Modélisation du jeu de Pentago

Un rapport écrit et soumis à monsieur Laurent Bourgois
requis pour

Projet Semestre 7 du cursus ingénieur

par

Hanane HAOUD

(2000571)

Ecole CentraleSupélec
Département d'ingénierie
Université Paris-Saclay
2024

Résumé

Ce projet se concentre sur la modélisation informatique du jeu Pentago en utilisant Python, avec des implémentations pour un mode à deux joueurs et un mode avec adversaire IA. Pentago est un jeu de plateau stratégique où l'objectif est d'aligner cinq billes d'affilée, avec la particularité de pouvoir tourner l'un des quatre quadrants après chaque mouvement. L'objectif principal de ce projet est de développer une version logicielle robuste de Pentago permettant un jeu fluide entre deux joueurs humains ainsi que contre une IA intelligente.

Le projet est structuré pour couvrir toutes les phases critiques du développement, y compris l'analyse des mécanismes du jeu, la conception de l'architecture logicielle, et l'implémentation de la logique du jeu et de l'interface utilisateur. Pour le composant IA, l'algorithme Minimax avec élagage Alpha-Beta est utilisé pour créer un adversaire compétent capable de prendre des décisions stratégiques. (((Les performances et les stratégies de l'IA sont évaluées à travers une série de tests contre des joueurs humains pour garantir une expérience de jeu stimulante et engageante.)))

Des tests sont effectués pour valider la fonctionnalité et la stabilité des deux modes de jeu. Les résultats démontrent que le jeu implémenté respecte fidèlement les règles de Pentago, offrant une expérience agréable pour les joueurs humains. L'IA s'avère être un adversaire compétent, capable de faire des mouvements tentant d'imiter la réflexion stratégique humaine.

Ce projet sert non seulement de démonstration d'une modélisation de jeu efficace et d'une intégration d'IA, mais pose également les bases pour des améliorations futures telles que des stratégies IA plus avancées, des capacités multijoueurs en réseau, et des fonctionnalités de jeu étendues. Par cette initiative, compte tenu du fait que le jeu de Pentago a déjà été modélisé, nous cherchons à explorer et obtenir une expérience supplémentaire de comparaisons d'algorithmes d'intelligence artificielle.

Remerciements

Je souhaite exprimer ma gratitude à mon professeur encadrant, Laurent Bourgois, pour ses conseils et son soutien tout au long de ce projet.

Je remercie également mes collègues et amis pour leurs retours constructifs, ainsi que ma famille pour leur soutien moral.

Enfin, je tiens à remercier l'ensemble du corps enseignant et le personnel administratif de CentraleSupélec pour avoir fourni un environnement d'apprentissage enrichissant et propice à la réalisation de ce projet.

Table des matières

Table des figures

CHAPITRE 1

Introduction

Le but de ce travail est de modéliser le jeu de plateau Pentago informatiquement. Ceci ayant pour but de permettre de pouvoir jouer des parties sans support physique de jeu avec ou sans adversaire physique.

1.1 Le jeu

1.1.1 Règles du jeu

Le jeu de Pentago se joue sur un plateau de forme carré contenant six trous par six (36 trous au total). Ce plateau est lui-même subdivisé en quatres quadrants de trois trous par trois.

FIGURE 1.1 – Plateau du jeu de Pentago

La version originale du jeu se joue à un contre un. Un joueur dispose de boules noires tandis que le deuxième dispose de boules blanches.

Une partie se déroule comme suit : Le premier joueur place la boule de sa couleur sur un emplacement du plateau puis choisis un quadrant à tourner de 90 degrés dans le sens horaire ou anti-horaire, c'est à ce moment seulement que son tour est terminé. Le deuxième joueur peut alors procéder de la même façon pour jouer son tour.

Le gagnant est le premier joueur à avoir cinq boules de sa couleur alignées à l'issue de son tour.

1.1.2 Stratégies

Il existe donc quatre manières d'aligner les cinq boules de la même couleur :

— Sur la même ligne horizontale : une ligne horizontale gagnante est une ligne débutant sur le bord droite ou gauche du plateau et alignant cinq boules. Il y a donc deux façons de gagner par ligne horizontale. La ligne gagnante s'étale sur deux quadrants différents.

— Sur la même ligne verticale : de la même façon, il est possible d'aligner les boules sur les verticales du plateau. De même, il existe deux façons de gagner par lignes d'emplacements alignés à la verticale. La ligne gagnante s'étale sur deux quadrants différents.
— Sur les diagonales entourant celles du milieu : Il y a une seule façon d'aligner ses boules sur chacune de ces diagonales, il y en a quatre. La ligne gagnante s'étale sur trois quadrants.
— Sur les diagonales du milieu : Il y a deux façons d'aligner ses boules sur chacune de ces diagonales, en débutant sur un coin différent. La ligne gagnante s'étale sur deux quadrants.

1.2 Objectifs du projet

1.2.1 Description des fonctionnalités voulues

Le jeu modélisé a pour ambition de présenter sur une interface graphique simple le plateau et de permettre dans un premier temps à deux joueurs de jouer l'un contre l'autre dessus.

Ensuite, dans un second temps, le joueur pourra jouer en autonomie contre l'ordinateur, l'ordinateur pourra, soit jouer aléatoirement, soit réfléchir à ses coups et choisir les meilleurs coups possibles.

Le projet a donc pour ambition de choisir la meilleure façon pour l'ordinateur de prédire ses coups, en termes de rapidité de jeu, de choix du meilleur chemin jusque une des configurations gagnantes, et de victoire.

Le pentago est un jeu fini/fermé/ ... Ceci nous dit qu'il, existe un moyen de connaître la meilleure stratégie (peut-être mettre ca dans une autre catégorie)

CHAPITRE 2

Etat de l'art

2.1 Jeux de plateaux modélisés

Les jeux de plateau ont toujours occupés une place importante dans l'univers ludique offrant des expériences interactives et stratégiques aux joueurs. Les jeux tels que les échecs ou les dames occupant depuis de nombreuses années une place privilégiée dans le coeur des joueurs malgré la création de jeux de société plus récents tels que le Monopoly ou le Scrabble.

Depuis l'avènement de l'informatique, ces jeux ont commencés à être modélisé afin de simuler des parties plus ou moins complexes et de les rendre accessibles à un plus large public. L'un des premiers jeux à avoir été simulé est celui des échecs.

FIGURE 2.1 – Deep Blue d'IBM bat le champion du monde Kasparov

Alan Turing, l'un des pionniers de l'informatique a écrit l'un des premiers algorithmes pour jouer aux échecs en 1948. A défaut d'ordinateur assez puissant, il a dû simuler manuellement son algorithme. L'année suivante, Claude Shannon écrit "Programming a Computer for playing chess" où il fait même mention de l'algorithme Minimax. Des programmes de plus en plus sophistiqués voient le jour dans de nombreux lieux jusqu'en 1997 où une étape majeure est franchie lorsque le champion du monde Garry Kasparov est vaincu par le robot Deep Blue d'IBM.

Ce jeu a permis de nombreuses avancées en intelligence artificielle et certains programmes d'échecs comme Alphazero et Stockfish permettent de surpasser les meilleurs joueurs humains. D'autres jeux tels que le jeu de go, les dames (jeu "résolu" en 2007) le Backgammon, le Poker et bien d'autres ont aussi été modélisé dans le même temps.

