Université de Caen Normandie

Projet annuel - Semestre 1

Master 1 Intelligence Artificielle et Facteurs Humains

Algorithmes pour les jeux avec modèles d'adversaire

Encadré par :
M. Bruno Zanuttini

Réalisé par :
Redwan Omari
Eldis Ymeraj

Année Universitaire 2024-2025

Table des matières

Table des figures

Listings

Liste des tableaux

1 Introduction générale

Les jeux adversariaux, comme le Morpion et le Domino, sont des moyens intéressants pour apprendre et expérimenter des stratégies de prise de décision. Ils nous permettent d'explorer comment un joueur peut prendre les meilleures décisions face à un adversaire, tout en considérant les règles et les défis spécifiques de chaque jeu.

Dans ce projet, nous avons choisi d'étudier deux jeux très différents : le Morpion, où toutes les informations sont visibles pour les joueurs, et le Domino, où une partie des informations reste cachée. Ces deux jeux nous ont permis d'étudier des problèmes variés et de tester différentes approches pour résoudre ces situations.

Notre objectif a été de créer un système qui peut réfléchir et prendre des décisions intelligentes. Pour cela, nous avons utilisé l'algorithme MinMax, qui est capable de calculer les meilleurs coups possibles dans une partie. Nous avons aussi ajouté un modèle d'adversaire pour rendre les parties plus réalistes, ce qui permet à notre système de mieux anticiper les choix des autres joueurs.

Nous avons aussi travaillé pour organiser notre code de manière simple et claire, en créant des règles communes pour les deux jeux. Cela nous permet d'imaginer d'ajouter d'autres jeux ou stratégies dans la deuxieme partie de ce porjet.

Dans ce rapport, nous allons expliquer ce que nous avons fait, comment nous avons réfléchi à chaque étape, et quels résultats nous avons obtenus.

2 Objectifs du projet

2.1 Problématique générale

Dans les jeux adversaries, chaque joueur doit non seulement choisir les meilleures actions pour lui-meme, mais aussi anticiper ce que son adversaire pourrait faire. Cette réflexion devient encore plus difficile dans les jeux à information incomplète, comme le Domino, ou les joueurs ne connaisssent pas toutes les informations, comme les dominos restants dans la pioche ou ceux de l'adversaire.

La difficulté principal est d'apprendre à modéliser et à comprendre le comportement de l'adversaire. En utilisant cette information il devient possible de développer des stratégies plus intelligentes et plus efficaces, adaptees à chaque situation. Ce projet nous permet donc d'explorer comment un joueur peut réfléchir de manière optimale, en s'appuyant sur des algorithmes comme MinMax, tout en tenant compte des décisions probables de l'adversaire.

2.2 Description des défis spécifiques liés aux jeux à information complète et incomplète

Les deux jeux que nous avons étudiés, le Morpion et le Domino, posent des défis differentes selon qu'ils soient à information complète ou incomplète :

- **Jeux à information complète** : Dans ces jeux, comme le Morpion tous les joueurs peuvent voirl'état complet du jeu à tout moment. Le principal défi est de calculer les meilleurs coups possibles en explorant les arbres de décision. Cela peut devenir tres complexe si le jeu a beaucoup de possibilités. Pour rendre les calculs plus rapides et efficaces, nous avons utilisé l'algorithme MinMax pour cette premiere partie, mais pour la deuxième partie du projet nous prévoyons d'ajouter Alpha-Beta.
- **Jeux à information incomplète** : Dans ces jeux, comme le Domino certaines informations sont cachées, par exemple les dominos que l'adversaire a en main. Cela rend les décisions plus difficiles, car il faut faire des suppositions sur ce que l'adversaire pourrait jouer. Pour résoudre ce problème, nous avons intégré un modèle d'adversaire qui aide à prédire les actions possibles de l'adversaire et à adapter notre stratégie en conséquence.

2.3 Résultats attendus à courte et moyen terme

Les résultats de ce projet se déclinents en deux étapes : les objectifs à courte terme, correspondant à la première phase du projet (semestre 1), et les objectifs à moyen terme, qui seront développés au cours du deuxième semestre.

