r

RICOCHET ROBOTS

EPSI

RESUME

Réalisation d'une intelligence Artificielle qui résoudre la situation de jeu Ricochet Robots et d'amener le robot de la couleur correspondante sur l'objectif en un minimum de coups

Réalisé par

Khaled FKIRI
Baptiste CHEVET

Ricochet Robots

I. Introduction

Durant le projet NAO et Intelligence Artificielle I4, nous nous intéressons à un jeu de société qui existe sous le nom de Ricochet Robots. C'est un jeu de société ou les joueurs doivent déplacer des robots en faisant le moins coups possible pour atteindre un objectif. Chaque partie fait ainsi appelle à beaucoup de réflexions.

Ce projet nous a donc amené à travailler sur des connaissances qui sont nouvelles pour nous, notamment sur le développement des algorithmes de résolution.

A travers ce document, nous présenterons dans un premier temps le jeu et les méthodes que nous avons utilisé ainsi que l'algorithme de résolution.

II. Presentation

Le jeu Ricochet Robots est un jeu de société qui a été réalisé en 1999 par l'Allemand Alex Randolph. Ce jeu peut se jouer seul ou à plusieurs. Le plateau du jeu représente une grille sur laquelle sont représentés différents murs. Ce plateau est divisé en quatre parties double-face avec des murs positionnés de manières différentes de chaque côté. On obtient ainsi une carte avec une des 16 combinaisons de grille de jeu possible. On place ensuite les quatre robots aléatoirement sur la grille.

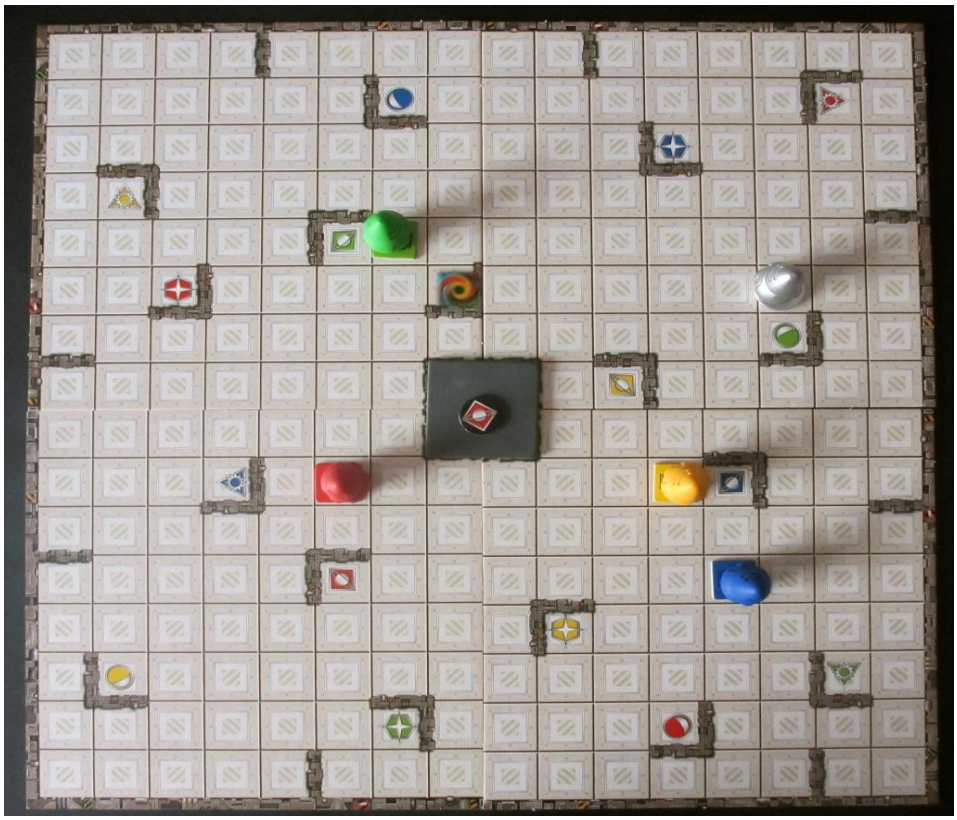


Figure 1 : Carte du jeu Ricochet Robots

Comme on le voit sur la figure 1, les cibles de couleurs sont positionnées dans les angles qui forment les murs.

