Projet : Roborally

1 Introduction

Roborally est un jeu de société élaboré par Richard Garfield en 1994. Le but de ce projet est de créer un programme capable de jouer de manière autonome à une version simplifiée du jeu. Chaque joueur incarne un petit robot qui doit trouver son chemin dans une usine. L'usine est parsemée de tapis roulants et de plateaux tournants qui vont perturber les déplacements du robot. Roborally est une course, et le but est donc de se déplacer le plus rapidement possible. Ce projet a pour but de vous faire appliquer les notions sur les graphes et les algorithmes de plus court chemin associés pour calculer automatiquement les meilleures séquences de coups possibles pour déplacer le robot.

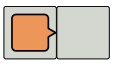
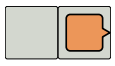
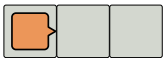
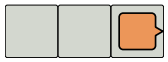
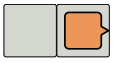
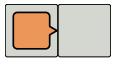
En terminant la première partie de ce projet, vous devriez normalement être capable de formuler un problème de plus court chemin à l'aide de la théorie des graphes, et d'appliquer l'algorithme de Dijkstra pour obtenir une solution à ce problème. Il est important que l'algorithme utilisé soit bien l'algorithme de Dijkstra et non une variante de votre cru.

En abordant la seconde partie, vous pourrez faire preuve de créativité pour définir la stratégie du robot dans des situations de jeu plus complexes. Vous serez amenés à vous questionner sur les performances de votre solution, en terme de complexité spatiale et temporelle et en terme de qualité du résultat obtenu.

2 Règles du jeu

2.1 Robot

Votre robot est défini par une position et une orientation. Pour déplacer le robot, sept actions sont possibles : avancer de une, deux ou trois cases, reculer de une case, tourner à gauche ou a droite, faire demi-tour. La direction dans laquelle le robot avance ou recule est déterminée par son orientation. Tourner le robot permet de changer son orientation. Les effets des différentes actions sont illustrés ci-dessous.

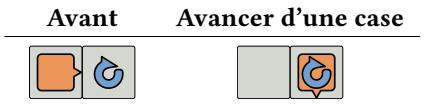
Action	**Avant**	**Après**
Avancer d'une case		
Avancer de deux cases		
Avancer de trois cases		
Reculer d'une case		
Tourner à gauche		
Tourner à droite		
Faire demi tour		

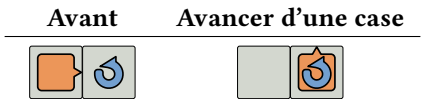
Lorsque le robot sort du plateau de jeu, il est détruit et ne peut plus faire la moindre action.

2.2 Plateau de jeu

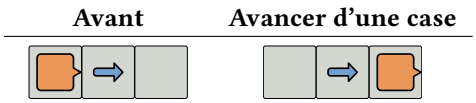
Après chaque déplacement du robot, les éléments du plateau sont mis en œuvre. Ces éléments peuvent être des tapis roulants, des tapis roulants rapides ou des plateaux tournants. Il ne vous est normalement pas nécessaire de comprendre dans le détail comment ces éléments fonctionnent, car le code fourni comprend un programme qui traite ces éléments pour vous. Dans tous les exemples fournis, le robot est placé dans une situation initiale, puis il est illustré dans son état après avoir effectué l'action « avancer d'une case » puis avoir appliqué les éléments du plateau.

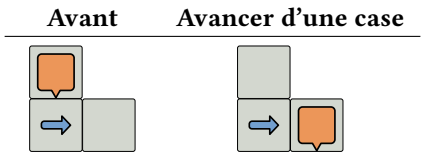
Dans l'ordre, les tapis roulants sont appliqués, puis les plateaux tournants. Les plateaux tournants font tourner votre robot, vers la droite ou vers la gauche selon leur type.



