Sorbonne Université
Master ANDROIDE

Algorithmes pour le jeu mastermind

UE de projet M1

Walid Ghenaiet – Romain Noda

2024

Table des matières

Chapitre 1

Introduction

L'objectif de ce projet est d'étudier différentes stratégies existantes pour le jeu Mastermind, de les implémenter et de les comparer à travers des expérimentations numériques. Nous développerons également une interface de jeu permettant à un joueur humain de jouer contre une intelligence artificielle.

La première étape du projet consiste à mettre en œuvre une stratégie optimale pour résoudre le jeu. Nous allons implémenter l'algorithme de Knuth, pour son efficacité en termes de nombre maximum d'essais. Cela signifie qu'il peut deviner n'importe quel code en un maximum de cinq essais. Cette implémentation sera utiliser pour en tirer des statistiques du jeu, produire un adversaire IA capable de résoudre Mastermind de manière efficace.

Nous développerons une interface graphique pour le jeu, comprenant une fonctionnalité permettant aux joueurs d'affronter l'IA pour voir qui peut deviner le code en premier. Cette fonctionnalité propose différents niveaux de difficulté pour l'IA, allant de facile à difficile.

Ensuite, nous introduirons une fonction "Aide" qui offrira aux joueurs une assistance pendant le jeu. Les conseils donnés seront basés sur les propositions précédentes du code-breaker et les indices de feedback donnés par le code-maker.

Enfin, nous concevrons une stratégie de jeu simple que les joueurs humains pourront mémoriser et utiliser pour gagner le jeu et augmenter leurs chances de gagner contre l'adversaire IA.

Chapitre 2

État de l'art

2.1 Mastermind

Le jeu Mastermind est un jeu de déduction classique, publié pour la première fois en 1972. Il se présente généralement sous la forme d'un plateau (voir Figure 1) perforé comprenant 10 rangées de 4 trous, chacun pouvant accueillir des billes de différentes couleurs. Les 6 couleurs disponibles sont le rouge, le bleu, le jaune, le vert, le blanc et le noir. De plus, des petits pions sont utilisés pour indiquer à chaque étape combien de billes sont au bon endroit ou de la bonne couleur.

Il existe de nombreuses variantes de ce jeu, notamment en ce qui concerne le nombre de rangées (10 ou 12) ou de trous (4 ou 5), les couleurs des billes (6 ou 8), ainsi que la possibilité d'inclure la même couleur plusieurs fois dans le même code. Le but du jeu est de trouver le code, une combinaison de couleurs, en un minimum de coups.

Dans ce projet, nous utiliserons une configuration de plateau comprenant 10 rangées de 4 trous, avec un choix de 6 couleurs pour les billes.

FIGURE 2.1 – Le plateau du jeu

2.2 Knuth

Dans les années 1976-77, Donald Knuth, un informaticien et mathématicien américain ainsi que professeur à l'université Stanford, a démontré que le jeu pouvait toujours être résolu en un maximum de 5 coups [1].

L'algorithme de Knuth [2] suit les étapes suivantes (les couleurs sont représentés par les numéros de 1 à 6) :

1 Créer un ensemble S contenant les 1296 codes,
2 La proposition initiale est 1122,
3 Jouez la proposition et obtenez la réponse des pions noirs et blancs
4 Si la réponse est 4 pions noirs alors le jeu est fini, l'algorithme s'arrête.
5 Sinon retirez depuis l'ensemble S les codes qui ne donneraient pas la réponse obtenu,
6 La prochaine proposition est choisi par le technique de minimax, le score d'un code est défini par le pire parmi toutes les réponses possibles. Le code choisi sera celui qui a le plus petit score parmi les codes valides si possibles, sinon parmi les 1296 codes.
7 A répéter depuis l'étape 3.

2.3 Swaszek

En 2000, Peter Swaszek propose un algorithme simple en sacrifiant l'optimalité en nombre de coups mais qui donne un résultat assez bonne en nombre de coups moyen.

Algorithme	Tours moyen
Koyama and Lai (Optimal)	4.340
Knuth	4.478
Swaszek	4.638

TABLE 2.1 – Comparison de nombre de tours moyen

L'algorithme de Swaszek [3] suit les étapes suivantes (les couleurs sont représentés par les numéros de 1 à 6) :

1. Créer un ensemble S contenant les 1296 codes,
2. La proposition initiale est 1122,
3. Jouez la proposition et obtenez la réponse des pions noirs et blancs
4. Si la réponse est 4 pions noirs alors le jeu est fini, l'algorithme s'arrête.
5. Sinon retirez depuis l'ensemble S les codes qui ne donneraient pas la réponse obtenu,
6. La prochaine proposition est choisi aléatoirement parmi les codes valides,
7. A répéter depuis l'étape 3.