Durant une partie, un des joueurs pioche une des cibles, l'objectif est d'amener le robot de la couleur sur la cible correspondant en faisant un minimum de coups possible ainsi les robots se déplacent en ligne ou en colonne.

En effet, une fois une direction choisie (haut, bas, droite ou gauche) ils ne peuvent s'arrêter tant qu'ils ne rencontrent pas un obstacle qui peut être un mur ou un autre robot. Les utilisateurs peuvent les déplacer (en conservent les règles de déplacement citées précédemment) de manière à créer des obstacles pour le robot principal selon ses besoins.

Chaque déplacement d'un des quatre robots se compte un coup. Dès qu'un des joueurs a trouvé une solution, on lance un compte à rebours et les autres joueurs on une minute pour trouver une solution avec moins de coups que le premier. A l'issue de ce délai, le joueur ayant la solution comptant le moins de coups présente sa solution et remporte la manche.

III. ORGANISATION DU TRAVAIL

En respectant les règles de la méthode Agile, on a découpé le projet en trois taches majeures :

- La partie étude de besoin : Une bonne partie de notre travail au début du projet est consacré à rechercher beaucoup d'informations qui nous permet de comprendre les principes du jeu. Nous devions en effet informer sur le jeu et les différentes solutions existantes qui nous ont donné une vision globale pour la résolution du jeu. Nous avons ensuite réalisé un cahier des charges. Enfin, nous avons passé du temps à chercher des algorithmes qui peuvent être utiliser pour obtenir une solution optimale.
- La partie développement : Une fois les informations réunies, nous avons commencé a développé une méthode qui permet de générer la carte du Ricochet Robots à partir des données d'un fichier CSV.
- L'intelligence Artificielle : Une fois que la carte générée, nous avons travaillé en parallèles à mettre en place l'algorithme choisit et l'intégration l'intelligence Artificielle censé résoudre la situation de jeu.

IV. DEVELOPPEMENT

1. La solution Réalisée

La solution proposée est une solution console développé avec langage Python 3.3.6 qui permet de :

- Générer la carte du jeu à partir d'un fichier CSV. Les données dans le fichier CSV sont des nombres entiers et des nombres décimaux, les 4 robots et l'objectif ont des coordonnées entiers et les murs ont des coordonnées décimales c'est-à-dire qu'un mur avec les coordonnées (5.5 ; 0) bloquerait un robot passant de (5 ; 0) à (6 ; 0).
- Positionner l'emplacement des quatre robots sur la carte générée qui ont des coordonnées entières à partir d'un fichier CSV.

2. Génération de la grille

La carte est générée en fonction des données du fichier CSV choisit (carte1.csv ou carte2.csv), les robots sont positionnés en fonction des données du fichier CSV choisit (depart1.csv ou depart2.csv) qui représente l'emplacement des quatre robots.