Résultats attendus à courte terme (Semestre 1) : Cette première phase a pour but de poser des bases solides pour les jeux à adversaires et d'implémenter des stratégies de prise de décision. Les résultats attendus sont :

- **Développement des jeux :**
  - Une version fonctionnelle du Morpion et du Domino avec des règles respectées et des interactions fluides.
  - L'introduction de l'abstraction pour faciliter l'ajout de nouvelles règles ou extensions futures.
- **Implémentation de l'algorithme MinMax :**
  - Un algorithme MinMax opérationnel dans les deux jeux, capable de prendre des décisions optimales en fonction des situations.
  - Tests de MinMax face à un adversaire simple pour vérifier la cohérence et l'efficacité de ses choix.
- **Simulations et validations :**
  - Des simulations de parties entre MinMax et l'adversaire simple pour s'assurer du bon fonctionnement de l'ensemble.
  - Analyse des résultats pour évaluer les forces et les faiblesses de MinMax dans les deux contextes (information complète et incomplète).

Résultats attendus à moyen terme (Semestre 2) : Dans la deuxième phase, l'objectif sera d'approfondir les travaux réalisés et d'introduire de nouvelles stratégies pour enrichir le système. Les résultats attendus incluent :

- **Développement de nouveaux modèles d'adversaires :**
  - Implémentation de plusieurs adversaires aux comportements variés (adversaires agressifs, défensifs ou aléatoires).
  - Création d'adversaires probabilistes capables de raisonner sur les informations cachées, notamment dans le Domino.
- **Intruduction plusieurs modèles d'adversaier à MinMax :**
  - Pour enrichir notre algorithme MinMax, nous proposons d'ajouter un paramètre permettant de passer une liste de modèles d'adversaires. Chaque modèle sera associé à une probabilité indiquant la probabilité que cet adversaire spécifique soit celui qui joue. Cette approche permet de mieux simuler des scénarios réalistes et d'adapter les décisions de l'algorithme en fonction des comportements probables des adversaires..
- **Comparaison des approches :**
  - Comparer les performances de MinMax contre différents modèles d'adversaires.
  - Analyser comment les stratégies utilisées par l'adversaire impactent les résultats obtenus par MinMax.
- **Analyse globale des performances :**
  - Évaluer les avantages et les limites de chaque méthode implémentée (MinMax, adversaires simples, adversaires probabilistes).
  - Identifier les situations où MinMax peut être améliorés.

3 Répartition des tâches

Nous avons réparti les tâches de manière équilibrée afin que chacun puisse contribuer à des parties spécifiques tout en travaillant ensemble sur les aspects essentiels du projet. Cette organisation nous a permis

de nous concentrer sur nos forces respectives tout en nous assurant que chaque partie du projet soit comprise et retravaillée par les deux membres. Voici comment nous avons organisé notre travail :

- **Eldis Ymeraj :**
  - Développement du jeu de Domino, y compris les règles, la gestion des coups possibles et les calculs de score.
  - Modification de l'algorithme MinMax pour qu'il prenne en compte le modèle d'adversaire dans un jeu à information incomplète comme le Domino.
  - Analyse des résultats des simulations pour le Domino et création des tableaux pour évaluer les performances.
- **Redwan Omari :**
  - Développement du jeu de Morpion, y compris la gestion du plateau, des conditions de victoire et des états terminaux.
  - Conception initiale de l'algorithme MinMax pour un jeu à information complète comme le Morpion.
  - Analyse des simulations du Morpion, avec interprétation des performances de MinMax dans un environnement complet.
- **Travail collaboratif :**
  - Définition des objectifs du projet et planification globale des étapes à réaliser.
  - Structuration des fichiers et création de l'architecture du projet pour garantir une modularité et une organisation claire.
  - Implémentation du modèle d'adversaire simple, utilisé dans les deux jeux pour tester et valider l'efficacité de MinMax.
  - Rédaction et amélioration du rapport, avec des échanges réguliers pour affiner les analyses et les explications.

