- Soutenance de projet -

Solveur de Ricochet Robots

Erwan Philippe MENSAH
TOURE Papa Samba Khary
TRAORE Daouda

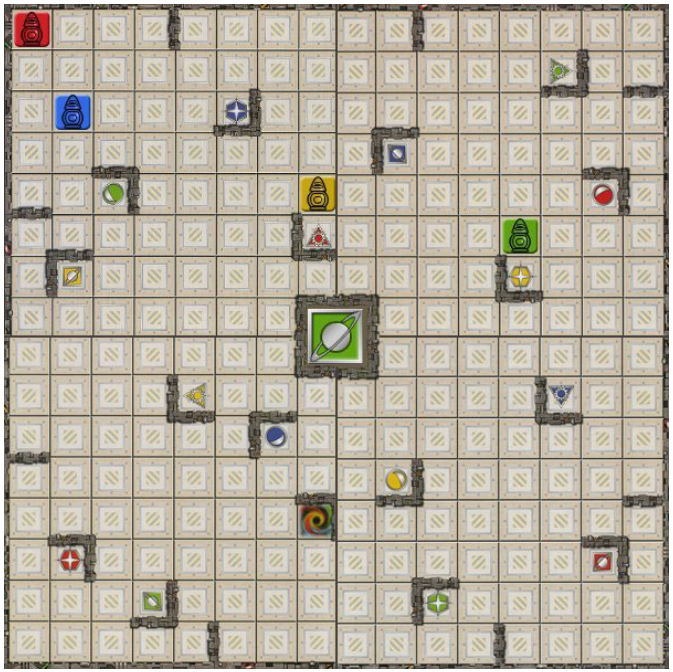
L2 Informatique
Groupe 2A

Sommaire

Introduction

- Le Ricochet Robots: jeu de société composé d'un plateau, de quatre de robots et de 17 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.
  - Implémentation de l'algorithme A*, optimisation de l'algorithme.

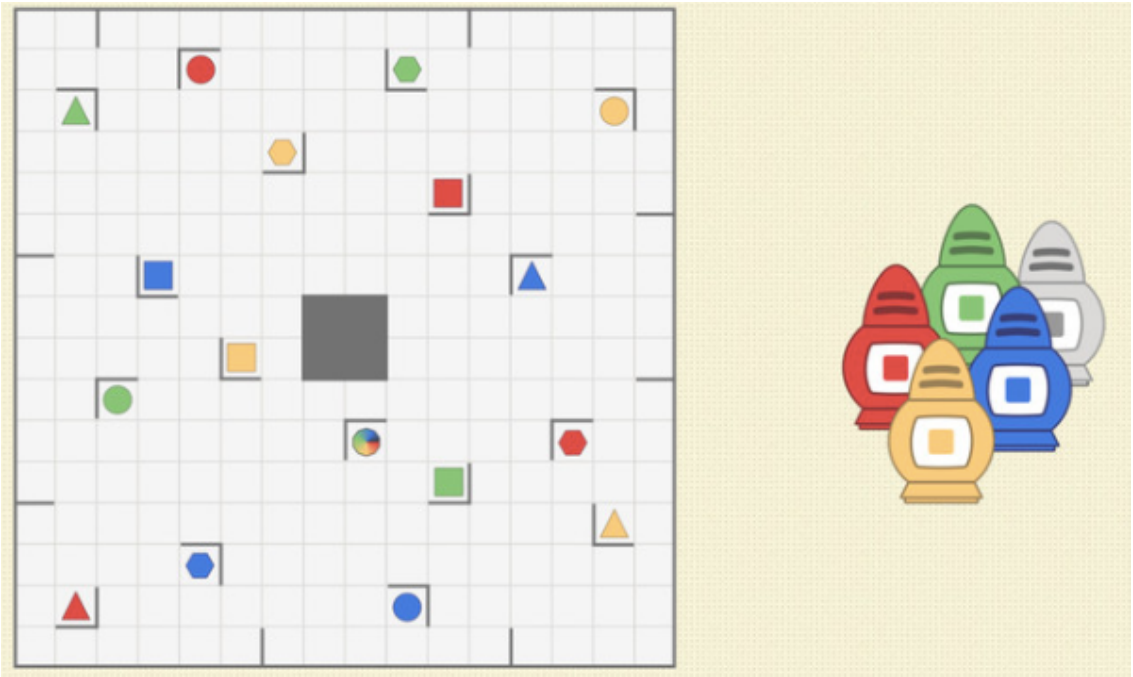
Exemple de Plateau



Remarque

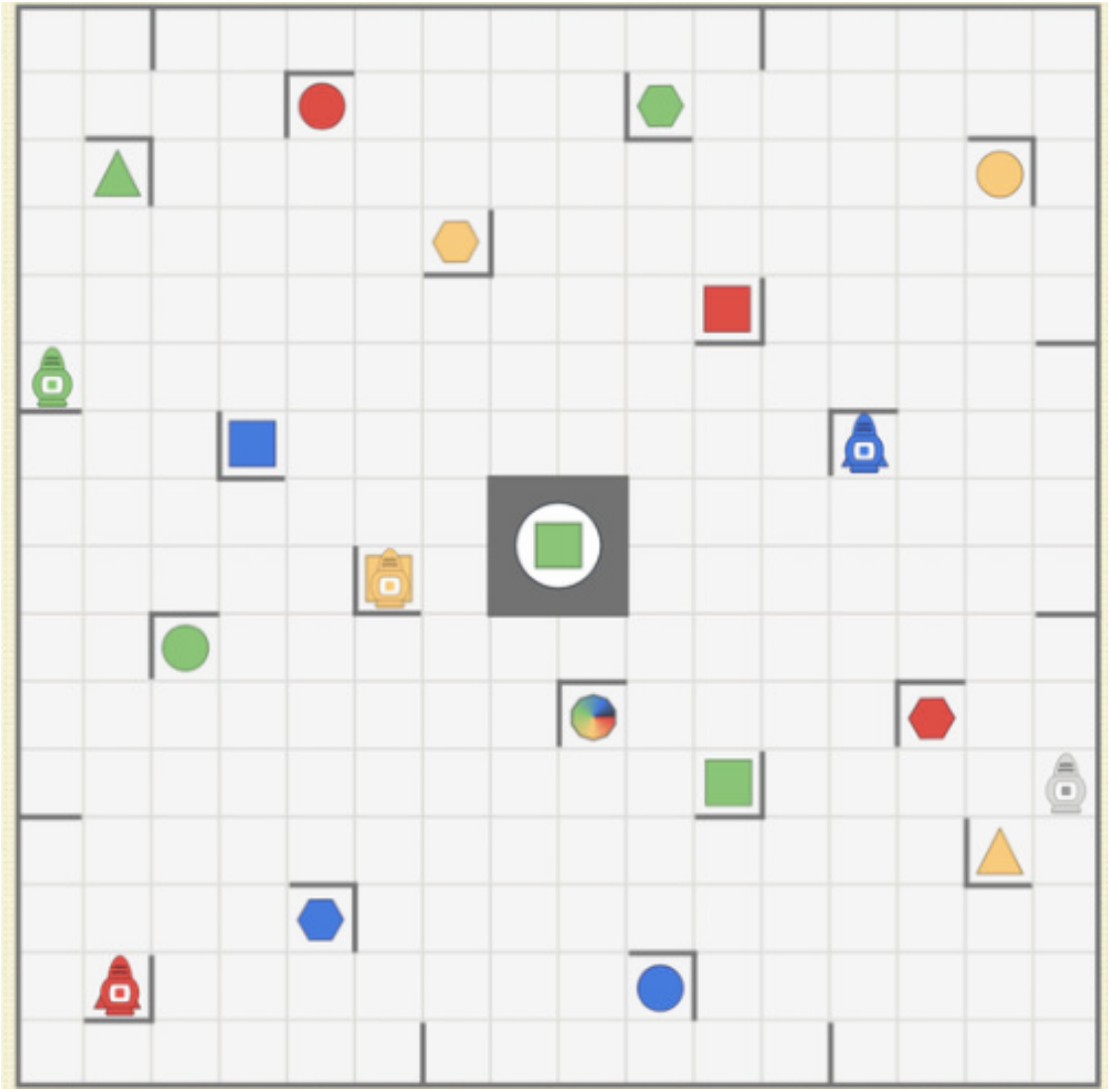
Nous avons 96 configurations possibles.

