INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Rapport de Pré-Étude et de Spécifications

Etienne ALLAIN, Aymen BERRAJAA, Noha DOAIF, Alp JAKOP,
Giorgi KUCHUKHIDZE, Aymane MENFAA, Ezgi OZEL

Encadrante: Laurence Rozé

Novembre 2023

Table des matières

1 Introduction

1.1 Cadre

Dans le cadre d'un projet de recherche sur l'interprétabilité des algorithmes d'apprentissage appliqués aux données tabulaires, les chercheurs font face à un défi. Prenons l'exemple d'un électrocardiogramme, où l'objectif est de concevoir une IA capable d'évaluer si l'électrocardiogramme est normal ou anormal. Ici un obstacle majeur se présente : une personne qui n'est pas spécialisée dans le domaine de la cardiologie n'a aucun moyen de vérifier l'exactitude des analyses produites par l'IA.

Afin de contourner cette problématique, notre projet se focalise sur un domaine familier aux chercheurs - un jeu de cartes simple, le jeu des "Cœurs" issu du jeu de cartes "Barbu", leur permettant ainsi de juger de la pertinence des résultats obtenus. Notre objectif est de travailler sur le jeu des "Cœurs", afin de créer une IA dont l'analyse peut être évaluée par les chercheurs sans nécessiter de compétences spécifiques dans un domaine hors de leur expertise.

1.2 Objectifs

Ce projet a pour but de créer une application mobile dédiée au jeu des "Cœurs". Il se déroulera en deux phases majeures : la première consacrée au développement de l'application, permettant aux utilisateurs de jouer contre des bots ou d'autres joueurs, et la seconde axée sur la création de ces bots, qui seront deux intelligences artificielles distinctes.

La première partie consiste donc à développer une application qui devra être capable de gérer différentes options de jeu et de s'adapter à différents types de joueurs, qu'il s'agisse de joueurs humains, d'IA ou même de joueurs aléatoires. L'objectif est de proposer une expérience de jeu fluide et adaptable.

L'autre partie du projet, se concentre sur le développement de deux intelligences artificielles spécifiquement conçues pour performer dans le jeu des Cœurs. La décision de développer deux IA, plutôt qu'une seule, repose sur notre volonté d'explorer différentes approches et de diversifier nos solutions.

Les deux IA que nous allons utiliser s'appuient sur les algorithmes de type réseau de neurones et Monte Carlo Tree Search, et ce choix a du sens pour des raisons spécifiques. Le réseau de neurones est idéal pour apprendre à partir de données complexes en entrée et prendre des décisions, ce qui est essentiel pour jouer aux Cœurs. Le MCTS, lui est une option couramment utilisée dans les jeux de cartes en raison de sa capacité à explorer de manière efficace les différentes possibilités. En utilisant ces deux approches, nous cherchons à explorer différentes méthodes pour créer des IA performantes pour ce jeu.

Enfin, pour tout faire fonctionner, il est important de relier les intelligences artificielles à l'application. Pour faire cela nous utilisons une architecture client/server où le client enverra au serveur toutes les informations décrivant l'état du jeu actuel. C'est ce qui permettra au serveur de collaborer avec les intelligences artificielles pour déterminer la meilleure carte à jouer, puis de transmettre cette décision au client.

1.3 Plan du Rapport

Pour résumer, dans ce rapport de pré-étude nous allons d'abord détailler le contexte de notre projet en étudiant l'état de l'art de l'IA dans le domaine des jeux et en abordant les règles du jeu de "Cœurs", puis nous présenterons l'application que nous souhaitons développer. Ensuite nous allons détailler le fonctionnement et les specifications des deux intelligences artificielles pour finir avec l'explication du mode de fonctionnement de la communication client-serveur. .

2 Contexte

Pour commencer ce rapport, nous plongerons d'abord dans l'état de l'art de l'intelligence artificielle appliquée aux jeux de plateau et de cartes, en mettant particulièrement l'accent sur le jeu de "Cœurs". En suivant cette mise en contexte, nous détaillerons les règles fondamentales du jeu de "Cœurs", jetant ainsi les bases nécessaires à la compréhension de notre projet.

2.1 Etat de l'Art

L'état de l'art de l'intelligence artificielle dans les jeux de plateau et de cartes a connu des avancées significatives ces dernières années, exploitant diverses approches, dont l'algorithme de recherche arborescente MCTS (Monte Carlo Tree Search) et l'utilisation de réseaux de neurones et de l'apprentissage par renforcement.

Un exemple marquant de la puissance de l'algorithme MCTS est AlphaGo, développé par DeepMind. AlphaGo a créé une onde de choc en surpassant des champions mondiaux au jeu de Go, réputé pour sa complexité stratégique. L'approche distinctive d'AlphaGo réside dans l'utilisation astucieuse du MCTS, un algorithme qui simule des milliers de parties possibles, évaluant les mouvements par des simulations aléatoires, et construisant progressivement un arbre de recherche pour déterminer les meilleurs coups. Cette méthodologie a permis à AlphaGo de maîtriser des stratégies complexes et de prendre des décisions tactiques exceptionnelles.

En ce qui concerne MCTS pour le jeu de "Cœurs", l'article de Freek Bax, intitulé "Determinization with Monte Carlo Tree Search for the card game Hearts", met en lumière la force de l'application de MCTS, en particulier de la déterminisation. Cette méthode permet de modéliser et anticiper les actions possibles des adversaires en examinant différentes configurations de mains. En résolvant les problèmes liés à l'information imparfaite grâce à des algorithmes de recherche d'arbres de jeu, cette approche améliore significativement les performances de l'IA. En d'autres termes, elle aide le programme à mieux comprendre les choix potentiels des adversaires en explorant diverses façons dont les cartes pourraient être distribuées parmi eux.

En parallèle, l'utilisation de réseaux de neurones a transformé la manière dont les agents d'IA abordent les jeux. Les réseaux de neurones, en particulier les réseaux de neurones profonds, sont capables d'apprendre à partir de données massives, ce qui les rend adaptés à la modélisation de stratégies complexes dans les jeux. Dans le contexte des jeux de plateau et de cartes, les réseaux de neurones sont utilisés pour évaluer les positions, prédire les actions des adversaires, et générer des stratégies sophistiquées.

Des jeux classiques tels que les échecs ou le Shogi (échecs japonnais) ont vu l'émergence d'IA basés sur des réseaux de neurones, améliorant considérablement leurs compétences tactiques et stratégiques. Les échecs classiques ont connu une évolution significative avec l'intégration de réseaux de neurones dans des agents. Un bon exemple est Leela Chess Zero, l'un des moteurs d'échecs les

plus puissants. En incorporant des réseaux de neurones, Leela a amélioré de manière substantielle ses compétences tactiques et stratégiques, se propulsant ainsi au sommet de la compétition échiquéenne informatique en battant par exemple une IA très connue et très puissante - le Stockfish.

Dans le domaine des jeux de cartes, les algorithmes d'apprentissage par renforcement ont également été appliqués avec succès à des jeux comme le Poker et le Blackjack. Des chercheurs tels que Michael Bowling et son équipe de l'Université de l'Alberta, Canada, ont travaillé sur l'IA "DeepStack" pour le Poker, tandis que des chercheurs de l'Université Carnegie Mellon, dont Tuomas Sandholm et Noam Brown, ont développé "Libratus", initialement conçu pour maîtriser le Poker, mais dont les avancées ont eu des répercussions sur d'autres jeux de cartes. Les deux IA ont produit de meilleurs performences que les joueurs professionnels de ces jeux de cartes.