Chapitre 3

Contribution

3.1 Language d'implémentation

Pour la réalisation de ce projet, il a été essentiel de choisir les outils adéquats. Nous avons décidé de le développer sous forme d'application web afin de séparer le code de l'interface de notre code logique. Pour la partie intelligence artificielle et les algorithmes, nous avons opté pour Python en raison de sa simplicité et de la disponibilité de bibliothèques telles que NumPy. Nous avons également choisi Flask pour l'utiliser comme passerelle API afin de connecter l'interface à la logique. Pour l'interface, nous avons opté pour JavaScript et Vue.js car cela nous a fourni une bonne structure de code tout en le gardant léger.

3.2 Implémentation des méthodes

3.2.1 L'arbre de Knuth

L'arbre de jeu se construit en se basant sur la stratégie de Knuth [1], cet arbre sera appelé l'arbre de Knuth dans la suite. Chaque nœud correspond à une situation du jeu, et une situation est représenté par les notations suivantes :

Soit n le nombre de codes valides,

- n, si $n \leq 2$,
- $n(y_1, y_2, y_3, y_4)$, si $n > 2$ et que la réponse au proposition $y_1y_2y_3y_4$ caractérisera de manière unique le code secret,
- $n(y_1, y_2, y_3, y_4, x)$, si $n > 2$ et que la situation après la proposition $y_1y_2y_3y_4$ ne caractérise pas de manière unique le code secret mais ne dépasse pas plus de 2 possibilités,
- $n(y_1, y_2, y_3, y_4, \alpha_{04}, \alpha_{03}, \alpha_{02}, \alpha_{01}, \alpha_{00}, \alpha_{13}, \alpha_{12}, \alpha_{11}, \alpha_{10}, \alpha_{22}, \alpha_{21}, \alpha_{20}, \alpha_{31}, \alpha_{30}, \alpha_{40})$, si $n > 2$ et que si la proposition $y_1y_2y_3y_4$ auquel répond i pions noirs et j pions blancs conduit à la situation α_{ij}.

Dans la représentation de l'arbre, on affichera que la proposition faite. La racine sera donc un nœud avec '1122' et les successeurs d'un nœud correspondent aux situations ayant eu un réponse de i pions noirs et j pions blancs. Les arcs représentent l'avancement à la situation α_{ij}.

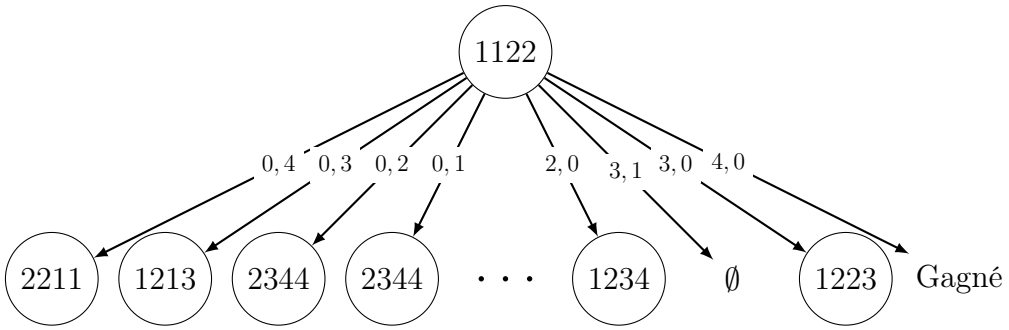

FIGURE 3.1 – Début de l'arbre de Knuth

Supposons que les fonctions suivantes existent :

- Une fonction pour filtrer l'ensemble des codes valides pour chaque situation α_{ij} : $filtrage(V, y_1y_2y_3y_4)$, avec
  - V : l'ensemble des codes valides à une situation,
  - $y_1y_2y_3y_4$: la proposition.
- Une fonction qui détermine la proposition à faire au tour suivant : $minimax(S, V)$, avec
  - S : l'ensemble de 1296 codes,
  - V : l'ensemble des codes valides à une situation.

