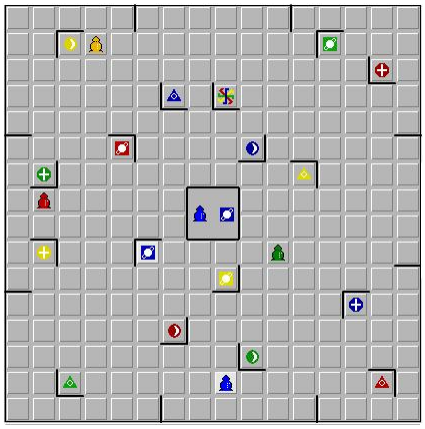
UNIVERSITÉ
CAEN
NORMANDIE

Université de Cean Normandie — UFR des sciences

Rapport sur Ricochet Robot



Réalisé par :

Kabori Hamza

Yassine El Harrab

Mohcine Tkhilla

Ahmad Faez Nasri

Ayoub Mazouak

Groupe 2A

Contents

1 Introduction

Durant notre projet d'année L2, nous nous intéressons à un jeu de société qui existe sous le nom de Ricochet Robots.

Ricochets Robots est un jeu de société créé par Alex Randolph et illustré par Franz Vohwinkel, édité en 1999 par Hans im Glück / Tilsit.

Le jeu est composé d'un plateau, de tuiles représentant chacune une des cases du plateau, et de pions appelés « robots ». La partie est décomposée en tours de jeu, un tour consistant à déplacer les robots sur un plateau afin d'en amener un sur l'une des cases du plateau. Les robots se déplacent en ligne droite et avancent toujours jusqu'au premier mur qu'ils rencontrent et on peut aussi bien y jouer seul qu'à un grand nombre de participants.

Comme c'est le cas pour beaucoup de jeux de société, il est possible de réaliser une version virtuelle du jeu, que ce soit sur ordinateur ou sur tablette/smartphone. Notre premier objectif durant le projet est donc de réaliser une version de Ricochet Robots utilisable sur PC.

Ce projet nous a donc amené à travailler sur des connaissances qui sont nouvelles pour nous, que ce soit JAVA jusqu'aux algorithmes de résolution.
Ce rapport a pour objectif de présenter comment nous avons travaillé et quels ont été les résultats de ce projet. A travers ce document, nous présenterons dans un premier temps le jeu et le développement sous JAVA. Ensuite, nous présenterons l'application que nous avons réalisée suivi de l'algorithme de résolution.

2 Organisation De Projet

Afin de faciliter la communication et le bon développement de notre conception de jeu, divers moyens ont été utilisés.

