UNIVERSITÉ
CAEN
NORMANDIE

- Soutenance de projet -

Solveur de Ricochet Robots

ANDRÉ Lorada
AUVRAY Théo

L2 Informatique
Groupe 1A

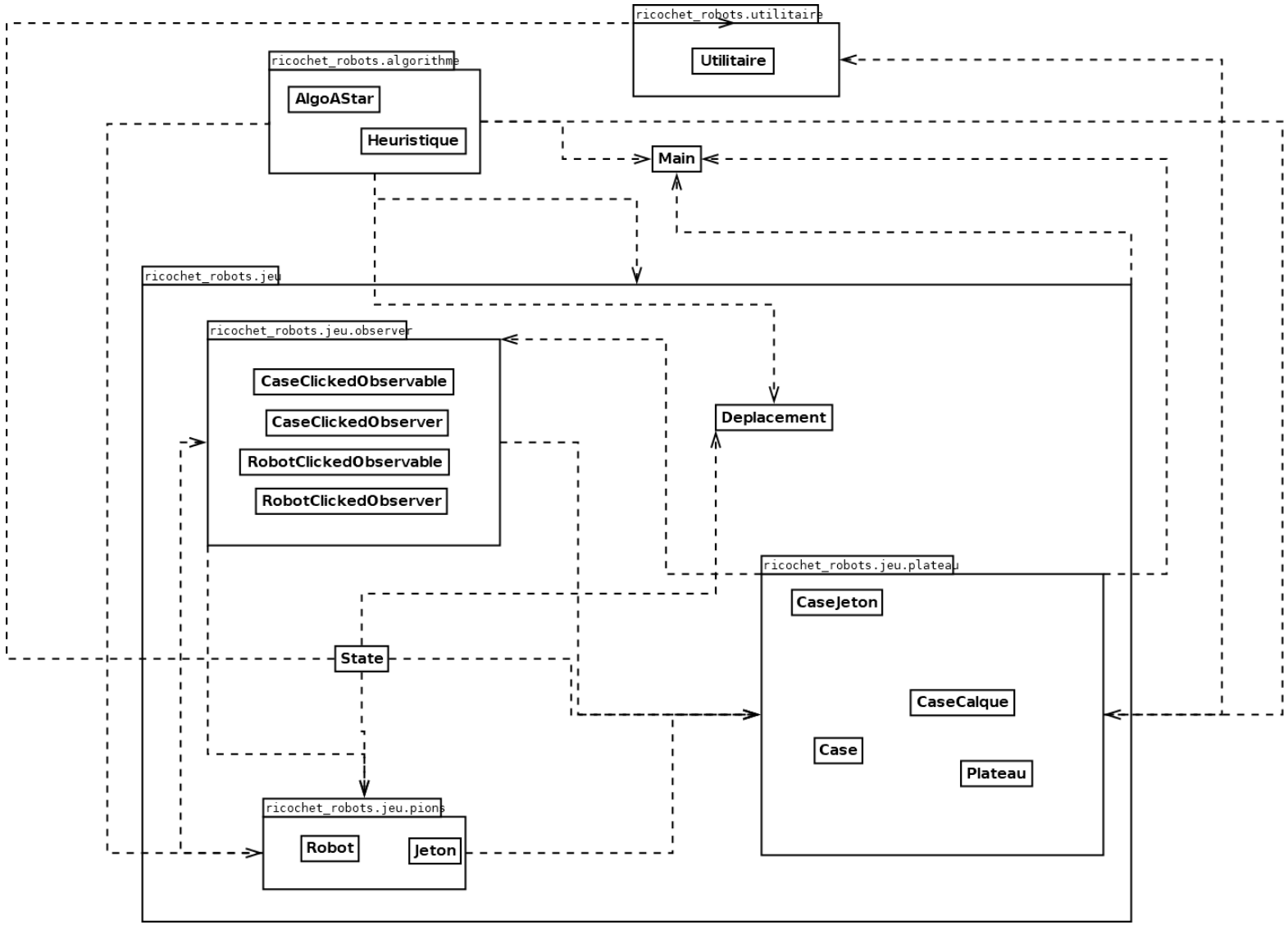
3 mai 2020

Sommaire

Introduction

- Le Ricochet Robots : jeu de société composé d'un plateau, de pions de robots et de jetons.
- Principe : Déplacer les robots pour atteindre la case qui correspond au symbole du jeton ayant été tiré aléatoirement. Le déplacement des robots s'effectue qu'en ligne droite jusqu'à rencontrer un obstacle (un robot ou un mur).
- Objectifs :
  - Conception du Ricochet Robots.
  - Réalisation d'une interface graphique.
  - Implantation de l'algorithme A*, optimisation de l'algorithme.

