ENSICAEN
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DE CAEN
& CENTRE DE RECHERCHE

Rapport de projet

Grand Prix

Encadrants de projet : Hidane Moncef et Christophe Rosenberger.

Table des matières

1 Introdution

Ce projet consiste en la réalisation d'un pilote en langage C pour GNU/Linux. Un pilote est un programme qui, en communication avec un gestionnaire de course *via* les entrée/sortie standards, va envoyer successivement des accélérations pour rejoindre un point d'arrivée, en respectant un certain nombre de contraintes qui sont détaillées dans l'énoncé du sujet écrit par Julien Gosme.

En particulier, les objectifs de ce travail sont de réaliser un pilote robuste qui respecte les conditions imposées par le gestionnaire de course et qui relie un point de départ à un point d'arrivée en un nombre de tours de jeu que l'on peut espérer optimal, et ce, pour tout type de carte[1].

2 Définitions et notations

Avant de comprendre ce qu'est un bon pilote[2], il est naturel de se demander quelle est sa définition[3] et comment ce dernier peut se comporter face à une carte quelconque. Le but de cette partie est donc de formaliser le problème en vue de poser des bases solides pour l'implémentation du programme.

2.1 Carte

Un carte de de taille $h \times w$ est une matrice de caractères parmi {`#`,`~`,`.`,`=`}.

[1] Éventuellement non connexe
[2] Un pilote qui gagne souvent les courses !
[3] Et donc comment le représenter

2.2 Pilote

Un pilote est la donnée d'un vecteur position p, d'une vitesse v et du nombre de boosts b dont il dispose. Le triplet (p, v, b) consitue **un état unique du pilote**.

2.3 Voisinage

Une fonction de voisinage Φ est une fonction donnant, pour tout état $e = (p, v, b)$ de pilote, un ensemble $\Phi(e)$ d'états accessibles à partir de e. On construit ce voisinage à partir des contraintes imposées par le gestionnaire : on parcourt la liste des accélérations possibles à partir de e et on regarde celles qui mènent vers un état correct.

3 Structures de données

Les principales structures de données utilisées dans ce projet sont les suivantes :

3.1 Map

Représentation d'une carte par un tableau 2D de caractères.

```
..............................
..............................
..............................
..............................
..##########################==...
..##########################==...
..##########################==...
..##########################==...
..##########################==...
..##########################==...
..##########################==...
..##########################==...
..............................
..............................
..............................
..............................
..............................
```

Exemple La carte *Droit au but* fournie avec le gestionnaire de course

3.2 Vertex

Représentation d'un sommet d'un graphe. Contient un élément, une clef et un pointeur vers un prédecesseur.

3.3 Graph

Représentation d'un graphe. J'ai choisi d'implémenter cette structure par un tableau 4D de pointeurs sur des sommets pour permettre un accès en temps constant à un état d'un pilote à partir d'un autre. Cette représentation facilite grandement la construction du voisinage $\Phi(e)$ d'un état e. Dans cette représentation, un état e est accessible dans le graphe G en temps constant par ses paramètres : $e = G[x][y][v][b]$[4]. Un accéleration a permettant d'atteindre une vitesse v' à partir de v l'état suivant de e par a est donné par $G[x+dx][y+dy][v'][b']$ où dx et dy sont les composantes de la vitesse v' et où $b' = b-1$ si la norme de a est supérieure à 2, $b' = b$ sinon.

3.4 Path

Chemin à partir d'un sommet renvoyé par l'algorithme de Dijkstra détaillé plus loin. Le chemin correspond à une liste de sommets.
Se référer aux fichiers d'en-tête pour plus de précisions concernant l'implémentation de ces structures.

