TP3 : Interface graphique et algorithmique

Le but de ce TP est de trouver un chemin sur une carte, et de l'afficher.

1 Modélisation et affichage d'une carte

Modélisation : Nous représenterons une `Carte` comme étant une grille de `Case`. Une `Case` peut être soit du `Terrain`, soit un `Mur` infranchissable. Une fois la carte créée, sa topologie ne changera pas.

Affichage : Pour afficher une carte, créez une classe `PanneauCarte` dérivant de `JPanel`. Son constructeur prendra une `Carte` en argument. Redéfinissez la méthode `void paint(Graphics g)` pour dessiner la carte.
Créez une classe `FenetreCarte`, dérivant de `JFrame`. Créez la méthode `void Afficher(Carte c)`, qui va créer un `PanneauCarte` et l'ajouter au contenu de la fenêtre.
Enfin, ajoutez la méthode `void Afficher()` à la classe `Carte`, qui crée une `FenetreCarte` et affiche la topologie de la carte.

Votre fenêtre ressemblera en quelque sorte à la figure 1.

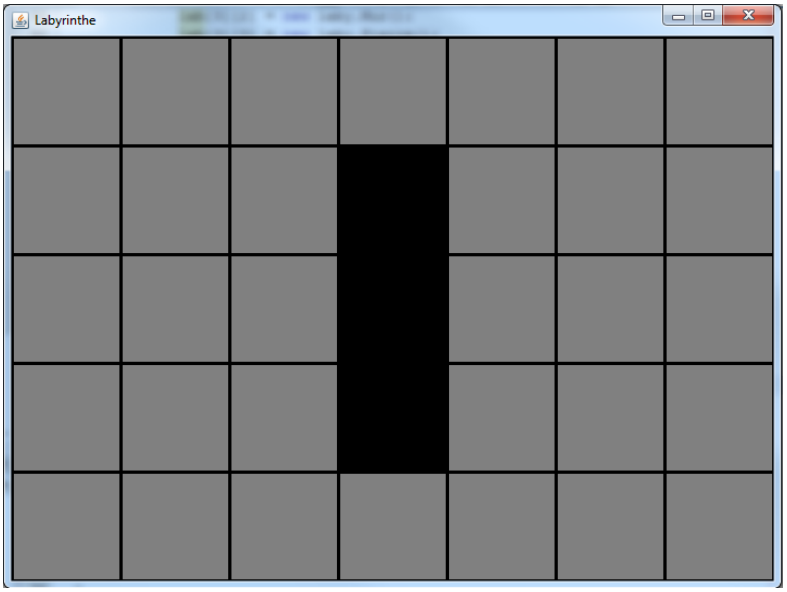

FIGURE 1 – Topologie de la carte

2 Algorithme A*

Notre objectif est de trouver le chemin le plus court sur une carte, entre une case de départ, et une case d'arrivée. Nous modéliserons un `Chemin` comme étant une liste de `Case`. Pour afficher un chemin, rajoutez les méthodes suivantes :

- `PanneauCarte(Carte carte, Chemin ch)` dans la classe `PanneauCarte` ;
- `void AfficherChemin(Carte carte, Chemin ch)` dans la classe `FenetreCarte` ;
- `void AfficherChemin(Chemin ch)` dans la classe `Carte` ;
- On modifiera également la fonction `void paint(Graphics g)` de `PanneauCarte`.

Pour trouver le chemin le plus court entre deux cases d'une carte, nous utiliserons l'algorithme A*.

A* : comment ça marche ?

Partons d'une situation initiale (figure 2). Des murs séparent le départ de l'arrivée.