Remarque : Bien que certaines parties aient été spécifiquement assignées, le projet a été développer de manière collaborative dès le début. Chaque membre a apporter des modifications et des amélioration aux parties initialement prises en charge par l'autre.

4 Méthodes utilisées

Pour permettre à notre programme de jouer contre un adversaire nous avons utilisé l'algorithme MinMax et un adversaire simple basé sur des choix aléatoires. Ces deux éléments nous ont permis de simuler des parties dans les jeux étudiés (Morpion et Domino) tout en testant les capacités stratégique de MinMax.

4.1 Algorithme MinMax

L'objectif de MinMax est de permettre au joueur principal maximisante (MAX) de prendre les meilleures décisions possibles en anticipant les actions de l'adversaire Minimisant (MIN). Dans notre projet, cet algorithme a été adapté pour inclure un modèle d'adversaire, ce qui le rend capable de simuler les décisions probables de l'adversaire avant de jouer. Le joueur Maximisant, maximise son gain en prédire la prochaine coupe d'adversaire.

Fonctionnement de l'algorithme : L'algorithme explore toutes les possibilités en fonction d'un profondeur donné de jeu en suivant une approche recurrsive.

```python
def minimax(jeu, profondeur, maximisant, modele_adversaire):
    if jeu.est_termine() or profondeur == 0:
        return jeu.evaluation()

    coups = jeu.coups_possibles()
    if not coups:
        return jeu.evaluation()

    if maximisant:
        meilleure_eval = float('-inf')
        for coup in coups:
            copie_jeu = jeu.copie()
            copie_jeu.jouer(coup)
            eval = MinMax.minimax(copie_jeu, profondeur - 1, False, modele_adversaire)
            meilleure_eval = max(meilleure_eval, eval)
        return meilleure_eval
    else:
        coup_predit = modele_adversaire.choisir_coup(jeu)
        if coup_predit:
            copie_jeu = jeu.copie()
            copie_jeu.jouer(coup_predit)
            return MinMax.minimax(copie_jeu, profondeur - 1, True, modele_adversaire)
        return jeu.evaluation()
```

Listing 1 – Algorithme MinMax avec modèle d'adversaire

Étapes :

- Si le jeu est terminé ou si la profondeur maximale est atteinte, l'algorithme retourne une évaluation de l'état actuel du jeu.
- Si c'est au tour de MAX, MinMax teste chaque coup possible et choisit celui qui maximise le score.
- Si c'est au tour de MIN, l'algorithme prédit un coup en utilisant le modèle d'adversaire et choisit l'option qui minimise les gains de MAX.
- Les valeurs sont propagées vers le haut de l'arbre pour déterminer le meilleur coup initial.

4.2 Adversaire simple

Pour cette première partie du projet, nous avons utilisé un adversaire simple, défini dans la classe `AdversaireSimple`. Cet adversaire suit une logique très basique : il choisit aléatoirement un coup parmi ceux qui sont possibles.

Fonctionnement de l'adversaire simple : .

```python
def choisir_coup(self, jeu):
    coups_possibles = jeu.coups_possibles()
    print(f"Coups possibles pour l'adversaire : {coups_possibles}")
    if not coups_possibles:
        print("L'adversaire ne peut pas jouer.")
        return None
    return random.choice(coups_possibles)
```

Listing 2 – Adversaire simple dans notre projet

Rôle de l'adversaire simple :

- Dans le Morpion, l'adversaire simple choisit une case disponible au hasard, sans réfléchir aux conséquences stratégiques.
- Dans le Domino, il sélectionne une pièce jouable de manière aleatoire parmi celles qu'il possède sans tenir compte des implications stratégiques à long terme.

Pourquoi un adversaire simple ? Cet adversaire est utilisé dans cette première partie du projet pour tester l'efficacité de l'algorithme MinMax dans un environnement basique. Il ne représente pas une véritable opposition strategique mais permet de valider les performances initiales de notre systeme. Dans la seconde partie du projet, nous prévoyons d'introduire des adversaires plus complexes et adaptatifs.