En ce qui concerne l'état de l'existant de l'utilisation des réseaux de neurones dans le jeu de cartes "Cœurs" que nous développons, nous pouvons se référer à l'article de Maxiem Wagenaar intitulé "Learning to play the Game of Hearts using Reinforcement Learning and a Multi-Layer Perceptron" qui a examiné l'efficacité de différentes configurations de réseaux neuronaux dans l'apprentissage du jeu. En expérimentant avec divers paramètres du réseau neuronal, tels que la fonction d'activation et le nombre de couches cachées, Wagenaar a réussi à configurer une IA qu'il n'a pas pu surpasser lui-même. Cela met en évidence la puissance de l'apprentissage par renforcement et de l'ajustement des paramètres du réseau pour atteindre des performances exceptionnelles dans un contexte assez complexe du jeu de "Cœurs". Cependant, l'auteur souligne également les limitations de son approche. L'IA n'ayant jamais été entraînée contre des joueurs professionnels de "Cœurs", son niveau de jeu optimal reste à déterminer. De plus, il reconnaît l'importance du facteur chance dans ce type de jeu, indiquant que même une IA hautement performante peut parfois subir des revers. Ces observations soulignent la nécessité d'une évaluation plus approfondie, peut-être en confrontant l'IA à des joueurs experts ou en explorant des ajustements supplémentaires pour améliorer la robustesse de l'IA face à des situations imprévisibles.

En conclusion, l'état actuel de l'IA dans les jeux de plateau et de cartes est caractérisé par une diversité d'approches réussies, allant de la puissance de MCTS à la sophistication des réseaux de neurones. Ces progrès ont non seulement redéfini les limites des capacités informatiques dans le domaine ludique, mais ont également ouvert la voie à des applications plus larges dans d'autres domaines de l'intelligence artificielle, ce qui est exactement l'objectif général de ce projet.

2.2 Le Jeu des Cœurs

Le jeu de "Cœurs" se base sur une manche spécifique appelée "Cœurs" d'un jeu plus complet appelé "Barbu". Ce jeu se base sur 32 cartes et notre version est conçue pour 4 joueurs. Chaque partie est divisée en 4x6 manches distinctes. Dans chaque série de 6 manches, un même joueur est désigné comme le "meneur", et c'est à lui de choisir le type de manche. Le joueur après le meneur, appelé "ouvreur", lance la manche en jouant une carte. Ensuite, pour les coups qui suivent, c'est le joueur ayant remporté le pli précédent qui commence. Chaque manche a ses propres objectifs, avec des noms spécifiques tels que "Plis", "Cœurs", "Dames", "Barbu", "Réussite" et "Total". Par contre, chaque manche se déroule de façon similaire : Le premier joueur demande une couleur. Les joueurs suivants doivent fournir une carte de cette couleur s'ils en ont, ou n'importe quelle carte sinon. Le joueur qui remporte le pli est celui qui a mis la carte la plus grande de la couleur demandée, sachant que l'ordre des cartes est : 7, 8, 9, 10, valet, dame, roi et as.

Dans la manche "Cœurs", l'objectif est soit de faire tous les Cœurs, soit de faire un minimum de Cœurs dans ses plis. Maintenant, regardons de plus près les deux stratégies que les joueurs peuvent

choisir.

1. **Stratégie "Faire le Minimum" :** Les joueurs qui adoptent cette approche cherchent à minimiser les pertes en remportant le moins de cœurs possible. Le joueur a une pénalité de 5 points par coeur dans ses plis.
2. **Stratégie "Tout Faire" :** À l'inverse, la stratégie "Tout Faire" consiste à remporter tous les cœurs disponibles, ce qui rapporte un bonus de 40 points. Cependant, il existe un risque, car ne pas réussir à remporter tous les cœurs entraîne des pénalités de 5 points par cœur.

3 L'Application

Dans cette section, nous allons présenter l'application que nous allons développer : "Coeur Artificiel". D'abord, nous allons expliquer le choix du nom avant de présenter Flutter qui est un framework mobile choisi comme base de programmation du projet. Ensuite, nous présenterons les modes de jeu que nous avons choisi pour finir avec le prototype de notre application.

En ce qui concerne le nom du projet, nous souhaitons partager les raisons qui ont motivé le choix de "Coeur Artificiel" :

— **Pourquoi "Cœur" ? :** On a opté pour le mot "Cœur" dans le titre pour évoquer le jeu de cartes "Cœurs". On souhaitait mettre en avant la place centrale de ce jeu.
— **Le côté Artificiel :** En ajoutant "Artificiel", on insiste sur le fait que des intelligences artificielles font partie du jeu.

Étant donné que le jeu est conçu pour une utilisation sur mobile, nous vous présenterons dans la partie suivante Flutter, le framework multiplateforme.

3.1 Choix de Flutter

Suite aux choix qui nous étaient proposés pour le développement de "Coeur Artificiel", nous avons choisi d'utiliser Flutter comme plateforme de développement. Les différents atouts de cette plateforme sont listés ci-dessous :

— **Multiplateforme :** La nature multiplateforme de Flutter signifie que notre jeu de cartes sera accessible à la fois sur les appareils IOS et Android, en utilisant une seule base de code. Cela réduit les coûts de développement et de maintenance, tout en garantissant une portée maximale pour notre jeu.
— **Performance élevée :** Flutter est réputé pour ses performances élevées. Grâce à son moteur de rendu 2D intégré, le jeu "Coeur Artificiel" fonctionne de manière fluide et réactive, offrant une expérience de jeu sans heurts, même sur des appareils mobiles moins puissants.
— **Widgets personnalisables :** Flutter propose un grand nombre de widgets prêts à l'emploi pour créer des interfaces utilisateur. De plus, il permet aux développeurs de créer des widgets personnalisés, ce qui est idéal pour concevoir des éléments de jeu de cartes uniques et interactifs. Ces widgets sont hautement personnalisables, ce qui signifie que nous pouvons créer une expérience de jeu visuellement attrayante et adaptée à nos besoins spécifiques.
— **Hot Reload :** L'une des caractéristiques les plus fascinantes de Flutter est le "Hot Reload". Il s'agit d'une fonctionnalité qui permet aux développeurs de voir instantanément

les changements apportés au code dans l'interface utilisateur de l'application en temps réel, sans avoir besoin de recompiler l'ensemble de l'application. Cela accélère considérablement le processus de développement, permettant aux concepteurs et développeurs de jeu de cartes de visualiser et d'itérer rapidement sur les modifications, ce qui est essentiel pour affiner l'expérience utilisateur.

— **Compatibilité avec les animations :** Les animations sont un élément essentiel de l'expérience de jeu. Flutter propose un puissant support pour la création d'animations fluides et engageantes, ce qui nous permet de donner vie à notre jeu de cartes de manière visuellement captivante.

— **Grande communauté de développeurs :** Flutter bénéficie d'une communauté de développeurs active et engagée. Cela signifie que nous avons accès à un large éventail de ressources, de bibliothèques tierces et de solutions aux problèmes courants, ce qui accélère encore le processus de développement.