Organisation du projet



Composition du plateau

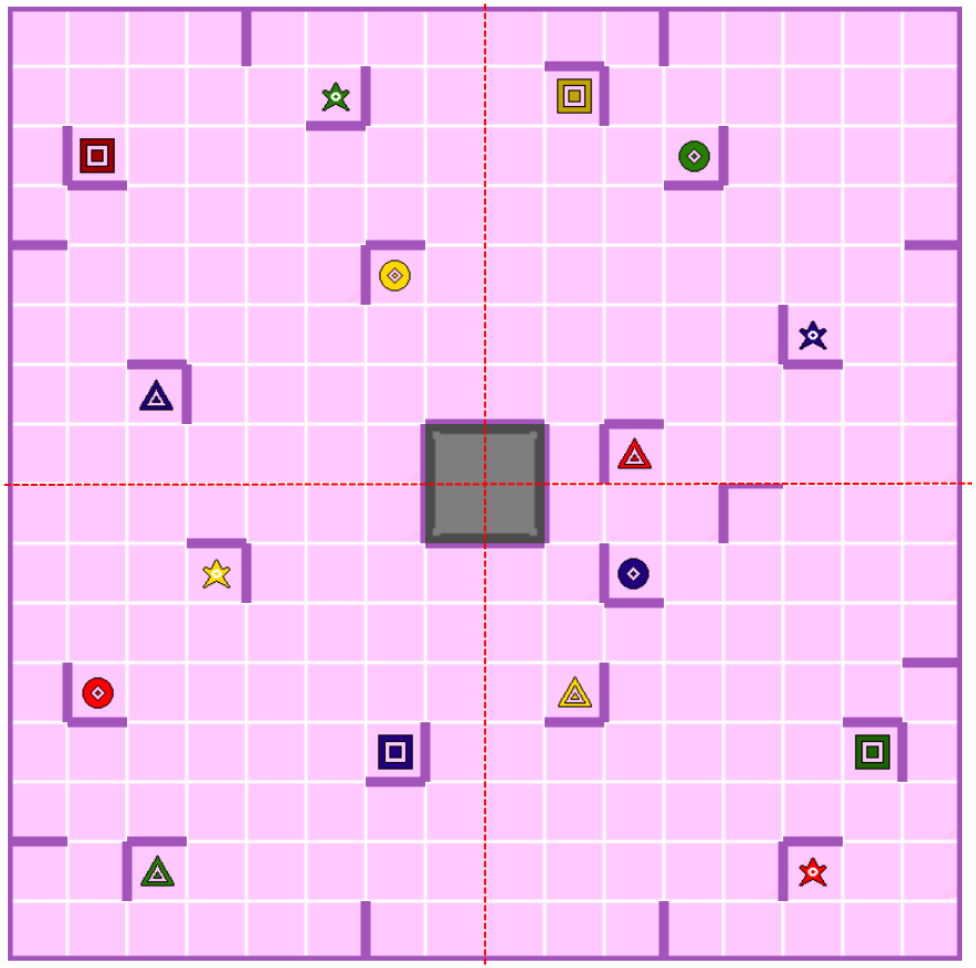


Figure – Liste des différents robots

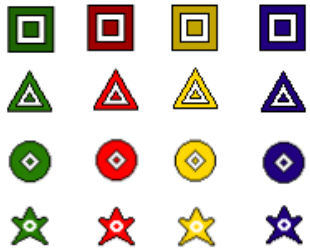
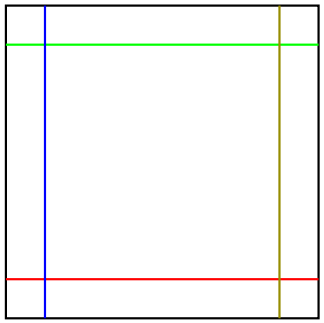


Figure – Liste des différents jetons

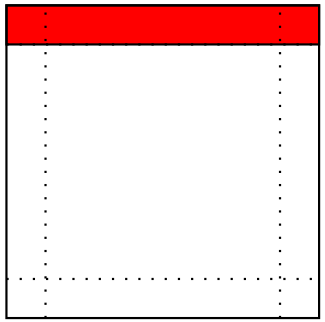
Figure – Liste des différentes cases