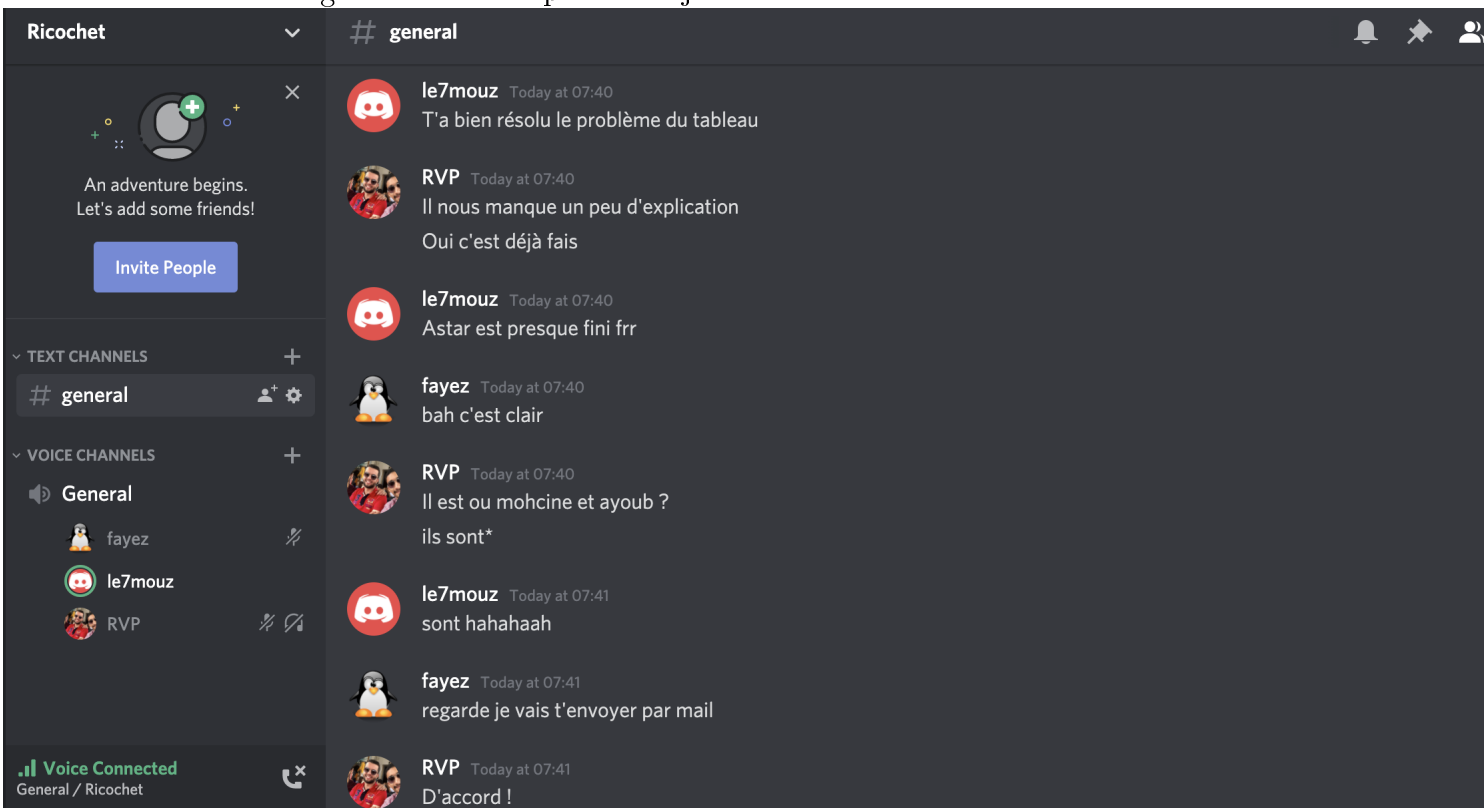
2.1 Gestion Du projet

Si l'on souhaite découper le projet en différentes parties on pourrait discerner 3 taches majeures :

- La partie étude : Une bonne partie de notre travail au début du projet fut consacré à rechercher beaucoup d'informations. Nous devions en effet nous informer sur le jeu lui-même Enfin, nous avons passé du temps à apprendre comment programmer sur JAVA ce fut pour nous l'occasion d'apprendre à programmer sur ce langage.
- La partie réalisation du jeu : Une fois que nous découvert les bases de la programmation Java, nous avons commencé à programmer l'application jeu.

2.1.1 Communication(Discord)

Pour faciliter la communication au sein du groupe, nous utilisons le service de messagerie Discord car tout le monde dans le groupe l'a déjà utilisé personnellement. Ce service vous permet de parler à travers un "serveur" gratuit où nous pouvons ajouter du texte ou des salons vocaux à volonté.