— **Documentation complète :** Flutter est livré avec une documentation complète, qui facilite l'apprentissage de la plateforme et la résolution de problèmes. Cela garantit que notre équipe de développement dispose de toutes les ressources nécessaires pour créer un jeu de cartes de haute qualité.

3.2 Modes de jeu

Dans notre projet, nous envisageons d'offrir aux joueurs trois modes de jeu principaux :

1. **Joueur Humain vs. IA (Intelligence Artificielle) :** Dans ce mode de jeu, le joueur principal affronte trois intelligences artificielles, dont le niveau de difficulté est soit facile, soit intermédiaire, soit difficile. L'utilisateur a la possibilité de sélectionner le niveau qui correspond à son expérience et à ses préférences. De plus, pour personnaliser davantage l'expérience de jeu, le joueur peut choisir de jouer contre des adversaires contrôlés par des algorithmes de type MCTS (Monte Carlo Tree Search) ou par des réseaux de neurones. Il est important de noter que, bien que notre approche utilise des algorithmes avancés tels que le Monte Carlo Tree Search (MCTS) et les réseaux de neurones, ces concepts peuvent ne pas être familiers à un utilisateur quelconque. Dans un premier temps, celà sera nécessaire pour nous en ce qui concerne le réglage de nos intelligences artificielles à travers l'entraînement par joueurs aléatoires. En plus, notre code pourra être utilisé par une équipe de recherche si besoin.
2. **Joueur Humain vs. Joueurs Humains :** Dans ce mode de jeu, le joueur principal affronte trois joueurs humains. Cela permet au joueur d'avoir une expérience de jeu plus équilibrée.

Après avoir étudié ensemble les différents modes de jeu, nous verrons dans la section suivante le prototype développé pour visualiser notre application.

3.3 Présentation du Prototype

Dans cette section, nous allons vous présenter notre application d'un point de vue purement externe à travers un prototype que nous avons réalisé.

(a) Prototype de l'interface principale de connexion au jeu

(b) Prototype de l'interface principale avec menu défilant

(c) Prototype de l'interface graphique du jeu de Cœurs

A l'ouverture de l'application, l'utilisateur verra l'interface principale de notre jeu, qui lui indiquera qu'il faut cliquer sur l'écran pour lancer le jeu, comme illustré sur la figure (a). Cette interface contient un boutton permettant de visualiser un menu défilant présentant les détails des règles du jeu comme le montre la figure (b). Au lancement du jeu, l'utilisateur fera face à l'interface graphique du jeu de Cœurs, qui contient les cartes visibles du joueur en bas, les cartes non visibles en haut, à droite, et à gauche des joueurs qui l'affrontent. Cette interface contient également un boutton de retour tout en haut permettant au joueur de quitter la partie, un boutton à l'extrême droite permettant de visualiser le tableau de score des joueurs, et un boutton de réglages pour personnaliser la partie. La figure (c) permet clairement de visualiser celà.

4 IA à base de réseau de neurones

Comme il a déjà été cité dans les sections précédentes, notre projet repose sur la création d'une application qui nécessitera des bots qui remplaceront des joueurs humains et qui devront donc être capables de jouer au jeu des Cœurs avec les mêmes capacités cognitives d'un joueur humain. Par conséquent, l'implémentation d'une IA est essentielle. Par ailleurs, le jeu des Cœurs est un jeu de cartes complexe qui comporte des éléments de stratégie et de prise de décision. Certaines approches de recherche exhaustive comme le minimax par exemple sont impraticables dans ce cas-là. Celà nous a donc permis d'orienter nos recherches et de nous interesser à deux approches differentes pour l'implementation de notre IA :

— Une approche par **réseau de neurones**
— Une approche s'appuyant sur l'algorithme **MCTS** (*Monte Carlo Tree Search*)

4.1 Réseau de Neurones

Un réseau de neurones est un modèle informatique inspiré du fonctionnement du cerveau humain. Il est conçu pour effectuer des tâches d'apprentissage automatique en traitant des données complexes. Il est composé de plusieurs "neurones" artificiels organisés en couches interconnectées.

Chaque neurone du réseau prend des données en entrée, effectue des calculs sur ces données, puis transmet un résultat à d'autres neurones. Ces neurones sont organisés en couches, généralement une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Les connexions entre les neurones, appelées "poids", sont ajustées pendant l'entraînement pour permettre au réseau de répondre de manière appropriée aux données d'entrée.

Nous devons également souligner l'importance des fonctions d'activation. Ces fonctions introduisent une non-linéarité dans les calculs des neurones, permettant au réseau de capturer des relations complexes et des motifs non linéaires dans les données. Les fonctions d'activation, telles que ReLU (Rectified Linear Unit) ou Sigmoid, sont appliquées à la sortie de chaque neurone. Elle influencent la manière dont l'information est propagée à travers le réseau. Cela donne au réseau la capacité d'apprendre des représentations hiérarchiques et des caractéristiques abstraites, ce qui lui permet d'améliorer sa capacité à résoudre des problèmes complexes d'apprentissage automatique.

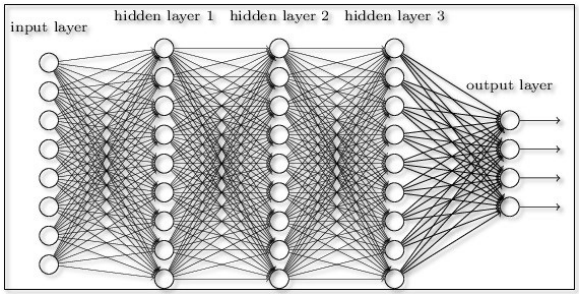

FIGURE 2 – Représentation d'un réseau de neurones

Nous pouvons voir dans la figure ci-dessus un niveau d'entrée sur la gauche, un niveau de sortie sur la droite, et trois couches cachées entre les deux. Les petits cercles représentent les neurones et les arêtes représentent quant à elles les connexions entre les neurones, appelées "poids".

L'apprentissage dans un réseau de neurones se produit en ajustant les poids pour minimiser les erreurs entre les prédictions du réseau et les données réelles. Une fois entraîné, le réseau de neurones peut être utilisé pour effectuer une variété de tâches, telles que la classification d'images, la reconnaissance de la parole, la traduction automatique, la prédiction de séquences, et bien plus encore.

Pour résumer, l'utilisation d'un réseau de neurones est adaptée aux problèmes de classification ou de régression sur des données statiques ou encore au traitement de données structurées. Par ailleurs, lorsqu'une tâche implique une prise de décisions séquentielles au fil du temps, et que les actions à prendre dépendent d'un contexte dynamique, ou alors implique des jeux où les interactions sont dynamiques et complexes, le réseau de neurones à ce moment-là devrait être combiné avec l'apprentissage par renforcement.

4.2 Apprentissage par Renforcement

L'apprentissage par renforcement est un domaine du *Machine Learning* qui se concentre sur la façon dont un agent interagit avec un environnement pour apprendre à prendre des décisions afin de maximiser une récompense cumulée. Il s'inspire en partie du comportement des organismes vivants qui apprennent à travers des essais et des erreurs pour optimiser leurs actions en fonction des conséquences.