L'algorithme pour la construction de l'arbre de Knuth est la suivante :

```
1  Function KnuthTree(S,V,y1y2y3y4) :
       Data : un ensemble des 1296 codes S, un ensemble des codes valides V, une
              proposition y1y2y3y4
       Résultat : l'arbre de Knuth
2      listCodesValides = filtrage(V, y1y2y3y4);
3      knuthTree = [] // contient un tuple (n, guess) ou (n, guess, knuthTree);
4      forall Vij ∈ listCodesValides do
5          if n > 2 then
6              next = minimax(S, Vij);
7              subKnuthTree =KnuthTree(S,Vij,next);
8              knuthTree[αij] = (n, y1y2y3y4, subknuthTree);
9          end
10         else
11             knuthTree[αij] = (n, y1y2y3y4);
12         end
13     end
14     return knuthTree;
```

Algorithme 1 : Algorithme de construction de l'arbre de Knuth

3.2.2 Calcul de coup max

A partir de l'arbre de Knuth, on peut calculer le nombre de coup restant pour trouver n'importe quel code valide.

Le code de la méthode est la suivante :

```
def calcul_max_guess_remaining(h, knuthTree):
    maxh = 0
    for alpha in knuthTree:
        if len(alpha) == 2:
            maxh = max(maxh, h+alpha[0])
        else:
            maxh = max(maxh, calcul_max_guess_remaining(h+1, alpha[3]))
    return maxh
```

Cette méthode parcours en profondeur l'arbre de Knuth et retourne l'hauteur de l'arbre, le paramètre h doit être à 0 lorsqu'on appelle la fonction.

3.2.3 Méthodes pour l'IA

Pour le joueur IA, nous avons initialement opté pour l'implémentation de l'algorithme de Knuth, puis laisser l'IA jouer en suivant l'arbre classique. Cependant, nous souhaitions ensuite disposer d'une méthode d'IA ajustable, pouvant être configurée pour offrir différents niveaux de difficulté. C'est là que nous avons décidé d'apporter une variation à la méthode de Swaszek : au lieu de simplement choisir un code valide au hasard parmi les codes possibles, nous avons ajouté une mutation. Avant chaque tentative, l'IA a une probabilité de muter le code, ce qui signifie qu'une position peut être changée en une couleur aléatoire.

Cette méthode permet de contrôler la puissance de l'IA en ajustant le taux de mutation. Pour créer trois niveaux d'IA, nous avons effectué des simulations afin de choisir le taux approprié pour chaque niveau. Vous trouverez ci-dessous deux graphiques, chacun généré avec 1000 parties par mutation. Le premier représente le nombre moyen de tentatives nécessaires pour trouver le bon code pour chaque mutation, et le deuxième représente le pourcentage de parties gagnées (c'est-à-dire que le code a été trouvé en moins de 10 tentatives).

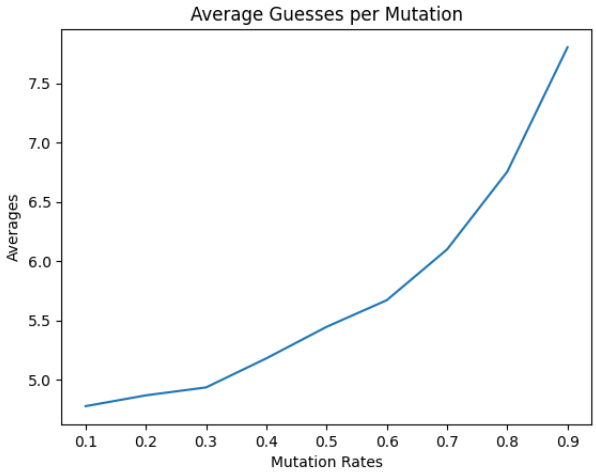

FIGURE 3.2 – Nombre moyen de tentatives nécessaires pour trouver le code par mutation