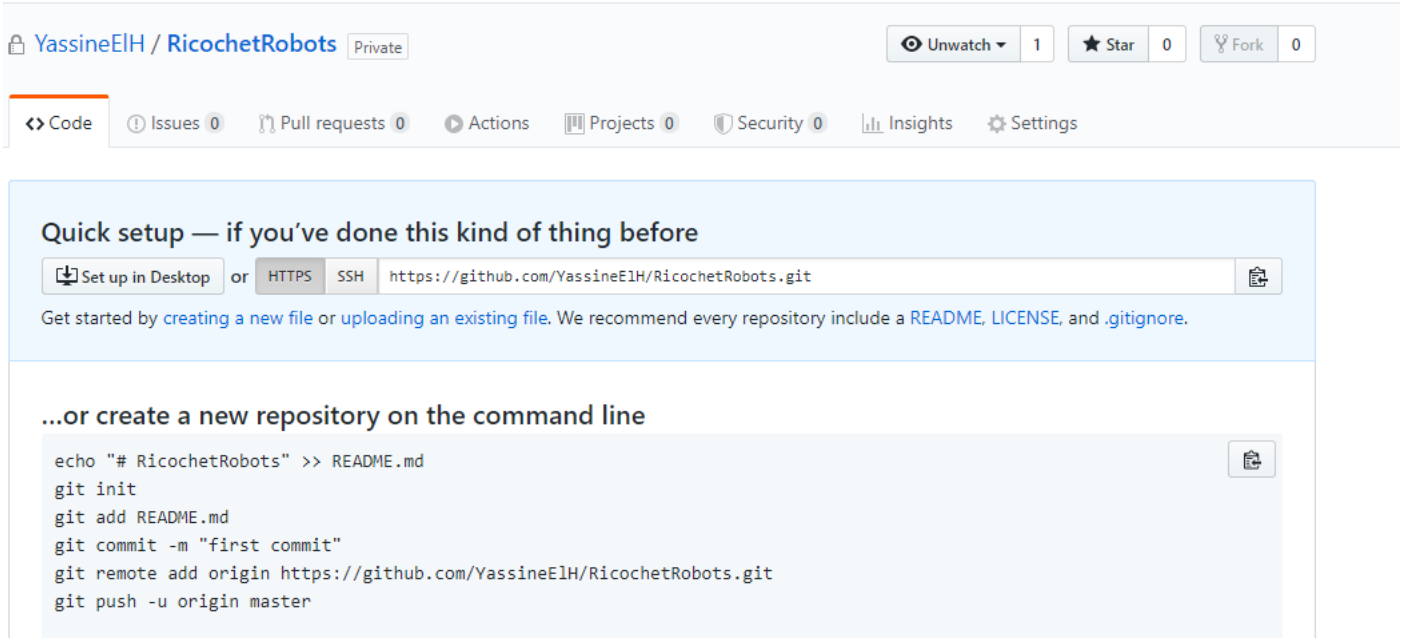


2.1.2 Hebergement du code

Pour faciliter la gestion de projet, nous avons utilisé à la fois ForgeUNICAEN et GitHub Cela facilite la création et la gestion de référentiels dans Git en Interface Web. D'autres fonctions sont également fournies sur ces plateformes, telles que Gestion des autorisations, visualisation des différentes soumissions, visualisation des activités L'utilisation supplémentaire de GitHub comme les projets permet de centraliser les projets Licence sur une seule plateforme.

2.1.3 Gestionnaire du vision(GIT-SVN)

Nous utilisons un gestionnaire de version pour centraliser et rendre le travail d'équipe plus efficace. Nous avons choisi Git, qui est le gestionnaire de versions que nous avons utilisé dans le projet précédent (FilRouge + Taquin). L'utilisation de Git facilite l'utilisation des branches et permet les validations sans connexion au serveur. Lorsque nous avons déterminé que la fonction ajoutée pouvait fonctionner normalement, cela a permis de faire plus de soumissions, qui ont été enregistrées localement et immédiatement envoyées toutes au serveur.



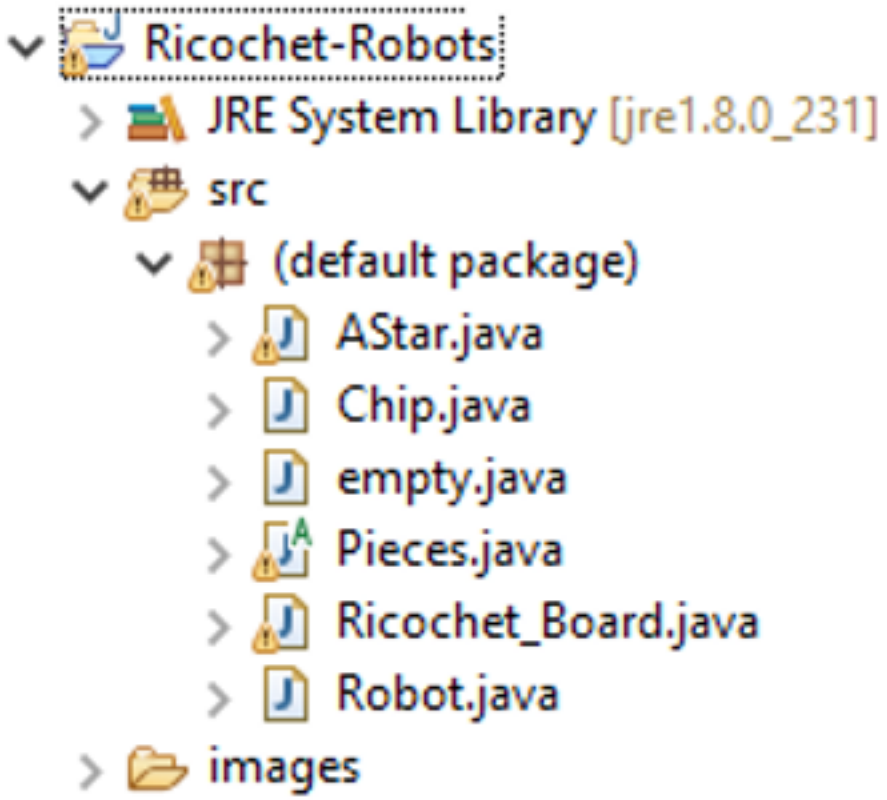
2.2 Répartition des taches

Concernant les membres du groupes on a éssayé a bien partitionné les taches d'une façon équilibré:

Tâches	Yassine	Mohcine	Ayoub	Hamza	Faez
Plateau(Ricochet_board) Création de la classe Plateau			X		
Création des quarts de plateau	X				
Assemblage des quarts de plateau		X	X		
Génération aléatoire du plateau		X			
Rotation des quarts de plateau					X
Affichage graphique du plateau	X			X	
Robot Création de la classe		X			X
Positionnement aléatoire des robots			X		
Collisions des robots avec les éléments					X
Déplacement des robots			X		X
Sélection d'un robot				X	
Affichage des robots et des socles				X	
Jeton (Chip) Création de la classe Jeton	X				
Génération aléatoire du jeton tiré		X	X		
Affichage graphique du jeton tiré	X				
Empty Création de la classe Case Jeton	X				
Positionnement des jetons		X			
Rotation des cases jeton				X	
Affichage graphique des cases jetons				X	
Pieces Implémentation du pattern observer	X				X
Score Création de la classe			X		X
Création du système			X		X
Deplacement (N'a pas marché)					
Création de l'état initial du jeu		X			
Séléction du robot qui doit jouer				X	
Définition de l'état gagnant		X		X	
Algorithme AStar Algorithme BFS	X			X	
Rapport Rédaction du rapport	X	X	X	X	X

