISIMA

Table des matières

Introduction

Dans le cadre des projets de première année, nous avions choisi de proposer notre propre sujet, à savoir l'élaboration d'un programme en C mettant en œuvre des algorithmes de pathfinding appliqué au jeu vidéo. En effet, l'univers vidéo ludique requiert régulièrement l'utilisation d'algorithmes de pathfinding, par exemple pour l'exploitation d'un compagnon qui doit suivre les ordres du joueur ou encore pour organiser les déplacement d'un ennemi contrôlé par une intelligence artificielle.

Notre objectif était donc de réaliser un jeu du chat et de la souris, dans lequel le chat aurait été géré par un algorithme de plus court chemin tandis que la souris aurait été dirigée par le joueur dans le but d'échapper au chat. Les objectifs et démarches précises de la mise en œuvre d'un tel projet sont détaillés ci-après.

Déroulement du projet

Organisation envisagée

Originellement, il avait été décidé de réaliser un jeu complet du chat et de la souris mettant en œuvre les algorithmes de plus court chemin de Dijkstra, Bellman-Ford et A*, en langage C. L'affichage graphique de ce jeu serait géré à l'aide de la bibliothèque SDL2. Il était également envisagé de comparer les différents algorithmes en fonction de leur efficacité ainsi que d'essayer d'implémenter une mise à jour du chemin trouvé en temps réel en fonction des changements d'environnement.

Le code pour ce projet aurait été développé en trois parties distinctes :

- L'ensembles des fonctions de gestion de la map
- Les algorithmes de pathfinding
- La gestion de l'affichage graphique avec SDL2

Réalisation du projet

Structures et organisation du code

Après réflexion, j'ai choisis de baser mes algorithmes sur des matrices qui sont traitées comme des graphes usuels : chaque point de la matrice correspond à au nœud du graphe correspondant et chaque arrête et symbolisée par l'adjacence de deux points. L'utilisation de matrices me permet un accès plus simple en deux dimensions du fait de l'utilisation de deux coordonnées, cela alourdit légèrement les algorithmes mais rend la transcription graphique plus intuitive.

Pour ce qui est de la map dans laquelle on souhaite trouver un chemin, le coût d'accès à une case suivante est simplement modélisé par un entier positif ou nul : plus cet entier est faible, plus l'accès est facile (0 pour un chemin) alors qu'un entier élevé correspond à un accès laborieux (par exemple des sables mouvants). Je représente également des zones inaccessibles par des entiers négatifs.

La gestion de ces différentes matrices utilisées pour le pathfinding est réalisée par les fichiers map.c et map.h

En ce qui concerne les algorithmes de plus court chemin, j'ai choisi de ne pas appliquer l'algorithme de Bellman-Ford pour me concentrer plus spécifiquement sur ceux de Dijkstra et A*.

Ces algorithmes ainsi que la gestion de l'affichage des chemins trouvée sont définis dans les fichiers path.c et path.h

La finalisation du chemin entre sa recherche et son affichage nécessite la gestion d'une pile dans laquelle il faut pouvoir stocker les coordonnées de points d'une matrice. Pour simplifier sa structure et réduire sa place en mémoire, la pile prend la forme d'une liste chainée pour laquelle chaque bloc alloué est effectivement nécessaire (une liste contiguë aurait une taille déraisonnable puisqu'il faudrait dans le pire des cas pouvoir stocker tous les points d'une matrice potentiellement grande, alors qu'en pratique seuls quelques point auront besoin d'être stockés) et dont le dernier élément est en tête. De la même façon, les coordonnées sont stockées sous la forme d'un entier x*nbcolonnes+y.

On retrouve toutes les fonctions de gestion de pile dans les fichiers pile.c et pile.h

Par manque de temps et comme j'ai dû réaliser ce projet seule, je n'ai pas abordé l'aspect graphique de ce projet avec SDL2. Cependant, une fois le chemin trouvé, ce dernier est affiché clairement dans une matrice dédiée.

Explication des algorithmes

— Gestion des matrices (fichiers map.c et map.h) :

Une matrice est d'abord allouée dynamiquement et initialisée avec une valeur choisie (0, -1 ou n).

La matrice correspondant au terrain est remplie à partir d'un fichier texte et une fois cela fait, la matrice des points visités est mise à jour en fonction des nœuds inaccessibles de la map, c'est-à-dire que l'on considère un point inaccessible comme visité.

La matrice de distance estimée, utilisée par l'algorithme A*, est remplie à l'aide de deux boucles for qui incrémentent la valeur de la distance au fur et à mesure que l'on s'éloigne du point d'arrivée.

J'ai réalisé une fonction qui me permet de connaitre le nombre de points inaccessibles de la matrice de terrain ainsi qu'une fonction qui permet l'affichage d'une matrice et une fonction de libération.

NOTE : le nombre de lignes et de colonnes des matrices est défini dans le fichier map.h donc il faut penser à le mettre à jour si l'on souhaite étudier des matrices de taille différente.

— Gestion de pile (fichiers pile.c et pile.h) :

Comme décrit précédemment, une pile des définie par une structure contenant la taille effective de la pile et pointant sur une liste chainée d'entiers, elle-même définie dans une structure. Toutes les fonctions de gestion de pile sont celles usuelles, à la différence près qu'un bloc empilé l'est en tête de liste donc lorsqu'on souhaite dépiler ou connaitre le dernier bloc il suffit de prendre le premier.