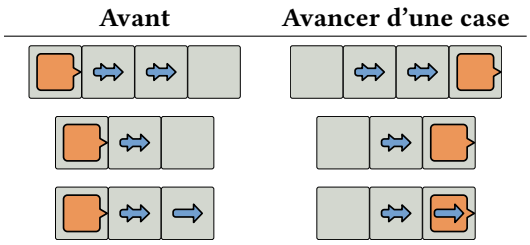


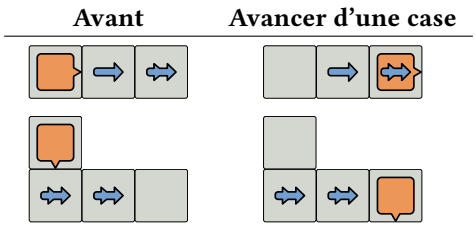
Les tapis roulants sont représentés par une simple flèche, et font avancer le robot d'une case dans la direction de la flèche.



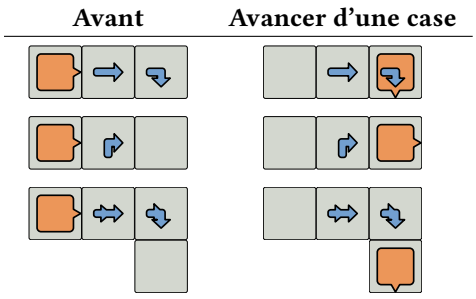


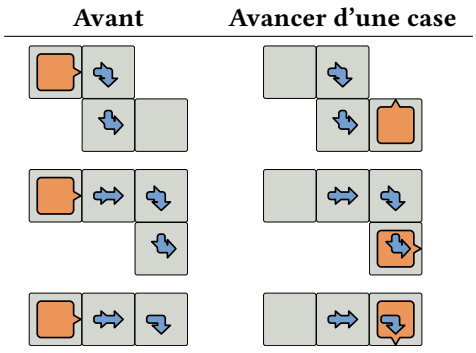
Les tapis roulants rapides sont représentés par une double flèche, et font avancer le robot de deux cases d'un coup, tant qu'il reste sur un tapis roulant rapide.



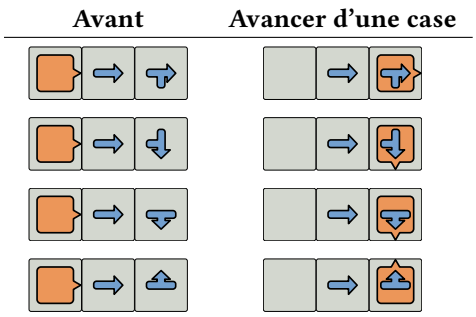


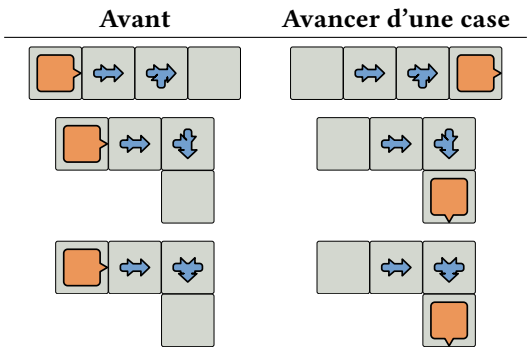
Certains tapis roulants peuvent également faire tourner le robot. Lorsque c'est le cas, un robot n'est tourné que s'il arrive sur la case suite à l'effet d'une autre case tapis roulant.





Il est également possible que certains tapis roulants se rejoignent. Les cases permettant de joindre des tapis roulants sont à considérer différemment en fonction de la direction via laquelle le robot arrive.