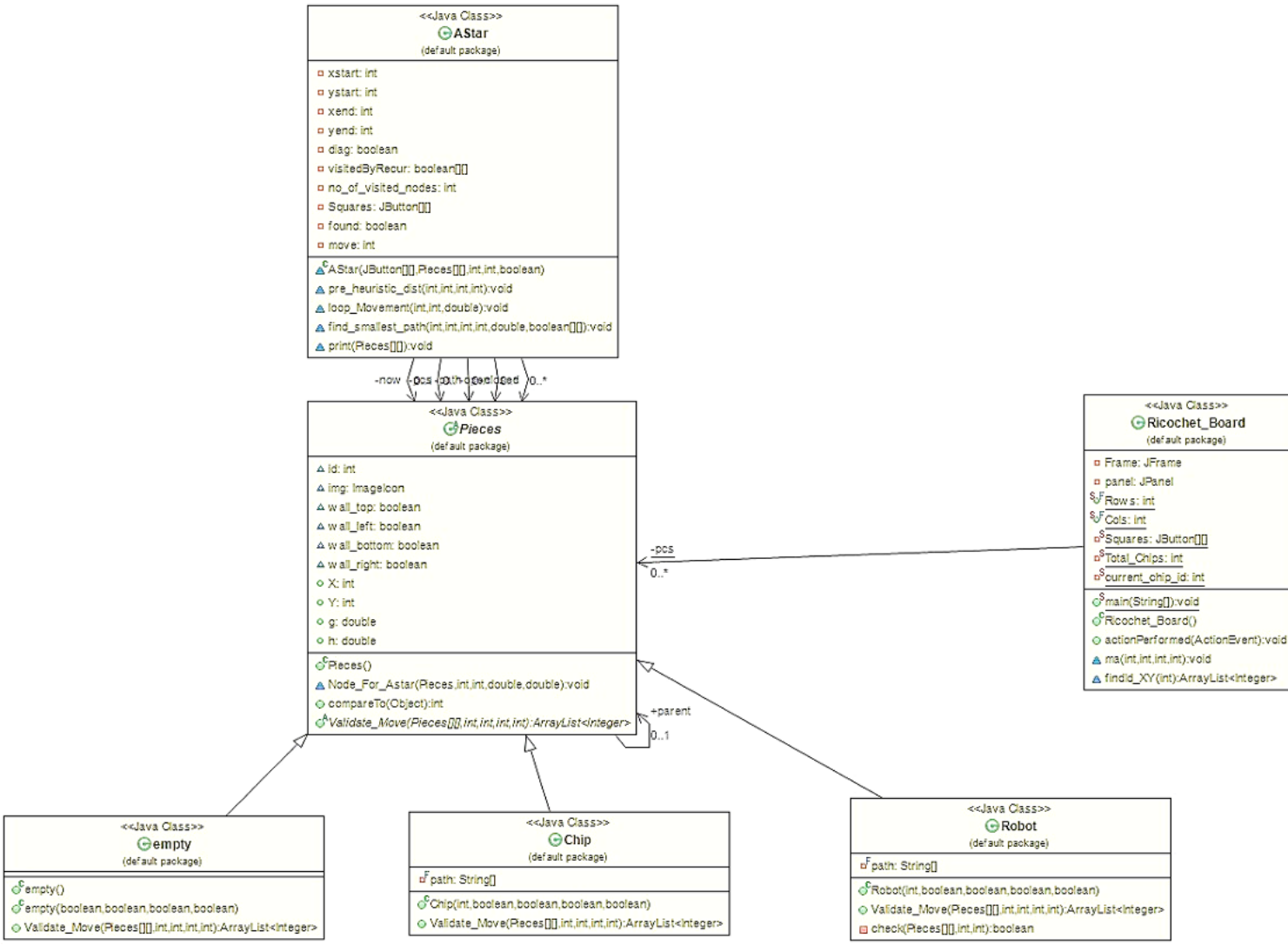
3 Architecture

3.1 La structure du projet



Le dossier Ricochet-Robots nous y retrouvons :
Src : contient le code (le package et les classes)
Images : contient des images qui vont servir à l'interface graphique de jeu
Doc : contient ce rapport et le diaporama pour la soutenance.