Voici les concepts clés de l'apprentissage par renforcement :

- **Agent** : L'entité qui apprend à prendre des décisions en interagissant avec l'environnement.
- **Environnement** : Le monde dans lequel l'agent évolue et prend des actions.
- **État** : Une représentation de l'environnement à un moment donné. Les états capturent les informations pertinentes pour la prise de décision.
- **Action** : Les choix disponibles à l'agent à chaque étape, qui influencent l'état suivant.
- **Récompense** : Une valeur numérique qui indique la qualité de la décision de l'agent à un moment donné. L'objectif de l'agent est de maximiser la récompense cumulée sur le long terme.

Cet apprentissage repose également sur le concept de politique. En d'autres termes, une politique signifierait la stratégie qu'adopte l'agent pour choisir des actions en fonction des états de son environnement. Cette politique peut être déterministe ce qui signifie le fait de faire correspondre un état à une action spécifique. Cependant, les politiques généralement utilisées sont non deterministes et reposent sur la notion d'exploration-exploitation où l'agent doit trouver un équilibre entre explorer de nouvelles actions et choisir des actions déjà jugées optimales.

En conclusion, et en nous mettant dans le contexte du jeu des Cœurs, le réseau de neurones est utilisé pour comprendre les schémas complexes et prendre des décisions basées sur les données structurées du jeu, quand parallèlement l'apprentissage par renforcement permet l'ajustement des actions au fil du temps en interagissant avec l'environnement du jeu, cherchant à maximiser les récompenses cumulées. Par conséquent, la création d'une intelligence artificielle capable de jouer au jeu des Cœurs nécessite l'association d'un réseau de neurones à l'apprentissage par renforcement pour permettre à cette IA d'apprendre de manière flexible et adaptative. Par ailleurs cette approche a été le sujet d'une thèse d'un projet de licence qui explore l'utilisation d'un réseau de neurones, plus précisément d'un perceptron multicouche (MLP), en combinaison avec l'apprentissage par renforcement pour apprendre à jouer au jeu de cartes "Hearts". [2]

4.3 Spécification de l'Algorithme

Le développement de la première intelligence artificielle pour le jeu de "Cœurs" repose sur une approche exploitant les capacités du apprentissage par renforcement, plus précisément le Deep Q-learning. Cette section se plonge davantage dans la méthodologie adoptée.

Algorithm 2: Algorithme Deep Q-Learning

Data: Ensemble d'états S, ensemble d'actions A, fonction de récompense R, réseau de neurones Q

Result: Politique optimale π^*

Initialiser Q avec des poids aléatoires;
Initialiser $Q_{target} = Q$;
for *épisode = 1 à nombre d'épisodes* **do**
    Générer un ensemble de données D : ;
        Initialiser l'état initial s ; ;
        for *pas de temps = 1 à nombre de pas par épisode* **do**
            Créer le vecteur d'entrée à partir de l'état actuel du jeu;
            Sélectionner une action a avec une politique d'exploration-exploitation;
            Exécuter l'action a et observer la récompense r et le nouvel état s';
            Ajouter (s, a, r, s') à D;
    Effectuer l'apprentissage : ;
        for *epoch = 1 à nbEpoch* **do**
            Traiter D par minibatch : ;
                Calculer y_i : ;
                    r si dernier épisode;
                    $r + \gamma \max_a Q(s', a')$ sinon;
                Calculer la perte $\mathcal{L} = (Q(s, a) - y_i)^2$;
                Mettre à jour les poids de Q en utilisant la descente de gradient;
                Tous les *nbPas*, faire $Q_{target} = Q$;

Le processus d'entraînement de l'IA commence par générer un grand nombre d'épisodes. À chaque carte jouée, un vecteur d'entrée est créé à partir de l'état du jeu, incluant des informations telles que le numéro du pli, la main du joueur, le nombre de cœurs tombés, la couleur demandée, etc.

Ce réseau de neurones est soumis à plusieurs itérations d'entraînement. Le vecteur d'entrée, représentant l'état du jeu, traverse des couches cachées non linéaires avec des fonctions d'activation ReLU, introduisant de la non-linéarité au modèle et facilitant l'apprentissage de schémas complexes.

En fonction des résultats des mouvements effectués, les poids associés à chaque mouvement sont ajustés via la rétropropagation des gradients dans PyTorch. Les poids sont mis à jour de manière positive en cas de victoire et négative en cas de défaite, permettant au modèle d'apprendre des stratégies complexes au fil du temps.

Pour mesurer l'écart entre les prédictions et les résultats réels du jeu, une fonction de perte est utilisée. Dans ce contexte, la Mean Squared Error (MSE) est choisie comme fonction de perte. La MSE vise à minimiser la différence quadratique entre les valeurs Q prédites et les valeurs Q cibles, facilitant la convergence du réseau de neurones.

Une fois l'entraînement terminé, l'IA est prête à être utilisée. Lorsqu'une requête est reçue du client, représentant un état du jeu spécifique, le modèle analyse les informations d'entrée et propose une action recommandée. Cette action recommandée est celle avec le poids le plus élevé. Ce processus est rapide et permet une interaction en temps réel avec le jeu.

4.4 Architecture

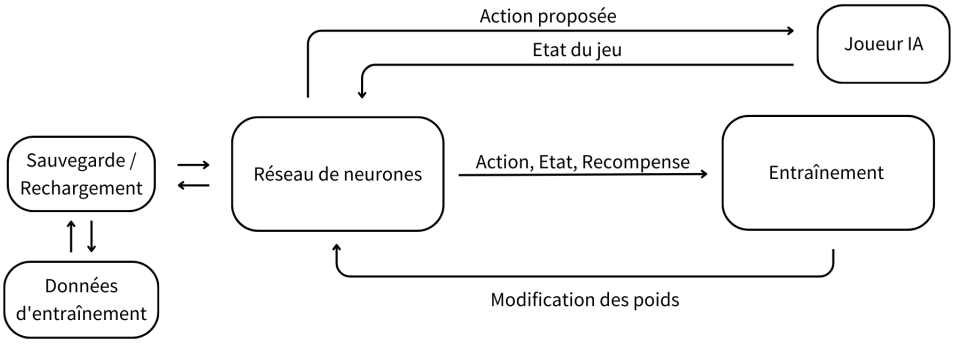

FIGURE 3 – Schéma de l'architecture générale de l'IA à base de réseau de neurones.

Notre système est organisé de manière à ce que tous ses éléments travaillent ensemble, comme illustré dans la Figure 3. Chaque partie de cette architecture a un rôle spécifique qui contribue au fonctionnement global du système.

Le **Réseau de Neurones** est au cœur de notre système. Il apprend et prend des décisions. Le lien bidirectionnel avec la sauvegarde/rechargement permet au réseau de stocker des données d'entraînement et de les récupérer au besoin. En même temps, la connexion avec le processus d'entraînement assure que l'entraînement reçoit des informations importantes sur les actions prises, l'état du jeu et les récompenses obtenues pendant son processus.

La partie **Sauvegarde/Rechargement** simplifie la gestion des données d'entraînement. En se connectant au réseau de neurones de manière bidirectionnelle, elle facilite le stockage et la récupération efficaces des informations nécessaires pour améliorer constamment le modèle.