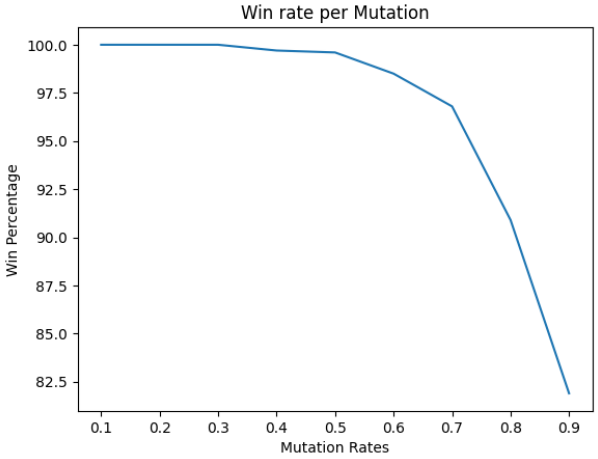

FIGURE 3.3 – Pourcentage de parties gagnées par mutation

Après avoir analysé les résultats, voici les configurations pour chaque niveau :

Expert étant donné que c'est la meilleure IA, nous avons conservé l'algorithme d'origine sans mutations (taux de mutation 0).

Normal nous lui avons attribué le taux de mutation le plus élevé garantissant une victoire dans 100% des parties avec une moyenne de 5 tentatives par partie (taux de mutation 0,3).

Facile nous lui avons attribué le taux de mutation le plus élevé garantissant une victoire dans au moins 90% des parties avec une moyenne de 6,6 tentatives par partie (taux de mutation 0,8).

3.2.4 Générations des indices

Chaque jeu nécessite un petit bouton d'aide, que ce soit pour les débutants qui apprennent encore le jeu ou les joueurs expérimentés confrontés à une situation difficile. Nous avons souhaité créer un système capable de générer ces indices pour nos joueurs, les indices étant :

1. Rédigés en langage naturel pour être facilement compréhensibles.
2. Basés uniquement sur les connaissances que le joueur peut déduire de ses coups précédents et leurs retour.

Pour générer des indices, nous commençons par réduire l'ensemble des codes possibles. Ensuite, nous parcourons tous les codes possibles et essayons de trouver des motifs. Nous extrayons des informations sur la présence de chaque couleur à chaque position et sur le nombre d'occurrences de chaque couleur par code. Ensuite, nous utilisons ces données pour générer nos indices en langage naturel à partir de nos modèles pré-écrits.

Voici les modèles que nous avons utilisés pour nos indices :

— Il y a exactement X <Couleur>.
— Il y a au moins X <Couleur>.
— Il n'y a pas de <Couleur> dans le code.
— Il y a au plus X <Couleur>.

— La position I est <Couleur>.
— La position I n'est pas [<Couleur>].
— La position I peut être [<Couleur>].

```
1  Function GenerateHints(PossibleCodes) :
       Data : une liste PossibleCodes contenant les codes possibles
       Résultat : born_inf une liste qui donne le nombre d'occurrence minimum de
                  chaque couleur, born_sup une liste qui donne le nombre d'occurrence
                  maximum de chaque couleur, possible_per_position une liste qui
                  contient pour chaque position l'ensemble des couleurs possibles
2      born_inf ← 4 pour chaque couleur;
3      born_sup ← 0 pour chaque couleur;
4      possibles_per_position ← ensemble vide pour chaque position;
5      for code in possible_codes do
6          for i to 4 do
7              possibles_per_position[i].add(code[i]);
8          end
9          for color in colors do
10             born_inf[color] ← min(born_inf[color], nombreOccurence(code, color));
11             born_sup[color] ← max(born_sup[color], nombreOccurence(code, color));
12         end
13     end
14     hints ← hintsToNaturalLanguage(born_inf, born_sup, possibles_per_position)
15     return hints;
```

Algorithme 2 : Algorithme de generation des indices

3.3 Interface

Notre interface consiste des trois sections :

3.3.1 Section gauche

Cette section permet à l'utilisateur d'interagir avec l'application. En haut, nous avons deux boutons : "Reset" pour recommencer le jeu et "Hint" pour obtenir un indice. En dessous, on a le code secret caché avec un bouton "Show" pour le révéler. Ensuite, nous avons un menu où l'utilisateur peut choisir un niveau d'IA ou jouer sans IA. Enfin, la partie qui permet à l'utilisateur de jouer un coup. Il sélectionne une couleur dans la palette ci-dessous, colore son code, puis clique sur "Guess".

3.3.2 Section centrale

Il s'agit du tableau de jeu de l'utilisateur qui se met à jour à chaque fois que l'utilisateur clique "Guess". En haut de cette section, nous affichons le nombre minimum d'essais que l'utilisateur peut effectuer pour trouver le code s'il utilise la méthode de Knuth à partir de ce moment-là.