Principe du jeu



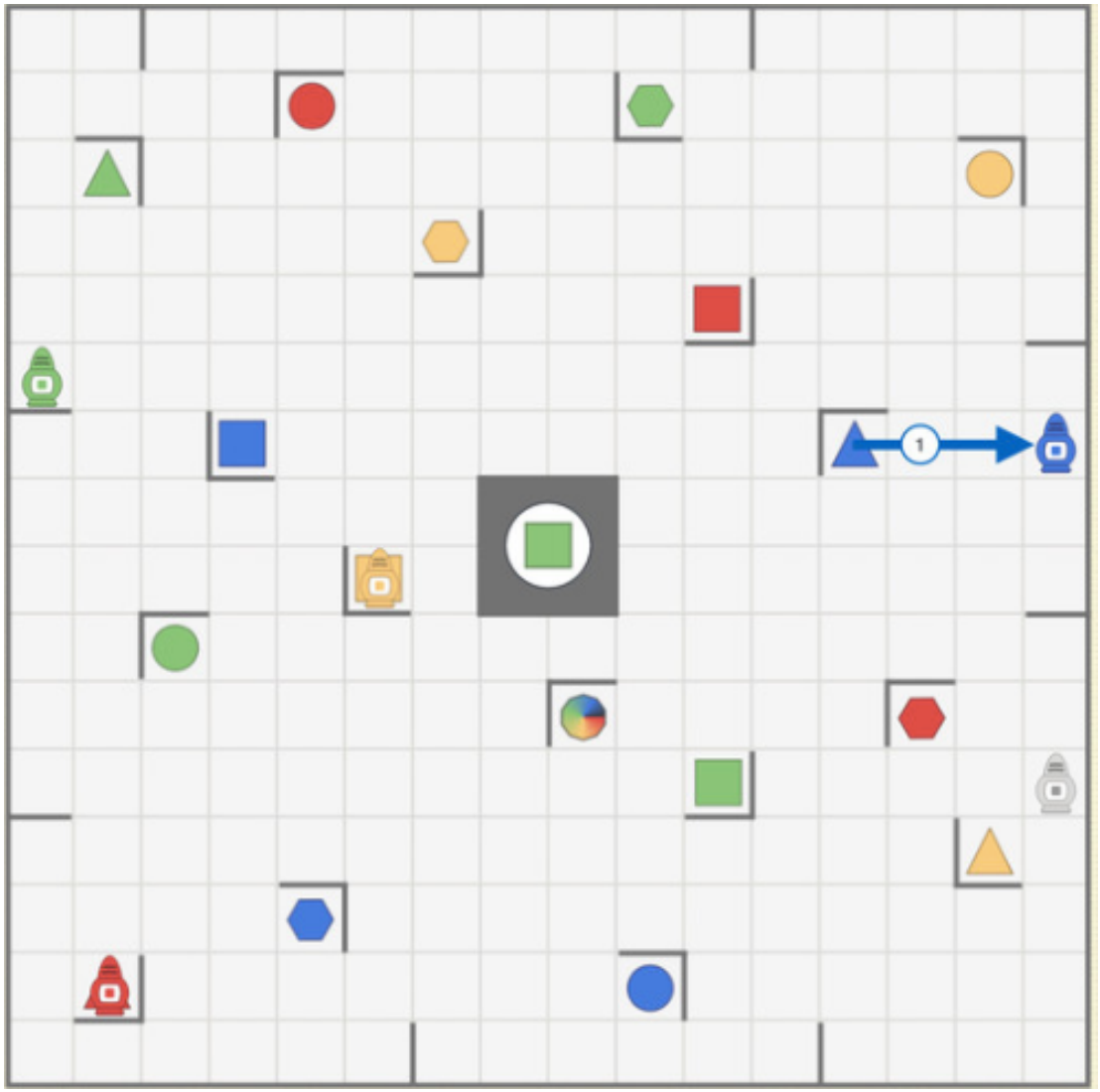
- Comment cela fonctionne?

Principe du jeu



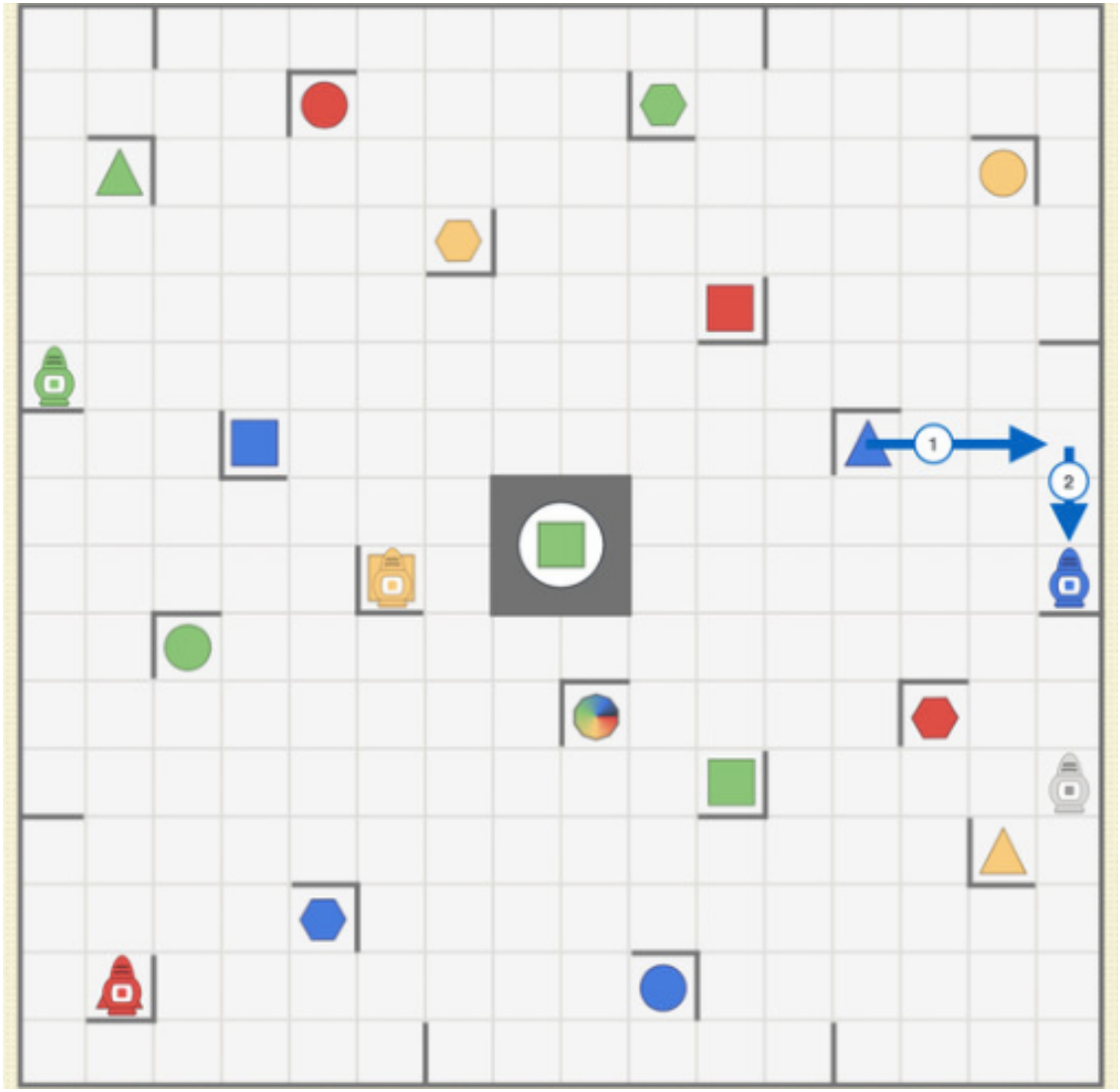
- Mouvements pour amener le robot Vert à la cible carré verte.

Principe du jeu



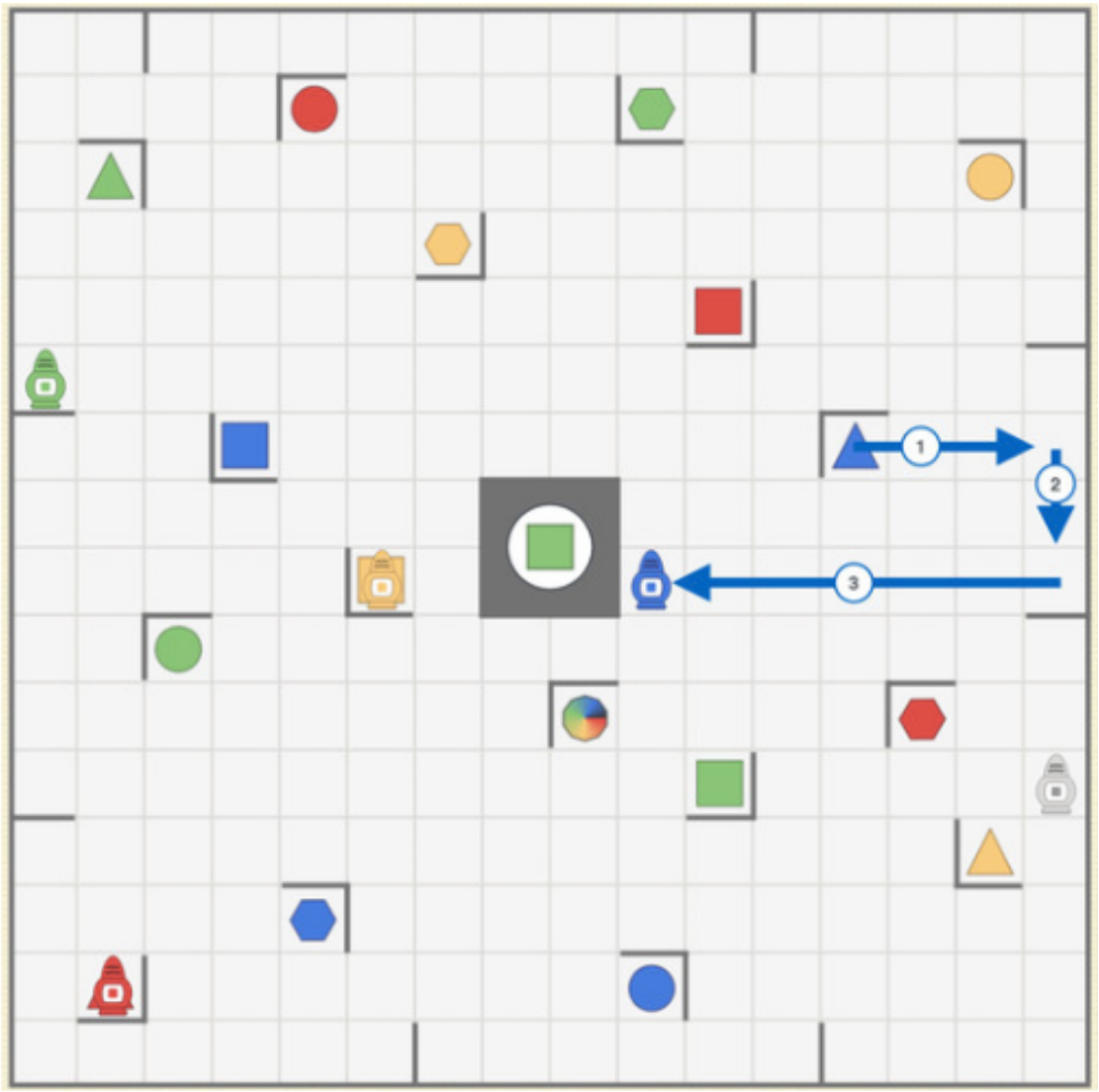
- Mouvements pour amener le robot Vert à la cible carré verte
- Bleu $\longleftrightarrow$ droite