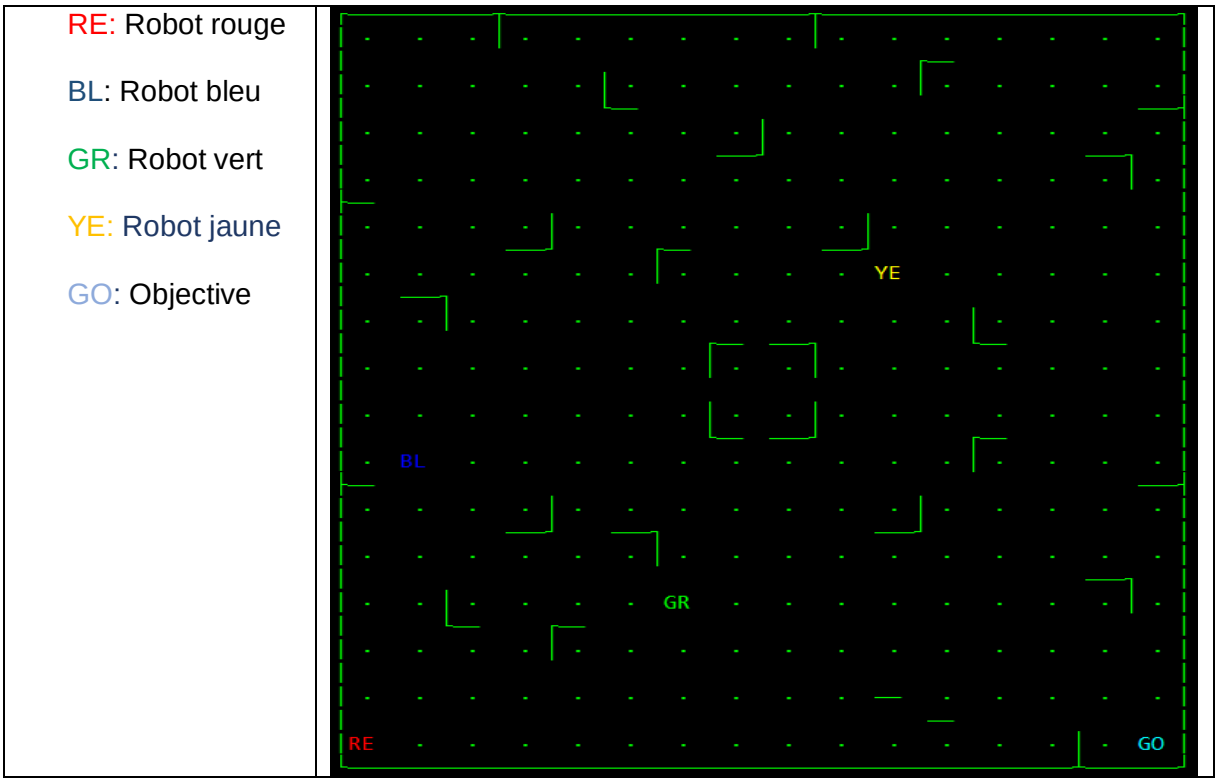


Figure 2 : Carte du jeu Ricochet Robots générée

- Tout d'abord, on place des murs autours de la grille, ainsi les murs forment le carré du milieu.
- On place deux murs extérieurs, un coté vertical et un coté horizontal.
- On ajoute ensuite les murs intérieurs, et je place les angles constitués de deux murs.
- Une fois que les murs extérieurs et les angles sont placés, on obtient une carte contenant 17 angles et 8 murs extérieurs comme sur la grilles du jeu.
- Enfin, il ne reste plus qu'à placer les robots.

V. L'INTELLIGENCE ARTIFICIELLE

Une fois que la carte du jeu est générée, nous avons commencé à travailler sur l'intelligence Artificielle du jeu. Le but de cette IA est de trouver en un minimum de temps une solution optimale à une situation dans Ricochet Robots. C'est-à-dire qu'à partir de la disposition du plateau de jeu et de la position de chaque robot, l'IA doit fournir en un temps raisonnable une liste de coups à effectuer afin de mener le robot principal à son objectif et présenter le chemin la plus courte. La solution doit être optimale, la liste fournie doit donc être la plus petite possible.

1. L'algorithme utilisé

Après une langue recherche des informations du jeu Ricochet Robots, nous avons découvert un algorithme de parcours en langueur (en anglais BFS : Breadth First Search). Cet algorithme permet de parcourir tous les nœuds en fonction de leur distance à l'origine. Il

explore les cartes par cercle concentriques de plus en plus grands. Il fonctionne sur tout type de graphe, cyclique ou non, orienté ou non.

Le BFS procède de la façon suivante :

- Il prend le premier nœud (distance 0)
- traite tous les nœuds qui sont à une distance de 1
- traite tous les nœuds à une distance de 2
- puis tous les nœuds à une distance de 3, et ainsi de suite.

Prince de l'algorithme :

1) On prend le premier nœud de la fille d'attente des nœuds à traiter.
2) Pour visiter ce nœud je le marque comme visité.
3) Chacun de ses voisins non visités et je les ajoute à la fin de la file d'attente des nœuds à visiter
4) On reprends le 1) tant qu'il reste des nœuds à traiter dans la file d'attente

```
Procedure BFS(Graphe G, Noeud Origine)
{
   File aTraiter
   Marquer Origine comme visite
   aTraiter.enfiler(Origine)

   Tant que aTraiter.nonVide()
   {
      Noeud N = aTraiter.defiler()

      Pour chaque voisin V de N
         Si V non visite
            Marquer V comme visite
            aTraiter.enfiler(V)
   }
}
```