3.3.3 Section droite

Cette section est uniquement visible lorsque l'IA est active. Pendant le jeu, l'utilisateur ne peut voir que les feedbacks que l'IA reçoit, mais il ne peut pas voir ses propositions tant qu'il n'a pas gagné la partie.

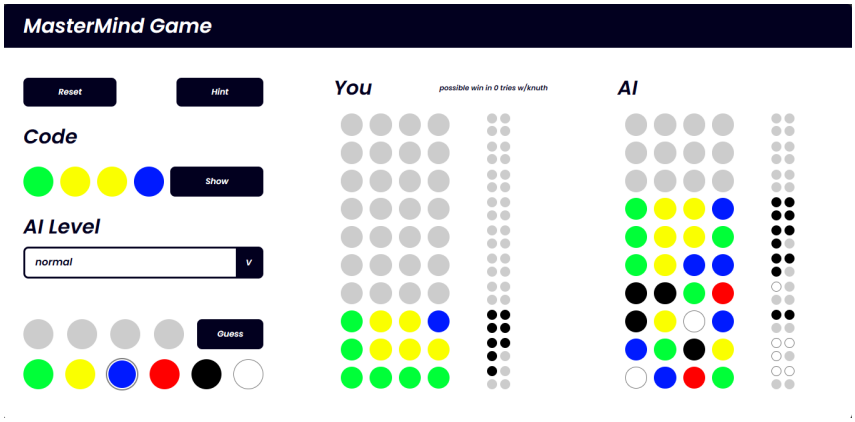

FIGURE 3.4 – Interface de jeu

3.4 Stratégie humaine

Tous les algorithmes que nous avons utilisés jusqu'à présent basent leurs prédictions sur l'ensemble des codes possibles. Cependant, étant donné qu'il est impossible pour un humain de mémoriser des centaines de codes et de réaliser de grands calculs mentaux, nous avons développé une stratégie basée sur les tests et les modifications. À chaque essai, nous proposons une version modifier de la proposition précédente.

Voici l'idée de fonctionnement de la stratégie :

1. Premier coup :
   - Commencez par un code où l'on met la même couleur dans les 4 positions. Si **n** pions noirs sont obtenus, cela signifie que cette couleur existe **n** fois dans le code correct.
   - S'il n'y a pas de pions noirs, cela signifie que cette couleur n'est pas présente dans le code.
2. Boucle :
   - Conservez les couleurs dont nous savons qu'elles sont présentes (même si leurs positions exactes sont inconnues) et remplissez les positions restantes avec une couleur que nous n'avons pas encore testée.
   - Si le nombre de pions noirs/blancs augmente, cela confirme la présence de cette couleur dans le code.
   - Si nous obtenons des pions blancs, nous testons les permutations uniquement avec les couleurs dont nous savons qu'elles sont présentes, mais dont nous ne connaissons pas les positions correctes, jusqu'à ce qu'elles soient déterminées.

3.4.1 Exemple d'utilisation

Dans cet exemple, le joueur a commencé avec la proposition (vert, vert, vert, vert). D'après le retour, nous savons qu'il n'y a pas de vert dans le code, nous passons donc à la prochaine couleur. À partir du retour sur (jaune, jaune, jaune, jaune), nous pouvons voir que nous avons deux jaunes dans le code, nous les gardons donc et remplissons le reste avec la couleur suivante. Après (jaune, jaune, bleu, bleu), le nombre de pions n'a pas changé, ce qui signifie qu'il n'y a pas de bleu dans le code, mais un pion est devenu blanc, ce qui signifie qu'un de nos jaunes est à la mauvaise position. Nous déplaçons donc un jaune et remplissons les 2 positions vides avec la couleur suivante. Après (jaune, rouge, jaune, rouge), nous n'avons obtenu que des pions noirs, ce qui signifie que nous avons correctement positionné les jaunes. Cependant, nous avons également remarqué une augmentation du nombre de pions, ce qui signifie qu'il y a un rouge dans le code. Nous le gardons et remplissons le reste avec la couleur suivante. Après (jaune, rouge, jaune, noir), nous remarquons qu'il n'y a pas d'augmentation du nombre de pions, donc il n'y a pas de noir, mais aussi qu'un pion est devenu blanc. Comme nous savons que les jaunes sont à leur bonne position, nous déplaçons un rouge et remplissons la place vide avec la couleur suivante. Après (jaune, blanc, jaune, rouge), nous obtenons 4 pions noirs, ce qui signifie que nous avons trouvé le code secret en 6 essais.