Principe du jeu



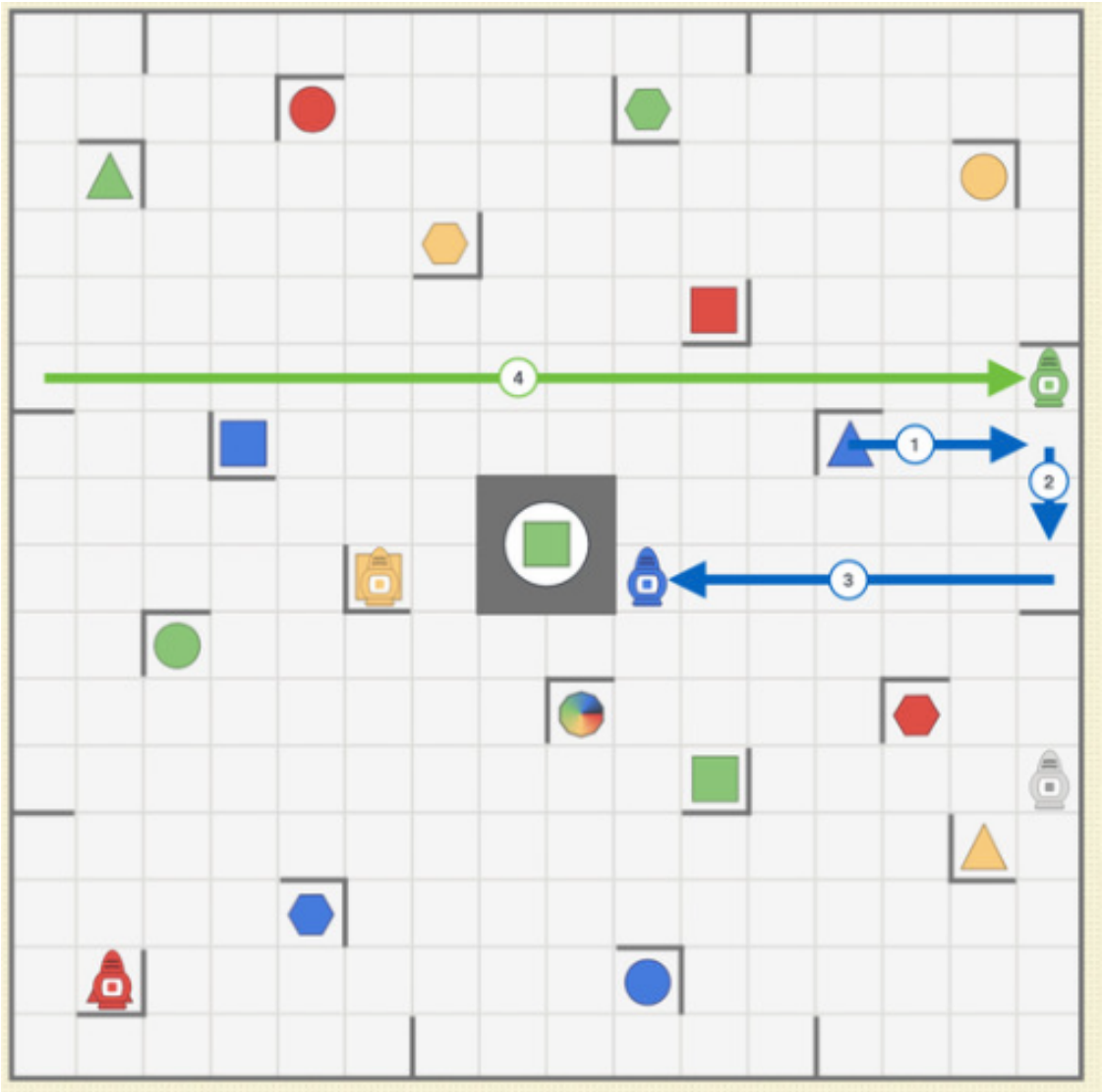
- Mouvements pour amener le robot Vert à la cible carré verte.
- Bleu $\longleftrightarrow$ droite
- Bleu $\longleftrightarrow$ bas

Principe du jeu



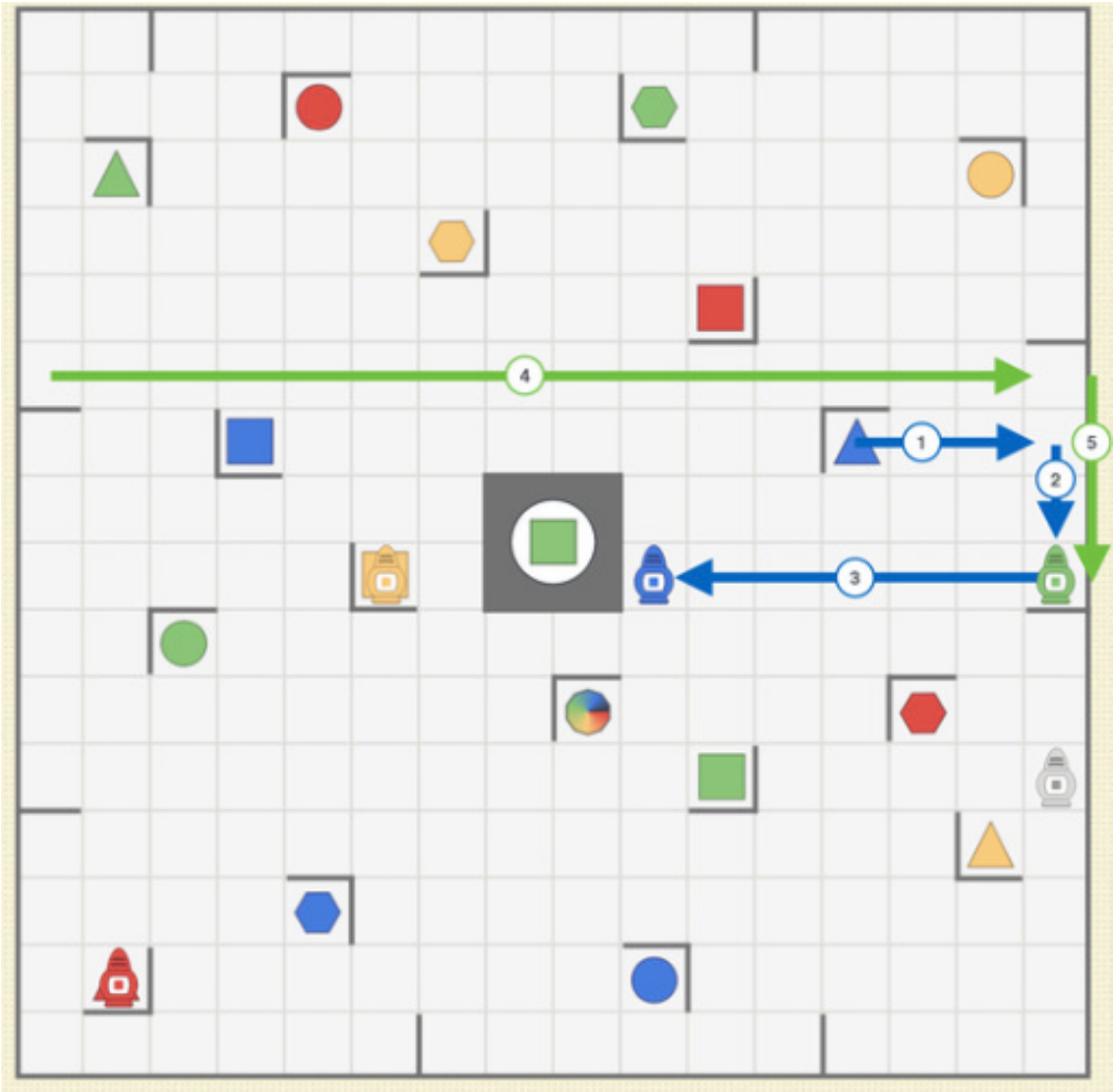
- Mouvements pour amener le robot Vert à la cible carré verte.
- Bleu $\longleftrightarrow$ droite
- Bleu $\longleftrightarrow$ bas
- Bleu $\longleftrightarrow$ gauche