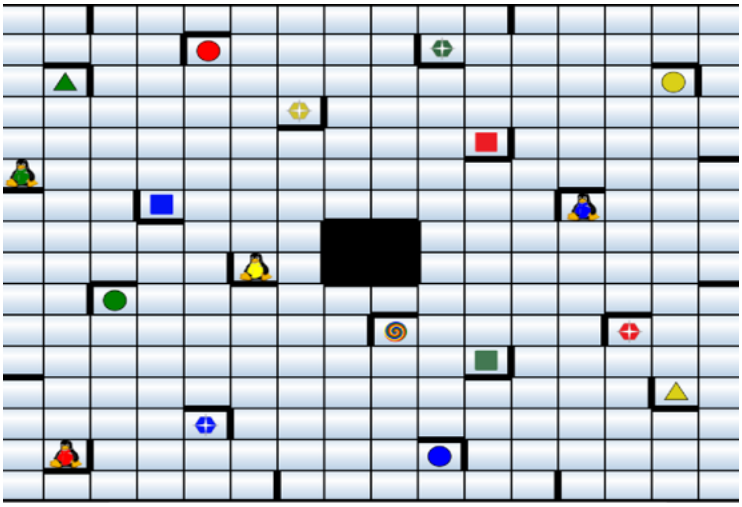
3.2 Diagramme des classes



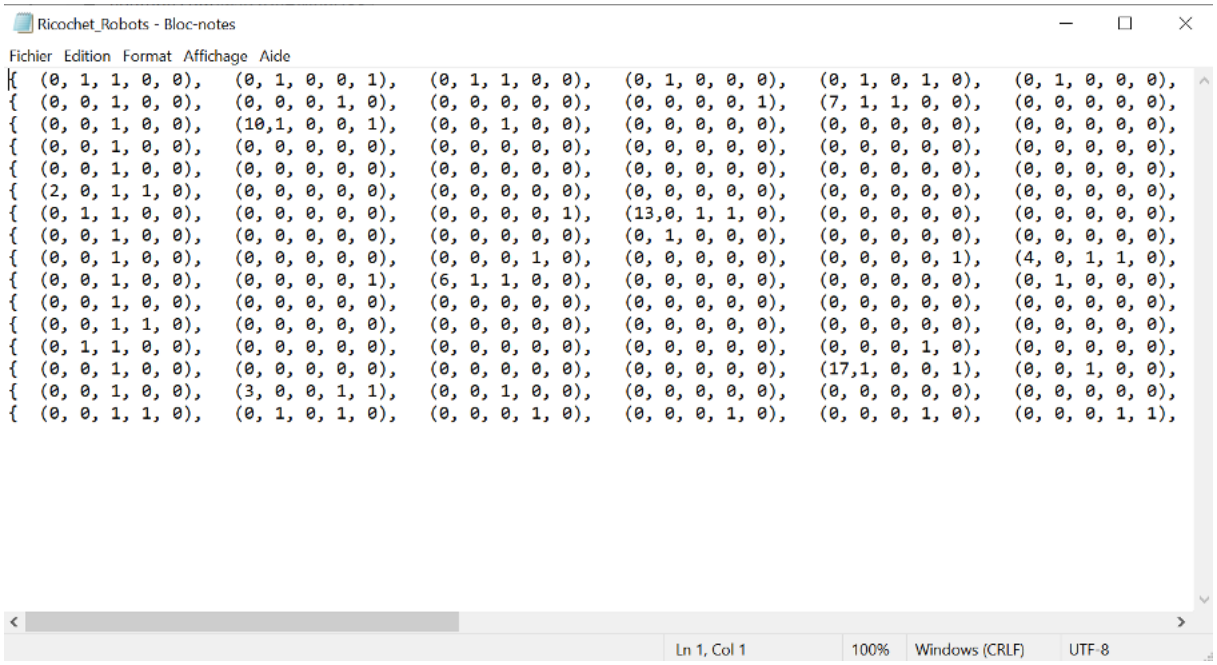
4 Le Jeu

4.1 La map des pions

Création du plateau : Le plateau du Ricochet Robot est un ensemble de "mini-plateaux" qui, une fois assemblés, forment ce plateau. Dans la version du jeu de société, il existe quatre morceaux de plateaux, chaque morceau ayant deux faces. Dans notre version, nous avons voulu construire ce plateau en un seul morceau ayant la forme d'un carré 16x16, qui était suffisant pour la réalisation du plateau du Ricochet Robot.



Les cases du plateau : Chaque case de ce plateau représente un bouton caractérisé par un entier placé en premier et 4 autres booléens (1,1,0,0,0). L'entier contient l'id, le premier booléen représente l'état du délimiteur supérieur (si ce booléen est à 1 ça veut dire que cette case contient un mur supérieur), le deuxième booléen représente l'état du délimiteur à gauche, le troisième représente l'état du délimiteur inférieur et le dernier représente l'état du délimiteur à droite. Toutes ces données sont enregistrées dans un fichier qui consiste à initialiser le plateau selon ces données.



