Université de Caen Normandie

Projet annuel - Semestre 2

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 différentes : le Morpion, où toutes les informations sont visibles pour les joueurs, et le Domino, où une partie des informations reste caché. 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 crée un système capable de réfléchir et de s'adapter à différents contexts de jeu. Pour cela, nous avons utilisé l'algorithme MinMax, connue pour sa capacité à simuler les coups futurs et à choisir les meilleures actions. Dans un premier temps, nous avons utilisé un modèle simple d'adversaire, permettant de valider l'approche dans un environnement contrôlé.

Ensuite, nous avons amélioré notre système pour qu'il prendre en compte plusieurs modèles d'adversaires crédibles, mis à jour dynamiquement au fil du jeu. Cette nouvelle version, plus réaliste, permet à l'algorithme MinMax d'éliminer progressivement les modèles incompatibles avec le comportement réel de l'adversaire. Nous avons également corriger les effets de bord qui existaient dans les versions initiales du jeu, notamment dans la gestion des scores.

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 adversairs, chaque joueur doit non seulement choisir les meilleures actions pour lui-même, mais aussi essayer de deviner ce que l'adversaire va faire. Cela devient encore plus difficile dans les jeux à information incomplète, comme le Domino, où certaines informations sont cachées, par exemple les dominos de l'adversaire ou ceux qui restent à piocher.

Le principaleéfi est donc de réussir à modéliser le comportement de l'adversaire, c'est-a-dire faire des hypothèses sur sa manière de jouer. En utilisant cette information un joueur peut adapter sa stratégie et prendre de meilleures décisions.

Dans la première partie du projet, nous avons utilisé une version simple de l'algorithme MinMax avec un seul modèle d'adversaire. Ensuite nous avons ameelioré notre méthode en ajoutant plusieurs modèles d'adversaires possibles, qui sont mis à jour pendant la partie. Si un modèle devient incohérent avec les actions observées, il est supprimé. Cette nouvelle approche permet de simuler plus finement l'incertitude et de mieux réagir pendant le jeu.

Grâce à cette évolution, notre projet passe d'un raisonnement basique à une prise de décision plus réaliste, où le joueur ne choisit plus seulement en fonction d'un scénario, mais en tenant compte de plusieurs possibilités crédibles.

Ce projet nous a donc permis d'explorer comment un joueur peut prendre de bonnes décisions, meme quand certaines informations sont manquantes, en utilisant l'algorithme MinMax et des modèles d'adversaires.

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 défies differentes selon qu'ils soient à information complète ou incomplète :

— **Jeux à information complète** : Dans ces jeux, comme le Morpion, tous les joueurs peuvent voir l'état complet du plateau à tout moment. Le principal défi est de choisir le meilleur coup en explorant les différents possibilités. Cela peut devenir compliqué si le jeu a beaucoup de coups disponibles. Pour reesoudre ce probleme, nous avons utilisé l'algorithme MinMax, qui permet de simuler plusieurs coups à l'avance et de choisir le meilleur. Cette partie est plus simple à gérer, car il n'y a pas d'incertitude.
— **Jeux à information incomplète** : Dans ces jeux, comme le Domino, certaines informations sont cachés, comme les dominos de l'adversaire ou ceux qui restent à piocher. Cela rend la prise de décision plus difficile, car on ne peut pas tout prévoir. Dans la première version du projet, nous avons utilisé un MinMax classique avec un seul modèle d'adversaire. Mais ce modèle ne suffisait pas, car il ne permettait pas de s'adapter aux coups réels de l'adversaire.

Pour améliorer ceci, nous avons ensuite développé une version plus avancée où plusieurs modèles d'adversaires sont considérés. À chaque tour, les modèles qui ne correspondent pas auxactions de l'adversaire sont retirés. Cela rend l'algorithme plus réaliste et plus intelligent. Mais cette approche a aussi posé de nouveaux défis, comme le fait de bien gérer les cas où aucun modèle ne reste crédible, ou encore d'éviter les effets de bord dans le code (par exemple quand une fonction modifie un score sans qu'on le veuille).

