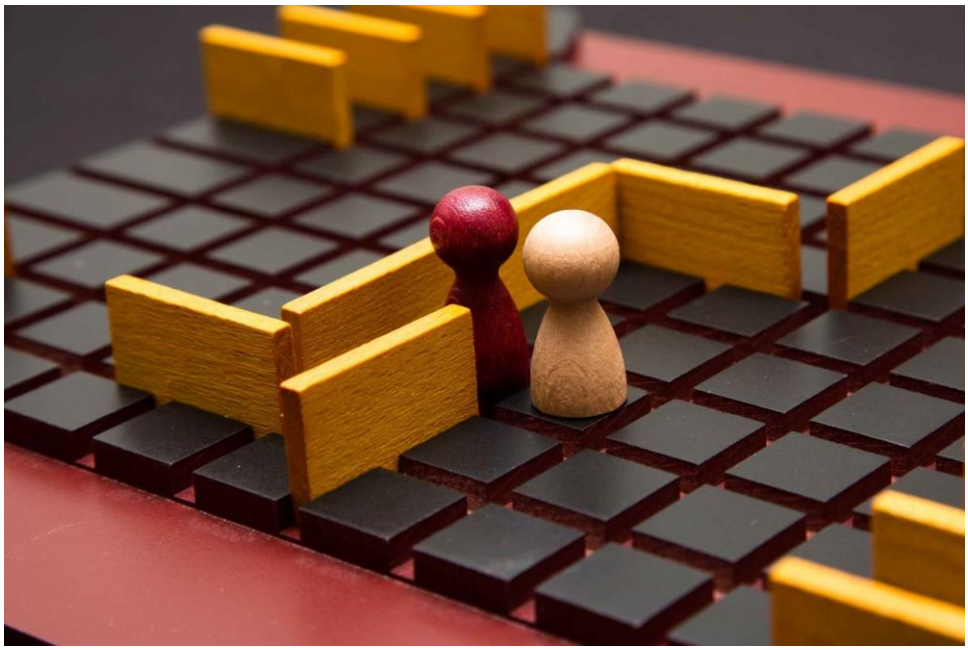
JEU DU QUORIDOR

Réalisé par : ***Ertugrul SEBUKHAN (3803088)***
Yasmine *AZ*IB (28618109)

Encadré par : **Yoones MIRHOSSEINI**

Année universitaire : **2022/2023**



SORBONNE UNIVERSITÉ

Partie1 : README

Nous avons organisé notre code de la manière suivante :

Une classe S0 (la classe mère) :

Elle contient tous les attributs nécessaires à l'initialisation du jeu tels que la position initiale des joueurs, les murs, les objectifs, ...etc. La fonction __init__ construit la grille et initialise les attributs cités ci-dessus. Elle contient aussi des fonctions utilitaires nécessaires pour jouer un coup. Toutes les stratégies héritent de cette classe. On a fait le choix de la surcharger pour alléger le code des autres classes et du main qui lance le jeu.

Quatre classes représentant les stratégies Alea, S1, S2, S3 :

Chaque classe hérite de S0 et définit une méthode 'jouer' implémentant la stratégie dont elle est responsable.

Fichier contenant la méthode Main :

Ce fichier permet de créer le Game (Murs, Joueurs, Grille...) et de créer les stratégies et faire jouer les joueurs à tour de rôle. La création de la stratégie se fait par appel au constructeur en lui passant trois paramètres : Game, identifiant du joueur et une liste composée de sa position initiale et celle de son adversaire.

Partie2 : Description des stratégies

Stratégie 1 :

Le joueur qui fait appel à la stratégie essaie de poser des murs devant et devant à droite de son adversaire, sinon devant et devant à gauche. Si aucune de ces deux conditions n'est vérifiée, il avance selon A*.

Stratégie 2 :

Le même principe que la stratégie 1 mais notre joueur ne pose des murs que s'il est bien parti pour gagner, c'est à dire que si le chemin de notre joueur vers son objectif le plus proche est plus court que le chemin de l'adversaire vers son objectif le plus proche.

Stratégie 3 :

Notre joueur calcule le chemin de l'adversaire via A*. Il calcule la case du prochain saut de ce chemin et met un mur dessus (s'il en a encore). Il pose le deuxième mur à droite du premier s'il peut. S'il ne peut pas, il regarde le saut d'après et recommence le processus jusqu'à ce qu'il ait essayé toutes les cases du chemin de l'adversaire. S'il n'a plus de murs à placer ou s'il n'arrive pas à placer un mur sur le chemin de l'adversaire, il se déplace selon A*.

Partie3 : Comparaison des résultats

Dans ce tableau, le joueur 1 joue en premier. Le joueur qui joue la stratégie random perd toujours, quel que soit la carte, qu'il joue en premier ou non.

MAP	*Joueur 1*	*Joueur 2*	*Vainqueur*
mini	*s1*	*s2*	*s1*
mini	*s2*	*s1*	*s2*
mini	*s1*	*s3*	*s1*
mini	*s3*	*s1*	*s1*
mini	*s2*	*s3*	*s2*
mini	*s3*	*s2*	*s2*
big	*s1*	*s2*	*s2*
big	*s2*	s1	*s1*
big	*s1*	*s3*	*s3*
big	*s3*	*s1*	*s1*
big	*s2*	*s3*	*s3*
big	*s3*	*s2*	*s2*

Nombre de victoires sur la mini map des différentes stratégies :

- S1 : 50%
- S2 : 50%
- S3 : 0%

Nombre de victoires sur la grande map des différentes stratégies :

- S1 : 33%
- S2 : 33%
- S3 : 33%

- S3 est très faible dans la petite map. S3 est meilleure dans la grande map et est meilleure lorsqu'elle est jouée en deuxième.

- S1 et S2 donnent de bons résultats sur la petite map

- Sur la grande map, toutes les stratégies ont le même taux de réussite.

- Quand deux joueurs jouent la même stratégie, c'est le joueur qui fait le premier coup qui gagne.

Remarque :

Nous avons essayé d'implémenter la stratégie MIN-MAX dans le fichier 'minmax.py' mais cela fait des erreurs quand on essaie de l'exécuter.

Conclusion :

Toutes les stratégies que nous avons implémentées donnent de bons résultats, les unes mieux que les autres selon la carte de jeu. Aucune stratégie n'est dominante, ce qui cohérent dans un jeu de stratégie comme le QUORIDOR.