Les jetons Pour les jetons on a créé une classe Chip qui décrit chaque jeton par 5 caractéristiques, qui sont : l'id, et les quatres délimiteurs (Haut, Gauche, Bas, Droit). On a associé à chaque jeton une image indépendante pour le différencier des autres. Un jeton est placé sur une case du plateau donc c'est un bouton.

```
public Chip(int ChipId, boolean top, boolean left, boolean bottom, boolean right) throws IOException{
 id = ChipId;

 img = new ImageIcon(ImageIO.read(new File(path[ChipId-5]))
            .getScaledInstance(30, 30, Image.SCALE_SMOOTH));//-5 because id sent from 5

wall_top = top;
wall_left = left;
wall_bottom = bottom;
wall_right = right;
h = -1;
}
```

4.2 Les Robots

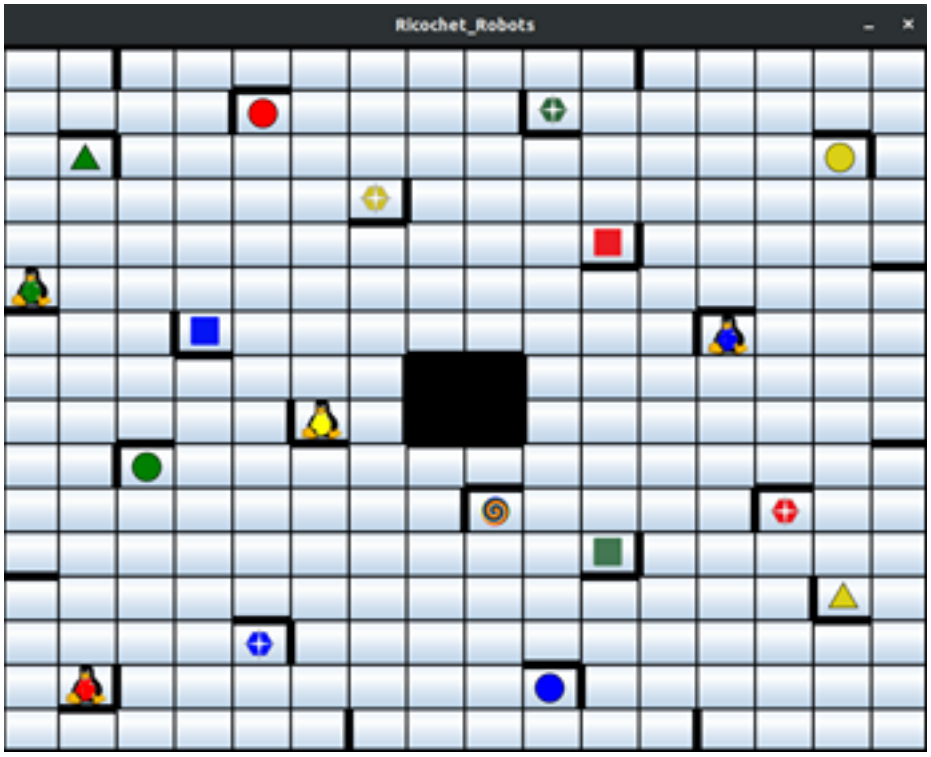
On a 4 robots (rouge, vert, bleu, jaune) qui sont définis dans la classe Robot par 5 caractérisitiques : L'id et les 4 délimiteurs. Ces derniers ont une relation avec les délimiteurs de la case actuelle qui fait une copie de ses délimiteurs et vérifie si le robot peut passer par un certain chemin ou pas.

```
public Robot(int RoboId,  boolean top, boolean left, boolean bottom, boolean right) throws IOException{
    id = RoboId;

    img = new ImageIcon(ImageIO.read(new File(path[RoboId-1]))
            .getScaledInstance(45, 38, Image.SCALE_SMOOTH));//-1 because id sent from 1

    wall_top = top;
    wall_left = left;
    wall_bottom = bottom;
    wall_right = right;
    h = -1;
}
```

Ces robots ne sont pas placés arbitrairement mais ils sont placés selon les données figurant sur le fichier. Ce préplacement consiste à éviter la surcharge d'une case par plusieurs jetons ou par un robot et un jeton.