Principe du jeu



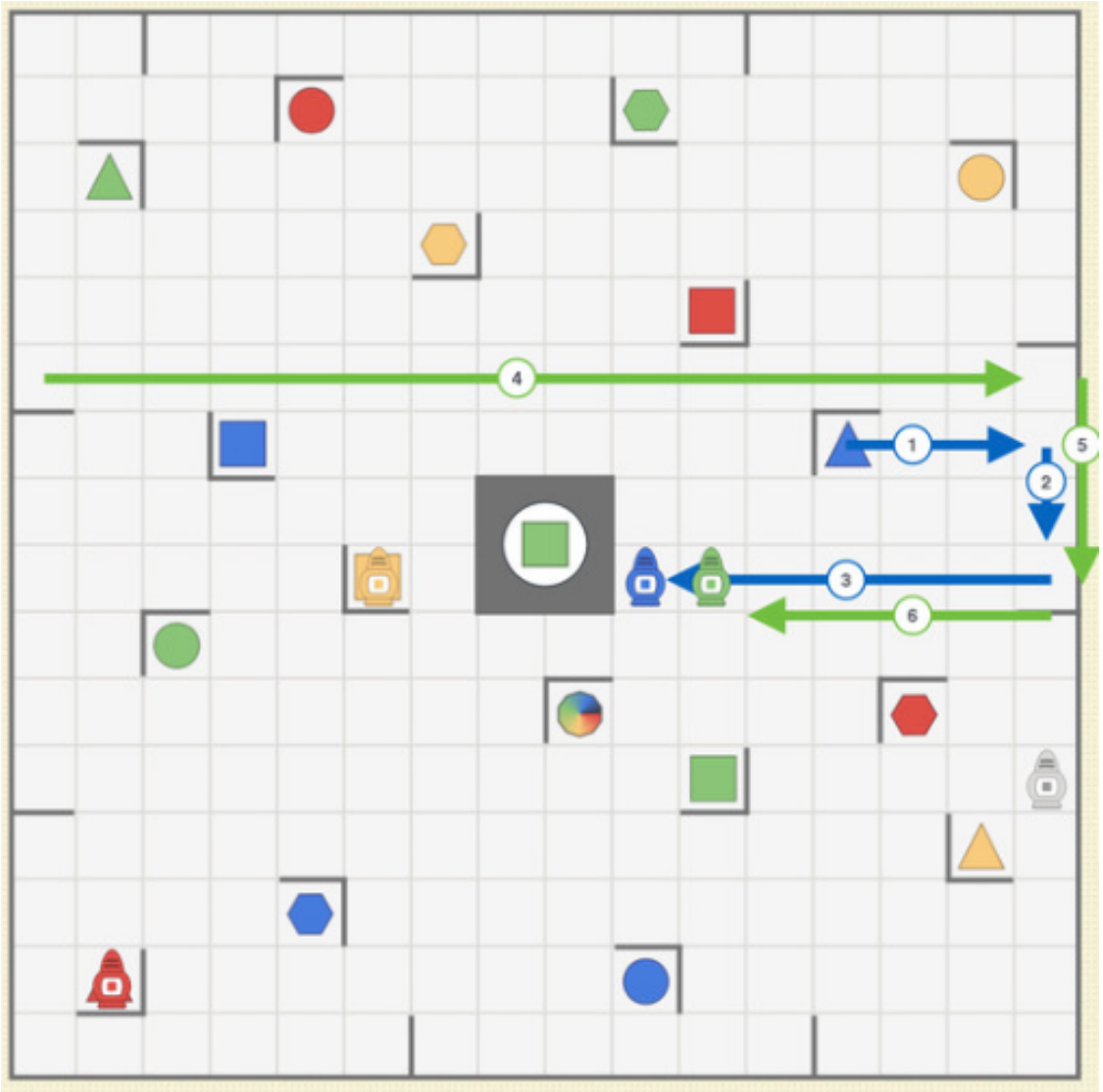
- Mouvements pour amener le robot Vert à la cible carré verte.
- Bleu $\longleftrightarrow$ droite
- Bleu $\longleftrightarrow$ bas
- Bleu $\longleftrightarrow$ gauche
- Vert $\longleftrightarrow$ droite

Principe du jeu



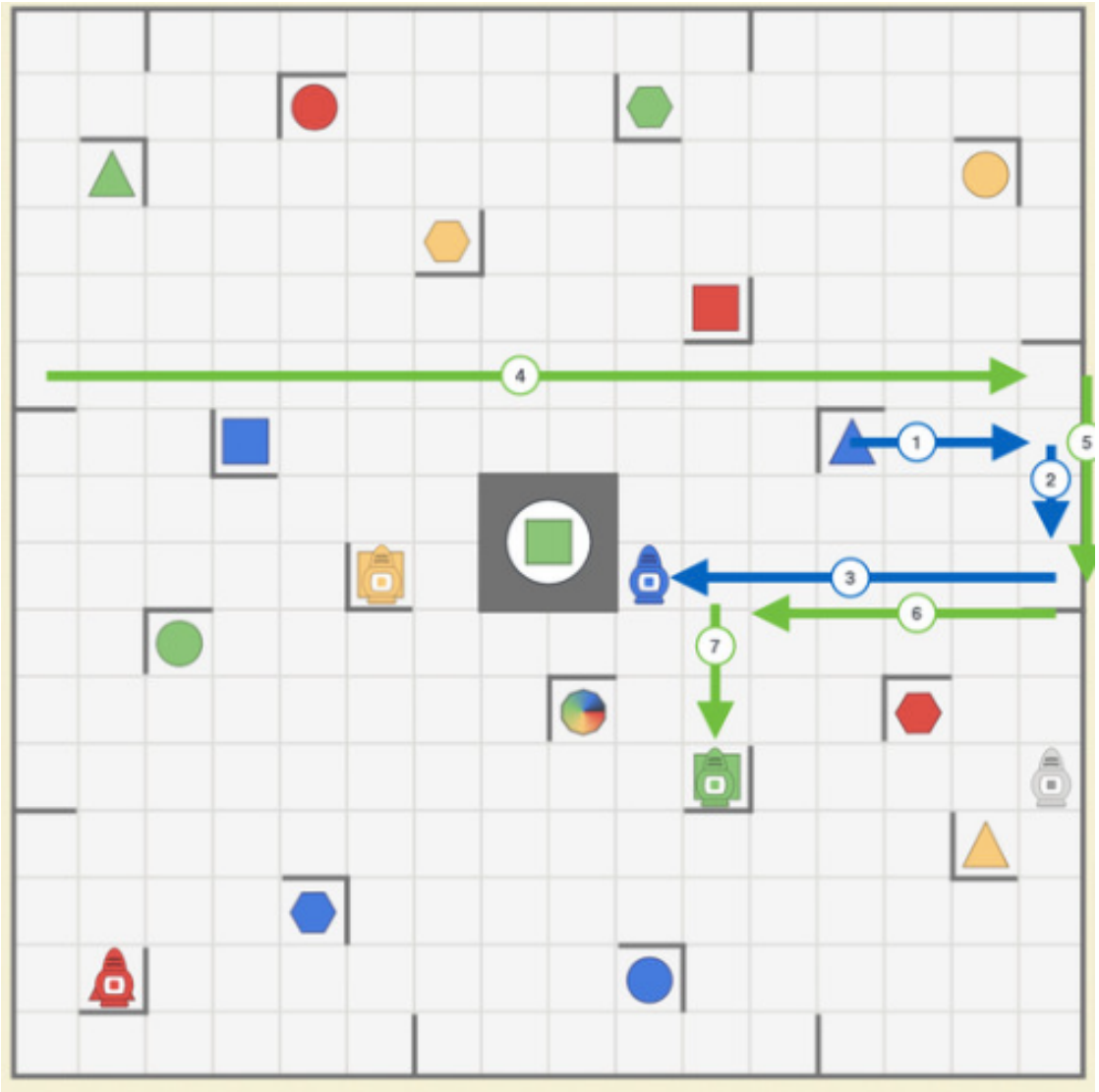
- Mouvements pour amener le robot Vert à la cible carré verte.
- Bleu $\longleftrightarrow$ droite
- Bleu $\longleftrightarrow$ bas
- Bleu $\longleftrightarrow$ gauche
- Vert $\longleftrightarrow$ droite
- Vert $\longleftrightarrow$ bas

Principe du jeu



- Mouvements pour amener le robot Vert à la cible carré verte.
  item Bleu $\longleftrightarrow$ droite
- Bleu $\longleftrightarrow$ bas
- Bleu $\longleftrightarrow$ gauche
- Vert $\longleftrightarrow$ droite
- Vert $\longleftrightarrow$ bas
- Vert $\longleftrightarrow$ gauche