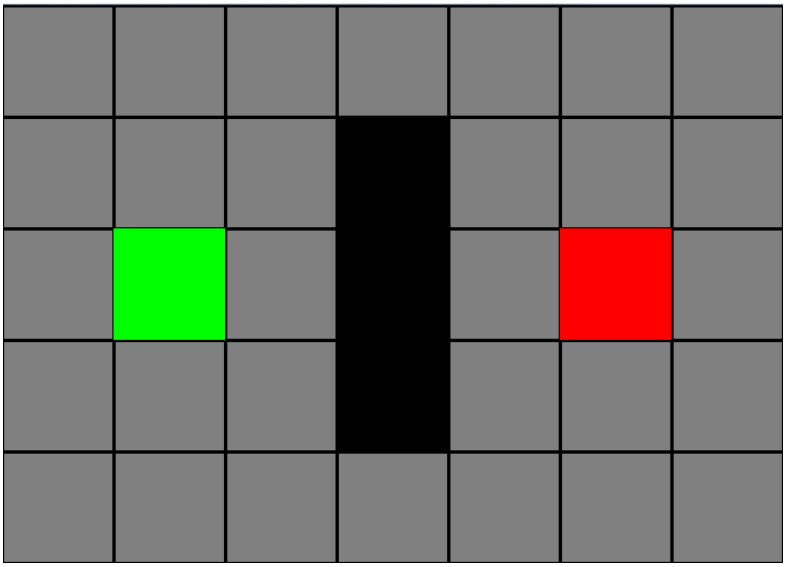

FIGURE 2 – Situation initiale

Le principe de A* est de partir de la case de départ, de visiter à chaque étape la case qui nous semble la plus intéressante, d'observer ses voisins, et de recommencer jusqu'à ce qu'on arrive sur la case d'arrivée. Pour cela, nous utilisons deux listes, que nous appelons *liste ouverte* et *liste fermée*. La liste ouverte peut être vue comme l'ensemble des cases que nous pouvons visiter à la prochaine étape, tandis que la liste fermée est l'ensemble des cases déjà visitées. Commençons par insérer la case de départ dans la liste ouverte. Pour l'instant, celle liste ne contient que la case départ. Commençons par les instructions suivantes :

1. Retirons de la liste ouverte la case A de départ. Ajoutons cette case A à la liste fermée.
2. Considérons maintenant toutes les cases valides (donc en excluant les murs) adjacentes à la case A, et ajoutons-les à la liste ouverte. Sauvegardons le fait que A est la *case parent* de ces cases.

À ce point, nous avons la situation suivante (figure 3).
Maintenant, il nous faut choisir et visiter l'une de ces cases, et répéter plus ou moins le processus. Mais quelle case choisir ?

Score et heuristique

Nous allons choisir dans la liste ouverte la case qui a le plus petit score (appelé coût F). Le coût F d'une case C de la liste ouverte est calculée de la manière suivante : $F(C) = G(C) + H(C)$, où :

- $G(C)$ est le coût "dépensé" pour arriver jusqu'à la case C,
- $H(C)$ est le coût "estimé" nécessaire pour aller de la case C, à la case d'arrivée. H est appelé *heuristique* de l'algorithme. Bien entendu, nous ne pouvons pas savoir, avant de trouver le chemin, quel va être le coût nécessaire. Il s'agit donc d'une estimation.

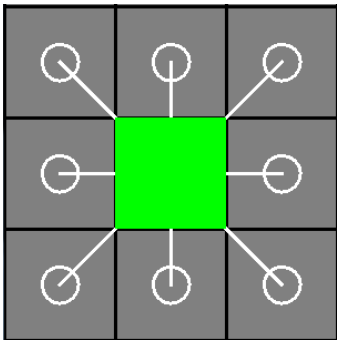

FIGURE 3 – Voisins du départ

À chaque étape de l'algorithme, nous allons visiter la case de la liste ouverte ayant le plus petit coût F. Nous rajouterons ses voisins à la liste ouverte, mais sans enlever les cases qui y sont déjà. Regardons plus précisément comment calculer le coût F.

Comme nous l'avons vu, G est la coût réellement dépensé pour arriver jusqu'à C. Décidons, pour simplifier les calculs, que se déplacer d'une case à une case adjacente coûte 10 s'il s'agit d'un mouvement horizontal ou vertical, ou 14 s'il s'agit d'un mouvement en diagonal. Pourquoi 14 ? Tout simplement parce que la diagonale d'un carré de côté 1 mesure $\sqrt{2}$, soit environ 1.41. Si nous notons P la case parent de C, nous avons alors $G(C) = G(P) + d(P, C)$, où $d(P, C)$ vaut soit 10, soit 14.

H peut être calculé de différentes manières. Nous pouvons par exemple utiliser la distance de Manhattan entre C et la case d'arrivée. Nous multiplions cette distance par 10, pour rester cohérent avec le calcul de G.

Dans la figure 4, F, G, et H sont indiquées pour chaque case de la liste ouverte.