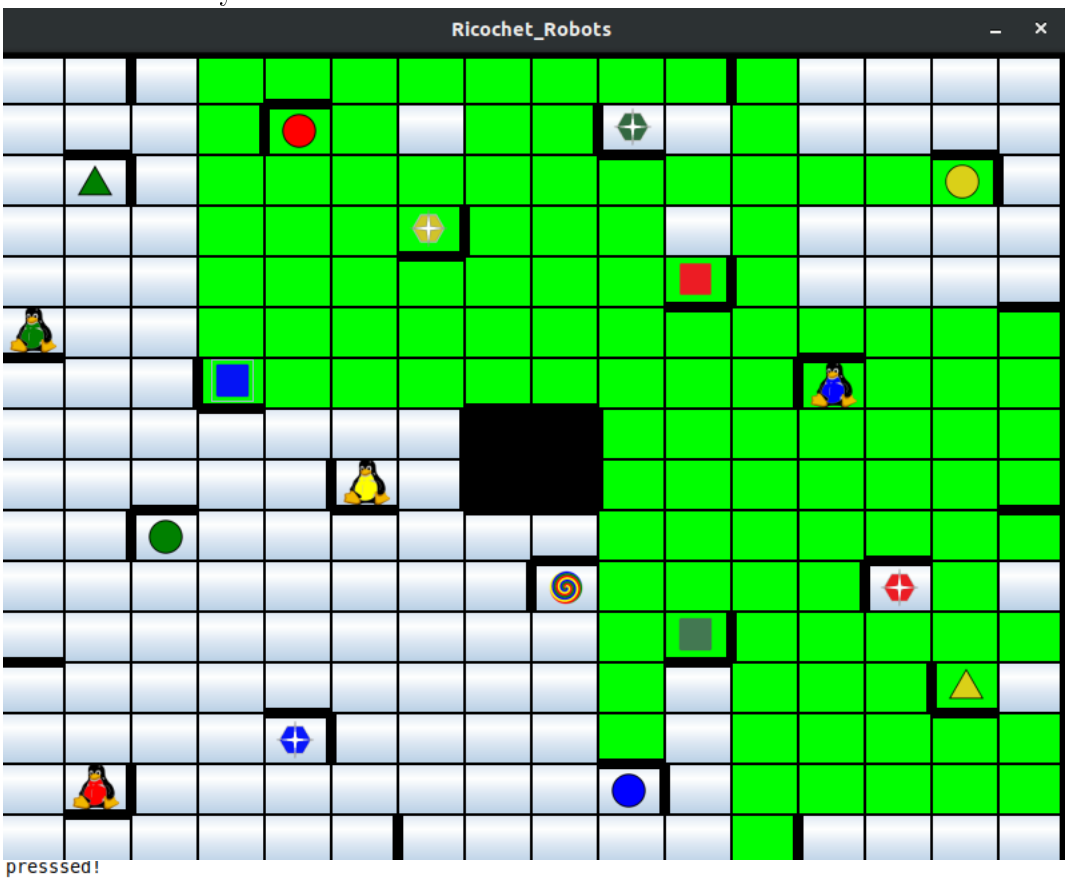
4.3 Le déroulement du jeu

Pour récapituler ce qui ce passe dans le jeu, comme fonctionnement on a le jeu dans l'état initial qui s'éxecute avec le fichier .jar, aprés on a comme interface graphique les quatre tableau créé dans la classe Ricochet Board qui contient 16 lignes et 16 colonnes, qui sont connécté les uns les autres avec une fonctions listener, cette dérnière qui capte le bouton dont on a cliqué.



Et des murs qui sont généré avec la classe Empty , qui utilise un fichier text qui contient plusieurs liste de 5 digits, ces digits signifie l'emplacement des murs, les robots et leurs chip.Le premier digit signifie la couleur du robot (bleu:1,vert:2,rouge:3,jaune:4).s'il est 5 c'est le ID du chip,les autres quatres digit sont en binaire soit 1 ou 0,alors la liste est constitué de (ID,HAUT,GAUCHE,BAS,I

veut dire qu'il y'a la mur dans ce coté la et vice versa.Après, en cliquant sur le robot, la distance heuristic est calculé en utilisant la récursivité.Malheureusement on peut pas jouer plusieur partie à la foie ,il faut redemarrer le jeu pour rejouer.La distance Heuristic nous donne le minimun mouvements pour y arriver , mais le robot ici fait plusieurs mouvements parce que à cause de l'algorithme du jeu les mouvements sont calculés mais ici en Ricochet robot, LE robot ne peut arréter sauf s'il y'avait des obstacles ou murs.




```
presssed!
13
heuristic distance:
3 3 2 1 2 2 2 2 2 2 2 2 3 3 3 3
2 2 2 1 2 2 3 2 2 3 3 2 3 3 3 3
3 4 2 1 2 2 2 2 2 2 2 2 2 2 2 6
2 2 2 1 2 2 2 2 2 2 3 2 3 3 3 3
2 2 2 1 2 2 2 2 2 2 2 2 3 3 3 3
2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2
4 3 3 0 1 1 1 1 1 1 1 1 4 3 3 3
3 3 3 3 2 2 2 0 0 2 2 2 3 3 3 3
3 3 3 3 2 2 2 0 0 2 2 2 3 3 3 3
4 3 3 3 2 3 2 3 3 2 2 2 3 3 3 3
3 3 3 3 2 3 2 3 3 2 2 2 3 5 3 4
3 3 3 3 2 3 2 3 3 2 2 2 3 3 3 3
3 3 3 3 2 3 2 3 3 2 3 2 3 3 3 4
4 3 4 4 4 3 2 3 3 2 3 2 3 3 3 3
4 3 3 3 3 3 2 3 3 3 3 2 3 3 3 3
4 5 4 4 4 4 2 3 3 3 3 2 4 4 4 4
```