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 II) : Dans la phase finale du projet, notre objectif était de rendre notre système de décision plus robust, que ce soit pour des jeux à information complète comme le Morpion, ou incomplète comme le Domino. Nous avons atteint cet objectif en développant plusieurs améliorations importants, applicables à différents types de jeux :

— **Développement de nouveaux modèles d'adversaires :**
  — Nous avons implémenté plusieurs adversaires différets : un adversaire aléatoire, un adversaire jouant toujours le premier coup, un autre jouant le dernier coup possible et un adversaire qui choisit le domino avec la somme la plus élevée.
  — Ces modèles permettent de tester le comportement de l'algorithme face à différents styles de jeu, simples ou stratégiques.
— **Utilisation de plusieurs modèles crédibles avec mise à jour dynamique :**

  - L'algorithme MinMax a été enrichi pour prendre en compte une liste de modèles d'adversaires possibles.
  - Après chaque coup adverse, les modèles incohérents sont éliminés. Cela permet au joueur de s'adapter progressivement au style de l'adversaire réel, en réduisant l'incertitude au cours de la partie.
- **Deux versions de MinMax implémentées :**
  - La première version (`JoueurMinMax`) adopte une stratégie Max–Min–Expectimax : pour chaque coup, elle simule tous les modèles d'adversaire crédibles individuellement, et retient l'évaluation minimale obtenue. Cette approche approximative est adaptée aux jeux à information complète comme le Morpion, mais reste limitée dans les contextes incertains comme le Domino, car elle n'écarte pas les modèles incompatibles avec le comportement simulé.
  - La seconde version (`JoueurMinMax2`) est plus réaliste : pour chaque coup adverse simulé, elle ne conserve que les modèles d'adversaire qui attribuent une probabilité non nulle à ce coup. L'arbre de décision est ainsi adapté dynamiquement en fonction des modèles compatibles avec chaque situation, ce qui rend l'évaluation plus cohérente et précise dans un contexte à information incomplète.
- **Gestion de l'incertitude par l'espérance :**
  - Nous avons introduit un raisonnement par espérance dans notre algorithme. Lorsque plusieurs coups sont possibles selon un modeele d'adversaire, l'algorithme calcule une moyenne pondérée des évaluations, selon les probabilités associées à chaque coup.
  - Cette méthode est utilisée pour tous les modèles d'adversaires, qu'ils soient aléatoires ou déterministes, dès qu'une distribution de probabilité est fournie.
  - Cela permet de mieux gérer les cas où plusieurs réactions adverses sont possibles, et d'évaluer plus finement les décisions à prendre.

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.

- **Eldis Ymeraj :**
  - Développement du jeu de Domino, y compris les règles, la gestion des coups possibles et les calculs de score.
  - Participation à l'implémentation et à l'évolution de l'algorithme MinMax pour les jeux à information incomplète.
  - 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.
  - Participation à l'implémentation et à l'évolution de l'algorithme MinMax pour les jeux à information complète.
  - 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.
- **Tâches transversales réalisées à deux :**
  - Implémentation de l'algorithme MinMax prédictif (`JoueurMinMax`).
  - Implémentation de l'algorithme MinMax réaliste (`JoueurMinMax2`).
  - Ajout de plusieurs modèles d'adversaires : aléatoire, premier coup, dernier coup, domino fort.
  - Correction de certains éléments dans les classes de jeu et dans les algorithmes MinMax (effets de bord, gestion du score, cohérence des méthodes).
  - Réalisation d'expériences et de simulations dans les deux jeux, et comparaison des performances des différentes versions de l'algorithme.

Remarque : Bien que certaines tâches aient été attribuées à un membre en priorité, la majorité des éléments ont été développés de manière collaborative. Chacun a contribué à toutes les parties du projet en corrigeant, testant ou enrichissant le travail de l'autre. Il n'y a donc pas de séparation stricte : les deux membres ont une responsabilité partagée sur l'ensemble du projet.

4 Méthodes utilisées

Pour permettre a notre programme de jouer intelligemment contre un adversaire, nous avons utilisé deux versions avancées de l'algorithme MinMax, adaptées aux jeux à information complète et incomplète : une version nommé prédictive, et une version réaliste.

Ces deux approches exploitent une liste de modèles d'adversaires, chacun représentant un style de jeu possible. Chaque modèle fournit une distribution de probabilités sur les coups qu'un adversaire pourrait jouer à un moment donné. Le joueur MinMax utilise cette information pour évaluer les situations de jeu en fonction des réponses attendues.

- **MinMax prédictif (JoueurMinMax)** : cette version utilise un modèle d'adversaire choisi à l'avance. Elle simule le déroulement de la partie en supposant que l'adversaire joue toujours selon ce modèle. Lorsqu'un coup adverse est incertain, elle effectue un calcul d'espérance sur les coups possibles, pondérés par leurs probabilités. Cette approche est efficace mais sensible à l'exactitude du modèle choisi.
- **MinMax réaliste (JoueurMinMax2)** : cette version va plus loin en maintenant une liste de modèles d'adversaires crédibles, mise à jour dynamiquement. À chaque coup de l'adversaire, elle élimine les modèles incompatibles avec ce coup. Lors de la simulation, pour chaque coup possible, elle sélectionne uniquement les modèles qui auraient pu effectivement jouer ce coup, et fait une évaluation réaliste du jeu. Cette méthode est plus robuste car elle s'adapte au comportement observé.