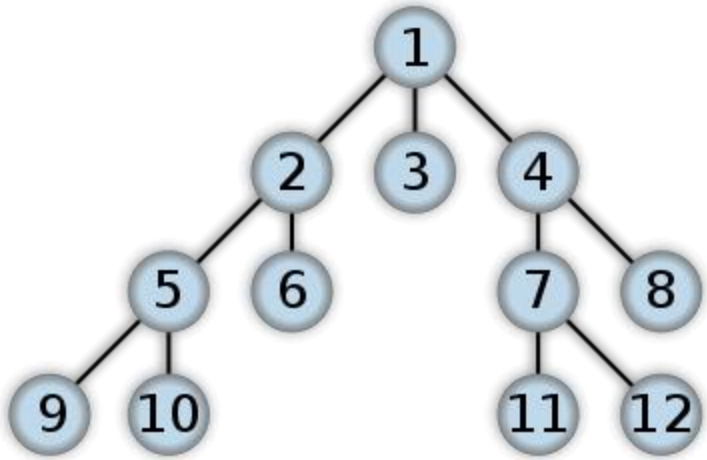


Figure 3 : Fonctionnement du BFS

2. Intégration du BFS au jeu Ricochet Robots

Dans le cadre de ricochet Robots, les nœuds du graphe que nous allons parcourir représentent les différents états que peut prendre le jeu au cours d'une partie et les arrêtes représentent les coups qui peuvent être effectués. La racine (nœud à l'origine de tous les autres) est donc l'état initial de la partie. C'est-à-dire la position de tous les pions au début de la partie. En appliquant l'algorithme du BFS, l'intelligence artificielle va ainsi tester toutes les solutions en un coup, puis toutes celles en deux coups, etc. En théorie, l'IA est donc sûre de trouver la solution optimale à condition d'avoir le temps et de chercher et d'avoir suffisamment de mémoire.

Le problème de cette méthode, si elle n'est pas optimisée, est que l'IA prendra beaucoup de temps a résoudre le problème. En appliquant l'IA sans optimisation, résoudre un problème de 10 coups peut prendre quelques minutes. D'après l'étude de Michel Fogleman, la majorité des problèmes (prés de 90%) peuvent être résolus en 10 coups. Les situations les plus complexes dans Ricochet Robots peuvent être résolues en 22 coups minimum. Pour un tel problème de l'IA mettrais des siècles avant de trouver la solution.

3. Optimisation de la solution

L'objectif de ce projet est d'amener le robot de la couleur correspondante sur l'objectif en un minimum de coups, je suis obligé ainsi d'optimiser l'IA afin puisse produire un résultat en un minimum de coups possible.

Dans le jeu Ricochet Robots, il existe 16 coups différents. En effet, chacun des quatre robots peut se déplacer en haut, en bas, à gauche et à droite. Par conséquent, appliquer une recherche revient à exécuter chacun de ces coups dans les ordres différentes jusqu'à trouver la solution. Pour un problème qui peut être résolu en x coups, il faut examiner au maximum **1**6^x-1 positions. Ainsi, pour résoudre un problème en 22 coups revient à visiter 19,342,813,113,834,066,795,298,816 positions. Voyant la complexité de nature exponentielle, il devient évident qu'il est nécessaire d'éliminer des cas u.