Ces avancées démontrent également l'intérêt de la théorie des jeux dans d'autres domaines et l'importance de ces recherches et du développement de ces algorithmes. Pour

exemple, en économie et marchés financiers, la théorie des enchères et les modèles de marchés peuvent utiliser les modèles de jeux. Aussi, ces algorithmes trouvent un intérêt en négociation de conflits, en biologie et écologie (interaction entre espèces), en réseaux de communication (protocoles de communication, allocation de ressources), en cyber-sécurité (modélisation des attaques et des défenses) et dans bien d'autres domaines divers.

Cela démontre leur utilité pour résoudre des problèmes complexes et optimiser des systèmes dans des contextes très différents des jeux de plateau.

2.2 Techniques couramment utilisées

2.2.1 Algoritmes de recherche

Algorithme Minimax

L'algorithme Minimax est utilisé dans les jeux à somme nulle pour maximiser le score tout en minimisant les pertes possibles.

La recherche dans l'arbre de jeu avec l'algorithme minimax trouve le meilleur coup en supposant que les deux joueurs jouent de manière rationnelle. Le but est de maximiser ou de minimiser la valeur d'une fonction d'évaluation statique jusqu'à une certaine profondeur. (L'algorithme peut également bien fonctionner même si ces hypothèses sont un peu relâchées.) Cependant, la qualité d'un coup s'améliore généralement avec l'augmentation de la profondeur maximale. Cela rend habituellement l'algorithme minimax long à exécuter.

La théorie derrière minimax est que l'adversaire de l'algorithme tentera de minimiser la valeur que l'algorithme tente de maximiser (d'où "minimax"). Ainsi, l'ordinateur doit effectuer le mouvement qui laisse son adversaire capable de faire le moins de dégâts.

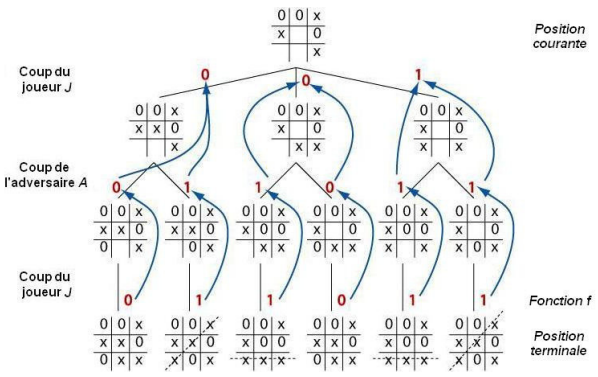

FIGURE 2.2 – Exemple de schéma d'exploration de Minimax dans le cadre du jeu du morpion

Le but de l'algorithme Minimax est de parcourir l'arbre en donnant une valeur à chaque noeud à l'aide d'une fonction de gain heuristique. Les noeuds terminaux doivent être de valeur : succès, échec ou nul. Cependant parcourir l'arbre complet est souvent impossible pour les jeux complexes ou comportant trop de chemins possibles. Le tictactoe,

version de jeu simple a lui-même déjà $9! = 362\ 880$ possibilités !

On peut alors choisir un niveau de profondeur de l'arbre à atteindre. L'algorithme cherche ensuite en parcourant l'arbre à choisir les noeuds de valeur maximale lorsqu'il joue et minimale lorsque l'adversaire joue. Il choisit alors le prochain coup qui maximise la valeure minimax.

L'algorithme minimax développe l'arbre jusqu'aux feuilles de la profondeur souhaitée. Les valeurs minimales de feuilles remontent à leur parent. Ensuite la valeur maximale remonte au parent et ainsi de suite. On appelle e un état du jeu (un état de l'arbre), h la fonction d'évaluation des feuilles, et minimax la valeur attribuée à un noeud. Alors :

— Si e est une feuille ou état final atteint :

$$\text{Minimax}(e) = \text{h}(e)$$

— Si joueur :

$$\text{Minimax}(e) = \max_{e' \in \text{Succ}(e)} \text{Minimax}(e')$$

On prend le maximum des sucesseurs

— Si adversaire :

$$\text{Minimax}(e) = \min_{e' \in \text{Succ}(e)} \text{Minimax}(e')$$

On prend le minimum des successeurs

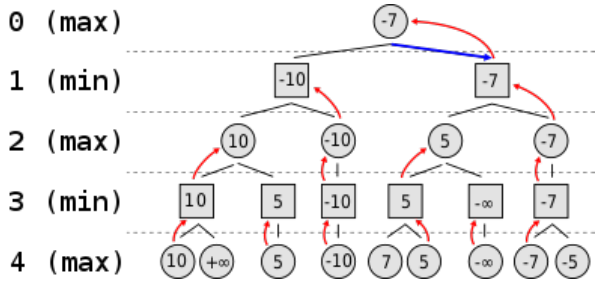

FIGURE 2.3 – Minimax pour une profondeur 4

Le but est donc de limiter les dégâts possibles de l'adversaire aux tours suivants, c'est une approche pessimisste. Le joueur peut alors sacrifier un gain plus gros ou plus rapide par peur que l'adversaire soit capable d'effectuer une série de meilleurs coups qu'il aura vu en réponse.

La difficulté principale réside dans l'évaluation heuristique de la fonction h. Effectivement, elle dépend fortement du jeu en question et de ses règles.

Les jeux comportant des éléments aléatoires, comme le backgammon, posent également des défis aux algorithmes minimax. Dans ces cas, une approche appelée "expectimax" est utilisée. En intégrant les probabilités que certains mouvements soient possibles pour chaque joueur (comme les probabilités des différentes combinaisons de dés) ainsi que leurs objectifs opposés, un ordinateur peut calculer une "valeur attendue" pour chaque nœud atteignable à partir de l'état actuel du jeu. À mesure que le jeu progresse, ces probabilités doivent être fréquemment mises à jour. Néanmoins, cette méthode permet de prendre des décisions éclairées.

Minimax avec élagage Alpha-Bêta

Comme vu précedemment, l'algorithme Minimax a pour plus grand défaut sa longueur, dû au fait qu'il évalue toutes les feuilles à la profondeur souhaitée. Le nombre de feuilles augmentant énormément avec la profondeur.

Une tentative de solution à ce problème est donnée par l'amélioration de l'algorithme Minimax par élagage Alpha-Bêta. L'élagage alpha-bêta est une optimisation de l'algorithme minimax qui réduit significativement le nombre de nœuds évalués dans l'arbre de jeu, sans affecter le résultat final. Cette technique fonctionne en éliminant les branches de l'arbre qui ne peuvent pas influencer la décision finale, en utilisant deux valeurs : alpha et bêta.

Alpha représente la valeur maximale garantie pour le joueur maximisant, tandis que bêta représente la valeur minimale garantie pour le joueur minimisant. Lors de la recherche, si l'algorithme découvre qu'un certain nœud ne peut pas améliorer le résultat pour l'un des joueurs (car il est moins favorable que les meilleures options déjà trouvées), il cesse d'explorer cette branche.

Cela permet de concentrer les ressources de calcul sur les parties de l'arbre de jeu les plus prometteuses, améliorant ainsi l'efficacité et la rapidité de l'algorithme minimax.