Principe du jeu



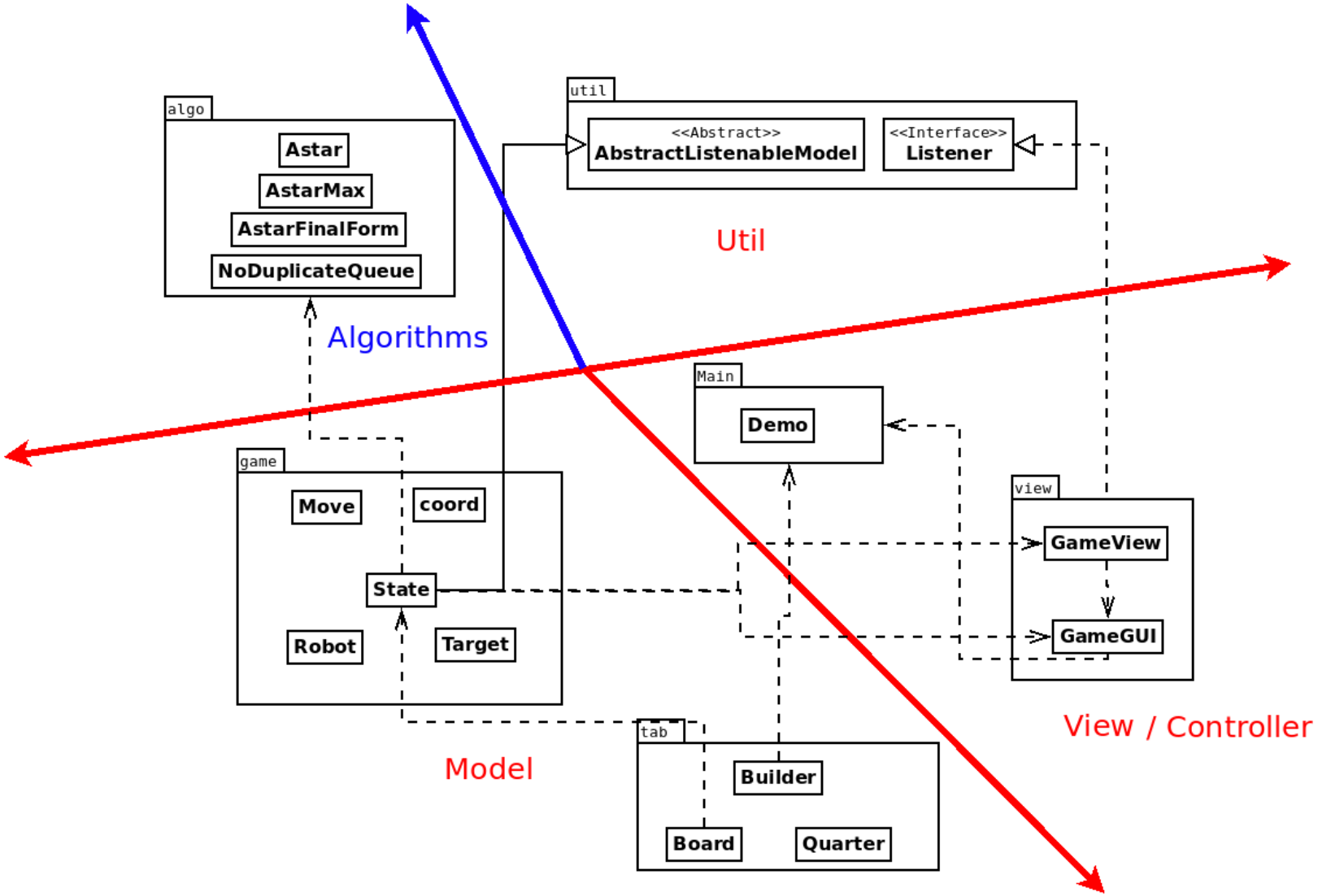
- Mouvements pour amener le robot Vert à la cible carré verte.
- Bleu $\longleftrightarrow$ droite
- Bleu $\longleftrightarrow$ bas
- Bleu $\longleftrightarrow$ gauche
- Vert $\longleftrightarrow$ droite
- Vert $\longleftrightarrow$ bas
- Vert $\longleftrightarrow$ gauche
- Vert $\longleftrightarrow$ bas

Les packages

Six packages

- ***Game*** : le moteur du jeu
- ***tab*** : création du plateau
- ***view*** : partie graphique
- ***algo*** : AStar
- ***util*** : implémentation MVC
- ***main*** : qui contient l'exécutable de l'application.

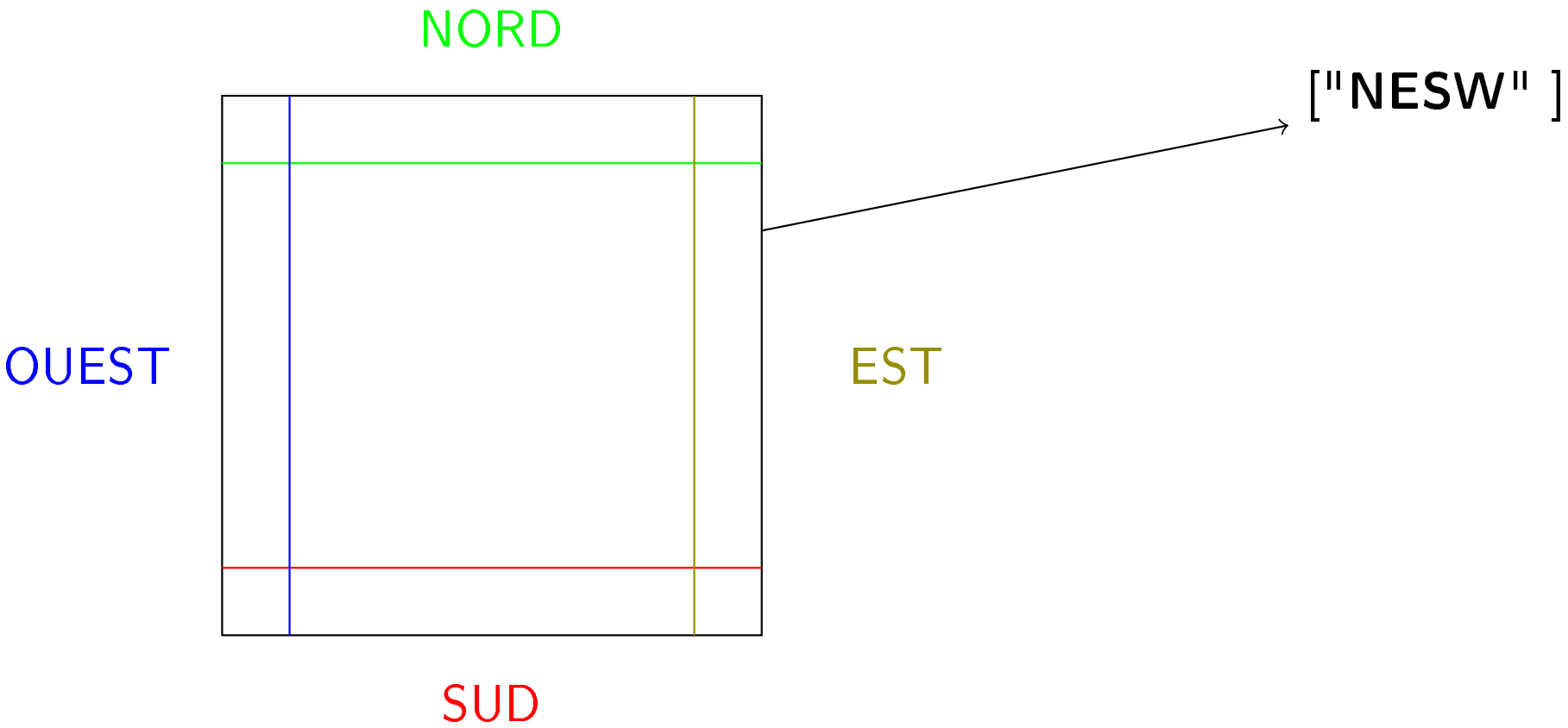
UML du projet



Représentation d'un case

- **'N'** indique la présence d'un mur au nord de la case.
- **'S'** indique la présence d'un mur au sud de la case.
- **'E'** indique la présence d'un mur à l'est de la case.
- **'W'** indique la présence d'un mur à l'ouest de la case.
- **'X'** indique que la case vide.
- les lettres **'R'**, **'B'**, **'G'**, **'Y'** désignent respectivement les couleurs des targets **red**, **blue**, **yellow**, **green**
- les lettres **'C'**, **'T'**, **'H'**, **'Q'** désignent respectivement la forme des targets **circle**, **triangle**, **hexagone**, **square**

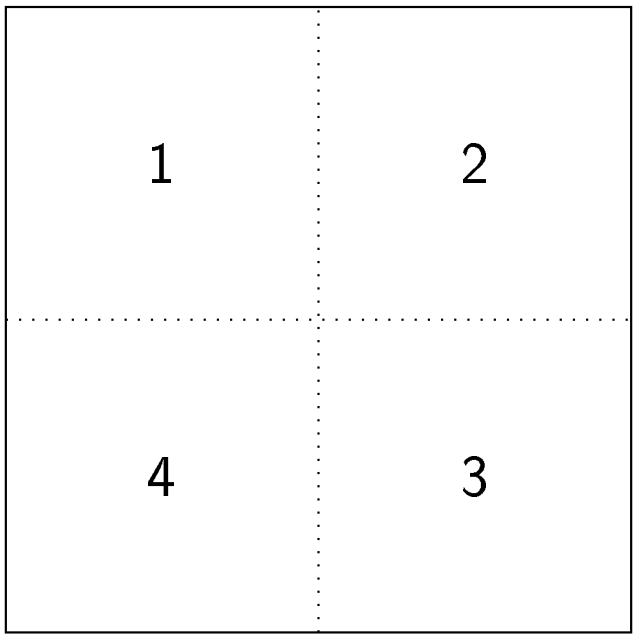
Exemple de case



Quartiers

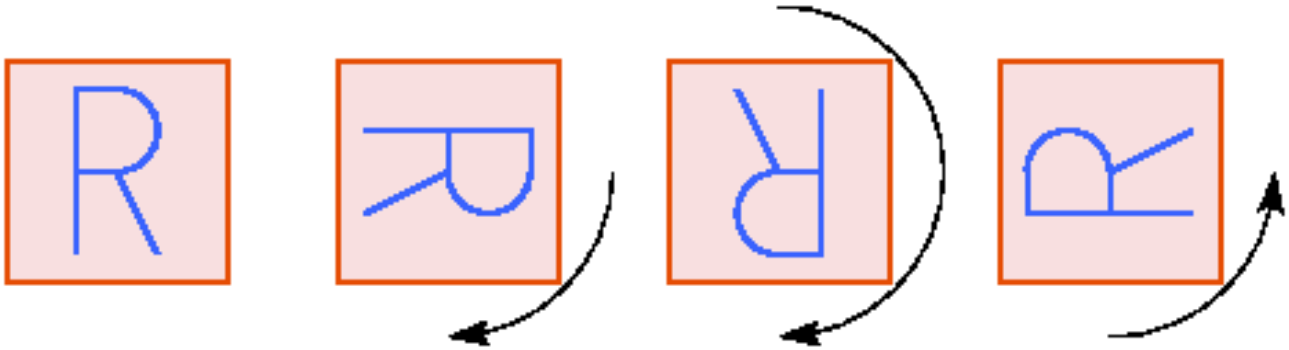
Composition

Le plateau est composé de 4 quartiers pouvant être placés aléatoirement.