- Eliminer les cas traiter

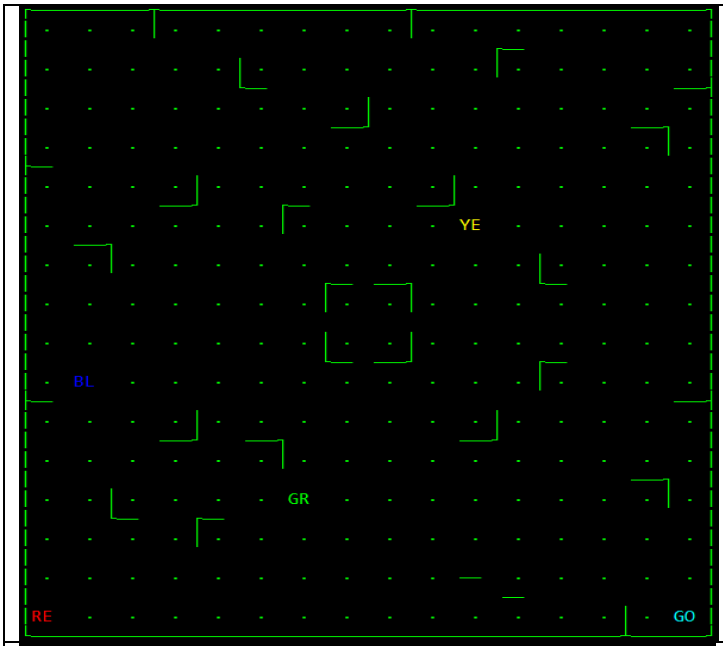
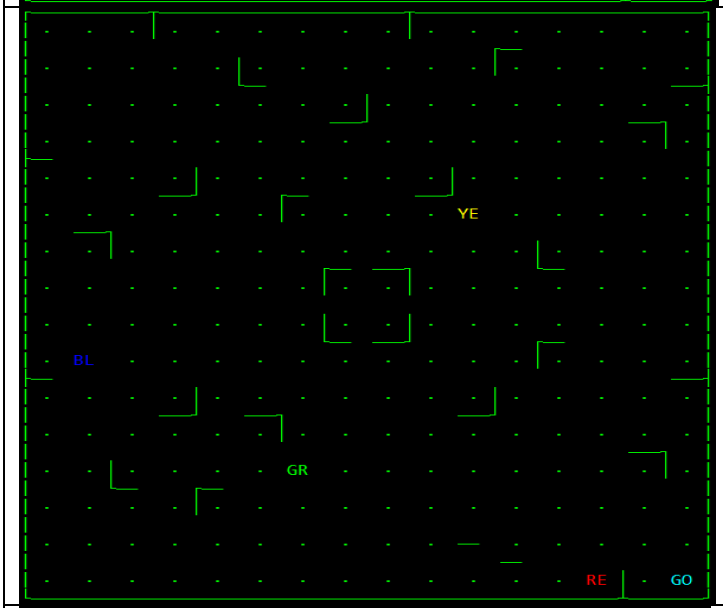
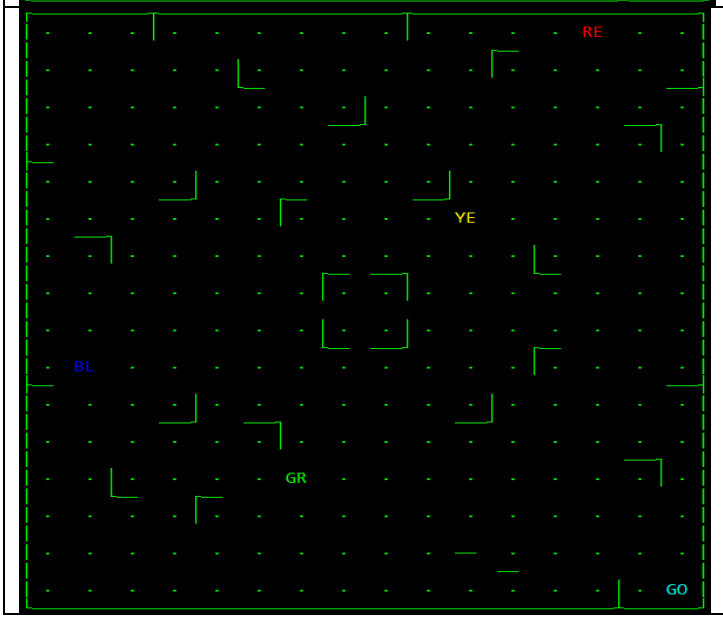
La première optimalisation est d'éliminer les cas déjà traités. C'est-à-dire que lorsqu'un cas traité, l'IA va mémoriser la position de tous les robots. Ainsi, si cette configuration revient plus basse dans la recherche, l'IA saura qu'il est inutile de la traiter de nouveau.