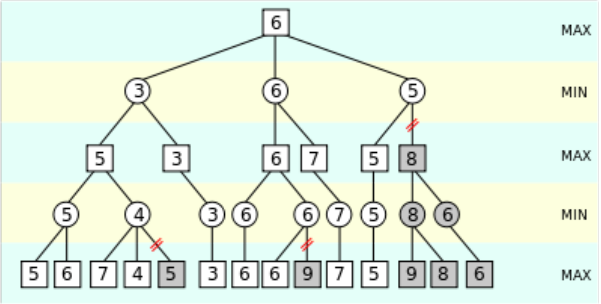

FIGURE 2.4 – Alpha-Beta pruning

Pour un état e, si c'est au tour du joueur maximisant, l'algorithme met à jour la valeur α en explorant les successeurs e' de e :

$$\alpha = \max(\alpha, \text{Minimax}(e'))$$

Si α devient supérieur ou égal à β, l'algorithme coupe (élague) les successeurs restants de e car ils ne peuvent pas influencer la décision finale :

$$\text{si } \alpha \geq \beta, \text{ alors couper}$$

Pour un état e, si c'est au tour du joueur minimisant, l'algorithme met à jour la valeur β en explorant les successeurs e' de e :

$$\beta = \min(\beta, \text{Minimax}(e'))$$

Si β devient inférieur ou égal à α, l'algorithme coupe (élague) les successeurs restants de e car ils ne peuvent pas influencer la décision finale :

$$\text{si } \beta \leq \alpha, \text{ alors couper}$$

L'élagage alpha-bêta permet ainsi de concentrer les ressources de calcul sur les parties les plus prometteuses de l'arbre de jeu. Les valeurs initiales de alpha et bêta dépendent de la nature du problème traité.

2.2.2 Intelligence artificielle

Dans cette section, des algorithmes différents de ceux de recherches plus déterministes et systématiques seront développés. Nous verrons des algorithmes plus basés sur l'apprentissage à partir de données ou de l'expérience.

Apprentissage par renforcement

L'apprentissage par renforcement représente une approche dynamique en intelligence artificielle, où un agent apprend à agir efficacement dans des environnements complexes en maximisant les récompenses cumulatives.
Inspiré par la psychologie comportementale, ce paradigme repose sur un processus d'interaction continu : l'agent prend des décisions, observe les conséquences de ses actions, puis ajuste son comportement pour optimiser ses gains à long terme.

Les techniques courantes incluent les processus décisionnels de Markov (MDP), les algorithmes de politique et de valeur, et leur application s'étend à divers domaines tels que la robotique avancée, la gestion autonome des stocks et même les jeux stratégiques complexes. Cette approche permet, par exemple, de s'adapter à la variabilité des situations réelles.

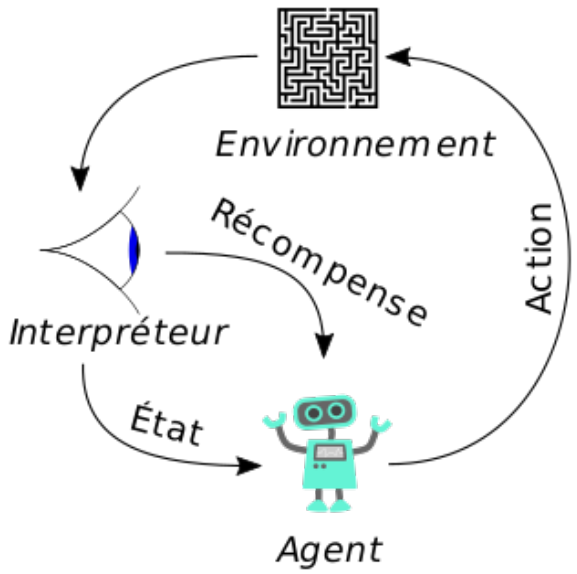

FIGURE 2.5 – Schéma illustratif de l'apprentissage par renforcement

Depuis un état s, l'agent utilise une politique π pour choisir une action a. Le but est d'optimiser la politique d'action π grâce à un système de récompenses positives et

négatives. La base de cette technique étant l'expérimentation, il est important de disposer d'un environnement simulé, ce qui s'adapte très bien à nos jeux de plateau simulés.

Suite à l'action a_t, le système passe de l'état s_t à l'état s_{t+1}, il en résulte un environnement possiblement modifié ainsi qu'une récompense r_t. On appelle fonction de transition la fonction permettant le passage de s_t à s_{t+1}. Une fois l'apprentissage jugé correct, il est possible de conserver la loi π pour l'utiliser.

Pour se placer dans le cadre des processus de décision markovien (MDP), la fonction de transition doit être une fonction stochastique (*), attribuant pour a effectué dans s, la probabilité de se retrouver dans chaque état s', cette distribution de probabilité devant être indépendante des actions et états précédents. Si l'environnement est déterministe, la fonction de transition renvoie directement l'état s'.

Dans le cas des jeux de plateaux à deux joueurs, on peut envisager le problème de la manière suivante : les états sont définis par la position actuelle de toutes les pièces du jeu, chaque action spécifie où la prochaine pièce doit être placée, la fonction de transition est déterminée par les règles du jeu, et la récompense est positive lorsque l'agent gagne la partie et négative lorsqu'il perd. Pour commencer l'apprentissage de la politique de l'agent sur le jeu à 2 joueurs, il faut donner un algorithme (soit hasard complet, soit un peu intelligent) à l'adversaire de cet agent.

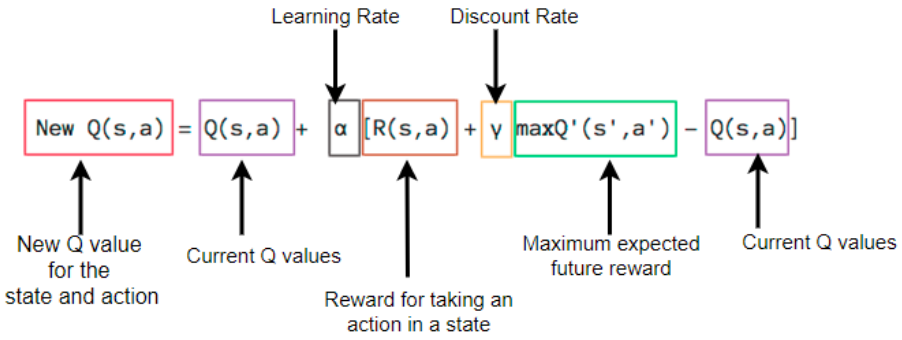

FIGURE 2.6 – Q-learning

Une fois notre environnement S,A,T,R créé, il nous faut débuter l'apprentissage, différentes méthodes sont possibles : Q-learning, SARSA, DQN, Policy gradient method ... Ces méthodes ont pour but d'approcher au mieux possible la politique optimale π_*.

Le Q-learning est une méthode d'apprentissage par renforcement qui permet à un agent d'apprendre à prendre des décisions optimales en interagissant avec son environnement. Cette méthode repose sur l'utilisation d'une fonction de qualité, ou fonction Q, qui évalue la qualité d'une action spécifique prise dans un état donné. L'objectif est de maximiser la récompense cumulée au fil du temps. Pour ce faire, l'agent met à jour la valeur de Q en utilisant la formule suivante :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right],$$

où s et a représentent respectivement l'état et l'action actuels, r est la récompense obtenue après avoir pris l'action a, s' est le nouvel état après l'action, α est le taux d'apprentissage et γ est le facteur de discount qui détermine l'importance des récompenses futures.