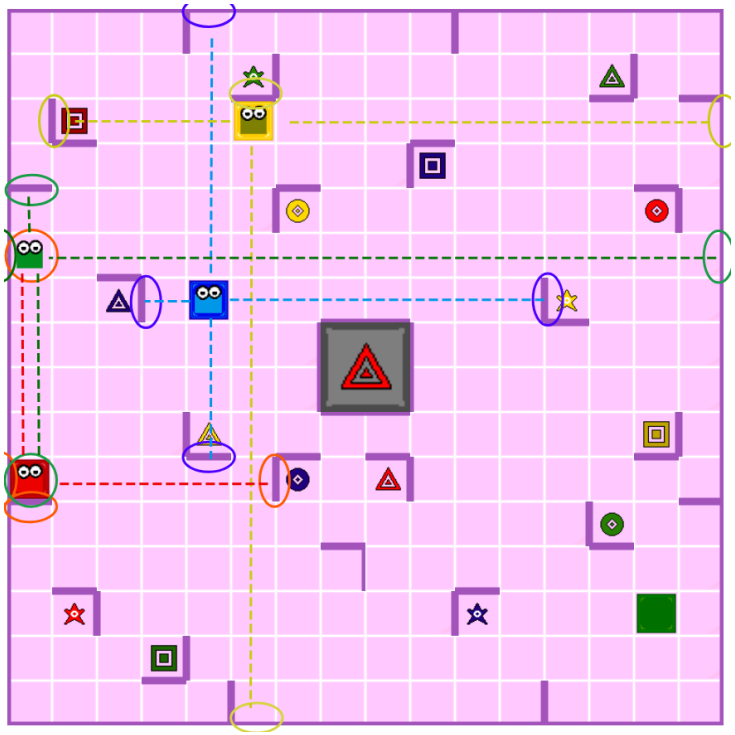


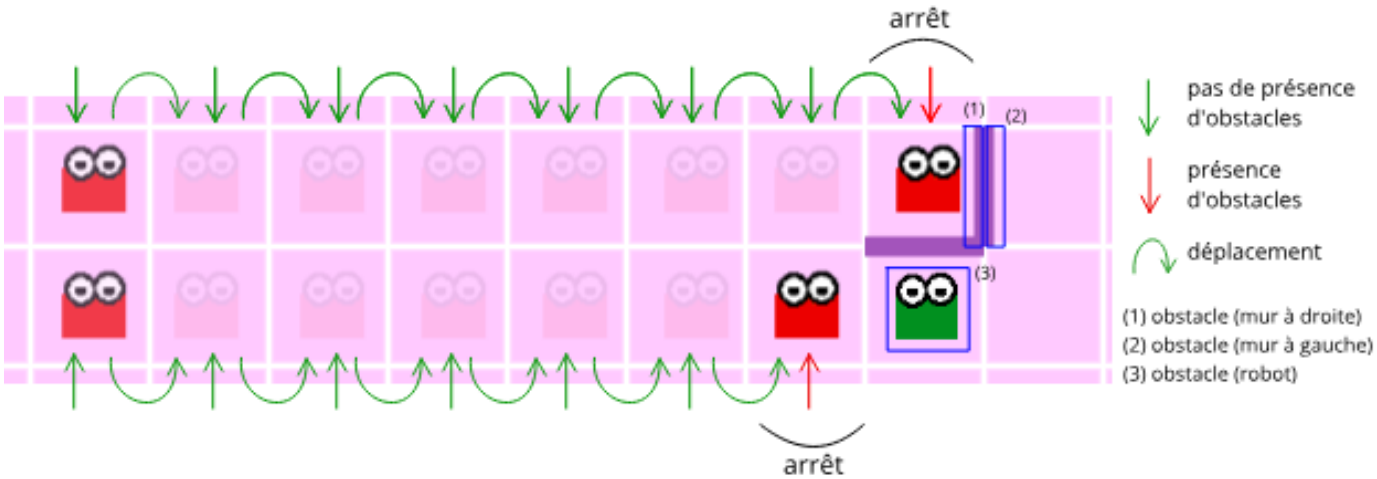
Placement et rotation des quartiers

- Premier quartier inchangé.
- Second : rotation de 90°
- Troisième : double rotation de 90°
- Quatrième : triple rotation de 90°



Déplacement Robots



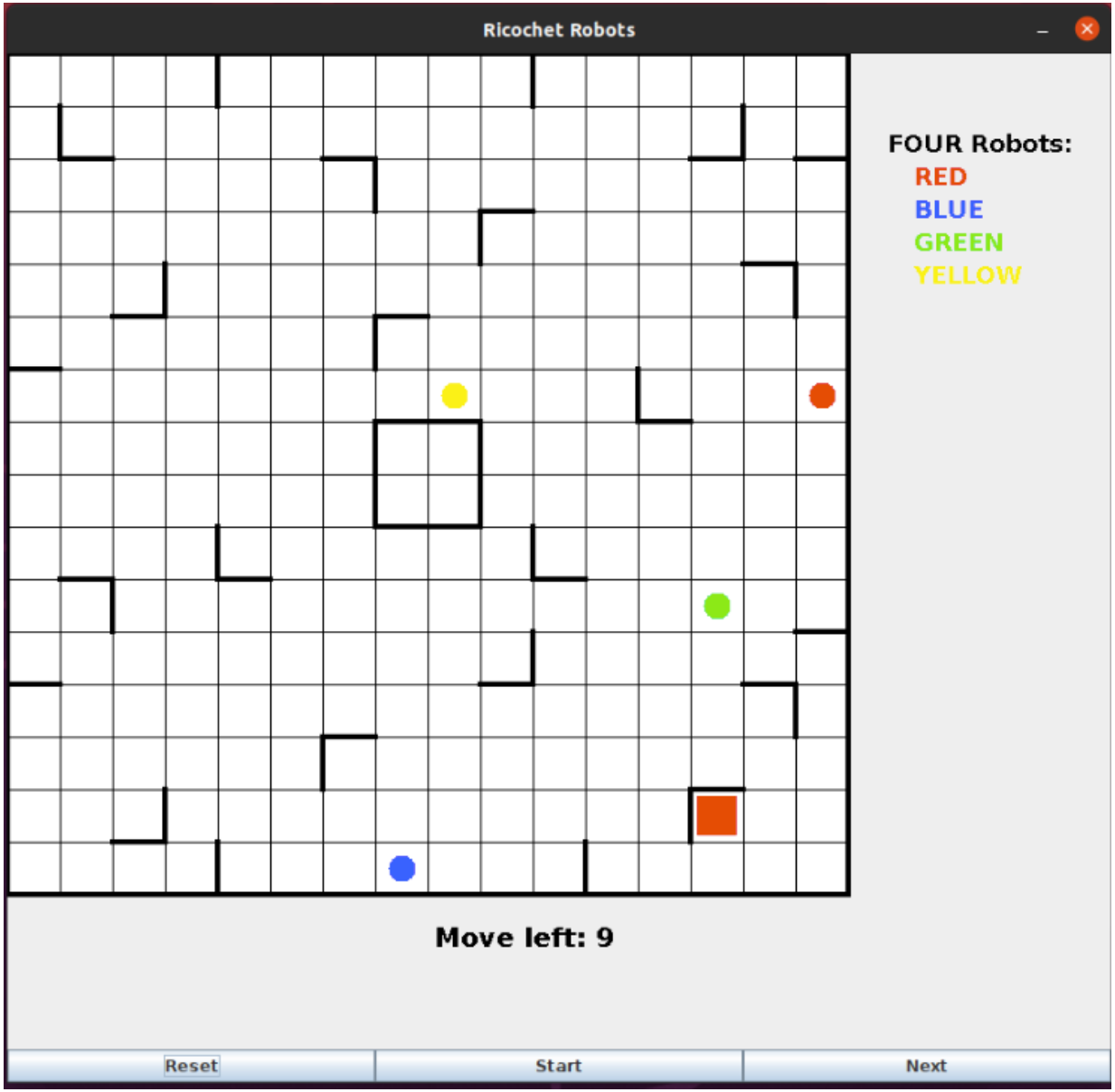


Représentation graphique

L'interface est composée:

- Des 4 robots avec leur couleur respectives.
- De la cible.
- De trois boutons ***START,NEXT,RESET***
- Du nombre de moves nécessaire pour atteindre la cible

Représentation graphique



Présentation

AStar

- A* aussi appelé ***A*** - ***Star*** est un algorithme de parcours de graphe et de recherche de parcours.
- Il a été publié en 1968 par Peter Hart et Bertram Raphael.[1].
- Peut être considéré comme une extension ***Dijkstra*** [2].
- A* parvient à obtenir de meilleurs résultats grâce à une heuristique guidant sa recherche.

Pourquoi une heuristique?