Composition des cases

HAUT

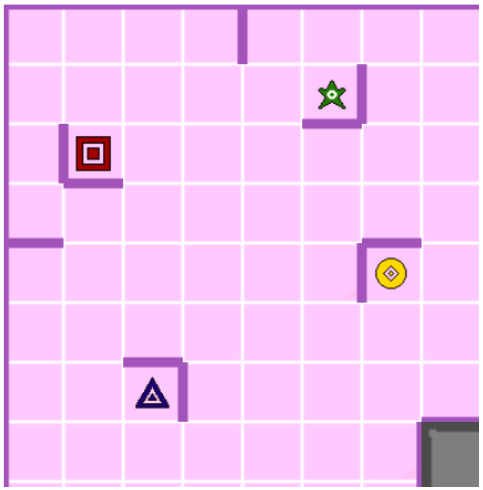
DROIT  GAUCHE [HAUT, DROIT, BAS, GAUCHE]

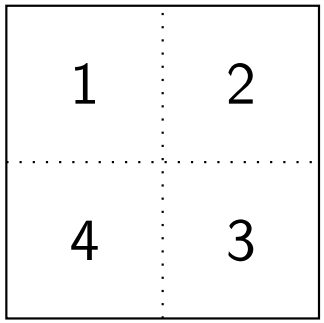
BAS

 ⟶ [True, False, False, False] ⟶ [1,0,0,0]

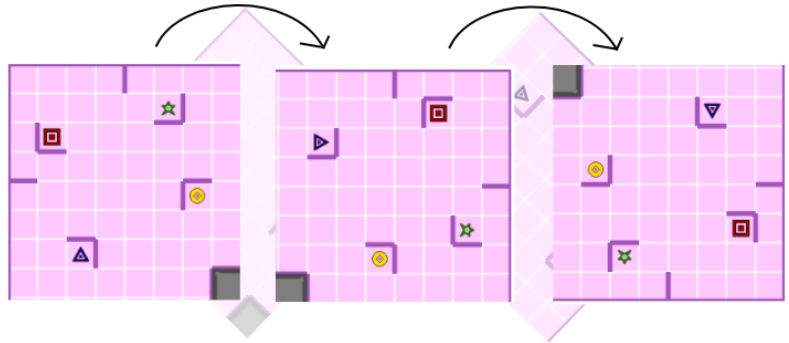
Positionnement des mini-plateaux

On affecte par exemple à ce plateau la position 3.