En plus de cette formule de mise à jour, il est crucial de considérer la politique d'exploration et d'exploitation. L'agent doit équilibrer l'exploration (essayer de nouvelles actions pour découvrir leur effet) et l'exploitation (utiliser les connaissances actuelles pour maximiser la récompense). Une stratégie courante est la politique ϵ-greedy, où l'agent choisit une action aléatoire avec une probabilité ϵ et la meilleure action (celle qui a la plus haute valeur Q) avec une probabilité $1 - \epsilon$. La formule de la politique ϵ-greedy est :

$$\text{choisir une action } a = \begin{cases} \text{une action aléatoire} & \text{avec probabilité } \epsilon, \\ \arg\max_a Q(s, a) & \text{avec probabilité } 1 - \epsilon. \end{cases}$$

Dans le cadre de la simulation de jeux de plateau, le Q-learning est particulièrement utile car il permet à l'agent d'apprendre à jouer de manière optimale à partir de zéro, en explorant différentes stratégies et en ajustant progressivement ses décisions en fonction des résultats obtenus. Par exemple, dans un jeu comme les échecs ou le go, l'agent peut évaluer la qualité de chaque coup possible en fonction de la configuration actuelle du plateau, et choisir les actions qui maximisent ses chances de victoire à long terme. Grâce à cette méthode, l'agent peut s'améliorer continuellement, devenant ainsi de plus en plus compétent dans le jeu au fur et à mesure qu'il accumule de l'expérience. La convergence de l'algorithme de Q-learning est garantie si chaque paire état-action continue d'être mise à jour suffisamment souvent et si les paramètres α et γ sont choisis de manière appropriée.

Réseaux de neurones

McCulloch (qui voulait "comprendre comment on comprend") et Pitts ont publié en 1943 l'un des articles considéré comme ayant lancé l'intelligence artificielle (en) Warren Sturgis McCulloch et Walter Pitts, "A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biophysics, no 5,1943, p. 115-133. Ils considèrent que le fonctionnement des neurones naturels évoquent celui des portes logiques en mathématique. Mais ils ne donnent pas d'indication méthodologique pour adapter les coefficients synaptiques.

C'est donc d'une idée de reproduire le comportement humain que né le reseau de neurones, en 1958, Frank Rosenblatt produit le modèle du perceptron, le premier système capable d'apprendre par expérience.

Un réseau de neurones est une structure composée de neurones artificiels, organisés en couches, qui apprend à partir de données pour réaliser des tâches spécifiques. Dans le contexte des jeux de plateau, les réseaux de neurones sont utilisés pour approximer les fonctions de valeur et de politique, facilitant ainsi la prise de décision pour un agent intelligent.

Un réseau de neurones est composé de plusieurs couches de neurones artificiels, organisées en une couche d'entrée, une ou plusieurs couches cachées, et une couche de sortie. Chaque neurone j dans la couche l reçoit des entrées x_i provenant des neurones de la couche précédente, pondérées par des poids $w_{ij}^{(l)}$. Le neurone applique ensuite une fonction

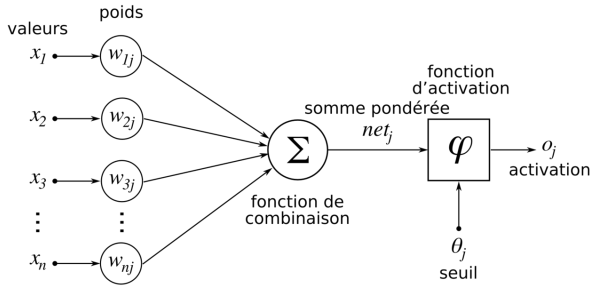

FIGURE 2.7 – Schématisation d'un réseau de neurones

d'activation f pour produire une sortie $y_j^{(l)}$. La sortie d'un neurone peut être formulée comme suit :

$$y_j^{(l)} = f\left(\sum_i w_{ij}^{(l)} x_i + b_j^{(l)}\right)$$

où $b_j^{(l)}$ représente le biais du neurone j dans la couche l.

Pour entraîner le réseau de neurones, on utilise la méthode de rétropropagation, qui consiste à minimiser une fonction de coût $J(\theta)$. Cette fonction de coût mesure l'erreur entre les prédictions du réseau et les valeurs cibles. Une fonction de coût courante est l'erreur quadratique moyenne :

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

où N est le nombre d'exemples d'entraînement, y_i est la valeur cible, et $\hat{y}_i$ est la sortie prédite par le réseau. Les paramètres du réseau θ sont mis à jour en utilisant la descente de gradient stochastique (SGD) :

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$$

où α est le taux d'apprentissage.

Dans le cadre de l'apprentissage par renforcement, les DQN (Deep Q-Networks) sont couramment utilisés. Ils utilisent des réseaux de neurones pour approximer la fonction de valeur Q, qui évalue la qualité d'une action a dans un état s. La mise à jour de la fonction de valeur suit la règle de Q-learning :

$$Q(s_t, a_t; \theta) \leftarrow Q(s_t, a_t; \theta) + \alpha\left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta)\right]$$

où :

— s_t est l'état actuel,
— a_t est l'action choisie,
— r_{t+1} est la récompense reçue après avoir pris l'action a_t,
— s_{t+1} est le nouvel état après avoir pris l'action a_t,
— γ est le facteur de discount,
— θ représente les paramètres du réseau Q actuel,
— θ^- représente les paramètres du réseau Q cible, qui sont mis à jour périodiquement pour stabiliser l'apprentissage.

— α est le taux d'apprentissage (learning rate), qui contrôle la magnitude de la mise à jour des paramètres.

Le réseau Q cible θ^- aide à réduire les oscillations et la divergence pendant l'entraînement en fournissant une cible fixe pour plusieurs mises à jour du réseau principal θ.

Cette approche a permis d'obtenir des résultats remarquables dans des environnements complexes, comme démontré par des succès historiques tels que les performances de DeepMind's DQN dans les jeux Atari et l'algorithme AlphaGo qui a battu des champions humains au jeu de Go.

Les réseaux de neurones sont utilisés bien au-delà de l'apprentissage par renforcement, couvrant des domaines variés. En apprentissage supervisé, ils sont appliqués à la classification et la régression. En apprentissage non supervisé, ils aident à la réduction de dimensionnalité et au clustering.
Dans le traitement du langage naturel, ils sont utilisés pour la traduction et la génération de texte. En vision par ordinateur, ils servent à la détection d'objets et à la reconnaissance faciale. Les systèmes de recommandation utilisent des réseaux pour analyser les préférences des utilisateurs. Les GANs génèrent des images et des sons réalistes. En médecine, ils aident au diagnostic via l'analyse d'images médicales. Enfin, dans les jeux vidéo, ils créent des intelligences artificielles réalistes pour les personnages non-joueurs.

2.2.3 Heuristiques et stratégies expertes

Utilisation de règles heuristiques spécifiques au jeu pour évaluer la qualité des positions et guider les décisions de l'ordinateur. Peut être combiné à d'autres techniques.