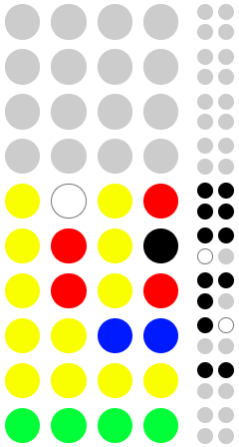

FIGURE 3.5 – Exemple d'un jeu avec la stratégie humaine

3.4.2 Force

La puissance de cette stratégie réside dans sa simplicité et sa mémorabilité. Elle ne nécessite aucune compétence particulière ; n'importe qui peut l'apprendre et commencer à l'appliquer immédiatement.

3.4.3 Limite

Pour rendre la stratégie aussi simple, de la même manière que nous pouvons obtenir les bonnes couleurs du premier coup, nous pouvons aussi avoir de la malchance et essayer toutes les mauvaises couleurs au début, ce qui peut entraîner de mauvaises performances ou même la perte de la partie.

Chapitre 4

Conclusion

En conclusion, ce projet a permis une exploration approfondie de diverses stratégies pour le jeu Mastermind, allant des algorithmes d'AI à une approche centrée sur l'humain. En mettant en œuvre et en comparant l'algorithme de Knuth, l'approche de Swaszek et une stratégie humaine, nous avons acquis des perspectives sur les forces et les limitations de chaque méthode. Une bonne addition pourrait être l'étude de l'algorithme de Koyama et Lai, qui est l'algorithme optimal en termes de nombre de coups moyens. De plus, le développement d'une interface interactive permet aux joueurs de s'engager avec le jeu et de tester leurs compétences contre différents niveaux d'opposants IA. L'incorporation d'un système de génération d'indices ajoute une dimension éducative au jeu. Une amélioration future pourrait consister à offrir aux joueurs des informations sur les déductions logiques en ajoutant de l'explicabilité aux indices donnés en étudiant exactement comment chaque élément du jeu est responsable de la génération d'un indice donné. Dans l'ensemble, ce projet a non seulement approfondi notre compréhension des stratégies du Mastermind, mais a également démontré la polyvalence des techniques d'IA dans les contextes de jeu.

Bibliographie

[1] Donald E. Knuth. « The Computer as Master mind ». In : *Journal of Recreational Mathematics* no. 9 (1976-77). URL : https://www.cs.uni.edu/~wallingf/teaching/cs3530/resources/knuth-mastermind.pdf (pages 3, 4).

[2] *Mastermind (board game)*. URL : https://en.wikipedia.org/wiki/Mastermind_(board_game) (page 3).

[3] P. Swaszek. « The mastermind novice ». In : *Journal of Recreational Mathematics* no. 30 (2000) (page 3).

Annexe A

Cahier des charges

La première étape du projet consiste à mettre en œuvre une stratégie optimale pour résoudre le jeu. Nous allons implémenter l'algorithme de Knuth, pour son efficacité en termes de nombre maximum d'essais. Cela signifie qu'il peut deviner n'importe quel code en un maximum de cinq essais. Cette implémentation produira un adversaire IA capable de résoudre Mastermind de manière efficace.

Nous développerons une interface graphique pour le jeu, comprenant une fonctionnalité permettant aux joueurs d'affronter l'IA pour voir qui peut deviner le code en premier. Cette fonctionnalité propose différents niveaux de difficulté pour l'IA, allant de facile à difficile.

Ensuite, nous introduirons une fonction "Aide" qui offrira aux joueurs une assistance pendant le jeu. Les conseils donnés seront basés sur les devinettes précédentes du code-breaker et les indices de feedback donnés par le code-maker.

Enfin, nous concevrons une stratégie de jeu simple que les joueurs humains pourront mémoriser et utiliser pour gagner le jeu et augmenter leurs chances de gagner contre l'adversaire IA.

Annexe B

Manuel utilisateur

Un manuel utilisateur permettant la prise en main de votre code.

1. Lancement du jeu depuis le fichier git :
   (a) prérequis : à installer les dépendances suivantes :
      — pip install Flask
      — pip install Flask-Cors
      — pip install python-dotenv
   (b) python3 api/server.py
   (c) un localhost est lancer en `http://127.0.0.1:5000`, ouvrez ce localhost dans un navigateur
2. Lancement du jeu depuis le serveur en ligne : lien