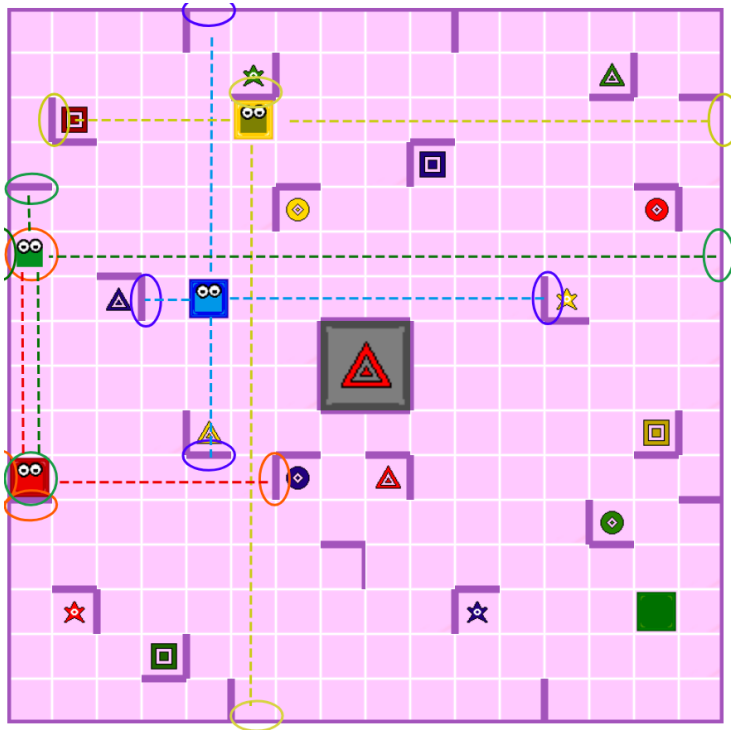
1. Application d'une rotation de 90 degrés du mini-plateau :

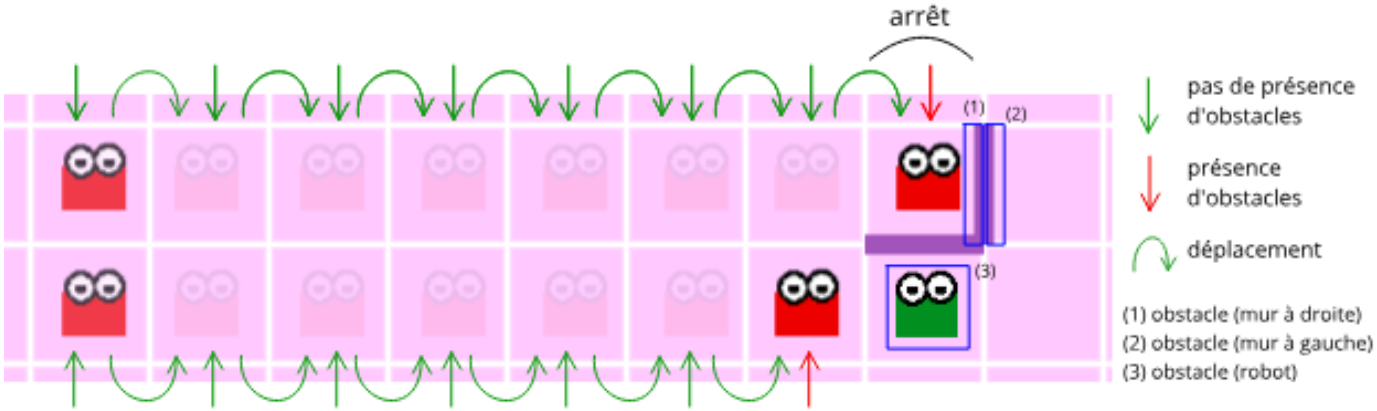


2. Application d'une rotation de 90 degrés de chaque case du mini-plateau :

[1,0,0,1] - [1,1,0,0] - [0,1,1,0] - [0,0,1,1]