5 L'Algorithme A*

L'algorithme A* est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final. Il utilise une évaluation heuristique sur chaque nœud pour estimer le meilleur chemin y passant, et visite ensuite les nœuds par ordre de cette évaluation heuristique. Dans notre cas les noeuds représentent un état du jeu et nous definissons une relation d'ordre qui permet de choisir un noeud plutôt qu'un autre. Cette relation d'ordre est définie par la fonction Eval(noeud) On a Eval(noeud) = g(n) + h(n) avec : — g(n) :le coût du chemin entre le noeud de départ et le noeud n — h(n) :le coût estimé du chemin donné par l'heuristique g(n) est très facile à définir. Dans notre cas il est stocké dans nos états sous la forme "Nombre de déplacement déja effectués". Mais en ce qui concerne l'heuristique cela est plus compliqué car il est démontré que l'algorithme A* est optimal si h(n) est une heuristique admissible. Cela signifie que h(n) ne doit jamais surestimer le coût pour atteindre le but alloué à un noeud. De plus il a aussi été montré que l'algorithme A* est optimal si h(n) est consistante. Une heuristique est dite consistante si, pour tout noeud n et pour tout successeur n(prime) de n généré par une action a, le coût estimé pour atteindre le but en partant de n n'est pas supérieur au coût de l'étape pour aller à n(prime)plus le coût estimé pour atteindre le but à partir de n . Nous avons donc essayé plusieurs heuristiques que nous allons détailler ci-dessous.

Concernant notre developpement de cet algorithme pour chaque robot un calcul sera fait avec une création d'un arbre qui determine les mouvements possibles (heuristic moves) avec cela on peut lire la profendeur de l'arbre et comme ça la recherche sera le plus vite possible

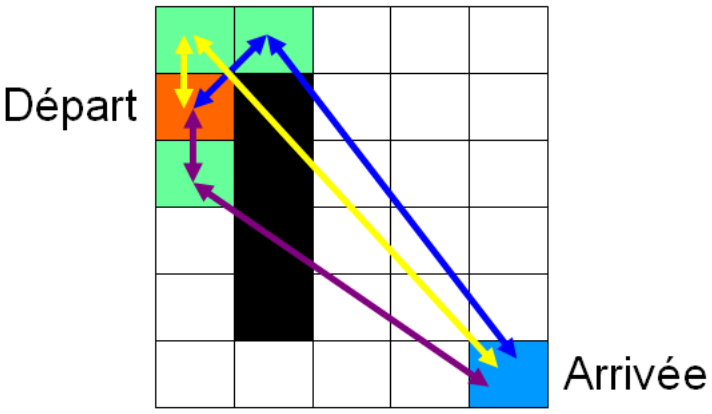


Figure 1: Fonctionnement de A*

6 Conclusion

6.1 Améliorations Possibles

Si ce projet devait être à refaire, nous adopterions dès le début du projet des normes de codage à respecter impérativement. En effet, il nous est arrivé bien trop souvent que l'application ne fonctionne pas de la façon désirée car, travaillant tous les cinq sur le même code et de plus les problèmes du confinement. Certains éléments n'étaient pas implémentés de la même façon partout. Par exemple, à un moment, nous avions des conventions différentes pour exprimer la couleur des pions. De plus, nous adopterions également une organisation plus modulaire au projet. Cela nous permettrait de modifier des bouts de code sans pour autant casser tout le projet. Finalement, le développement de l'application serait bien plus rapide car, à présent, nous connaissons les contraintes du développement JAVA. Ainsi, nous aurions nettement plus de temps pour optimiser et ajouter de l'intelligence artificielle.

6.2 Résumé

Dans l'ensemble, nous pouvons d'abord dire que ce projet est tout aussi intéressant en termes de connaissances et de défis. En effet, au début du projet, on commençait tout juste à utiliser la programmation Java. Par conséquent, ce projet nous permet de mener une formation JAVA et de consolider ses capacités en ce dernier. De plus, le projet s'articule autour du développement d'un jeu complexe, et nous sommes en mesure de tester nos capacités. Travailler en équipe est également une expérience enrichissante car elle nous permet de découvrir les limites que nous pouvons rencontrer lorsque nous ne travaillons pas seuls.