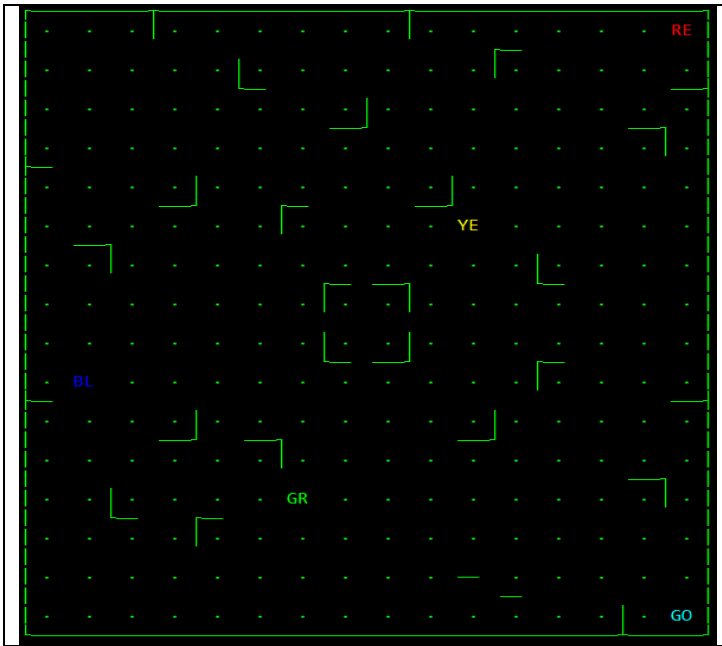
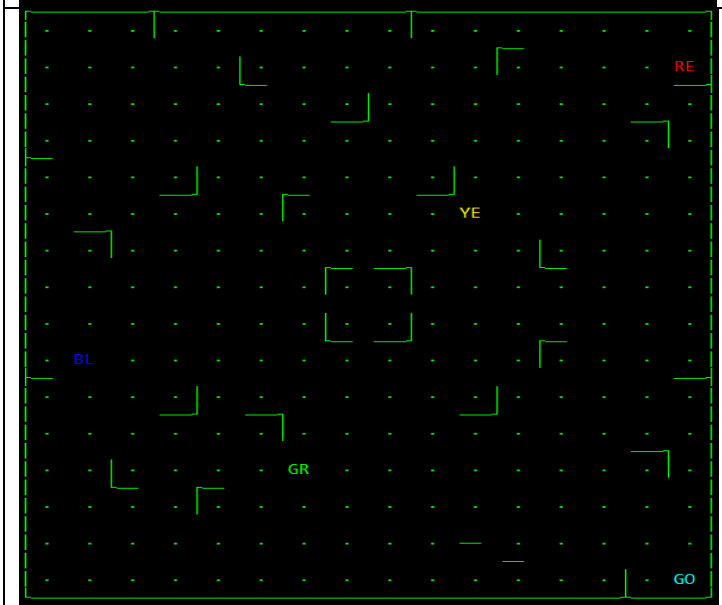
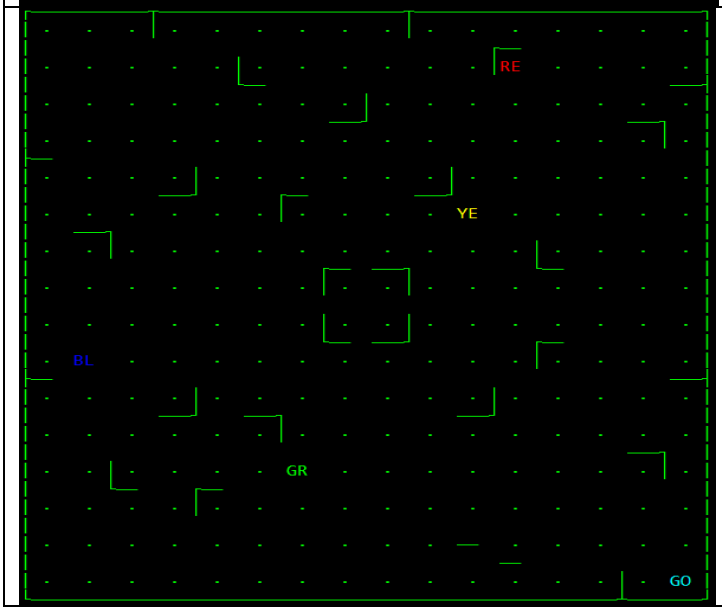
- Ignorer les déplacements du robot qui sont coinsés au même endroit
- Ignorer les déplacements du robot qui nous ramènent à l'emplacement précédent du robot
- Ignorer les déplacements du robot nous amène à un endroit suffisamment exploré
- Créer un nouvel état de déplacement à ajouter dans la file d'attente