L'**Entraînement** est essentiel pour affiner les performances du réseau de neurones. En interagissant avec le réseau, il transmet la modification des poids et garantit que le réseau ajuste ses paramètres en fonction des résultats de manière itérative.

Enfin, le **Joueur IA** est constamment en communication bidirectionnelle avec le réseau de neurones. En envoyant des informations sur l'étape du jeu et en recevant des actions proposées, cette interaction assure une synchronisation efficace entre le modèle d'IA et le jeu réel.

Cette architecture favorise une collaboration fluide entre ses composants, créant ainsi un environnement intégré propice à l'apprentissage, à la prise de décision et à l'interaction avec le jeu des Cœurs.

4.5 Détails Techniques

- **Utilisation de PyTorch :** PyTorch a été choisi plutôt que TensorFlow en raison de sa facilité d'utilisation et de son interface intuitive, offrant ainsi une expérience de développement plus conviviale.

- **Gestion des Dépendances avec Poetry :** La simplicité de la gestion des dépendances et la création d'environnements virtuels cohérents avec Poetry simplifient le processus de déploiement et d'entretien de notre solution.

- En ce moment, nous n'envisageons pas de travailler sur le contrôle du niveau de difficulté de cette IA.

4.6 Entrées/Sorties

Cette section détaille les aspects spécifiques des entrées et sorties du modèle d'intelligence artificielle à base de réseau de neurones développé pour le jeu de "Cœurs". Un exemple d'état du jeu est fourni, mettant en évidence la manière dont le modèle interprète ces données pour générer des décisions intelligentes.

4.6.1 Exemple d'État du Jeu

Illustrons le fonctionnement du modèle avec un exemple d'état du jeu. Chaque composante de l'état (décrit ci-dessous) est codée pour permettre au modèle de comprendre pleinement la situation actuelle du jeu.

- **Numéro du Pli :** 3

- **Main du Joueur :** [seven_hearts, eight_hearts, king_spades, ace_diamonds, queen_clubs]

- **Cœurs Tombés Avant le Pli Courant :** 4

- **Nombre de Cœurs du Joueur 0..3 :** [2, 1, 0, 1]

- **Couleur Demandée dans le Pli Courant :** spades

- **Cartes Déjà Jouées dans le Pli :** [ten_spades, jack_spades]

- **Cartes Restantes dans les Mains des Autres Joueurs :** [[queen_hearts, ace_spades, [...], [...], [...], [...]]

4.6.2 Action de Sortie

La sortie de l'IA correspond à la carte optimale à jouer dans l'état actuel du jeu. En d'autres termes, c'est l'action ayant la "Q-value" la plus élevée selon la "Q-function" apprise.

En résumé, l'importance cruciale des entrées et sorties dans le fonctionnement de notre modèle d'intelligence artificielle pour le jeu de "Cœurs" se manifeste à travers des exemples concrets, offrant ainsi un aperçu du processus décisionnel. Cela met en évidence la capacité du modèle à prendre des décisions intelligentes et stratégiques au sein d'un contexte de jeu dynamique.

5 IA à base de MCTS

5.1 MCTS

L'Intelligence Artificielle (IA) basée sur l'algorithme Monte Carlo Tree Search (MCTS) est particulièrement bien adaptée aux jeux de stratégie comme le jeu des Cœurs, où les actions possibles sont nombreuses, et où nos décisions ont un fort impact dans l'optimisation de nos coups suivants. L'algorithme MCTS se démarque par sa capacité à explorer rapidement un arbre des actions possibles. Il est notamment utilisé dans les jeux de société tels que les échecs, les dames, et même des jeux de cartes comme le poker.

5.1.1 Fonctionnement de l'Algorithme MCTS

L'algorithme MCTS fonctionne en suivant un processus itératif en quatre étapes : Sélection, Expansion, Simulation et "Backpropagation".

1. **Sélection :** L'arbre est exploré à partir du nœud racine en sélectionnant à chaque étape le nœud enfant qui maximise une certaine mesure de qualité. La sélection est basée sur l'équilibre entre l'exploration (visiter de nouveaux nœuds) et l'exploitation (sélectionner les nœuds les plus intéressants en fonction de la connaissance actuelle). Elle s'arrête lorsqu'un nœud feuille est atteint, c'est-à-dire lorsqu'un nœud n'a pas tous ses enfants (actions possibles) créés.
2. **Expansion :** Une fois que la sélection atteint un nœud non exploré (une feuille), de nouveaux nœuds enfants sont créés pour représenter les actions possibles à partir de l'état du jeu défini par le nœud sélectionné. Le choix de l'action à ajouter est souvent aléatoire : un seul nœud enfant parmi les actions possibles est ajouté à l'arbre.
3. **Simulation :** Une simulation est ensuite effectuée à partir du nœud enfant nouvellement créé (l'action choisie), généralement jusqu'à atteindre un état terminal du jeu. Cette simulation est basée sur une politique aléatoire, i.e. les décisions prises par l'ensemble des joueurs lors de la simulation sont aléatoires.
4. **"Backpropagation" :** La récompense obtenue à la fin de la simulation est propagée vers le haut de l'arbre, en mettant à jour les statistiques des nœuds visités : du noeud depuis lequel la simulation a été lancée jusqu'à la racine. Dans notre cas, les statistiques sont le nombre de visite du noeud et la somme des scores obtenus à la fin des simulations.

Ce processus de 4 étapes constitue ce que l'on appelle une itération. Il rend un arbre mis à jour pour le chemin pris par les différentes étapes. Il se répète pour un certain nombre d'itérations, permettant ainsi à l'algorithme MCTS de converger vers la meilleure action à prendre en fonction de l'état du jeu.

5.2 Determinization

5.2.1 Introduction au "Determinization"

Le concept de "Determinization" joue un rôle déterminant dans l'IA MCTS. Ce concept permet de gérer l'incertitude au sein des jeux de cartes où la distribution des cartes est inconnue, comme dans le jeu des Cœurs. Dans notre contexte, le Determinization consiste à prendre en considération différentes hypothèses sur la distribution des cartes entre les joueurs. Chaque hypothèse représente une possible distribution du jeu de cartes. Elles sont générées aléatoirement tout en respectant certaines contraintes comme le nombre de cartes par joueur.

5.2.2 Utilisation du Determinization dans l'IA MCTS

L'utilisation du Determinization est essentielle pour permettre à l'IA MCTS de prendre les meilleures décisions possibles dans un contexte incertain : nous n'avons aucune connaissance sur les mains des adversaires et donc sur les actions possibles qu'ils peuvent prendre. Au lieu de se baser sur une distribution de cartes fixe, l'IA génère alors un ensemble d'états possibles en modifiant la distribution des cartes à chaque itération de l'algorithme MCTS. Cela signifie que l'IA MCTS simule des parties dans lesquelles elle suppose différentes distributions de cartes parmi les joueurs. De cette manière, cela permet de mieux anticiper les réponses des adversaires et d'adopter une stratégie plus robuste vis-à-vis de leurs actions.

5.3 Architecture Prévue/Spécifications

5.3.1 Architecture Générale

L'IA MCTS (Monte Carlo Tree Search) est un composant clé de notre projet sur le jeu des Cœurs. Son rôle est d'aider les joueurs à prendre la meilleure décision lors de leur tour. L'IA MCTS est conçue comme un module distinct du jeu, capable de communiquer avec le moteur du jeu (l'application) pour recevoir des informations sur l'état actuel.