[4] Attention, une vitesse v est représentée ici par son index, donc un seul entier ; ceci étant possible du fait que l'ensemble des vitesses de norme $\leq$ à 5 est fini et donc qu'on peut le numéroter

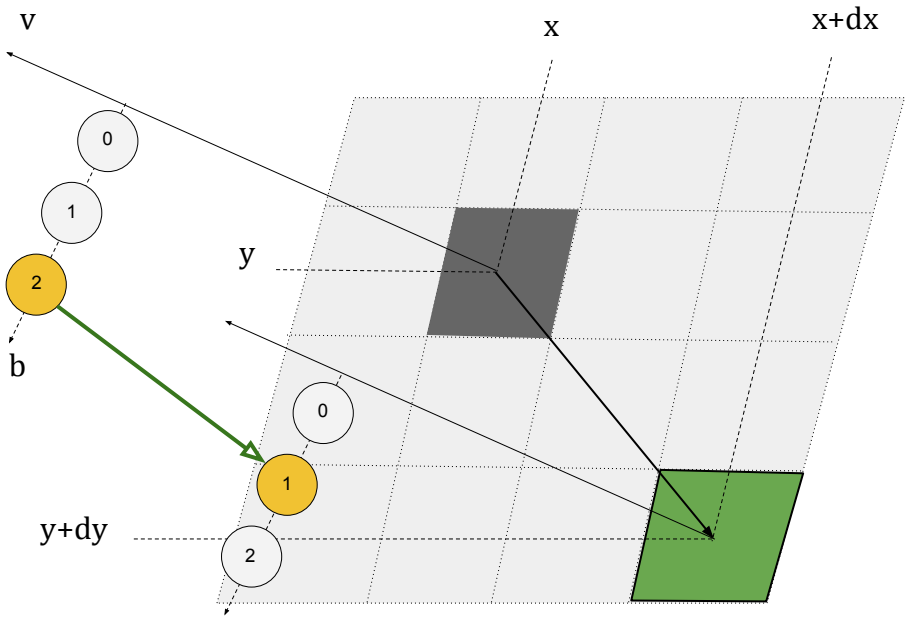

Figure 1: Représentation du graphe des états possibles

4 Algorithme implémenté

4.1 Calcul d'un plus court chemin

Parmi les différentes stratégies qui peuvent être mises en place pour trouver un chemin de longueur optimale, j'ai choisi d'implémenter une version [5] de l'algorithme de Dijkstra pour trouver le plus court chemin dans le graphe des états possibles du pilote.

Cet algorithme, détaillé ci-dessous, permet de calculer le plus court chemin dans le graphe des états possibles d'un pilote en temps polynômial. L'extraction du minimum, qui se fait en $O(log_2(n))$ repose sur l'utilisation des tas de Fibonacci, structure de donnée conçue par Michael L. Fredman et Robert E. Tarjan en 1984 pour optimiser l'algorithme de Dijkstra. Pour cette partie[6] seulement, j'ai utilisé une implémentation disponible à l'adresse suivante :

`http://resnet.uoregon.edu/~gurney_j/jmpc/fib.html`

Description de l'algorithme de Dijsktra Soit un graphe G de fonction de voisinage Φ et soit un point de départ a.

E1. Pour tout sommet s du graphe des états G, faire $d(s) \leftarrow \infty$

E2. $d(a) \leftarrow 0$

E3. Ajouter a au tas F.

E4. Tant que $F \neq \emptyset$, extraire le sommet u de distance minimale et mettre à jour chaque $v \in \Phi(u)$:

E4.1 Si $d(v) > d(u)+1$, faire $d(v) \leftarrow d(u)+1$ et $pred(v) \leftarrow u$.

E4.2 Si v est un état correspondant à une position d'arrivée, on peut retourner directement ce sommet.

Construction du chemin Le chemin se construit facilement en remontant à partir du sommet renvoyé par l'algorithme précédent, vu qu'on a stocké dans chaque sommet son prédecesseur.

4.2 Mise à jour dynamique du chemin

Il faut garder à l'esprit que la course se déroule en présence d'autre pilotes et que l'on ne peut bien évidemment pas, à un instant donné, aller vers une case occupée par un autre pilote. Cela implique qu'il faut vérifier à chaque tour de jeu la validité de l'accélération que l'on s'apprête à envoyer au gestionnaire.

[5] Originellement, cet algorithme très simple ne faisait que calculer la longueur du plus court chemin dans un graphe. Une légère modification nous permet de récupérer ce chemin

[6] Relativement complexe

E1. Si on n'a pas de chemin, on construit un plus court chemin jusqu'à l'arrivée.