Déplacements et collisions des robots





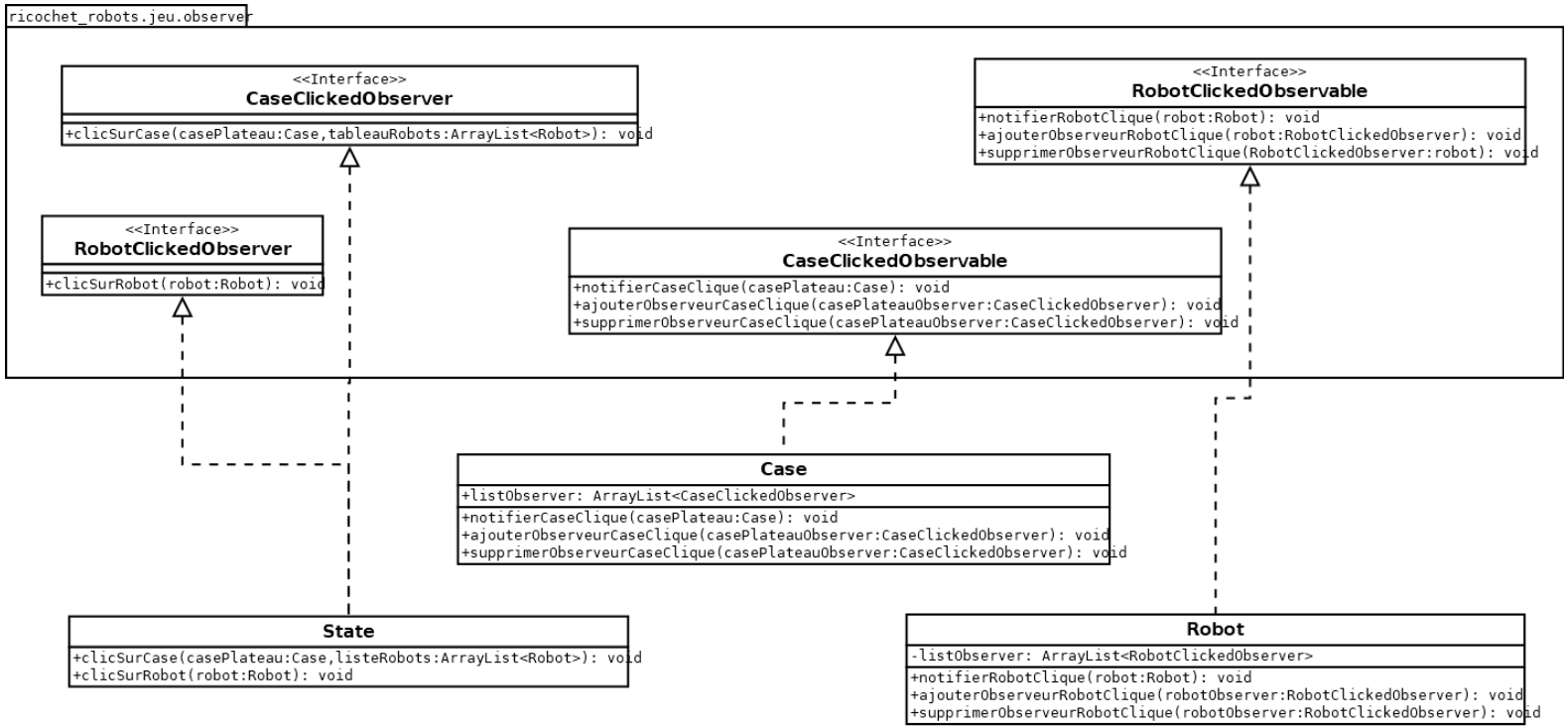
Algorithme de déplacements des robots

Algorithme 1 : Déplacement d'un robot jusqu'à collision

Entrées : La direction de déplacement

```
1  si direction == haut alors
2  |   tant que pas de mur en haut de la case actuelle et
   |     pas de mur en bas de la case suivante et
   |     !estUneCollisionRobot(direction, robot) faire
3  |   |   deplacement d'une case vers le haut
4  |   fin
5  fin
6  sinon si direction == bas alors
7  |   tant que pas de mur en bas de la case actuelle et
   |     pas de mur en haut de la case suivante et
   |     !estUneCollisionRobot(direction, robot) faire
8  |   |   deplacement d'une case vers le bas
9  |   fin
10 sinon si direction == gauche alors
11 |   tant que pas de mur a gauche de la case actuelle et
   |     pas de mur a droite de la case suivante et
   |     !estUneCollisionRobot(direction, robot) faire
12 |   |   deplacement d'une case vers la gauche
13 |   fin
14 sinon si direction == droite alors
15 |   tant que pas de mur a droite de la case actuelle et
   |     pas de mur a gauche de la case suivante et
   |     !estUneCollisionRobot(direction, robot) faire
16 |   |   deplacement d'une case vers la droite
17 |   fin
```