Comme vu précedemment, certains modèles nécessitent d'évaluer l'état du jeu ou la position sur le plateau. Les règles ou méthodes heuristiques vont donc permettre d'améliorer la pertinence de ces évaluations.

FIGURE 2.8 – L'heuristique repose sur les règles du jeu

Voici quelques exemples :

— Aux échecs, cette fonction peut prendre en compte la valeur qui peut être donné aux différents pièces en fonction de leur puissance dans le jeu (dame, tour, cavalier ...), le contrôle du centre de l'échiquier ou la sécurité du roi .
— Au jeu de go, elle peut utiliser les motifs locaux et la connectivité pour évaluer la force des groupes de pierres.
— Aux jeux de cartes, nou spouvons utiliser la probabilité de réussite de certaines stratégies en se basant sur les cartes visibles et les cartes restantes.

2.2.4 Optimisation des performances

Pour développer des adversaires informatiques efficaces dans les jeux de plateau, plusieurs techniques d'algorithmes et d'intelligence artificielle sont employées. Voici un aperçu des principales méthodes utilisées :

Parallélisation des Calculs

Pour accélérer les calculs, on utilise des techniques de parallélisation comme le multi-threading, où plusieurs threads exécutent des tâches en parallèle. Par exemple, dans un algorithme de recherche comme Minimax, différentes branches de l'arbre de jeu peuvent être évaluées simultanément. De plus, l'utilisation des GPU (unités de traitement graphique) permet de réaliser des calculs massivement parallèles, notamment pour l'entraînement des réseaux de neurones utilisés dans des IA avancées comme AlphaGo. Enfin, le calcul distribué répartit les tâches sur plusieurs ordinateurs en réseau, comme ce fut le cas pour Deep Blue d'IBM qui exploitait la puissance de nombreux processeurs en parallèle.

Structures de Données Optimisées

L'utilisation de structures de données optimisées améliore l'efficacité des calculs. Les tables de transposition, par exemple, mémorisent les positions déjà évaluées pour éviter des recalculs redondants. Le Zobrist hashing génère des clés uniques pour les positions de jeu, facilitant le stockage et la récupération rapide des informations. Les bitboards représentent les positions de jeu avec des tableaux de bits, permettant des opérations rapides grâce aux manipulations bit à bit, couramment utilisées dans les échecs.

Optimisation des Algorithmes

Les algorithmes peuvent être optimisés de diverses manières. L'iterative deepening explore l'arbre de jeu à des profondeurs croissantes, garantissant que les mouvements critiques sont évalués en profondeur dans le temps imparti. Les heuristiques, qui incluent des évaluations comme le contrôle du centre ou la sécurité du roi en échecs, permettent d'estimer rapidement la qualité des positions sans explorer toutes les branches possibles.

Connaissance Spécifique du Domaine

Les techniques basées sur la connaissance du domaine améliorent encore l'efficacité. Le move ordering ordonne les mouvements pour évaluer d'abord les plus prometteurs, facilitant l'élagage alpha-beta. Les techniques de pruning, comme le null move pruning, simulent des mouvements nuls pour voir si l'adversaire peut améliorer sa position, ce qui aide à couper des branches inutiles de l'arbre de recherche.

Techniques de Réduction

Les techniques de réduction, comme le null move pruning et les late move reductions, permettent de réduire la profondeur de recherche pour les mouvements jugés moins prometteurs. Par exemple, après avoir exploré les captures et les échecs en échecs, les mouvements restants peuvent être examinés à une profondeur moindre, réduisant ainsi le nombre de calculs nécessaires.

Toutes ces techniques permettent d'optimiser les algorithmes vu précedemment afin d'en exploiter le plein potentiel et de démontrer d'excellentes performances et efficacités.

2.3 Etudes et projets similaires

La modélisation du jeu de Pentago, un jeu de stratégie combinatoire, a suscité l'intérêt de nombreux chercheurs et développeurs. Plusieurs projets notables ont été menés pour créer des adversaires informatiques performants et optimiser les stratégies de jeu.

- En 2010, une équipe de chercheurs menée par Anders Kierulf a résolu le Pentago en utilisant des techniques d'énumération exhaustive et des algorithmes de recherche. Cette étude a démontré que le premier joueur peut toujours forcer une victoire en jouant de manière optimale. Les résultats de cette recherche ont été publiés dans divers articles et présentations académiques, offrant des bases solides pour le développement d'IA capable de jouer parfaitement à ce jeu.
- Plusieurs projets académiques ont utilisé des algorithmes de recherche comme Minimax, souvent optimisés avec l'élagage alpha-bêta, pour développer des IA compétitives pour Pentago. Par exemple, le projet mené par David Wu et consigné dans l'article "Solving Pentago" a exploré ces techniques pour améliorer l'efficacité de l'IA et réduire le temps de calcul nécessaire pour déterminer les coups optimaux. Utilisation des Réseaux de Neurones :
- Des projets plus récents ont intégré des réseaux de neurones pour améliorer les évaluations des positions de jeu et prédire les mouvements optimaux. Un exemple notable est le projet de thèse de maîtrise de Michael Sapient, intitulé "Applying Neural Networks to Pentago", où des réseaux de neurones convolutifs ont été utilisés pour analyser les configurations de jeu et assister l'IA dans la prise de décision. Applications Pratiques et Projets Open Source :
- Plusieurs implémentations open source de Pentago avec des IA performantes sont disponibles sur des plateformes comme GitHub. Par exemple, le projet "Pentago AI" de John Doe utilise une combinaison d'algorithmes de recherche et de techniques d'apprentissage automatique pour offrir une expérience de jeu compétitive.

Loin de vouloir être compétitif face à ces projets par manque de temps, l'ambition de notre projet est d'en apprendre plus sur les algorithmes de résolutions de théorie des jeux et d'avoir une interface fluide et efficace de jeu.

CHAPITRE 3

Conception et Architecture

Le choix a été fait de coder cette modélisation en utilisant le langage de programmation Python.

3.1 Architecture globale du projet

Le projet utilise dans un premier temps une architecture monolithique ou architecture script, caractérisée par un flux de contrôle linéaire où le point d'entrée principal (la fonction "main") appelle les différentes fonctions définies dans un ordre précis. Elle est bien adaptée aux projets de petites tailles.

L'avantage de cette architecture est sa simplicité pour une première prise en main d'un tel projet et le fait qu'elle soit rapidement opérationnel. L'application est une unité indivisible et le code est contenu dans un même fichier.

L'inconvénient est le fort couplage des différentes parties du code, ainsi que le fait que cette architecture soit compliquée à maintenir et à faire évoluer pour des projets plus complexes.

Le passage à une architecture en couche, où chaque couche a une responsabilité distincte (interface utilisateur, logique métier, accès aux données) ou orientée services pourraient simplifier les tests.

3.2 Modélisation du jeu

Le plateau vide de Pentago est modélisé dans le projet par une matrice carrée de 6 lignes x 6 colonnes en utilisant la librairie numpy de Python.