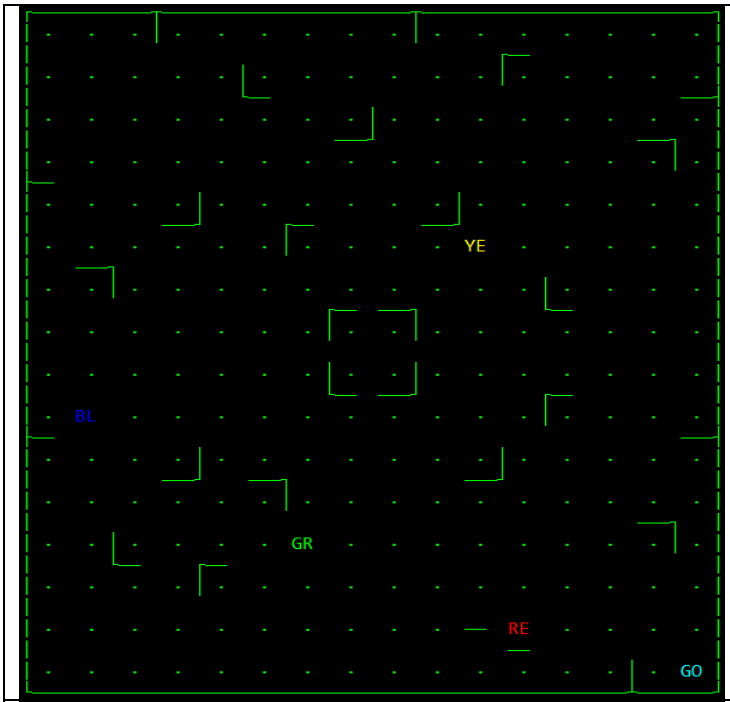
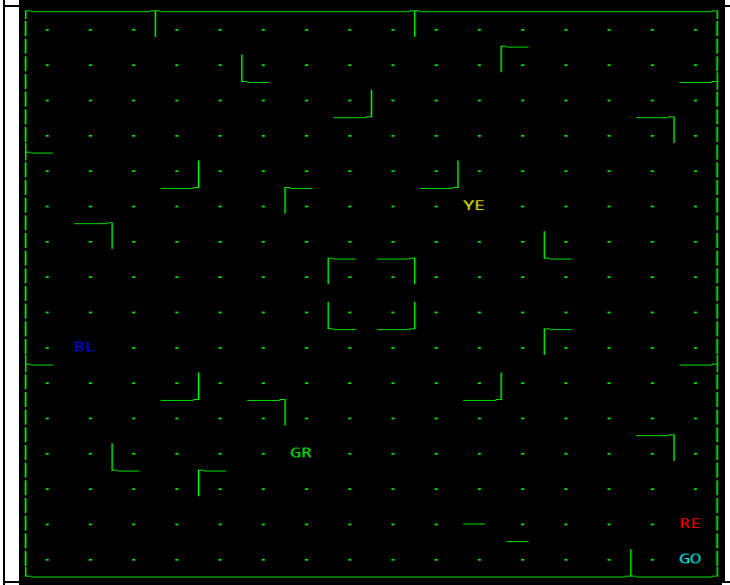
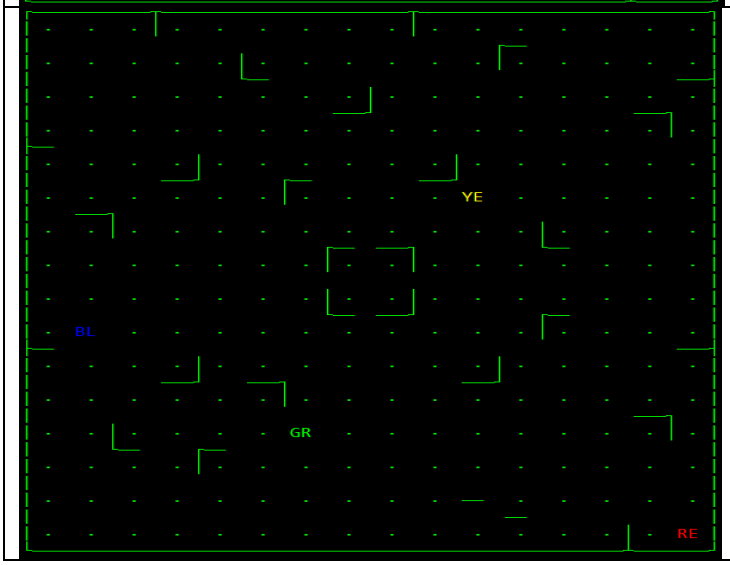
VI. TEST

Commençant par le premier : nous avons défini le robot rouge comme principal de cordonnées

(0, 0) et pour atteindre l'objectif de coordonnées (15, 1).