5 Implémentation de l'algorithme MinMax

Dans cette section, nous expliquons comment l'algorithme MinMax a été implémenté pour les deux jeux choisis : le Morpion et le Domino. Chaque jeu a ses particularités, que MinMax a été adapté à gérer. Nous illustrons ces adaptations par des exemples concrets basés sur des situations de jeu, accompagnés d'arbres de décision.

5.1 Implémentation pour le Morpion

Le Morpion est un jeu à information complète, où tous les joueurs ont une vue claire de l'état du plateau. Cela simplifie l'implémentation de MinMax, car aucune incertitude n'est impliquée.

Fonctionnement de l'algorithme : L'algorithme MinMax explore tous les coups possibles pour le joueur MAX (ici représenté par 'X'), en évaluant chaque état du plateau. La fonction d'évaluation attribue un score à chaque situation :

- **+10** : Victoire de l'algorithme (alignement de trois symboles).
- **-10** : Victoire de l'adversaire (alignement de trois symboles par l'adversaire).
- **+3** : Deux symboles alignés dans une ligne, colonne ou diagonale sans obstruction par l'adversaire.
- **+1** : Un symbole seul dans une ligne, colonne ou diagonale sans obstruction.
- **-3** : Deux symboles alignés par l'adversaire dans une ligne, colonne ou diagonale sans obstruction.
- **-1** : Un symbole seul de l'adversaire dans une ligne, colonne ou diagonale sans obstruction.
- **+2** : Bonus pour contrôler la case centrale (position stratégique).

Ces règles permettent d'évaluer les positions intermédiaires et d'orienter les décisions de l'algorithme vers des coups stratégiques, maximisant ses chances de victoire tout en minimisant celles de l'adversaire.

Exemple illustré : L'image ci-dessous montre un arbre de décision généré par MinMax dans une situation spécifique du Morpion. À chaque niveau, l'algorithme simule les coups possibles et évalue les états de jeu pour choisir le meilleur coup pour MAX.

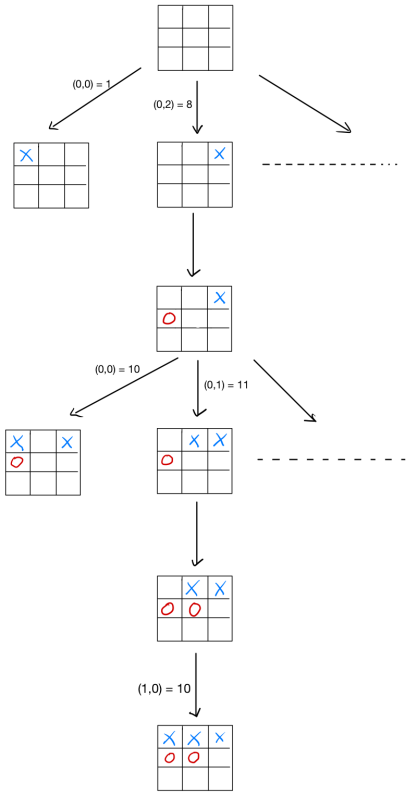

Figure 1 – Arbre de décision généré pour le Morpion à l'aide de l'algorithme MinMax.

Impact : Dans cet exemple, MinMax favorise les positions stratégiques, comme le centre du plateau, en anticipant les menaces adverses et en maximisant ses opportunités de victoire.

5.2 Implémentation pour le Domino

Le Domino est un jeu à information incomplète, comme mentionné précédemment, où les joueurs ne connaissent pas les dominos restants dans la main de l'adversaire. Cela rend l'implémentation de MinMax plus complexe, nécessitant des hypothèses sur les coups possibles de l'adversaire.

Fonctionnement de l'algorithme : Dans le Domino, chaque état de l'arbre de décision inclut :

- Les dominos sur le plateau.
- Les mains des joueurs, avec des hypothèses pour les pièces cachées.
- Le score actuel des joueurs.