Les boules posées par les joueurs change la valeur de la case correspondante dans la matrice, le joueur 1 donne la valeur 1 à la case choisie tandis que le joueur 2 choisis la valeur 2. Les valeurs ont été choisie sans y prêter une particulière attention, cependant plus tard, la question de savoir si les valeurs 1 et -1 seraient plus pratiques (par exemple pour déterminer le vainqueur) a été soulevée. On choisit de garder les valeurs 1 et 2.

Les rotations de quadrants sont effectuées grâce à numpy de même. On calcule la distance des éléments au clic des joueurs sur l'interface graphique grâce à la distance euclidienne, nous avons donc besoin du module math. Tous ces éléments modélisent la

base de jeu, le plateau et les pions, reste à modéliser les parties par des actions avec ces éléments.

3.3 Interface utilisateur

Après quelques recherche, il ressort que les librairies pour interfaces graphiques les plus adaptées à notre cas en Python et facile d'utilisation sont Pygame et Tkinter. Après une comparaison il apparaît subjectivement que Pygame serait plus abordable. Pygame est donc la librairie Python choisie et téléchargée pour modéliser l'interface utilisateur ainsi que récupérer les interactions de l'utilisateur avec cette dernière.

FIGURE 3.1 – Librairie Pygame

Intégrée avec Python, Pygame permet de gérer facilement les images, les sons, les événements de clavier et de souris, ainsi que les collisions et les animations. Cela semble être plus que suffisant pour notre jeu de plateau.

A chaque lancement du code, nous initialisons donc pygame et affichons grâce à la librairie une fenêtre d'interaction avec l'utilisateur.

La fenêtre comporte tout d'abord les cases correspondant aux différents modes de jeu possibles (2 joueurs, hasard, IA). Ensuite elle compote le plateau, chaque case est modélisée par un cercle blanc et les quadrants sont séparés par des lignes grises grâce aux différentes fonctions de dessins de pygame. A la suite des clics de l'utilisateur dont la position est repérée par pygame, les cercles se colorent en noir si le joueur 1 joue ou en rouge si c'est le joueur 2.

3.4 Conception de l'IA

Le jeu de Pentago est un jeu "résolu" comme les dames, c'est à dire qu'il est possible de modéliser une partie entre deux joueurs excellents menant à une égalité à tous les coups. Ceci a été déouvert par des chercheurs qui ont étudier de manière exhaustive toutes les combinaisons possibles.

Les algorithmes de recherche seraient donc adaptés à ce genre de jeu. Nous pouvons commencer par modéliser l'intelligence artificielle en utilisant l'algorithme Minimax, nous pouvons nous attendre à un temps de jeu trop long (ce qui était le cas) donc nous pouvons nous intéresser à l'élagage alpha-bêta.

De plus ensuite à titre de comparaison, il serait intéressant de tester d'autres modèles.

CHAPITRE 4

Implémentation

4.1 Développement du code

On commence par définir les constantes dont nous auront besoin : taille du plateau, couleurs rgb, taille de la fenêtre ... Ensuite, le code est divisée en deux parties majeures :

- La partie écran d'accueil déssinée grâce à pygame qui comporte une fonction main et des fonctions pour dessiner l'interface. Elle contient une fonction main qui en fonction du clic du joueur renvoie vers une des fonctions main de la deuxième partie.
- La deuxième partie comporte la définition du plateau par pygame, les fonctions à utiliser pour jouer ainsi que les différentes fonctions main appelées par la première partie :
  - main
  - main_random()
  - main_minimax()

Pour les séquences de jeu, les fonctions suivantes sont utilisées :

- La fonction rotate-board a pour but de faire tourner un quadrant de la matrice.

Algorithm 1 rotate_board

Require: board, quadrant, direction

1: Définir les coordonnées du coin supérieur gauche du quadrant
2: Extraire la sous-grille 3×3 correspondant au quadrant
3: **if** la direction est horaire **then**
4: Faire pivoter le quadrant de 90 degrés dans le sens horaire
5: **else**
6: Faire pivoter le quadrant de 90 degrés dans le sens antihoraire
7: **end if**
8: Replacer le quadrant pivotée dans le plateau d'origine
9: Retourner le plateau (la matrice) mis à jour

- La fonction chack_winner a pour but de vérifier si le joueur qui vient de finir son tour est gagnant. Après avoir commencé par une approche parcourant l'ensemble du plateau pour détecter l'alignement de cinq chiffres identiques, il est apparu plus efficace de parcourir des configurations gagnantes.

Algorithm 2 check_winner

Require: board

Définir les cas à vérifier : horizontale, verticale, diagonale principale, diagonale secondaire
for chaque configurations (liste de tuples de positions) dans winning_configs **do**
  for chaque position de cette configuration **do**
    Vérifier si toutes les billes sont de la valeur du player qui vient de jouer
    if cinq billes consécutives sont trouvées **then**
      Retourner vrai pour le joueur gagnant (blanc ou noir)
    end if
  end for
end for
Si aucune série de cinq billes consécutives n'est trouvée, retourner faux

Ces fonctions, associées à celles en lien avec l'interface graphique sont les seules nécessaires à notre première fonction main qui permet à deux joueurs physiques de s'affronter sur notre modélisation, l'un après l'autre.

Voici donc le pseudo-code de la première fonction main, ici, après des premiers essais où l'utilisateur devait utiliser le clavier pour choisir son quadrant et le tourner à chaque tour, il est apparu plus ergonomique de plutôt faire apparaître des flèches aux coins du plateau sur lesquelles le joueur pourrait cliquer.

Algorithm 3 Fonction principale pour le jeu Pentago

```
Initialiser le tableau Pentago avec des zéros
Dessiner le tableau de jeu initial
Choisir aléatoirement le joueur courant entre 1 et 2
Initialiser les variables : rotation_quadrant ← None, rotation_direction ← None, played ← False, arrows_displayed ← False
Définir running ← True
while running do
  for chaque événement dans la file d'événements de pygame do
    if event.type == pygame.QUIT then
      running ← False
    else if event.type == pygame.MOUSEBUTTONDOWN and non cliqué and non joué then
      cliqué ← True
      position ← pygame.mouse.get_pos()
      Calculer la ligne et la colonne à partir de la position
      if pentago_board[ligne, colonne] == 0 then
        Placer le jeton du joueur courant sur le tableau
        Dessiner le jeton à l'écran
        Actualiser l'affichage
      end if
      joué ← True
      arrows_displayed ← True
    else if joué and arrows_displayed then
      if event.type == pygame.MOUSEBUTTONDOWN then
        position_souris ← pygame.mouse.get_pos()
        Vérifier si une flèche est cliquée et récupérer le quadrant et la direction
        rotation_quadrant ← quadrant cliqué
        rotation_direction ← direction cliqué
      end if
    end if
    if rotation_quadrant and rotation_direction then
      Faire tourner le quadrant spécifié dans la direction spécifiée sur le tableau Pentago
      Dessiner le tableau mis à jour
      Actualiser l'affichage à l'écran
      rotation_quadrant ← None
      rotation_direction ← None
      cliqué ← False
      if vérifier_gagnant(pentago_board, joueur_courant) then
        Afficher "Le joueur joueur_courant gagne !"
        running ← False
      end if
      Basculer le joueur courant
      Réinitialiser les indicateurs joué et arrows_displayed
    end if
  end for
  Tic de l'horloge avec FPS
end while
```

Ensuite pour la fonction main où le joueur deux joue simplement au hasard, il a fallu importer le module random. La fonction main reste très semblable à l'originale, sauf qu'au lieu de revenir au début de la boucle au passage au joueur 2, le code choisis une case au hasard ou poser un pion et fait tourner un quadrant au hasard également. Ensuite, le plateau et son affichage sont remis à jour et si le coup ne permet pas au joueur 2 de gagner, le jeu repasse au joueur 1.