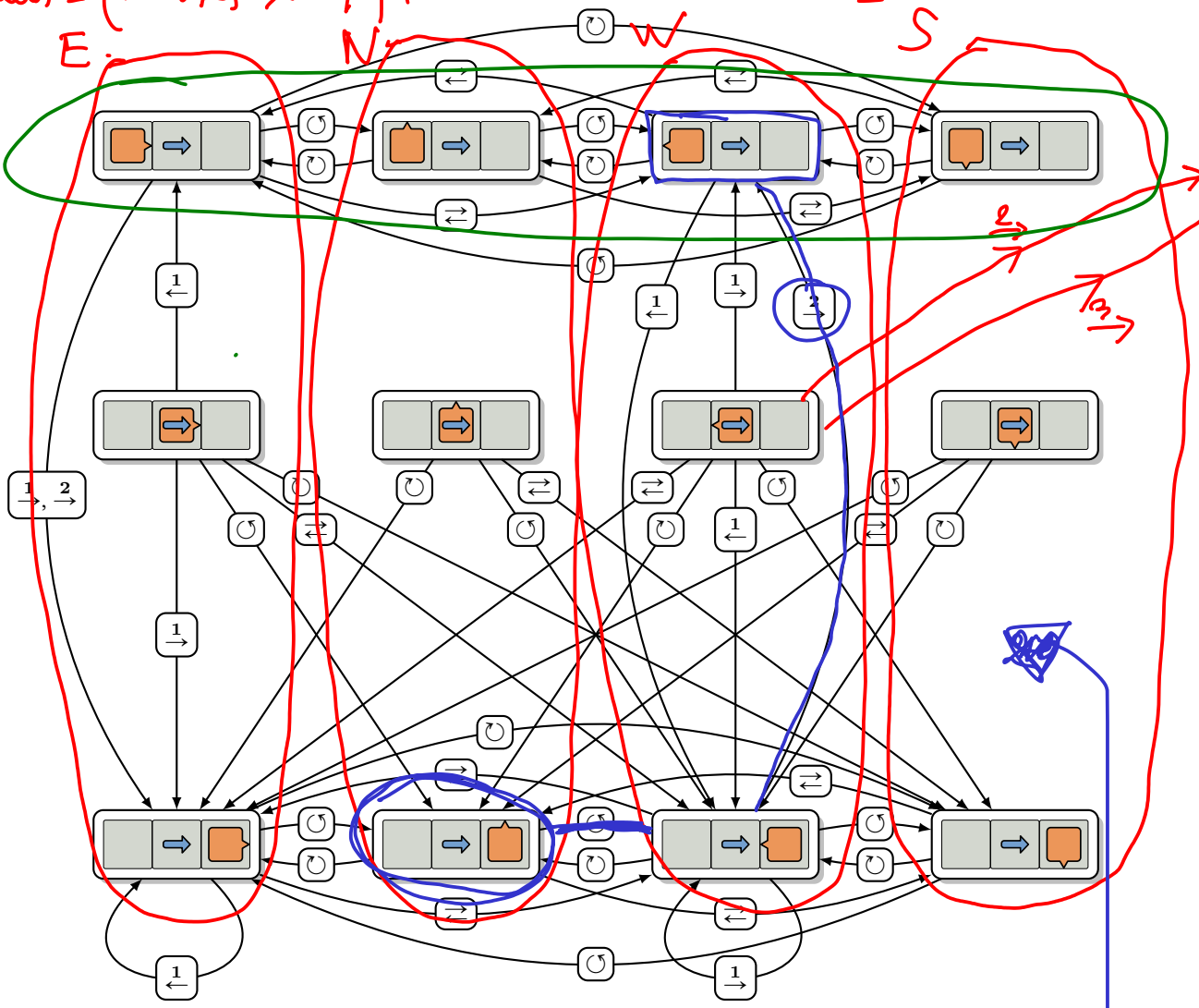
2.3 Déroulement du jeu

En début de partie, le robot d'un joueur est placé sur le plateau. Une autre case du plateau est définie comme but à atteindre. L'objectif du joueur est d'amener son robot le plus rapidement possible sur la case objectif.