La fonction d'évaluation utilise des règles spécifiques pour attribuer un score à chaque coup :

- **+5 points** : Jouer un domino valide.
- **-10 points** : Piocher un domino ou passer un tour.
- **+20 points** : Forcer l'adversaire à piocher.
- **+15 points** : Jouer un domino qui connecte les deux extrémités du plateau.
- **+30 points** : Bonus pour victoire.

Exemple dans une situation spécifique : Voici un arbre de décision pour une situation spécifique dans le Domino. MinMax évalue les coups possibles pour MAX (joueur 1) en simulant les impacts sur le score selon les règles définies.

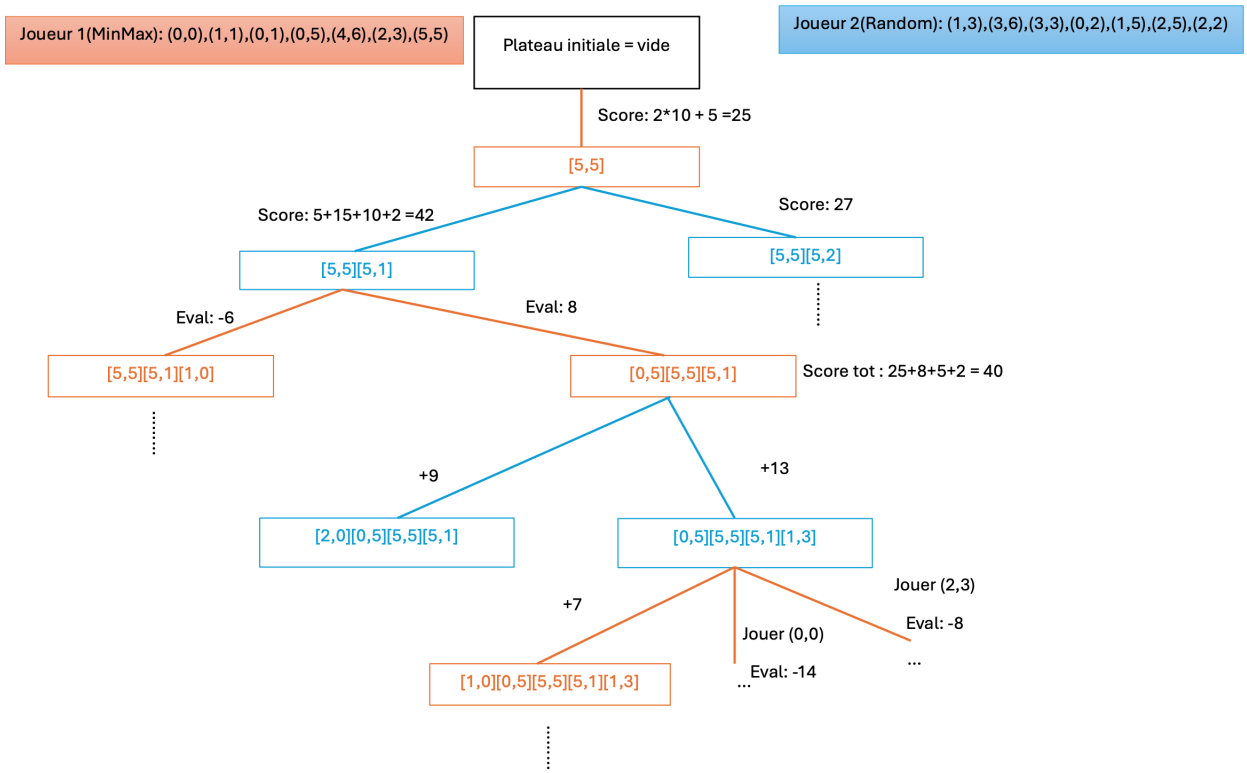

Figure 2 – Arbre de décision généré pour le Domino à l'aide de l'algorithme MinMax.