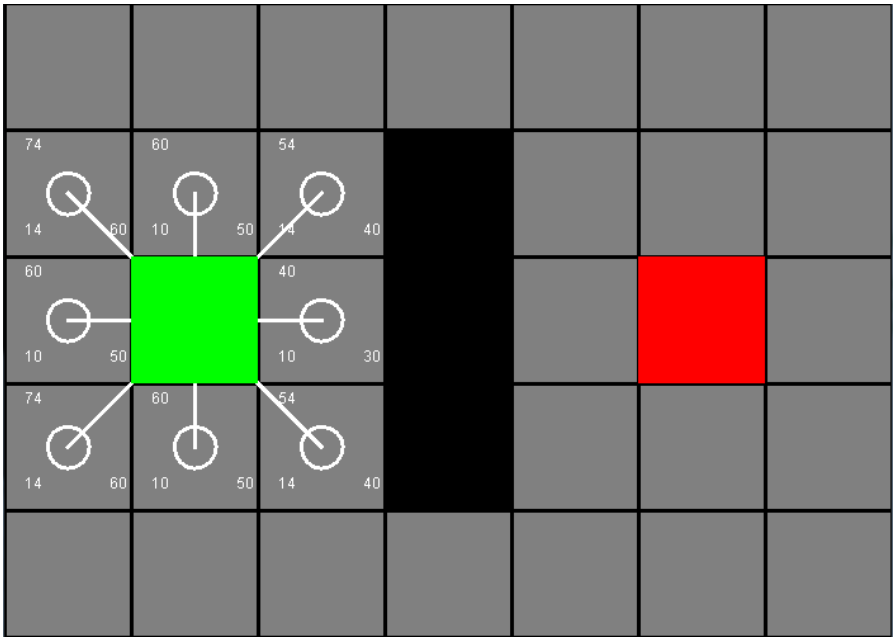

FIGURE 4 – Scores

Pour l'étape suivante de l'algorithme, nous considérons la case C de la liste ouverte avec le plus petit coût F (dans l'exemple, il s'agit de la case à droite de la case départ). Nous suivons ensuite ces étapes :

1. Enlevons C de la liste ouverte, et rajoutons-la dans la liste fermée.
2. Pour chaque voisin valide V de C, ajoutons V à la liste ouverte s'il n'appartient pas encore à la liste ouverte. Sauvegardons C comme étant la case parent de V. Un voisin est valide si ce n'est pas un mur, et s'il n'est pas déjà dans la liste fermée.

3. Si V appartient déjà à la liste ouverte, comparons l'ancien et le nouveau coût G de V. Si l'ancien est le plus petit, alors nous ne faisons rien. Dans le cas contraire, nous remplaçons le coût G (et donc le coût F) de V, et nous changeons également sa case parent, qui devient C.

Nous recommençons ensuite ces étapes jusqu'à ce que la case arrivée soit ajoutée à la liste fermée. Nous pouvons alors reconstruire le chemin.

Reconstruction du chemin

À partir de la liste fermée, nous pouvons reconstruire le chemin à l'envers. En partant de la case arrivée, nous ajoutons son parent au chemin, puis le parent du parent, puis ..., et ce jusqu'à retrouver la case de départ.

Résumé

Une fois compris, l'algorithme A* est finalement assez simple :

1. Ajouter la case de départ à la liste ouverte.
2. Répéter :
   (a) Soit C la case de la liste ouverte avec le plus petit coût F.
   (b) Enlever C de la liste ouverte, et la rajouter dans la liste fermée.
   (c) Pour chaque case adjacente V de C :
   - Si V est un mur, ou si V est déjà dans la liste fermée, ne rien faire.
   - Sinon, si V n'est pas encore dans la liste ouverte, l'ajouter. Calculer les coûts de V, et enregistrer C comme case parent de V.
   - Si V appartient déjà à la liste ouverte, comparer les coûts G et ne garder que la version de V avec le coût G le plus petit.

   (d) S'arrêter :
   - si la case d'arrivée n'est pas trouvée, et que la liste ouverte est vide. Dans ce cas, il n'y a pas de chemin possible.
   - si la case d'arrivée est ajoutée à la liste fermée. Dans ce cas, on reconstruit le chemin.

La figure 5 montre le résultat de l'algorithme. Les cases de la liste fermée sont en orange, celles de la liste ouverte sont en blanc.

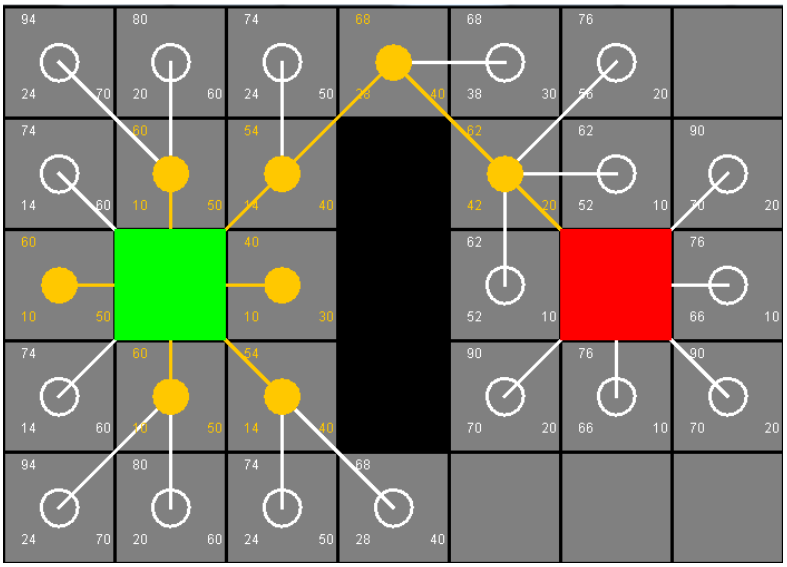

FIGURE 5 – Scores