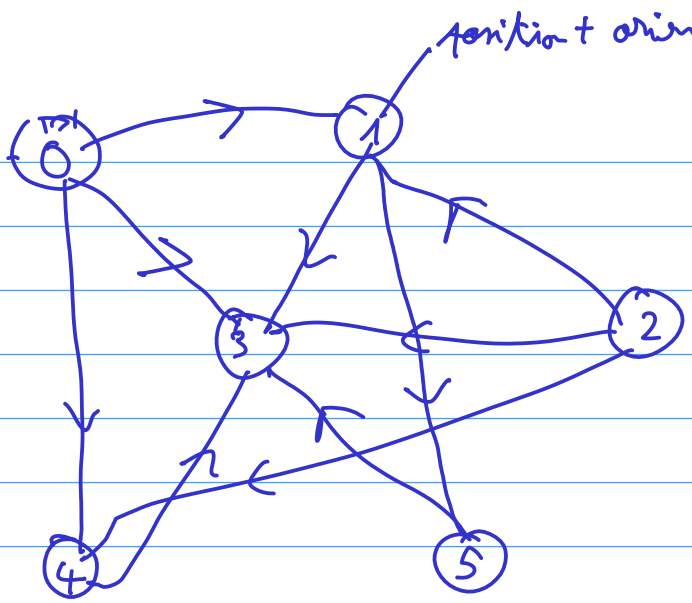
À chaque tour de jeu, un joueur reçoit neuf cartes prises au hasard dans un paquet de cartes. Chaque carte permet de réaliser l'une des sept actions décrites en section 2.1. Le joueur choisit cinq cartes parmis les neuf et les place dans l'ordre de son choix pour déterminer la séquence d'actions réalisée par son robot lors du tour.

3 Modélisation du problème

Nous vous proposons de modéliser le problème sous la forme d'un graphe. Chaque nœud du graphe correspond à une position et une orientation possible du robot. Un nœud de ce graphe possède sept voisins : un pour chaque action de jeu possible, avancer d'une, deux ou trois cases, tourner, etc. En suivant une arête, le nœud obtenu correspond à la position et à l'orientation du robot une fois avoir appliqué l'action, puis appliqué les tapis roulants et plaques tournantes. Un tel graphe est illustré ci-dessous. Pour plus de clarté, les arêtes correspondant à des actions menant à la destruction du robot ne sont pas indiquées. Il conviendrait donc sur ce graphe de rajouter un nœud « robot détruit » et de rediriger vers ce nœud toutes les arêtes sortantes absentes du graphe ci-dessous. Les sept voisins de ce nœud mènent vers lui-même.



$\overset{1}{\rightarrow}, \overset{2}{\rightarrow}, \overset{3}{\rightarrow}$: avancer $\overset{1}{\leftarrow}$: reculer $\circlearrowleft, \circlearrowright$: tourner $\rightleftarrows$: faire demi tour