Pour pallier le manque d'informations sur les pièces adverses, MinMax simule des états où les pièces sont attribuées aléatoirement, permettant ainsi d'évaluer les coups même en présence d'incertitudes.

Dans cet exemple, MinMax joue un rôle essentiel dans la maximisation des points tout en forçant l'adversaire à piocher, un élément stratégique important dans le Domino. Toutefois, lorsque plusieurs coups possibles obtiennent la même évaluation, l'algorithme MinMax sélectionne systématiquement le premier coup rencontré dans la liste des coups possibles. Ce comportement déterministe peut parfois limiter la diversité des choix stratégiques, notamment dans des situations où plusieurs options équivalentes existent. Une amélioration future pourrait inclure un choix aléatoire parmi ces options pour diversifier les décisions de MinMax.

5.3 Résumé des performances

L'algorithme MinMax a démontré une capacité à :

— Identifier les meilleures stratégies dans un environnement à information complète comme le Morpion.

— Gérer l'incertitude dans un jeu à information incomplète comme le Domino, tout en maximisant les scores selon des règles spécifiques.

Ces résultats valident l'efficacité de l'algorithme dans les deux types de jeux étudiés, tout en mettant en lumière des pistes d'amélioration pour des adversaires plus complexes à explorer dans la suite du projet.

6 Structure du projet

6.1 Arborescence des fichiers

Voici l'organisation des fichiers dans notre projet. Chaque dossier et fichier a une fonction bien définie, ce qui facilite la compréhension et la modification de projet.

```
.
|-- Adversaire
|--  |--  Adversaire.py
|--  |--  AdversaireSimple.py
|--  |--  __init__.py
|-- Jeu
|--  |-- Domino.py
|--  |--  Jeu.py
|--  |--  JeuInformationComplete.py
|--  |--  JeuInformationIncomplete.py
|--  |--  Morpion.py
|--  |--  __init__.py
|-- Main.py
|--  MinMax.py
```

Listing 3 – Arborescence des fichiers

6.2 Description des fichiers et dossiers

Le projet est organisé de manière modulaire, ce qui permet une séparation claire des responsabilités.

Dossier `Adversaire`

Ce dossier contient les classes liées aux différants adversaires dans les jeux.

- **Adversaire.py** : Une classe abstraite qui définit le comportement générique des adversaires. Tous les adversaires spécifiques héritents de cette classe.
- **AdversaireSimple.py** : Implémente un adversaire simple et générique. Cet adversaire joue de manière aléatoire dans le Domino et le Morpion.
- **`__init__`.py** : Permet de considérer ce dossier comme un module Python, facilitant l'importation des classes.

Dossier `Jeu`

Ce dossier contient les implémentations des jeux ainsi que les classes abstraites pour généraliser leurs logique.

- **Domino.py** : Définit les règles et la logique du jeu de Domino, y compris la gestion des dominos, des scores et du plateau.
- **Jeu.py** : Une classe abstraite qui sert de base pour tous les jeux, avec des méthodes comme `est_termine()`, `evaluation()` ou `jouer()`.
- **JeuInformationComplete.py** : Hérite de `Jeu.py` et gère les spécificités des jeux à information complète, comme le Morpion.
- **JeuInformationIncomplete.py** : Hérite de `Jeu.py` et gère les jeux à information incomplète, comme le Domino.
- **Morpion.py** : Contient la logique et les règles du Morpion, comme la gestion du plateau et la détection des victoires.
- **`__init__`.py** : Permet de considérer ce dossier comme un module Python.

Fichiers principaux

Ces fichiers sont à la racine du projet et assurent les fonctions clés.

- **Main.py** : Le fichier principal pour lancer le projet. Il permet de choisir entre le Morpion et le Domino, et il gère le déroulement des parties.
- **MinMax.py** : Contient l'implémentation de l'algorithme MinMax, qui est générique et peut être utilisé dans les deux jeux sans modifications supplémentaires.