E2. Tant que la partie n'est pas terminée, on récupère l'état suivant du pilote à partir du chemin précédemment calculé et on teste sa validité

E2.1 Si le coup est valide, on envoie l'accélération au gestionnaire de course.

E2.2 Si le coup est non valide, on reconstruit un nouveau chemin optimal en retirant les états de $\Phi(e)$ dont la position coïncide avec au moins l'une des positions des autres pilotes. On obtient un nouvel ensemble $\Phi'(e)$.

E2.2.1 Si $\Phi'(e)$ est vide, alors le pilote ne peut donner une autre accélération lui permettant de continuer et le gestionnaire l'arrête. On recommence depuis (E1) à vitesse nulle.

E2.2.2 Si $\Phi'(e)$ contient au moins un élément, on recalcule donc un nouveau chemin optimal à partir de la position courante et on continue à partir de (E2.1).

Cette mise à jour simple est détaillé dans le diagramme ci-dessous :

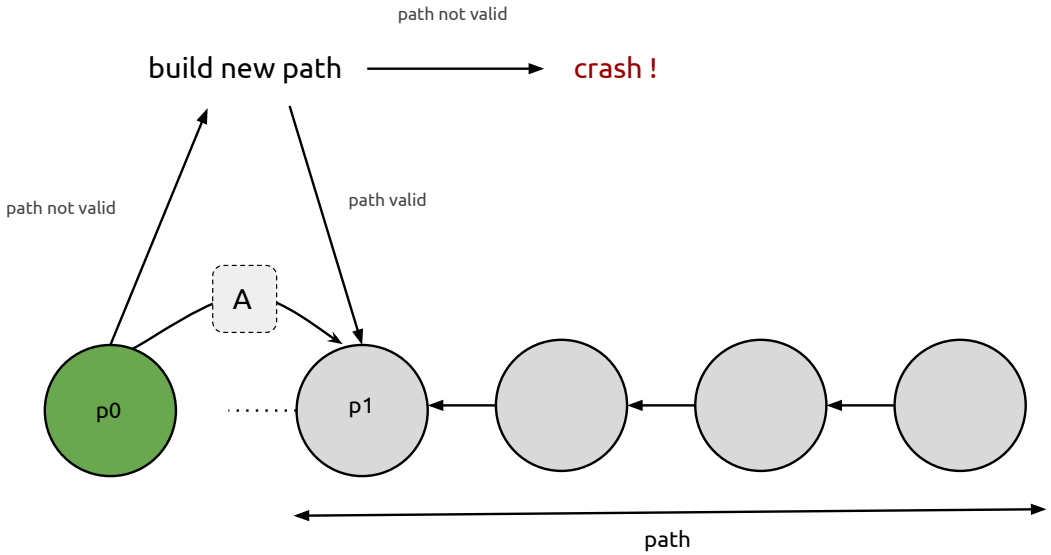

Figure 2: Mise à jour dynamique du chemin

5 Conclusion

Ce projet a été l'occasion d'implémenter et d'adapter l'algorithme de Dijsktra qui donne à partir d'un graphe des états possibles d'un pilote, un chemin optimal reliant ce même pilote à l'arrivée, sur tout type de carte [7].

Le pilote ainsi obtenu permet de calculer très rapidement un chemin optimal jusqu'à l'arrivée, pour des cartes de taille raisonnable. Il s'adapte en fonction de ses adversaires, en réévaluant le chemin si nécessaire. À titre d'information, voici quelques résultats obtenus[8] sur les cartes officielles, en comparaison avec les pilotes GPDriverOne et GPDriverTwo fournis avec le gestionnaire de course.

Carte/Pilote	GPDriverOne	GPDriverTwo	Mon pilote
Droit au but	13	9	6
Virages	112	75	39
Virages et sable	430	354	41
Serpent	257	77	33

[7] Il faut bien évidemment qu'au moins un chemin reliant le départ et l'arrivée existe
[8] En nombre de tours de jeu