listes d'adjacences

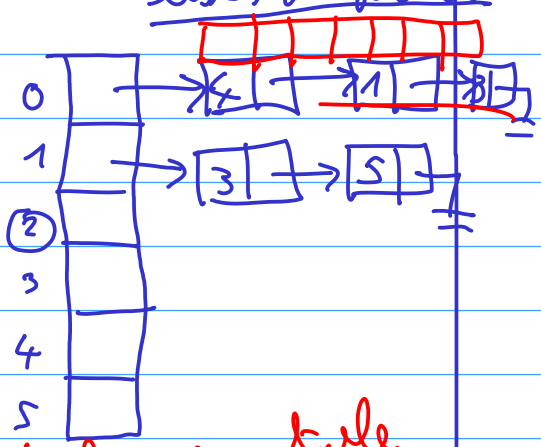


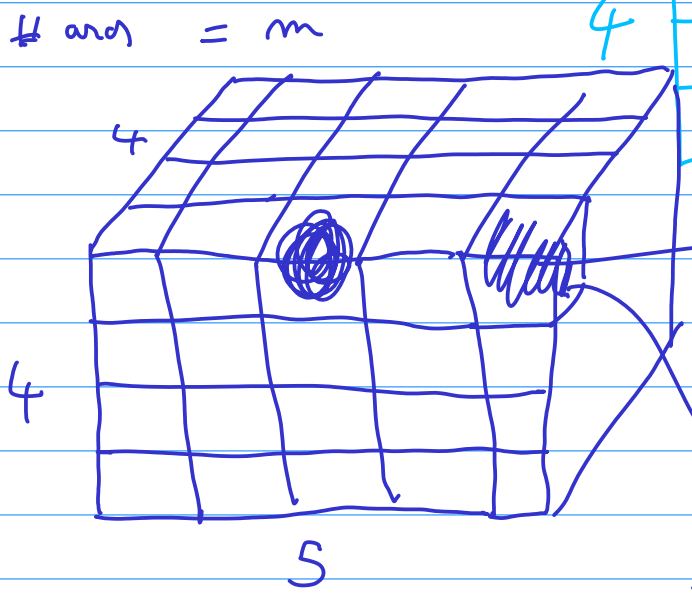
tableau ou table de hash

5

0 . . n-1

sommets = n

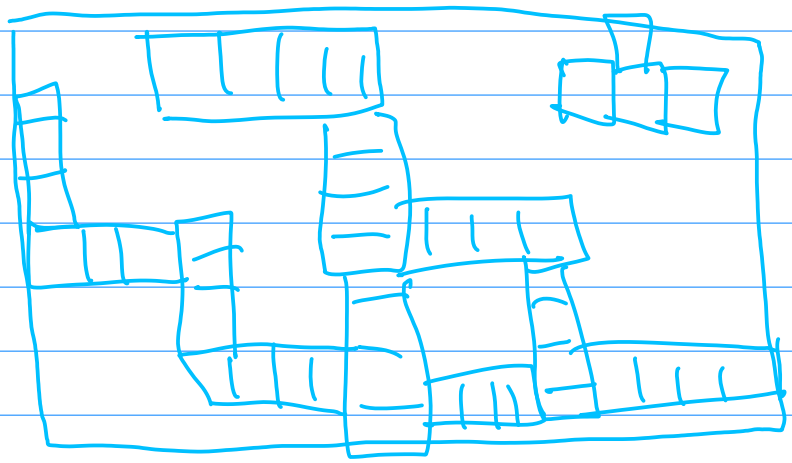
arcs = m

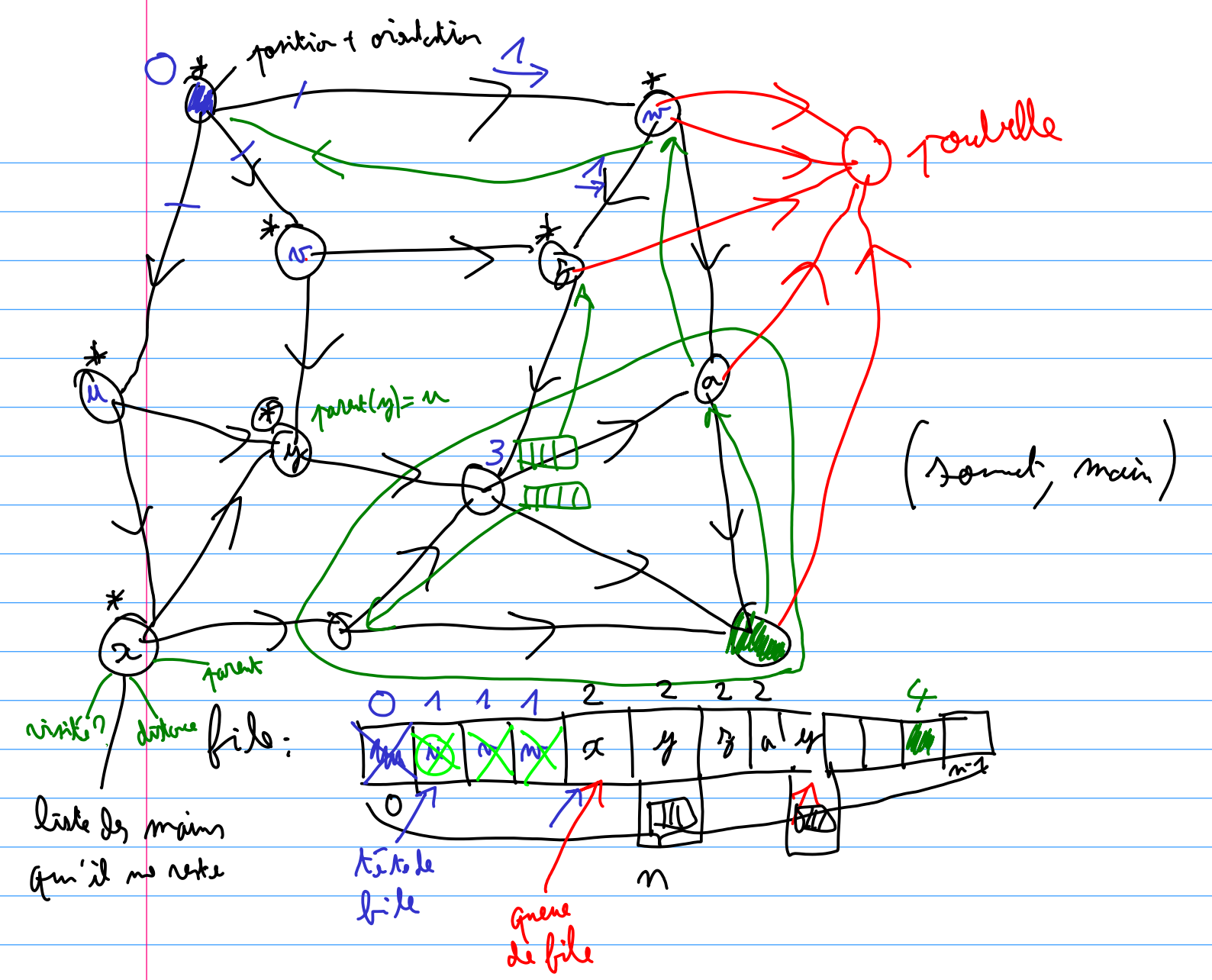


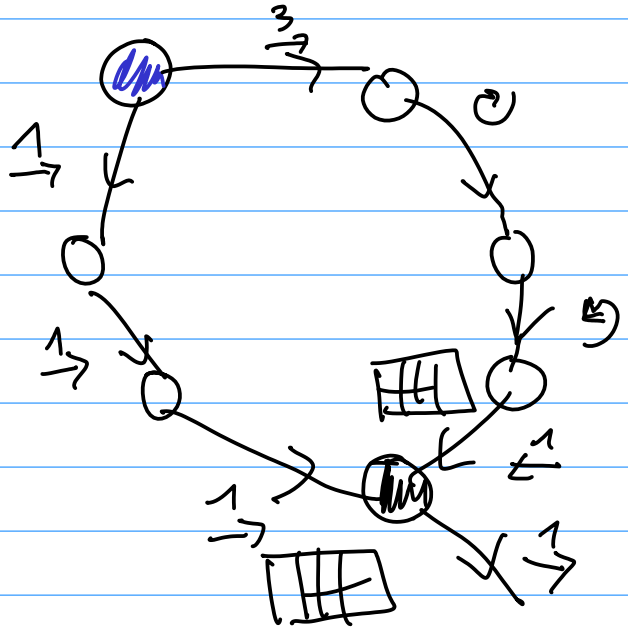
	0	1	2	3
4	5			6
2	...			
...			14	15

une cellule de cette grille matrice 3D = sommet de ce (ligne, colonne, orientation)

tableau de taille 7 ~~liste~~ stocke la liste des sommets voisins dans le graphe de jeu (il y en a 7)




position + orientation
nouvelle
parent(y) = u
3
(sommet, main)
parent
visité ?
distance
file :
liste des mains qu'il me reste
0 1 1 1 2 2 2 2 4
n-1
tête de file
queue de file
n


3
1
1
1
1

4 Matériel fourni

4.1 Gestion du plateau de jeu

Le code fourni avec ce projet contient les outils pour charger un plateau de jeu, créer un robot, et jouer des coups. Vous n'avez donc pas à comprendre le détail des règles de déplacement du robot, il vous suffit d'utiliser la méthode "play" de la classe "Board".