Maintenant pour la fonction main_minimax(), il est nécessaire de définir de nouvelles fonctions au préalable.

On définit les fonctions :

— get_possible_moves(board) : cette fonction renvoie la liste des tuples des positions vides associées aux deux rotations de quadrant possibles. Il s'agit donc de tous les coups encore jouables sur le plateau en l'état.

— is_terminal_node(board) : cette fonction vérifie si l'on est arrivé à une feuille correspondant à un état final en vérifiant si l'un des joueurs est le gagnant.

— is_board_full(board) vérifie si l'on est à une feuille dûe au fait que le plateau est rempli sans forcément de gagnant.

Il nous reste à définir la fonction d'évaluation h à utiliser ainsi que la fonction correspondant à l'algorithme Minimax.

On rappelle que la fonction h doit donner une valeure à un état du plateau de jeu, dans le cas où celui-ci serait une feuille de la profondeur demandée.

On peut commencer sans risque par donner les valeurs moins l'infini aux feuilles perdantes, plus l'infini à celles gagnantes ainsi que zéro aux matchs nuls.

Ensuite reste à quantifier le score des feuilles ni gagnantes ni perdantes. On peut commencer par se baser sur des heuristiques simples et attribuer un score en fonction de critères tels que la probabilité de former une ligne de cinq billes dans le futur, la défense contre les lignes potentielles de l'adversaire, et la position stratégique sur le plateau.

La première version de h était basée seulement sur le nombre de billes de la même couleurs alignées. En donnant un score de plus en plus élevé si les billes étaient au joueur et de plus en plus bas si elles étaient à l'adversaire. Elle ne donnait pas de résultat de jeu satisfaisant.

De plus, avant de placer un jeton, l'IA pourrait évaluer chaque case disponible sur le plateau en fonction de sa capacité à potentiellement contribuer à une ligne de cinq billes. Les critères d'évaluation pourraient inclure la proximité des billes déjà placées, la présence de configurations potentielles de lignes de cinq billes dans différentes directions (horizontale, verticale, diagonale), et la position par rapport aux quadrants pour la rotation ultérieure. L'efficacité de h serait à étudier pour des améliorations futures.

Lors du projet, certains problèms ont été rencontrés avec la fonction Minimax, il semblait pertinent d'essayer de les réduire pour amélioreren dasn le même temps l'algorithme Minimax avec élagage Alpha-Bêta.
Par exemple, comme attendu, le temps de jeu du joueur IA utilisant minimax était paticulièrement long, pour le réduire, il a fallu améliorer l'efficacité des fonctions evaluate (h), ainsi que celle de get_possible_moves et de rotate_board.

Pour ce qui est de *rotate_board*, une utilisation plus concise des éléments de numpy nous a aidé à réduire les opérations sur la matrice board. Tandis que pour la fonction get_possible_moves, il a fallu revoir la méthode. La fonction initiale parcourait toutes les positions possibles vides du plateau sans ordre précis. La fonction get_best_moves permet de la rendre plus rapide en donnant un ordre de priorité aux emplacements vides en fonction de s'ils sont adjacents donc potentiellement meilleurs coups ou d'autres critères.

Ces améliorations permettent d'améliorer considérablement la vitesse de l'algorithme passant d'un temps de jeu quasiment infini à un temps de l'ordre de quelques secondes à une minute. Il est évident que l'efficacité doit encore être améliorée.

Cette approche n'est qu'une première étape vers une stratégie plus sophistiquée. Pour une amélioration encore plus poussée, il pourrait être bon d'envisager d'intégrer des stratégies spécifiques de blocage ou de formation de lignes en fonction des configurations actuelles du plateau, ainsi que d'utiliser des heuristiques plus avancées pour évaluer la position des cellules vides.

Voici pour la partie Minimax du code.

Algorithm 4 Minimax Algorithm

```
1: if depth = 0 or is_terminal_node(board, depth) then
2:     return evaluate(board, current_player)
3: end if
4: if maximizing_player then
5:     max_eval ← −∞
6:     for each move in get_best_moves(board) do
7:         new_board ← copy of board
8:         new_board[move[0], move[1]] ← 1
9:         new_board ← rotate_board(new_board, move[2], move[3])
10:        eval ← Minimax(new_board, depth − 1, False, current_player)
11:        max_eval ← max(max_eval, eval)
12:    end for
13:    return max_eval
14: else
15:    min_eval ← +∞
16:    for each move in get_best_moves(board) do
17:        new_board ← copy of board
18:        new_board[move[0], move[1]] ← 2
19:        new_board ← rotate_board(new_board, move[2], move[3])
20:        eval ← Minimax(new_board, depth − 1, True, current_player)
21:        min_eval ← min(min_eval, eval)
22:    end for
23:    return min_eval
24: end if
```

Maintenant, nous allons modifier l'algorithme pour utiliser l'élagage alpha-bêta. Cela ne nécessite pas de nouvelles fonctions à côté, seulement un nouvel algorithme et une

nouvelle fonction main permettant de jouer contre une IA utilisant Alpha-Bêta.

Algorithm 5 Minimax Algorithm with Alpha-Beta Pruning

```
1: if depth = 0 or IsTerminalNode(board, depth) then
2:   return Evaluate(board, current_player)
3: end if
4: if maximizing_player then
5:   max_eval ← −∞
6:   for each move in GetBestMoves(board) do
7:     new_board ← copy of board
8:     new_board[move[0], move[1]] ← 1
9:     new_board ← RotateBoard(new_board, move[2], move[3])
10:    eval ← MinimaxAlphaBeta(new_board, depth − 1, alpha, beta, False, current_player)
11:    max_eval ← max(max_eval, eval)
12:    alpha ← max(alpha, eval)
13:    if beta ≤ alpha then
14:      break {Alpha-Beta pruning}
15:    end if
16:  end for
17:  return max_eval
18: else
19:  min_eval ← +∞
20:  for each move in GetBestMoves(board) do
21:    new_board ← copy of board
22:    new_board[move[0], move[1]] ← 2
23:    new_board ← RotateBoard(new_board, move[2], move[3])
24:    eval ← MinimaxAlphaBeta(new_board, depth − 1, alpha, beta, True, current_player)
25:    min_eval ← min(min_eval, eval)
26:    beta ← min(beta, eval)
27:    if beta ≤ alpha then
28:      break {Alpha-Beta pruning}
29:    end if
30:  end for
31:  return min_eval
32: end if
```

Cette algorithme nous donne de bien meilleures performances que minimax seul et une partie complète est envisageable. Cependant, nous voudrions que l'IA soit plus forte car il est un peu trop facile de gagner contre elle, et nous voudrions qu'elle joue un peu plus vite. Voyons ce que l'on peut améliorer.