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

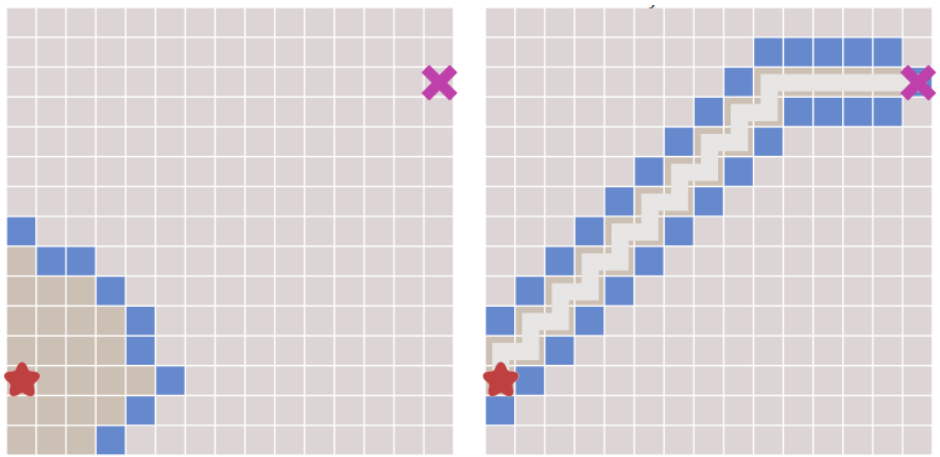
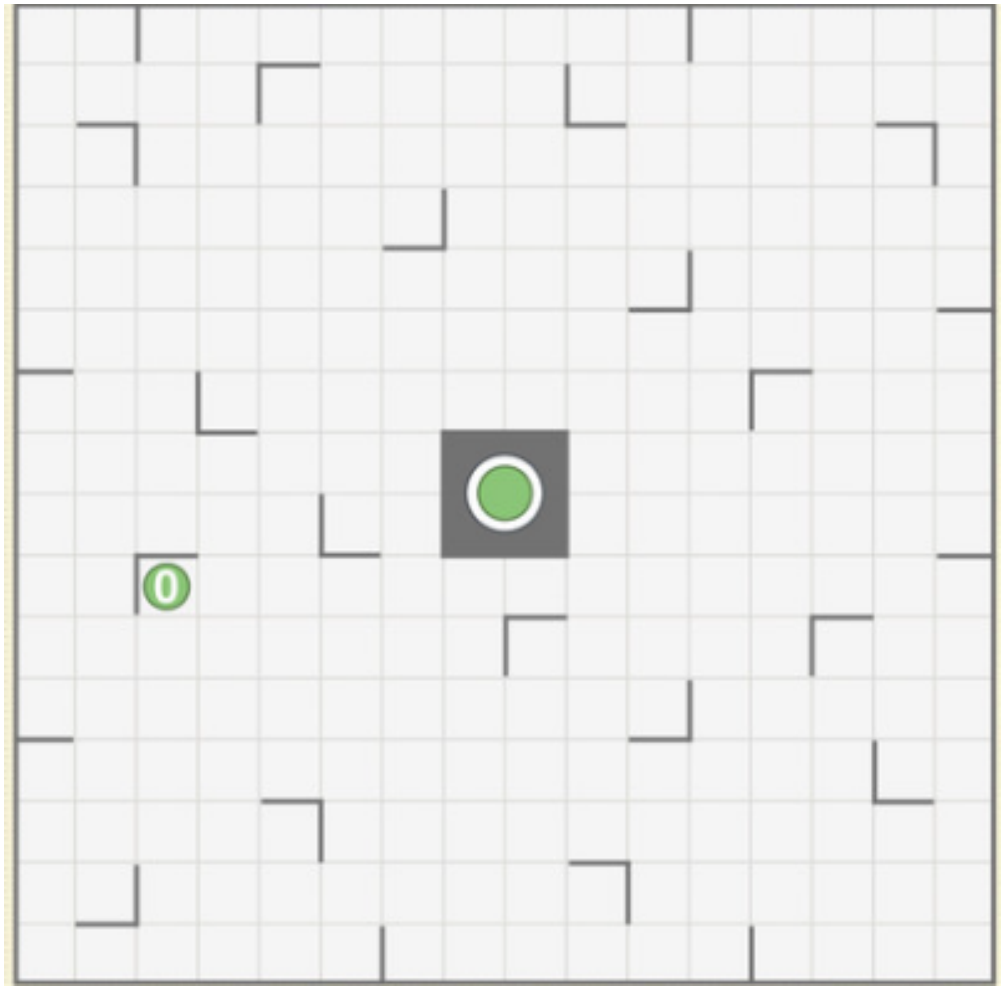