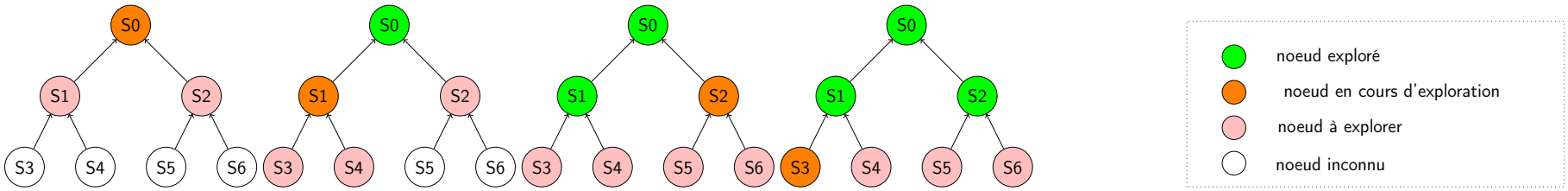
Sélection d'un robot



Première approche : l'algorithme BFS

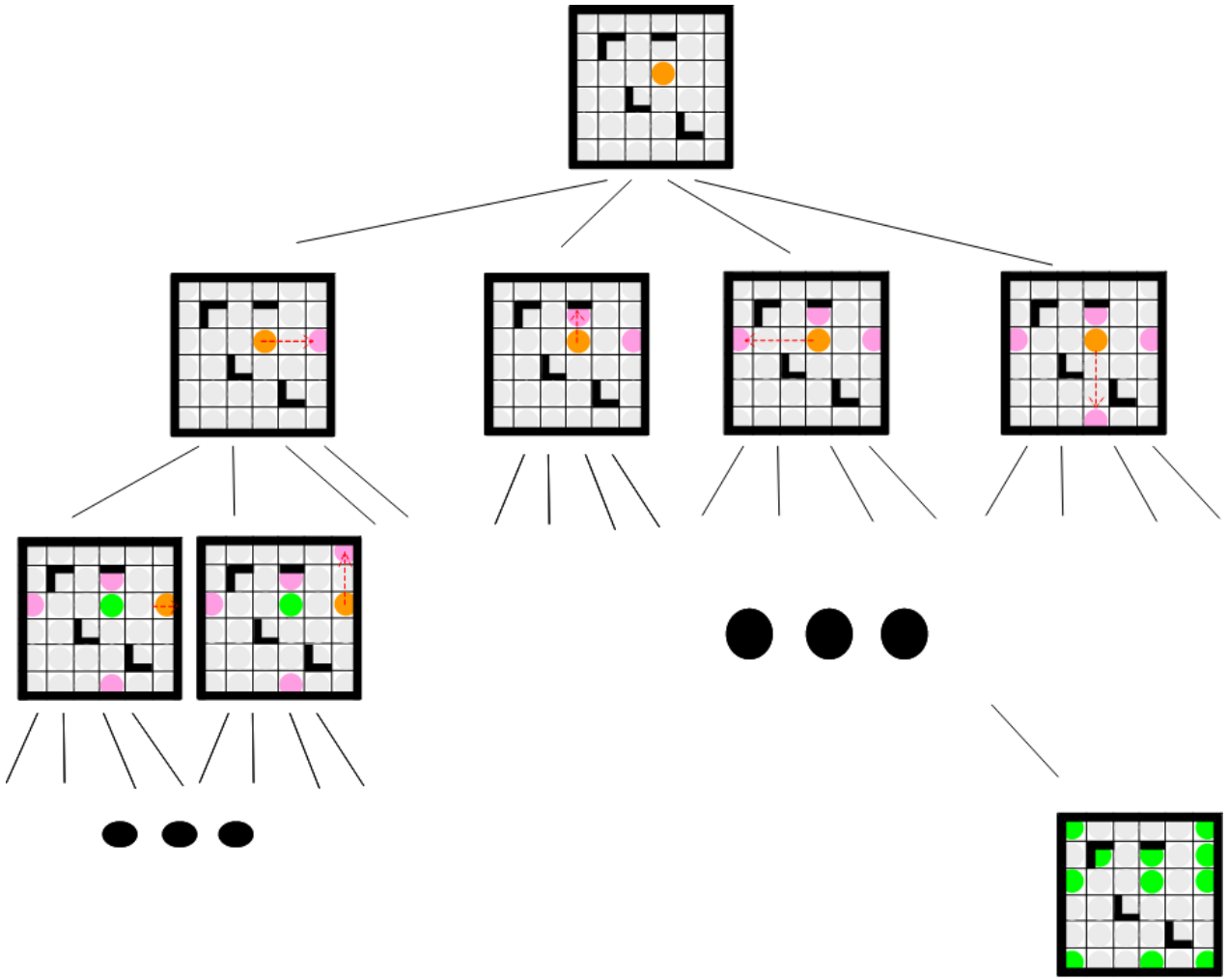
- Idée : Commencer par implémenter un algorithme de parcours de noeud et de le modifier afin de le transformer en algorithme A*.
- Avantage : Permet de bien comprendre le principe de parcours de noeud de ce type d'algorithme.