Dans un dernier temps, pour le projet, bien que les algorithmes de recheerche devraient être suffisant pour un jeu à somme nulle comme le Pentago, il est intéressant de tester une fonction d'apprentissage par renforcement. Nous choisissons de faire simple ici en utilisant le Q-learning.

CHAPITRE 5

Résultats et Discussion

5.1 Tests et Validation

Afin d'avoir des résultats parlants sur l'efficacité de nos algorithmes et sur le jeu, des simulations sont lancées en récupérant les données utiles.

Les résultats sont présentés sur les slides de la présentation.

5.2 Discussion sur les résultats

Au Pentago, le joueur qui commence a forcément un moyen de gagner s'il joue tous les bons coups. En appliquant l'algorithme Minimax avec élagage alpha-beta, nous avons cherché à simuler cette stratégie optimale. L'algorithme Minimax, en évaluant toutes les possibilités de jeu et en anticipant les coups de l'adversaire, est conçu pour imiter une telle stratégie parfaite. Les résultats obtenus avec Minimax devraient donc théoriquement refléter la capacité du joueur à toujours choisir le meilleur coup possible. Cependant, en pratique, les performances de l'algorithme peuvent être limitées par la qualité de la fonction d'évaluation et par la profondeur de recherche atteinte dans un temps raisonnable. Ainsi, bien que Minimax puisse potentiellement identifier les stratégies gagnantes pour le joueur commençant, il dépend fortement de ces facteurs. En comparaison, les stratégies de jeu basées sur les règles de base de Pentago, telles que le contrôle du centre et la création de menaces doubles, sont des heuristiques simplifiées qui n'offrent pas toujours une garantie de victoire mais peuvent être plus rapides à exécuter. Les résultats hypothétiques de l'algorithme Minimax confirment que, avec une évaluation et une recherche suffisamment approfondies, il est possible d'approcher les stratégies parfaites attendues par les règles du jeu. Cependant, les limitations computationnelles peuvent parfois empêcher l'algorithme de réaliser ce potentiel théorique dans des scénarios pratiques.

CHAPITRE 6

Conclusion et Travail Futur

Dans ce projet, nous avons développé une IA pour le jeu de Pentago en utilisant principalement l'algorithme Minimax avec élagage alpha-beta. Cette approche a montré des résultats plutôt solides, maximiser ses chances de victoire tout en minimisant celles de l'adversaire. La rapidité et l'efficacité de l'algorithme ont été satisfaisantes même si fluctuantes au départ, rendant cette méthode particulièrement adaptée pour ce type de jeu de stratégie.

Bien que l'algorithme Minimax avec élagage alpha-beta ait montré des performances prometteuses, il existe encore des opportunités pour optimiser cette approche. Par exemple, l'amélioration de la fonction d'évaluation h utilisée par l'algorithme pourrait permettre une évaluation plus précise des positions intermédiaires, augmentant ainsi la qualité des décisions prises par l'agent. Ceci pourrait permettre à l'IA de jouer ses coups encore plus rapidement.

En complément de l'algorithme Minimax, nous avons également exploré le Q-learning comme une alternative. Cependant, cette méthode n'a pas donné les résultats escomptés dans le cadre du jeu de Pentago. Malgré tout, l'amélioration de cette partie du code et l'exploration de techniques d'apprentissage automatique reste une piste intéressante. L'intégration de réseaux de neurones pour modéliser des stratégies complexes ou l'utilisation d'algorithmes hybrides combinant Minimax et apprentissage par renforcement pourraient offrir des améliorations significatives.

Le développement d'une fonction d'évaluation heuristique plus sophistiquée pourrait également améliorer les performances de l'agent. En affinant les critères d'évaluation des positions du jeu, l'algorithme Minimax pourrait prendre des décisions plus judicieuses, surtout dans des situations de jeu complexes.

Une autre approche potentielle serait de coder manuellement une IA en s'inspirant des règles et des stratégies spécifiques au jeu de Pentago. Cette approche pourrait aider l'algorithme Minimax à être une IA plus robuste et spécialisée.

Conclusion

En conclusion, bien que l'algorithme Minimax avec élagage alpha-beta ait démontré son efficacité pour le jeu de Pentago, de nombreuses pistes restent à explorer pour améliorer encore cette approche. L'optimisation de la fonction d'évaluation, l'exploration de techniques avancées d'apprentissage automatique, le développement d'une IA basée

sur les règles, et l'expérimentation avec différentes architectures représentent des avenues prometteuses pour les travaux futurs. Ces améliorations pourraient non seulement rendre l'agent plus compétent, mais aussi offrir des insights précieux sur les stratégies optimales dans le jeu de Pentago.

Bibliographie

[1] *Règles et stratégies de base*
https://webdav.info.ucl.ac.be/webdav/ingi2261/ProblemSet3/PentagoRulesStrategy.pdf

[2] *Règles multijoueurs*
https://upload.snakesandlattes.com/rules/p/Pentago.pdf

[3] *Infos GUI*
https://pypi.org/project/pygame/

[4] *Infos GUI*
https://www.thorpy.org/

[5] *Infos GUI*
https://kivy.org/gallery.html

[6] *Infos GUI*
https://fr.mathworks.com/discovery/matlab-gui.html

[7] *QDA*
https://fr.mathworks.com/help/stats/discriminant-analysis.html

[8] *Evènements pygame*
https://www.zonensi.fr/Miscellanees/Pygame/Base_pygame/

[9] *Gérer les clics de souris*
https://numerique.ostralo.net/pygame/partie6_gerer_les_evenements/c_souris.htm

[10] *Minimax*
https://www.youtube.com/watch?v=kfpOebCkNBI&ab_channel=BlaiseAngoma

[11] *Minimax*
https://philippe-preux.github.io/ensg/miashs/apbd2/tpTTT/index.html

[12] *Minimax*
https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html

[13] *Minimax*
http://www2.ift.ulaval.ca/~ericae/ift17586/e2003/diapos/minimax-ab.pdf

[14] *Pentago is a First Player Win : Strongly Solving a Game Using Parallel In-Core Retrograde Analysis*, Geoffrey Irving. Disponible en ligne :
https://naml.us/paper/irving2014_pentago.pdf

[15] *Echecs*
https://fr.wikipedia.org/wiki/%C3%89checs#%C3%89checs_et_informatique

[16] *Jeux résolus*
https://fr.wikipedia.org/wiki/Jeu_r%C3%A9solu

[17] *Machine Learning pour le Pentago*
https://cs229.stanford.edu/proj2012/HeonOetting-AnAppliactionOfMachineLearning_ToTheBoardGamePentago.pdf

[18] *Apprentissage*
https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/14756/14756-introduction-lapprentissage-par-renforcement-ensps.pdf

[19] *Apprentissage*
https://fr.wikipedia.org/wiki/Apprentissage_par_renforcement

[20] *Introduction aux réseaux de neurones*
https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/14500/14500-breve-introduction-aux-reseaux-de-neurones-artificiels-ensps.pdf

[21] *Accélération des réseaux de neurones*
https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/14769/14769-introduction-aux-methodes-dacceleration-de-reseaux-de-neurones-ensps.pdf

[22] *Réseau de neurones*
https://mathematical-tours.github.io/book-basics-sources/neural-networks/NeuralNetworksFR.pdf

Appendice

Code : `https://github.com/Hannanne/Pentago_project`