— Gestion du pathfinder (fichiers path.c et path.h) :

Les algorithmes de plus court chemin s'appuient sur plusieurs matrices :

— une matrice de terrain (la map) qui est invariante lors de l'appel des algorithmes

— une matrice de distance minimale calculée (à l'origine initialisée avec une valeur suffisamment grande pour être considérée comme infinie et mise à jour à chaque évaluation d'un point)

— une matrice de précurseurs initialisée à -1 pour marquer l'absence de précurseur et mise à jour également lors du traitement d'un point

— une matrice de distance estimée pour l'algorithme A* qui répertorie la distance d'un point de la matrice par rapport au point d'arrivée sans tenir compte d'éventuels obstables

Les algorithmes de Dijkstra et A* ont globalement la même forme car ils ne diffèrent que par leur choix du point suivant à traiter. Ces deux algorithmes se décrivent de la façon suivante :

Tant qu'on n'a pas visité tous les points (autant d'itérations qu'il y a de nœuds accessibles)
    On choisit le prochain point à traiter (fonctions différentes pour les deux algorithmes)
    Pour tous les voisins de ce point qui sont accessibles et pour lesquels le déplacement est moins coûteux que celui déjà calculé
        On met à jour la matrice de distance minimale
        On met à jour le précurseur de ce voisin
    On passe ensuite le point considéré en visité

La fonction de choix de l'algorithme de Dijkstra choisit le point non visité du terrain pour lequel la distance minimale est minimale. Au premier appel c'est le point de départ qui est choisi car il est le seul à avoir une distance minimale nulle et n'est pas encore visité.

La fonction de choix de l'algorithme A* choisit le point non visité du terrain pour lequel la distance minimale additionnée à la distance estimée à l'objectif est minimale. De la même façon que pour l'algorithme de Dijkstra, le premier point sélectionné est celui de départ car peu importe sa distance estimée au point d'arrivée, il est le seul à avoir une distance minimale nulle (non infinie).

Une fois la matrice des précurseurs établie à l'aide d'un des deux algorithmes précédant, une fonction permet de créer la pile correspondant au chemin obtenu en partant du point d'arrivée.

Ensuite, une autre fonction dépile ce chemin de façon à créer une matrice qui servira à afficher clairement le chemin emprunté entre le point de départ et le point d'arrivée.

NOTES :

— pour que ces algorithmes fonctionnent correctement, il ne faut pas qu'il y ait de blocs isolés dans la matrice de terrain

— il est important de remarquer que les deux algorithmes ici ne considèrent pas le voisin diagonal de la case étudiée de la même façon : pour A*, cette case est plus loin que pour Dijkstra.

Continuation possible

Puisque cela n'a pas été traité, il serait intéressant d'utiliser SDL2 pour afficher la progression d'un compagnon sur le chemin trouvé et créer un jeu complet autour de ces algorithmes de pathfinding. Il serait également intéressant de pouvoir adapter ces algorithmes en fonctions des capacités du personnage : par exemple, un chat pourrait sauter par-dessus une case et ainsi franchir un ruisseau tandis qu'une souris serait bloquée. Pour cela, il suffirait de décliner les algorithmes de plus court chemin selon ces deux entités, chat et souris. Par ailleurs, une génération aléatoire du terrain ou la gestion de niveaux de difficulté pourrait être ajoutés pour étoffer ce projet.

Problèmes rencontrées

J'ai éprouvé de nombreuses difficultés liées au fait que je me sois retrouvée seule au cours du projet. J'ai dû revoir les ambitions du projet à la baisse puisque le fait d'être seule a réduit considérablement le temps allouable à chaque objectif. En résulte l'abandon total de la partie graphique du jeu.

Par ailleurs, les fonctions de gestion de map réalisée par Marc-Antoine Lafarge, mon ancien binôme, étaient trop pointues et complexes pour être aisément reprises dans les algorithmes de recherche de plus court chemin. Il a donc fallu refaire des fonctions de gestion de matrice plus simples et efficaces.

J'ai également dû faire face à des difficultés personnelles concernant la réalisation même des fonctions. En effet, je me suis plusieurs fois retrouvée coincée sur de mauvaises pistes, situations débloquées en étudiant à nouveau le problème et en cherchant de la documentation concernant les problèmes rencontrés.

Bilan technique

La réalisation de ce projet m'a permis mieux maîtriser le langage C ainsi que de découvrir et de maîtriser la base de l'utilisation de SDL2 puisque j'avais d'ores et déjà commencé à l'utiliser avant de l'abandonner par manque de temps.

J'ai également put développer mon autonomie étant donné la situation de mon binôme. Par ailleurs, cela m'a permis de mieux me rendre compte du temps nécessaire à la réalisation de chaque partie d'un projet, de son élaboration à sa finalisation en passant par l'implémentation du code des différentes fonctions.

Conclusion

Bien que ce projet ne réponde pas en totalité à ses ambitions, il a été possible de mener à bien la partie centrale et essentielle de ce dernier, à savoir l'implémentation d'algorithmes de plus court chemin et leur utilisation pour déterminer un chemin « rapide » entre deux point sur une map.

Par ailleurs, ces différents algorithmes pourraient aisément être étoffés, moyennant un certain temps supplémentaire, en y ajoutant un aspect graphique et en créant tout un univers de jeu autour de cette base.