Robot est la classe permettant de modéliser un robot. Elle contient une position, encodée sous la forme de deux coordonnées ligne et colonne, et un status, qui indique l'orientation du robot, ou s'il est mort.

Board est la classe permettant de gérer le plateau. Le constructeur permet de charger un plateau à partir d'un fichier texte, un exemple vous est fourni via le fichier "board.txt". Le plateau peut également être sauvé ou exporté en json. La méthode qui vous intéresse le plus est la méthode "play" qui vous donnera étant donné un robot et un coup à jouer la position du robot après avoir joué le coup. C'est grâce à cette méthode que vous construirez votre graphe.

4.2 Application interactive

Le fichier "app.cpp" contient une petite application pour visualiser des plateaux, placer un robot, et jouer des coups. Une fois compilée, lancez la en fournissant en paramètre le fichier plateau à utiliser, et ouvrez votre navigateur web à l'adresse `http://localhost:8080`. Si tout fonctionne bien, vous devriez voir un plateau, pouvoir cliquer dessus pour placer un robot, et utiliser les boutons en dessous pour jouer des coups.

5 Consignes

5.1 Structures de données

Vous commencerez par élaborer une structure de données adaptée pour stocker le graphe de jeu. Votre structure devra donc stocker les différents noeuds possible du graphe de jeu, et les arêtes les reliant. Pour déterminer les arêtes, vous utiliserez la méthode "play" de la classe Board, en initialisant un robot selon le noeud de départ du graphe, et en lui appliquant un coup pour trouver le noeud voisin correspondant à ce coup.

Éléments clés évalués :

- pertinence de la structure de données
  - quel est l'espace mémoire requis en fonction du nombre de cases ?
  - quelles sont les opérations pour lesquelles elle est conçue ?
  - quelle est la complexité de ces opérations ?
- algorithme de construction
  - quelle est la complexité de l'initialisation ?
  - tous les plateaux sont ils gérés ? Même s'ils sont vides ou ont plusieurs morceaux ?
  - la construction a-t-elle été testée sur des plateaux simples ?
- propreté de la gestion de la mémoire
  - votre code a-t-il des fuites de mémoire ?
  - avez vous testé s'il a des fuites, localisé et tenté de résoudre le problème ?

5.2 Plus court chemin dans un graphe

À partir du graphe de jeu, d'une position initiale du robot (position et orientation) et d'une position d'arrivée, écrivez un algorithme permettant de calculer la suite de coups minimale pour passer de la position initiale à la position d'arrivée. Pouvez-vous également calculer la séquence de coups minimale pour rejoindre une case donnée (quelle que soit l'orientation du robot à l'arrivée) ?

Éléments clés évalués :

- correction de l'algorithme
  - la séquence de coups obtenue est-elle bien la plus courte ?
  - la solution obtenue a-t-elle été testée sur des cas simples ?
- mise en œuvre de vos connaissances et complexité
  - avez vous utilisé un algorithme classique ? Lequel ?
  - quelles structures de données ont été utilisées ?
  - quel est la complexité instructions / mémoire de l'algorithme ?