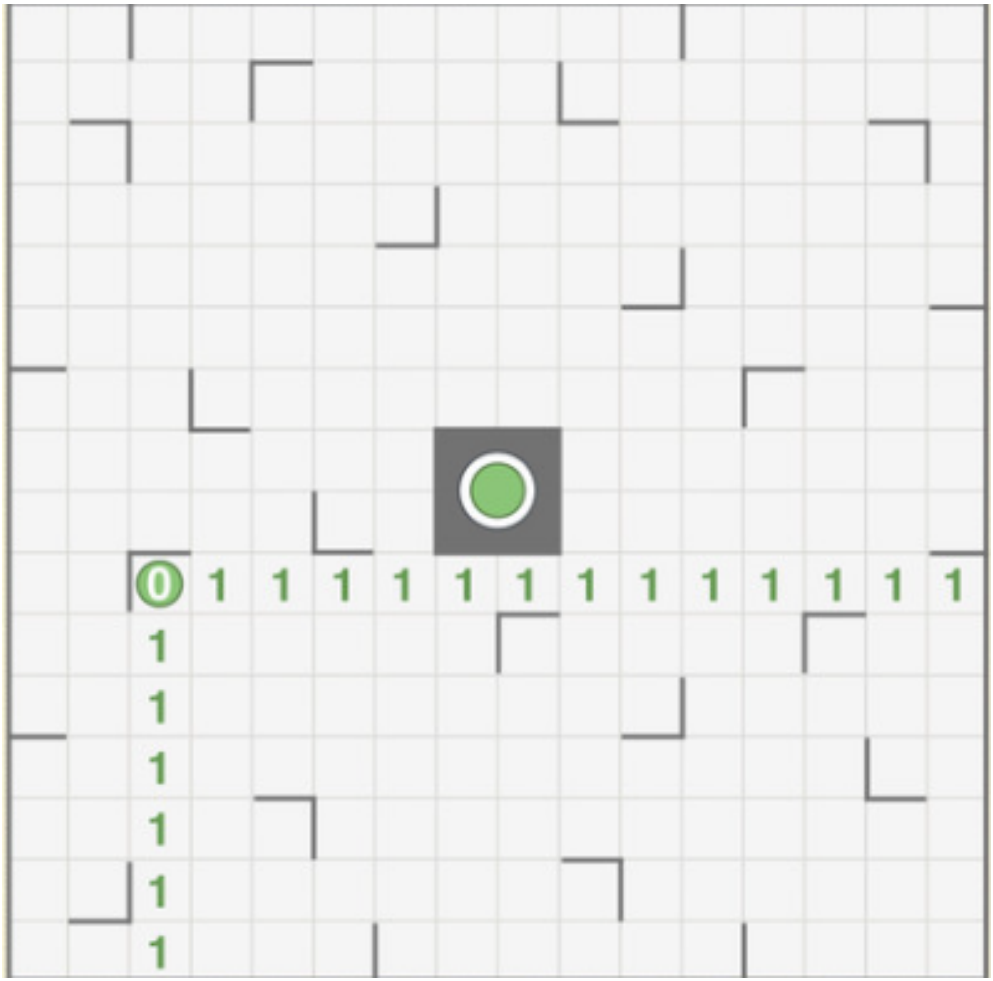
Figure: A* vs Dijkstra

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

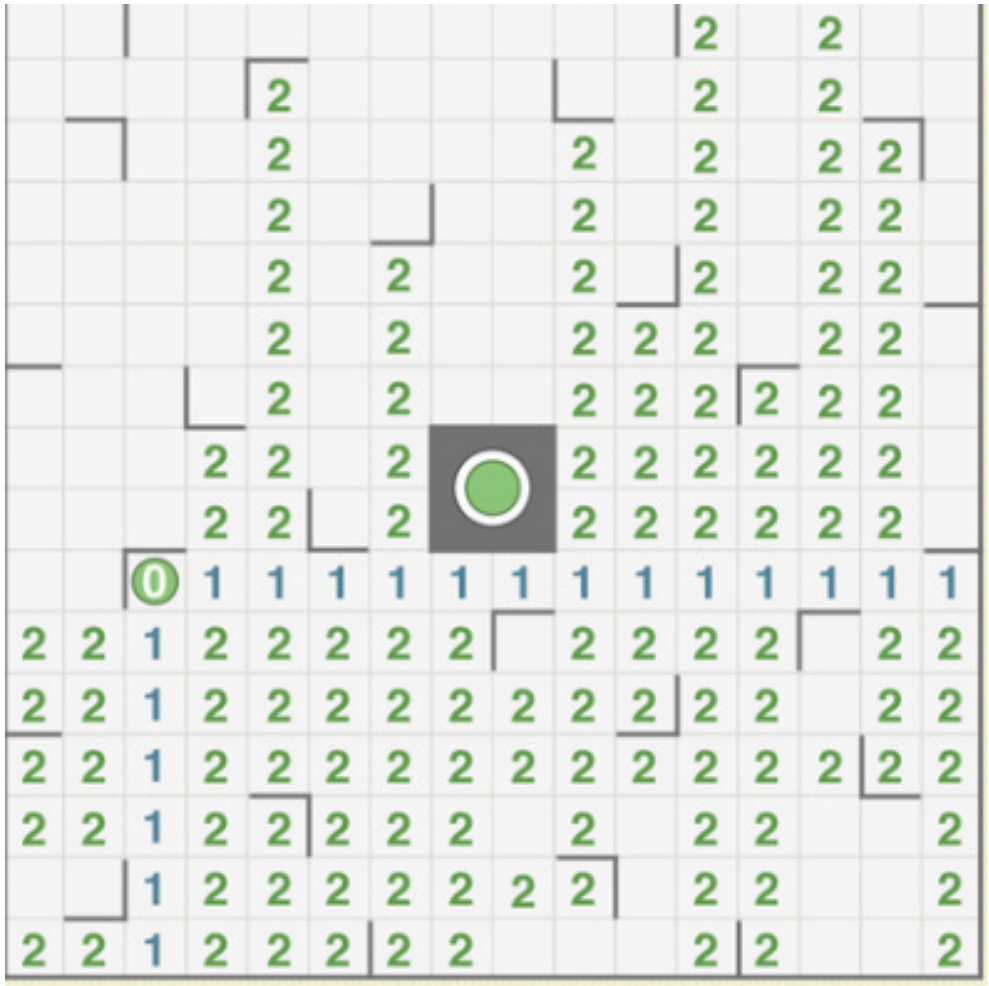
Mise en place



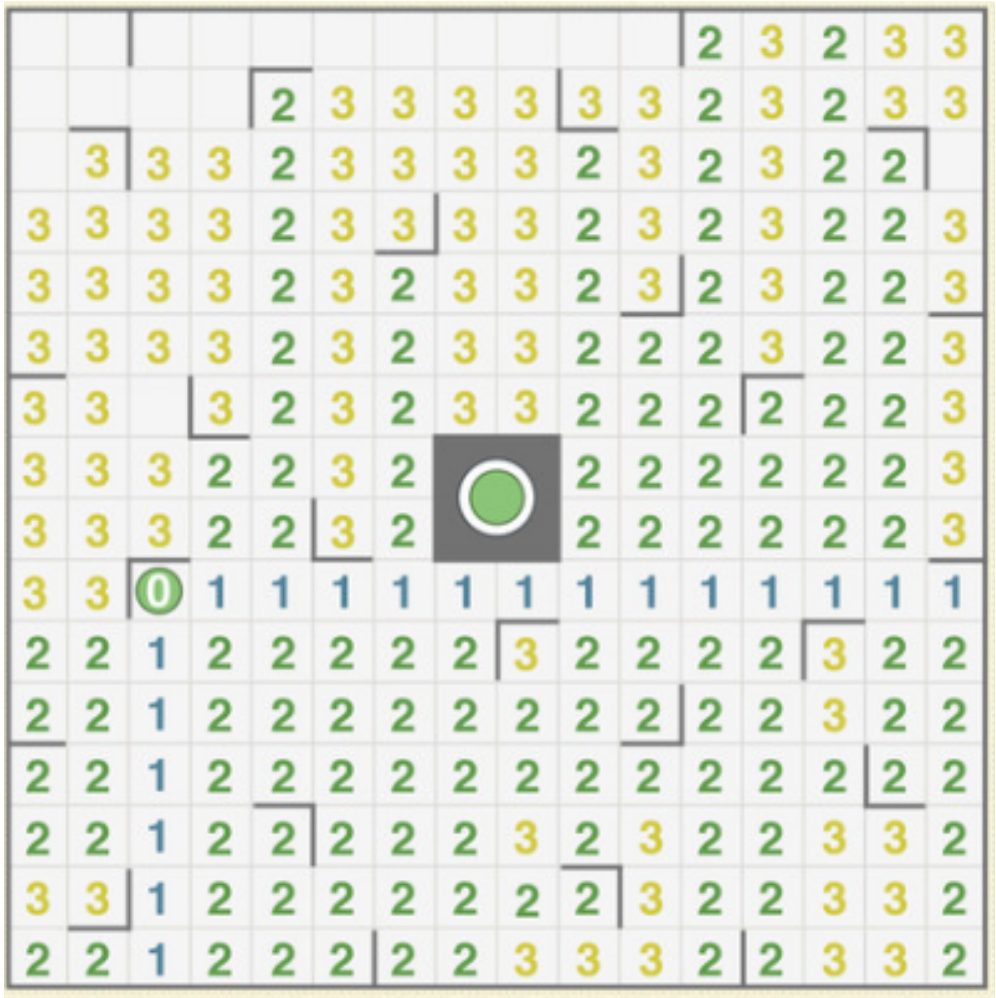
1 Move de la cible



2 Moves de la cible



3 Moves de la cible



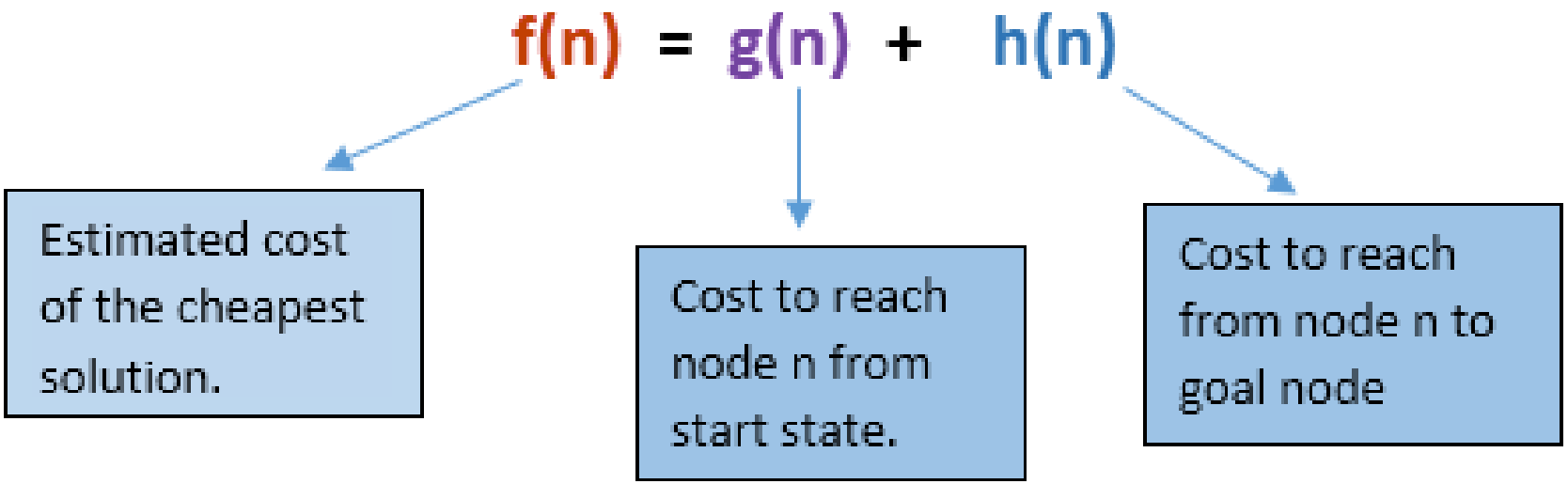
4 Moves de la cible

5 Moves de la cible

Figure: Tableau montrant en combien de mouvements à partir de chaque case on peut atteindre la cible [3]

FScore/GScore

- Le prochain état a étudié est celui dont le coût jusque là (***GScore***) + l'estimation du coût jusqu'à la cible (***Heuristique***) est le ***minimum***.



Code

Algorithm 1: ALGORITHME ASTAR_FINAL_FORM

Input : initialState, h
Output: listOfStates

```
1  openSet ← ∅
2  priorityQueue ← ∅
3  gScore ← new map < State, Int > with Default Value of +∞
4  fScore ← new map < State, Int >
5  startingState ← initialState
6  openSet.add(startingState)
7  priorityQueue.add(startingState)
8  gScore.put(startingState, 0)
9  fScore.put(startingState, h(startingState))
10 while priorityQueue is not empty do
11     currentState ← priorityQueue.poll()
12     if currentState.isOver() then
13         return reconstructPath(cameFrom, currentState)
14     end if
       /* On explore tous les états résultant d'un mouvement
          de robot                                          */
15     foreach neighbor of currentState do
16         tentativeGscore ← gScore[currentState] + 1
           /* S'il est meilleur que tout ce que nous avions
              jusqu'à présent nous le gardons               */
17         if tentiveGscore < gScore[neighbor] then
18             cameFrom[neighbor] ← currentState
               gScore[neighbor] ← tentativeGscore
               fScore[neighbor] ← tentativeGscore + h(neighbor)
19             if neighbor not in openSet then
20                 openSet.add(neighbor)
21                 priorityQueue.add(neighbor)
22             end if
23         end if
24     end foreach
25 end while
   /* openSet est vide et la target jamais atteinte       */
26 print("NO SOLUTION")
27 return ∅
```

Algorithm 2: ALGORITHME RECONSTRUCTPATH : RETOURNANT LE TRAJET

Input : HashMap cameFrom<State,State>
Input : currentState
Output: ListOfStates

```
1 listOfStates ← new List < State > ()
2 listOfStates.add(currentState)
3 while currentState ∈ cameFrom.Keys do
4     currentState ← cameFrom.get(currentState)
5     listOfStates.add(currentState)
6 end while
7 return listOfStates
```

Démo

Démonstration

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:

- Amélioration de l'interface graphique, car l'interface finale est très basique.
- Rajouter d'autres fonctionnalités comme pourvoir jouer par soi même.
- Faire des tests comparant différents type d'algorithmes.

Bibliographie

Références:

Wikipedia, algorithme a*.
https://fr.wikipedia.org/wiki/Algorithme_A*.

Wikipedia, algorithme dijkstra.
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

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