Elle est ainsi décomposée en trois parties, nous permettant d'adapter son fonctionnement à notre variante du jeu des Cœurs. L'une se charge de contrôler l'évolution de l'arbre en lui-même, ses nœuds, le concept de "Determinization" et tout ce qui se rapporte à l'algorithme MCTS. Nous avons ensuite une partie qui détaille les règles de notre jeu des Cœurs. Du pli jusqu'à une partie globale en passant par les manches, elle caractérise notre version d'IA à base de MCTS. La dernière partie décrit les cartes, une main d'un joueur ainsi que le jeu de cartes utilisé dans notre variante (allant des 7 aux As).

Cette organisation est inspirée de l'architecture d'une IA MCTS appliquée à une autre version du jeu des Cœurs en C# disponible sur GitHub. Cette implémentation utilise notamment un autre algorithme lors du processus itératif détaillé plus tôt, appelé UCT. Cet algorithme est utilisé pour la Sélection. La principale difficulté de l'algorithme MCTS étant de préserver un certain équilibre entre l'exploration et l'exploitation, UCT le gère en utilisant une formule composée de deux termes : l'un correspondant à l'exploration (élevé pour les nœuds avec peu de visites) et l'autre correspondant à l'exploitation (élevé pour les nœuds avec un haut ratio de victoires). Cette formule permet de calculer un score pour chaque nœud et ainsi choisir le plus prometteur à chaque itération. UCT améliore l'efficacité de la recherche et guide vers les parties les plus intéressantes de l'arbre de jeu.

Cet exemple nous a convaincu d'utiliser cette même approche pour guider la sélection des nœuds dans notre algorithme MCTS.

Plus généralement, cette implémentation rentre dans le cadre d'un projet de thèse de l'Université d'Utrecht qui explore l'application de l'algorithme Monte Carlo Tree Search (MCTS), spécifiquement sa variante Upper Confidence Bound (UCB), au jeu des Cœurs. [1]

5.3.2 Spécification de l'IA MCTS

L'IA MCTS utilise l'algorithme MCTS pour explorer les différentes actions possibles et évaluer leur qualité. Elle répète ce processus pendant un certain nombre d'itérations pour améliorer sa décision. L'objectif de ces itérations est d'affiner les statistiques des nœuds grâce aux simulations, d'explorer davantage l'arbre et la sortie finale est choisie parmi les nœuds enfants de celui

représentant l'état actuel du jeu donné en entrée. L'IA prend aussi en compte le concept de "Determinization" pour gérer l'incertitude liée à la distribution des cartes.

Pour mieux comprendre son fonctionnement, voici un pseudo-code illustrant le principe de l'algorithme MCTS jusqu'à la décision finale :

Algorithm 3: Algorithme MCTS pour le jeu des Coeurs

```
1  Variables :
       — V(N) : nombre de visites du noeud N
       — R(N) : somme des récompenses au noeud N
   for chaque itération do
       N = racine de l'arbre T;
       D = une "determinization" aléatoire de N;
       // Sélection
2      while N n'est pas une feuille do
3          Choisir un enfant M de N selon l'UCT;
4          N = M;
       // Expansion
5      Ajouter un M parmi les M possibles depuis N dans D à l'arbre;
6      N = M;
       // Simulation
7      r = résultat de la simulation à partir de N dans D;
       // Backpropagation
8      for chaque parent M de N jusqu'à la racine (N inclus) do
9          Incrémenter V(M) de 1;
10         Augmenter R(M) de r;
   // Choix de la meilleure action A à partir du noeud E correspondant à
      l'état donné en entrée
11 A = enfant M de E avec R(M)/V(M) maximal;
12 Retourner A;
```

L'algorithme montre le processus itératif en utilisant les statistiques évoquées plus tôt : le nombre de visites et la somme des récompenses pour chaque noeud. Chaque itération démarre en se positionnant à la racine de l'arbre et en considérant une hypothèse de distribution du jeu de cartes en accord avec l'entrée représentée par la "determinization" D. Puis, sans détailler leur fonctionnement, l'algorithme poursuit avec les 4 étapes : Sélection, Expansion, Simulation et Backpropagation. Enfin, après un certain nombre d'itérations, la décision finale est déterminée selon le critère suivant : l'action choisie est celle possèdant la récompense moyenne maximale parmi les noeuds enfants de celui correspondant à l'état fournit en entrée. Ce critère pourra être amené à changer en fonction des résultats lors de la phase de conception et n'est qu'un choix "arbitraire" pour le moment.

5.3.3 Contrôle du Niveau de Difficulté

Un aspect important de la spécification de l'IA MCTS est la possibilité de contrôler son niveau de difficulté. Cela peut être réalisé en ajustant les paramètres de l'algorithme MCTS, tels que le nombre d'itérations effectuées par l'IA.

5.4 Entrées/Sorties

5.4.1 Entrées

L'IA MCTS prend en compte un ensemble spécifique d'informations qui décrivent l'état actuel du jeu. Ces informations sont essentielles pour garantir la meilleure décision possible dans chaque situation. Voici les principales informations composant le vecteur d'entrée utilisé par l'IA MCTS :

- Numéro du Pli en Cours : Entier informant l'IA quel pli est en cours.
- Cartes Jouables dans la Main du Joueur : Liste de cartes définies par un rang (7 à As) et une couleur (Carreau, Pic, Trèfle ou Cœur) qui représentent donc les actions possibles.
- Cartes Jouées lors du Pli en Cours Avant notre Tour : Liste des cartes jouées par les autres joueurs lors du pli.
- Cartes Jouées depuis le Début du Jeu : Liste des cartes jouées depuis le premier pli.
- Combinaison des Cartes Possédées par les Autres Joueurs : Liste des cartes possédées par l'ensemble des autres joueurs, cela aide l'IA à ajuster ses hypothèses de distribution des cartes.
- Rang du Joueur : Entier informant la position du joueur dans le pli en cours.
- Nombre de Cœurs Ramassés par le Joueur : Entier représentant le nombre de Cœurs ramassés par le joueur jusqu'à présent.
- Nombre de Cœurs Ramassés par l'Ensemble des Joueurs : Entier représentant le cumul du nombre de Cœurs ramassés par chaque joueur jusqu'à présent.

5.4.2 Sorties

Une fois que l'IA MCTS a analysé l'état actuel du jeu en fonction de ces entrées, elle génère une sortie qui oriente l'action du joueur. La sortie de l'IA MCTS est donc l'action recommandée. Cette action est sélectionnée en utilisant l'algorithme MCTS, qui explore les différentes actions possibles pour déterminer la meilleure à partir de l'état du jeu actuel, i.e. celle qui possède la valuation maximale (récompense moyenne obtenue lors des simulations).

6 Communication Client-Serveur

Cette partie traite de la communication client-serveur entre nos intelligences artificielles et l'interface de notre application. Le client est donc un client Flutter qui représente l'interface du jeu et le serveur, un serveur Python qui héberge les intelligences artificielles du jeu. La communication entre les deux est établie grâce à l'utilisation de Flask pour le serveur et de HTTP avec la bibliothèque Dart pour le client. Cette section discutera du choix de ces outils, de l'architecture du projet, donnera un exemple concret de leur utilisation ainsi que les spécifications des communications entre le client et le serveur.