6.3 Principes d'organisation

Notre projet suit une structure modulaire pour rendre le code facile à comprendre et à étendre. Par exemple :

- Les comportements des adversaires sont séparés des règles des jeux. Cela permet d'ajouter facilement de nouveaux types d'adversaires sans modifier les jeux eux-mêmes.
- Chaque jeu a ses propres fichiers, mais ils partagent une base commune grâce à des classes abstraites. Cela garantit une cohérence et évite la duplication de code.
- L'algorithme MinMax est écrit de manière générique, ce qui évite de créer des versions spécifiques pour chaque jeu. Cela simplifie le développement et la maintenance.

7 Analyse des résultats obtenus

Ici on va analyser les simulations qu'on a réalisée pour les jeux de Morpion et de Domino. Le but est de voir comment l'algorithme MinMax se debrouille face à un adversare simple et de juger les resultats obtenus.

Pour le domino on se concentre sur les scores et le nombre de parties gagnées, en tenant compte du fait que c'est un jeu à information incomplète, cela veut dire que MinMax ne connait pas toutes les pieces de l'adversaire, ce qui rend ses décisions plus difficiles.

Pour le Morpion, qui est un jeu à information complete on regarde si MinMax arrive à utiliser ses capacités pour bien jouer et battre un adversaire simple. On va essayer de comprendre pourquoi l'algorithme gagne ou perd dans certains cas, en fonction des regles et des stratégies.

7.1 Morpion

Présentation des simulations : Nous avons realisé 20 parties simulées entre l'algorithme MinMax et un adversaire simple dans le jeu de Morpion. Le tableau ci-dessous récapitule les résultats de ces simulations, avec des informations comme le gagnant, le score de MinMax, et le score de l'adversaire.

Table 1 – Résultats des simulations pour le Morpion

Simulation	Score MinMax	Score Adversaire	Gagnant
1	10	-10	MinMax
2	12	-5	MinMax
3	8	-10	MinMax
4	10	-5	MinMax
5	15	-10	MinMax
6	10	-5	MinMax
7	10	-10	MinMax
8	12	-8	MinMax
9	10	-15	MinMax
10	-5	12	Adversaire
11	10	-10	MinMax
12	10	-5	MinMax
13	10	-5	MinMax
14	11	-8	MinMax
15	10	-5	MinMax
16	10	-5	MinMax
17	10	-8	MinMax
18	10	-8	MinMax
19	10	-8	MinMax
20	10	-8	MinMax

Analyse des résultats : Dans la majorité des cas (19 simulations sur 20), l'algorithme MinMax a remporté la partie, démontrant son efficacité pour anticiper les coups et maximiser ses chances de victoire. Toutefois, dans une simulation, l'adversaire a gagné. Cela peut être attribué au fait que l'algorithme MinMax utilise un modèle d'adversaire pour prédire les prochains coups, et que dans ce cas précis, le modèle d'adversaire choisit les coups de manière aléatoire. Ce caractère aléatoire peut parfois surprendre l'algorithme, car il n'est pas toujours optimisé pour gérer des décisions imprévisibles. Néanmoins, le faible taux de défaite montre que l'algorithme MinMax est robuste face à des stratégies adverses même aléatoires, tout en exploitant efficacement les opportunités présentes sur le plateau.

Ces simulations confirment que l'algorithme MinMax est bien adapté au jeu de Morpion et qu'il pourrait être encore amélioré pour faire face à des adversaires plus complexes.

7.2 Dominos

Pareil que pour Morpion, nous avons réalisé 20 parties simulées dans le jeu de Domino et on a ces resultats :

Table 2 – Résultats des simulations pour le Domino

Simulation	Score MinMax	Score Adversaire	Gagnant
1	149	252	Adversaire
2	204	92	MinMax
3	131	201	Adversaire
4	131	201	Adversaire
5	201	83	MinMax
6	149	214	Adversaire
7	117	249	Adversaire
8	210	146	MinMax
9	210	146	MinMax
10	81	196	Adversaire
11	290	98	MinMax
12	381	275	MinMax
13	265	107	MinMax
14	292	367	Adversaire
15	297	174	MinMax
16	238	62	MinMax
17	142	249	Adversaire
18	207	111	MinMax
19	115	225	Adversaire
20	238	91	MinMax

En regardant les résultats des simulations on peut remarquer que MinMax a remporté 11 parties sur 20. Ce résultat soulève quelques questions sur les performances de l'algorithme dans un jeu à information incomplète, comme le Domino qui est un cas precis.