	Etat initial les robots La position du robot principale « rouge » de coordonnées (0,0) {'robots': {'red': (0, 0), 'green': (6, 3), 'blue': (1, 6), 'yellow': (10, 10)} **'COST': 0**
	Le premier déplacement à droite du robot principale « rouge » qui a changé de position de (0, 0) à (13 ,0) et s'arrêté au premier mur {'robots': {'red': (13, 0), 'green': (6, 3), 'blue': (1, 6), 'yellow': (10, 10)} **'COST': 1**
	Le deuxième déplacement en haut du robot principale « rouge » qui a changé de position de (13, 0) à (13 ,15) et s'arrête au deuxième mur rencontré {'robots': {'red': (13, 15), 'green': (6, 3), 'blue': (1, 6), 'yellow': (10, 10)} **'COST': 2**

	Le 3ème déplacement en haut du robot principale « rouge » qui a changé de position de (13, 15) à (15 ,15) et s'arrête au 3ème mur rencontré {'robots': {'red': (15, 15), 'green': (6, 3), 'blue': (1, 6), 'yellow': (10, 10)} **'COST': 3**
	Le 4ème déplacement en haut du robot principale « rouge » qui a changé de position de (15, 15) à (15 ,14) et s'arrête à la rencontre du 4ème mur rencontré { {'robots': {'red': (15, 14), 'green': (6, 3), 'blue': (1, 6), 'yellow': (10, 10)} **'COST': 4**
	Le 5ème déplacement en haut du robot principale « rouge » qui a changé de position de (15, 14) à (11 ,14) et s'arrête au 5ème mur rencontré {'robots': {'red': (11, 14), 'green': (6, 3), 'blue': (1, 6), 'yellow': (10, 10)} **'COST': 5**

	Le 6ème déplacement en haut du robot principale « rouge » qui a changé de position de (11, 14) à (11 ,1) et s'arrête au 6ème mur rencontré {'robots': {'red': (11, 1), 'green': (6, 3), 'blue': (1, 6), 'yellow': (10, 10)} **'COST': 6**
	Le 7ème déplacement en haut du robot principale « rouge » qui a changé de position de (11, 1) à (15 ,1) et s'arrête au 7ème mur rencontré {'robots': {'red': (15, 1), 'green': (6, 3), 'blue': (1, 6), 'yellow': (10, 10)} **'COST': 7**
	Le 2ème déplacement en haut du robot principale « rouge » qui a changé de position de (15, 1) à (15 ,0) et s'arrête l'objectif correspond {'robots': {'red': (15, 0), 'green': (6, 3), 'blue': (1, 6), 'yellow': (10, 10)} **'COST': 8**

Dans ce test, l'IA a fait que déplacer le robot principal sans bouger les autres robots, pour atteindre l'objectif final, elle a fait 8 coups. En optimisant l'IA on doit bouger les autres robots pour aider le robot principal à atteindre son objectif au minimum du coup, on arrive à un résultat plus optimal de 4 coups.

VII. Conclusion

Pour conclure, nous pouvons tout d'abord dire que ce projet s'est révélé autant intéressant au niveau des connaissances qu'au au niveau du défi. En effet, au début du projet, on n'a pas assez de connaissances pour la programmation Python et aussi les règles du jeu Ricochet Robots. Ainsi, ce projet nous a permis de nous former en Python et enrichir nos compétences dans ce langage. De plus, le projet étant centré autour du développement d'une intelligence artificielle pour un problème compliqué, nous avons été en mesure de mettre à l'épreuve nos capacités. Travailler en équipe a également été une expérience enrichissante lorsque nous ne travaillons pas seul.

L'objectif de ce projet était de développer une solution pour recoudre le jeu Ricochet Ribots tout en implémentant une intelligence artificielle capable de résoudre et amener le robot de la couleur correspondante sur l'objectifs en un minimum de coups. Bien que nous ayons été capable de faire marcher l'intelligence artificielle, cet aspect peut néanmoins être améliorés.

Source du code : https://github.com/khaled44000/Ricochet-Robot