L'algorithme BFS (Breadth First Search en anglais), est un algorithme permettant le parcours d'un graphe ou d'un arbre étage par étage.



Application au Ricochet Robots

Pour de déplacement d'un seul robot :



Comparaison entre les algorithmes BFS et A*

Similarités :

- Algorithmes de parcours de noeuds, permettent la recherche de chemin.
- Le noeud suivant qui est pris correspond au premier élément d'une liste de noeuds à explorer.

Problèmes au BFS :

- La liste de noeuds à explorer contient suite à suite les voisins du noeud courant.
- Parcours long en testant tout les cas possibles.
- Parcours de noeuds inintéressant à la résolution du problème.

Solution :

- Prioriser certains coups afin atteindre plus rapidement le noeud solution.

Pourquoi une heuristique ?

- Permet d'aller plus rapidement vers la solution.
- Type d'heuristique le plus connu : heuristique du vol d'oiseau.
  - Calcule la distance du pion actuel par rapport à l'objectif
  - Priorise les états qui se rapprochent de l'objectif

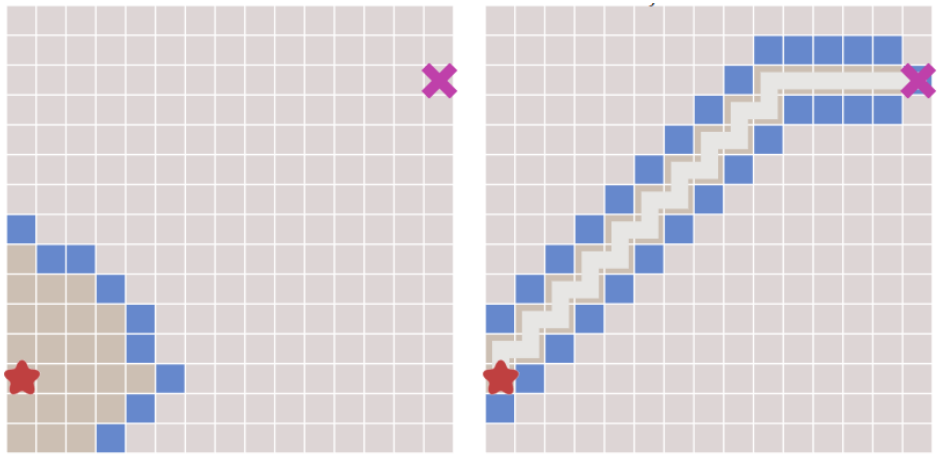


Figure – Algorithme A* avec et sans heuristique après 23 coups[1]

Cependant, cette heuristique n'est pas adaptée au Ricochet Robots.