6.1 Choix des Outils

6.1.1 Flask

Flask a été choisi comme framework de développement côté serveur en raison de sa légèreté, de sa simplicité et de sa flexibilité. Il offre un ensemble de bibliothèques et d'outils permettant de développer des applications web rapidement. En outre, il offre un support pour créer des API RESTful, ce qui est essentiel pour la communication entre le client et le serveur. Flask est idéal pour

les applications de petite à moyenne envergure, ce qui en fait un choix adapté à un jeu de cartes Cœurs.

6.1.2 HTTP avec Dart

Pour la communication client-serveur, Flutter utilise la bibliothèque HTTP avec Dart. HTTP est le protocole standard pour le transfert de données sur le web, ce qui en fait un choix naturel pour communiquer avec un serveur Flask. La bibliothèque HTTP de Dart permet d'effectuer des requêtes HTTP de manière simple et efficace. Elle offre un moyen fiable de communiquer avec le serveur Python, de transmettre des données et de recevoir des réponses.

6.2 Architecture Générale du Projet

L'architecture générale du projet est conçue pour assurer une communication fluide entre le client Flutter et le serveur Python. Voici un aperçu des principaux composants de l'architecture :

6.2.1 Schéma de l'architecture du projet :

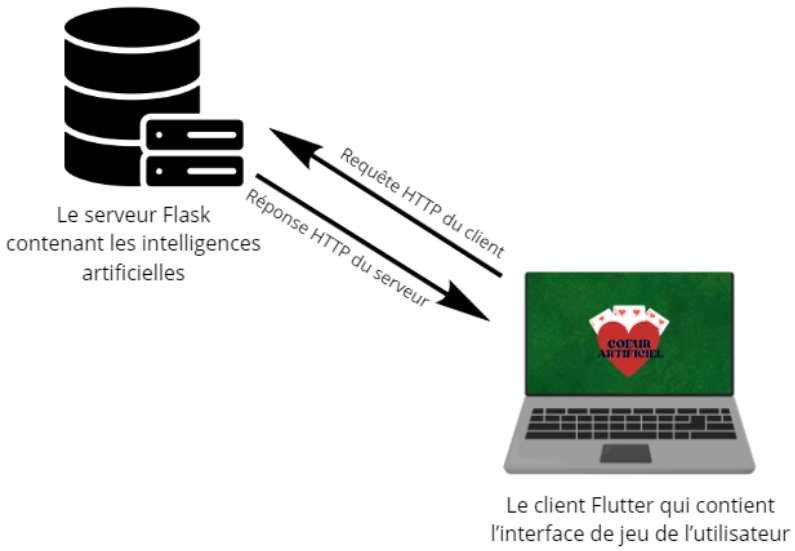

FIGURE 4 – Schéma de l'architecture générale du jeu des Cœurs.

6.2.2 Détails de l'architecture du projet :

- **Client Flutter** :
  - L'interface utilisateur du jeu des Cœurs est développée en utilisant Flutter.
  - Le client Flutter gère l'interaction avec le joueur, affiche le plateau de jeu et les cartes, et gère les événements utilisateur.
  - Il communique avec le serveur Python via des requêtes HTTP pour effectuer des actions de jeu, telle que demander à l'intelligence artificielle de jouer un coup.
- **Serveur Python (Flask)** :

  - Le serveur Python utilise le framework Flask pour créer un serveur web léger.
  - Il héberge l'intelligence artificielle du jeu, qui prend des décisions en fonction de l'état du jeu.
  - Le serveur expose une API RESTful qui permet au client de lui envoyer des données et de recevoir des réponses.
  - Il gère aussi la logique du jeu, le suivi des tours, le suivi des scores et les règles du jeu.
- **Communication Client-Serveur** :
  - La communication entre le client et le serveur est réalisée via des requêtes HTTP. Le client envoie des requêtes HTTP au serveur pour effectuer des actions, et le serveur renvoie des réponses avec les résultats.
  - Les requêtes sont envoyées à des points de terminaison (endpoints ou routes) spécifiques définis sur le serveur Flask.
  - Les données sont échangées au format JSON pour une compatibilité facile entre les deux parties.

L'architecture du projet est conçue pour garantir la cohérence du jeu entre le client et le serveur. Le serveur Python est responsable des intelligences artificielles et de leurs actions, tandis que le client Flutter gère l'interface utilisateur du jeu. Du côté du client ou du serveur, tous deux intègrent la logique du jeu. Cette logique du jeu côté serveur permet de contrôler les actions des intelligences artificielles et de déduire des informations à partir des données reçues du client. Du côté du client, l'intégration de la logique du jeu lui permet de superviser le déroulement de la partie.
Cette approche vise à minimiser le nombre de requêtes HTTP effectuées, assurant ainsi un déroulement fluide du jeu (Les appels au serveur se produisent uniquement lorsqu'une intelligence artificielle doit jouer).

6.3 Exemple d'Utilisation

Un exemple concret de l'utilisation des bibliothèques Flask et HTTP avec Dart a été illustré dans le développement d'un jeu de Morpion en Flutter. Dans ce scénario, le client Flutter représentait l'interface de jeu, permettant aux joueurs de faire des mouvements sur le tableau de Morpion. Le serveur Python, qui contenait une intelligence artificielle basée sur l'algorithme Minimax, servait à l'adversaire virtuel. Les deux parties communiquaient via une API exposée par le serveur Flask. Lorsque les joueurs effectuaient des mouvements sur le tableau, le client Flutter envoyait des requêtes HTTP au serveur Python pour transmettre ces actions. Le serveur Python analysait les coups et renvoyait des réponses avec les mouvements de l'IA. Cette utilisation réussie d'une architecture client-serveur avec Flask et HTTP dans le contexte du Morpion démontre la flexibilité de ces outils pour le développement de notre jeu des Cœurs.

Le lien du travail effectué est disponible sur GitLab à l'adresse :

https://gitlab.insa-rennes.fr/projet˙coeur/hearts

6.4 Spécifications des communications

Le processus de communication entre le client Flutter et le serveur Flask pour la gestion des IA, la collecte des données de jeu et la transmission des scores à la fin de la partie implique plusieurs types de requêtes HTTP.

6.4.1 Initialisation des IA :

Requête d'initialisation des IA (POST) :

End-Point : http ://localhost :8080/jeu/initialisation

Objectif : C'est une requête POST qui permet lorsque l'utilisateur a choisi les types d'intelligence artificielle contre qui il veut jouer, de les initialiser côté serveur afin de pouvoir lancer la partie. Ces données seront envoyées au serveur sous forme de JSON, qui contient l'identifiant de l'intelligence artificielle avec son type.

Exemple de JSON :

```
{
  "ia_players": [
    {
      "id": 1,
      "type": "MCTS"
    },
    {
      "id": 2,
      "type": "MCTS"
    },
    {
      "id": 3,
      "type": "NeuralNetwork"
    }
  ]
}
```

Réponse d'initialisation des IA :

Objectif : C'est la réponse à la requête POST d'initialisation des intelligences artificielles. Elle renvoie un code de retour, avec un message explicatif. Le code de retour peut être de type 200 si tout s'est bien déroulé, 400 si il y a eu un problème au niveau de la requête et 500 si il y a un problème au niveau du serveur.