L'algorithme évalue les coups possibles en anticipant les actions futures mais il est limité par la profondeur de recherche et les informations cachées comme les pièces dans la main de l'adversaire ou dans la pioche. Ces informations manquantes rendent laprise de décision plus complexe pour MinMax car il ne peut pas predire avec certitude les actions de l'adversaire. Cela peut expliquer pourquoi il perdcertaines parties malgré son processus d'évaluation.

L'adversaire joue de manière aléatoire parmi les coups possibles. Bien que ca semble d'etre très basique, la chance joue un rôle important dans un jeu à information incomplète. Parfois, l'adversaire peut avoir des dominos avantageux, ou MinMax peut etre contraint à des décisions sous optimales en raison d'une information insuffisante.

MinMax analyse les coups possibles en évaluant chaque situation selon une fonction d'évaluation qu'on a défini et la qualité de cet évaluation joue un role clé dans ses performances. Si la fonction d'évaluation ne reflète pas parfaitement la réalité du jeu cela peut entrainer des décisions sous-optimales, en plus lorsque plusieurs coups obtiennent la meme évaluation, MinMax sélectionne toujours le premier coup rencontré dans la liste des options possibles. Ce comportement bien que deeterministe peut parfois manquer de flexibilité stratégique ce qui pourrait expliquer certains résultats où les décisions de MinMax ne semblent pas toujours optimales.

Dans un jeu à information incomplète comme le Domino, les informations cachés et la chance jouent un role important. Les 20 simulations montrent que MinMax peut bien performer, mais ces résultats ne sont pas suffisants pour tireer des conclusions définitifes. Si on augmentait le nombre de simulations, les performances pourraient varier, avec potentiellement une amélioration pour MinMax ou un équilibre avec l'adversaire simple. Une analyse plus approfondie nécessiterait de comparer MinMax à d'autres adversaires, comme un adversaire utilisant une stratégie plus réfléchie afin d'évaluer ses forces et faiblesses dans un contexte plus complet.

Conclusion

Après avoir implémenté l'algorithme MinMax pour ces deux jeux adversaires, on peut dire que dans cette première phase du projet, MinMax arrive à évaluer ses coups et à choisir celui qu'il estime etre le meilleur. Cependant avec un adversaire aléatoire comme celui que nous avons utilisé, la situation peut devenir compliquée. De plus, le comportement déterministe de MinMax, qui sélectionne le premier coup parmi des options ayant la même évaluation, peut limiter la diversité stratégique de l'algorithme. Une approche introduisant un choix aléatoire parmi ces coups pourrait offrir une meilleure adaptation dans certains scénarios Comme nous l'avons mentionné, la chance joue un roole important ici, car un adversaire aléatoire ne base pas ses décisions sur l'état du jeu ou une stratégie particulière. Cela peut parfois rendre ses actions imprévisibles et dans certains cas, difficiles à contrer pour MinMax.

Malgré cela, MinMax a montré qu'il était capable d'évaluer ses coups de manière cohérente et d'anticiper les résultats possibles. Mais cela reste insuffisant dans certaines situations, surtout dans un jeu à information incomplète comme le Domino. Pour améliorer les performances de MinMax, il faudra explorer des ajouts comme l'intégration de probabilités, ou encore l'adaptation à des modèles d'adversaires plus avancés. Ces améliorations seront au centre de la deuxième partie du projet, où nous chercherons à rendre MinMax plus robuste et performant.

En conclusion, cette première phase a montré que MinMax peut déjà bien jouer et prendre des décisions éclairées, mais qu'il y a encore des marges d'amélioration pour mieux gérer les situations complexes ou imprévisibles.