Mise en place de l'heuristique

Figure – Exemple de carte heuristique pour le Ricochet Robots[2]

Application de l'algorithme A*

Algorithme 2 : Algorithme A*

Entrées : L'etat initial

```
1  etatInitial ← l'etat initial ;
2  PriorityQueue < State > frontier ← new PriorityQueue <> () ;
3  HashMap < State, State > cameFrom ← new HashMap <> () ;
4  HashMap < State, Integer > costSoFar ← new HashMap <> () ;
5  ArrayList < State > path ← new ArrayList <> () ;
6  ArrayList < Robot > listRobot ← new ArrayList <> () ;
7  ArrayList < Deplacement > listDeplacement ← new ArrayList <> () ;
8  etatInitial.setValCost(0) ;
9  frontier.add(etatInitial) ;
10 cameFrom.put(etatInitial, null) ;
   costSoFar.put(etatInitial, cout de l'etat initial) ;
11 tant que frontier n'est pas vide faire
12     noeudEnCours ← frontier.poll() ;
13     si objectif atteint alors
14         etatFinal ← noeudEnCours ;
15         break ;
16     fin
17     pour chaque direction faire
18         pour chaque robot faire
19             etatSuivant ←
                noeudEnCours.etatSuivant(direction, robot) ;
20             newCost ←
                cout de l'etat suivant + 1 + heuristique de l'etat suivant ;
21             si l'etat suivant est nouveau or
                le cout de l'etat suivant est plus petit que celui delaliste
                alors
22                 costSoFar.put(etatSuivant, newCost) ;
                   etatSuivant ← cout de l'etat suivant + 1 ;
                   frontier.add(etatSuivant) ;
                   cameFrom.put(etatSuivant, noeudEnCours) ;
23             fin
24         fin
25     fin
26 fin
```

```
1  current ← etatFinal ;
2  tant que current! = etatInitial and current! = null faire
3      path.add(current) ;
4      current = cameFrom.get(current) ;
5  fin
6  si l'etat actuel existe alors
7      path.add(etatInitial) ;
8      pour chaque etat du chemin faire
9          listDeplacement.add(dernier deplacement de l'etat) ;
10         listRobot.add(dernier deplacement de robot de l'etat) ;
11     fin
12     reverse(listRobot) ; reverse(listDeplacement) ;
13     pour chaquedeplacementderobot faire
14         affiche le robot et la direction à effectuer
15     fin
16 fin
17 sinon
18     affiche "pas de solution"
19 fin
```

Démonstration de l'application

Nous allons procéder à la démonstration de l'application du Ricochet Robots.

Conclusion

Ce projet était très intéressant. Nous ne connaissions pas le Ricochet Robots et c'était intéressant de se pencher sur les règles de ce jeu et de le recréer afin de pouvoir y jouer autrement que sous forme physique.

Améliorations possibles :

- La mise en place du jeton spirale.
- La mise en place d'un système de score.
- Une mise en duel entre l'IA et le joueur.
- L'ajout ou la suppression de robots sur le plateau.
- L'ajout d'obstacles (et/ou de malus) ou de bonus sur le plateau.
- Mise en place d'un système de difficulté.
- Mise en place du délai pour la réflexion et pour la réalisation des mouvements.

Bibliographie

Références :

 Red Blob Games.
Introduction to the a* algorithm.
https://www.redblobgames.com/pathfinding/a-star/introduction.html.

 Randycoulman.
A* algorithm.
https://speakerdeck.com/randycoulman/solving-ricochet-robots?slide=108.