Exemple de JSON :

```
{
  "status": 200,
  "message": "Les IA sont initialisées, la partie peut commencer."
}
```

6.4.2 Collecte des données de jeu par l'IA :

Requête de Collecte des Données de Jeu par l'IA (POST) :

End-Point : http ://localhost :8080/jeu/tour_ia.

Objectif : C'est la requête qui intervient lorsque c'est le tour d'une des intelligences artificielles de jouer. Le client envoie au serveur qui hébérge l'intelligence artificielle toutes les données nécessaires afin de pouvoir choisir le coup optimal, cela correspond à son vecteur d'entrée. Il envoie sous forme de JSON, l'identifiant de l'intelligence artificielle qui joue, la liste de toutes les cartes qui ont été joué depuis le début de la partie, il transmet aussi la liste des cartes que l'intelligence artificielle peut jouer ainsi que tous les plis que possède chaque joueur. Ces informations suffisent au serveur pour déduire le vecteur d'entrée dont a besoin l'intelligence artificielle.

Exemple de JSON : ("H"="Hearts", "D"="Diamonds", "S"="Spades", "C"="Clubs")

```
{
  "id": 1,
  "played_cards": ["KD", "10D", "QS","QD"],
  "possible_cards": ["7D", "8D","QH","9S","KC","7H","10S"],
  "tricks": [
    {
      "id": "0",
      "list_of_tricks": [
      {
      "number_trick": "1",
      "cards_trick": ["KD", "10D", "QS","QD"],
      }
      ]
    },
    {
      "id": "1",
      "list_of_tricks": [
      ]
    }
    {
      "id": "2",
      "list_of_tricks": [
      ]
    }
    {
      "id": "3",
      "list_of_tricks": [
      ]
    }
  ]
}
```

Réponse à la collecte des données de Jeu :

Objectif : C'est la reponse à la requête POST qui permet à l'intelligence artificielle de récupérer son vecteur d'entrée afin d'analyser et choisir le coup à jouer. Une fois que le coup à jouer est determiné, il est envoyé dans la réponse de la requête. On transmet au client l'identifiant de l'intelligence artificielle qui joue ainsi que la carte à jouer, on la joue puis on passe au tour du joueur suivant. Des codes de retour (200 si tout s'est bien déroulé, 400 si il y a eu un problème dans la requête, 500

si il y a un problème au niveau du serveur) sont également envoyés avec un message explicatif en fonction du code de retour.

Exemple de JSON :

```
{
  "status": 200,
  "message": "L'intelligence artificielle a choisi son coup.",
  "id": 1,
  "selected_card": "7D"
}
```

6.4.3 Transmission des scores :

Requête de transmission des scores en fin de partie :

End-Point : http ://localhost :8080/jeu/scores

Objectif : A la fin de chaque partie, le client fait une requête POST afin de transmettre aux intelligences artificielles le score qu'elles ont obtenu. Ces analyses de scores permettront aux intelligences artificielles d'optimiser leurs prochains choix pour les parties qui suivront et c'est ainsi qu'elles pourront s'améliorer.

Exemple de JSON :

```
{
  "scores": [
    {
      "id": "0",
      "score": -20
    },
    {
      "id": 1,
      "score": 0
    },
    {
      "id": 2,
      "score": 0
    },
    {
      "id": 3,
      "score": 40
    }
  ]
}
```

Réponse à la transmission des scores en fin de partie :

Objectif : Lorsque la requête HTTP est envoyée au serveur, un code de retour ainsi qu'un message explicatif est renvoyé au client afin de documenter l'état de la requête qui a été réalisé.

Exemple de JSON :

```
{
  "status": 200,
  "message": "Scores enregistrés avec succès."
}
```

Ainsi, le client et le serveur peuvent échanger les informations nécessaires pour gérer les intelligences artificielles, jouer la partie et enregistrer les résultats de manière organisée et efficace. Chaque requête est clairement définie en fonction de son objectif, ce qui permet une communication fluide et organisée entre le client et le serveur.

6.5 Récapitulatif

Le choix de Flask pour le serveur Python et de HTTP avec Dart pour le client Flutter a permis de mettre en place une communication client-serveur robuste pour le jeu de cartes Cœurs. L'architecture générale du projet garantit un fonctionnement harmonieux entre le client et le serveur, permettant aux joueurs d'avoir une bonne expérience de jeu. L'utilisation de ces outils permettra de créer une application de jeu de cartes Cœurs efficace et légère.

7 Conclusion

Cette première phase nous a permis de comprendre en profondeur la problématique du projet. Notre but est de créer une application mobile pour le jeu de cartes "*Cœurs*".

Pour ce faire, nous avons d'abord commencé notre pre-étude par comprendre les règles du jeu des *Cœurs*. Puis, nous avons lu des articles sur les deux algorithmes de l'IA à utiliser dans notre projet. Le choix des outils pour le développement du projet a également été effectué, suivi par la création d'une maquette pour l'interface utilisateur et l'installation des outils dont nous avions besoin afin de travailler sur les différents parties du projet. Nous avons également regardé des codes déjà existants sur les algorithmes et aussi sur l'interface utilisateur pour une meilleure comprèhension de l'utilisation des outils que nous avons choisis. Dernièrement, nous avons commencé à travailler sur des differents aspects de notre projet :

- La décision de nos entrées et sorties pour nos deux algorithmes d'IA,
- L'implémentation d'un *réseau de neurones* avec *apprentissage par renforcement* pour le jeu Morpion,
- L'implémentation du jeu du Morpion en Flutter, et une intelligence artificielle en Python coté serveur afin de tester le bon fonctionnement des bibliothèques choisies pour la communication client-serveur.

De plus, nous avons commencé à implementer quelques pages de notre application mobile sur Flutter pour prendre en main les outils que nous utiliserons pour le reste du projet.

Notre projet vise à développer une application mobile pour le jeu de cartes "*Cœurs*". Pour atteindre cet objectif, nous avons conçu l'architecture initiale de notre projet. Cette architecture englobe les structures de deux algorithmes distincts, à savoir la *Monte Carlo Tree Search (MCTS)* et *Deep Q Learning*. De plus, il intègre l'architecture qui établit la connexion entre ces algorithmes et le côté client de notre projet, représentant l'interface à travers laquelle les utilisateurs interagissent. L'interface client est implémentée sous la forme d'une application mobile utilisant *Flutter*, et la communication client-serveur est facilitée grâce aux technologies *Flask* et *Dart*.

La prochaine phase de notre projet consistera à effectuer une conception de notre application. Nous allons planifier et définir la direction globale du projet en tenant compte des exigences que nous avons identifiées lors de la phase de pré-étude et de spécification. Notre travail consistera à développer l'interface utilisateur et les deux algorithmes d'intelligence artificielle qui seront utilisés : le *réseau de neurones* avec *Deep Q Learning* et le *MCTS (Monte Carlo Tree Search)*.

Références

[1] Freek Bax. Determinization with monte carlo tree search for the card game hearts. Thesis Draft, 2020.

[2] Maxiem Wagenaar. Learning to play the game of hearts using reinforcement learning and a multi-layer perceptron. Bachelor's Project Thesis, 2017.