5.3 Joueur artificiel

En utilisant votre algorithme de recherche de plus court chemin (ou un autre que vous jugerez plus approprié), élaborez un programme permettant de sélectionner les actions à réaliser à chaque tour de jeu, comme décrit en section 2.3. Votre programme prendra en paramètre le graphe de jeu, l'état actuel du robot et les neuf actions qui lui sont possibles sous forme d'un tableau. Il calculera une séquence de cinq actions à jouer parmis les neuf disponibles. Le résultat sera fourni en réordonnant le tableau de 9 actions fourni en entrée pour que les cinq actions choisies figurent dans l'ordre au début du tableau.

Notez qu'il s'agit d'une question ouverte, et que nous n'attendons pas une solution en particulier. Vous écrirez également les fonctions permettant de tester votre programme.

Éléments clés évalués :

- capacité à trouver une solution au problème
  - la solution proposée amène-t-elle le robot à l'objectif si le tirage n'est pas trop mauvais ?
  - quelle est la complexité de l'algorithme mis en œuvre ?

5.4 Pour aller plus loin

Si vous en avez le temps, nous vous proposons plusieurs idées pour aller plus loin sur ce projet. Cette partie n'est à traiter qu'une fois le reste du projet fonctionnel, et ne donnera lieu qu'à des points bonus.

5.4.1 Prise en compte de la rareté des actions

Dans le jeu originel, lorsque le joueur tire les neufs cartes actions qu'il aura à disposition pour son tour, toutes les actions n'ont pas la même chance d'être tirée. Dans le paquet de cartes, il y a plus de cartes « avancer de 1 » que de cartes « avancer de 3 ». Étant donné les probabilités de tirer chaque action, vous pouvez tenter d'améliorer vos algorithmes pour les prendre en compte.

5.4.2 Mise en compétition des robots

Pour comparer vos stratégies à celles des autres binômes, vous pouvez utiliser la programmation générique afin d'écrire un programme qui puisse facilement intégrer les structures de données et fonctions élaborées par les autres groupes. Vous pourrez ainsi mettre en concurrence plusieurs stratégies sur le même tirage, et regarder le nombre de tours nécessaires pour arriver à l'objectif, ainsi que le temps d'exécution nécessaire pour y parvenir.

5.4.3 Pièges et points de vie

Le jeu initial ajoute également des cases piégées sur le plateau. Lorsqu'un robot s'y arrête, il perd un point de vie. Le nombre de points de vie initial est neuf : autant que de cartes tirées en début de manche. La perte d'un point de vie fait que le joueur tirera une carte de moins en début de tour. Lorsqu'il reste moins de cinq points de vie, les dernières actions de la séquence du robot sont bloquées : l'action choisie lorsque le point de vie a été perdu ne pourra plus être changée. Un robot n'ayant que trois points de vie ne recevra donc que trois cartes qu'il placera dans l'ordre de son choix en début de séquence, et ses deux derniers coups seront déterminés par les actions bloquées. Vous pouvez donc modifier le programme pour gérer la santé du robot et l'ensemble de pièges sur le plateau. Ces modifications pourront vous amener à chercher d'autres stratégies pour le robot.

5.4.4 Personnalisation du plateau

L'apparence des éléments de jeu est entièrement définie par le fichier svg fourni via le paramètre « style » des fonctions des fichiers board_display. En modifiant ce fichier, vous pouvez choisir une autre apparence pour le plateau et les éléments de jeu. À vous de voir comment le fichier est construit et utilisé dans le code d'affichage, pour pouvoir proposer votre propre fichier de style.

5.5 Modalités de rendu et évaluation

5.5.1 Composition et envoi de l'archive

Votre projet sera à déposer sous forme d'une archive sur TOMUSS. Il devra être facilemenent compilable par votre correcteur. Un fichier "readme" fournira les éléments clés pour sa prise en main. À l'issue du projet, un entretien oral sera réalisé avec votre correcteur pour évaluer votre maîtrise du code rendu.

S'il reste des bugs ou des fuites mémoire dans votre code, il sera pertinent de le mentionner, ainsi que les informations que vous avez : savoir que vous avez repéré et analysé un bug est une information importante pour l'évaluation de votre projet.