Ces deux variantes ont été implémentées de façon générique, compatibles avec n'importe quel jeu respectant notre API commune (comme le Morpion ou le Domino). Les adversaires utilisés sont également modulaires : certains sont aléatoires, d'autres déterministes, et tous fournissent une distribution de coups compatible avec notre système de simulation.

4.1 Algorithme MinMax prédictif

Dans cette version, appelée `JoueurMinMax`, le joueur MAX simule la partie en supposant que l'adversaire suit un modèle précis (par exemple un adversaire aléatoire ou stratégique). À chaque tour de l'adversaire (MIN), l'algorithme utilise ce modèle pour prévoir un ou plusieurs coups possibles, puis les évalue à l'aide d'une moyenne pondérée, appelée espérance.

Nous avons appelé cette methode prédictive car elle fait une hypothèse fixe sur le comportement de l'adversaire. Le modèle ne change pas au cours de la partie.

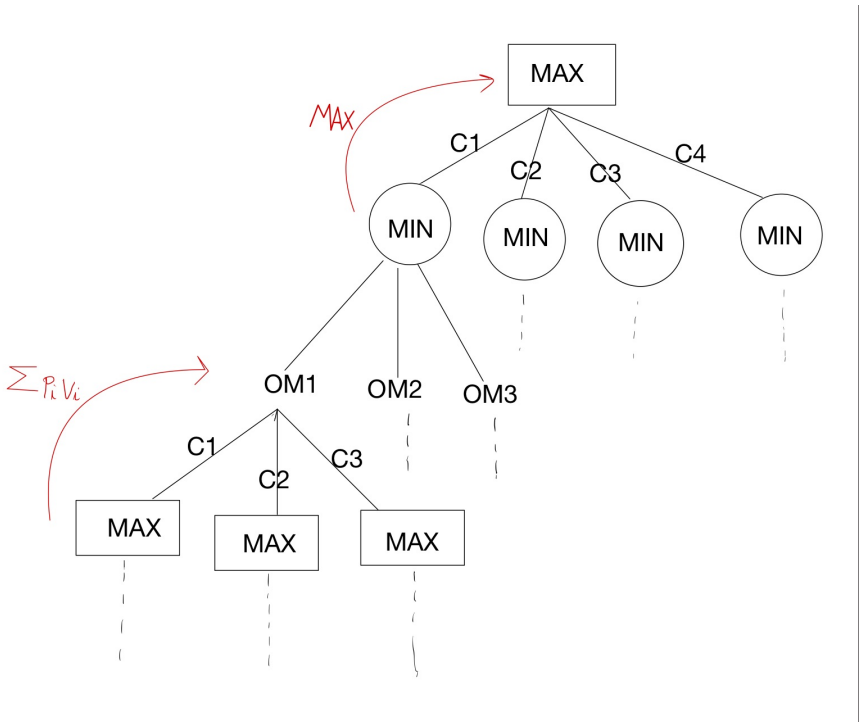

Figure 1 – Structure de l'algorithme MinMax prédictif avec modèle d'adversaire

Étapes de l'algorithme :

- Le joueur MAX explore tous les coups possibles (C1, C2, ...).
- Pour chaque coup, il simule les réponses de l'adversaire MIN selon le modèle choisi (OM1, OM2, OM3...).
- Si le modèle est aléatoire, les réponses sont évaluées en utilisant l'espérance mathématique : chaque coup est pondéré par sa probabilité (formule : $\sum p_i \cdot V_i$).
- Si le modèle est non-aléatoire, l'algorithme choisit directement le coup ayant la probabilité la plus élevée dans la distribution.
- L'évaluation est remontée dans l'arbre jusqu'à MAX, qui choisit le coup avec la meilleure espérance.

Avantages et limites :

- **Avantage :** L'algorithme est rapide et facile à mettre en place. Il donne de bons résultats si le modèle d'adversaire est proche de la réalité.
- **Limite :** Si l'adversaire joue différemment du modèle, les décisions peuvent être sous-optimales.

4.2 Algorithme MinMax réaliste

Cette version, appelée `JoueurMinMax2`, est plus avancée. Elle simule les coups de l'adversaire en s'appuyant sur plusieurs modèles crédibles à la fois. Contrairement à la version prédictive, ces modèles sont mis à jour à chaque étape de la simulation.

Lorsqu'un coup est joué par l'adversaire (MIN), l'algorithme vérifie quels modèles auraient pu proposer ce coup. Seuls ces modèles sont conservés pour continuer l'analyse. Ce filtrage dynamique permet d'éliminer les hypothèses incohérentes, et donc de prendre des décisions plus réalistes.

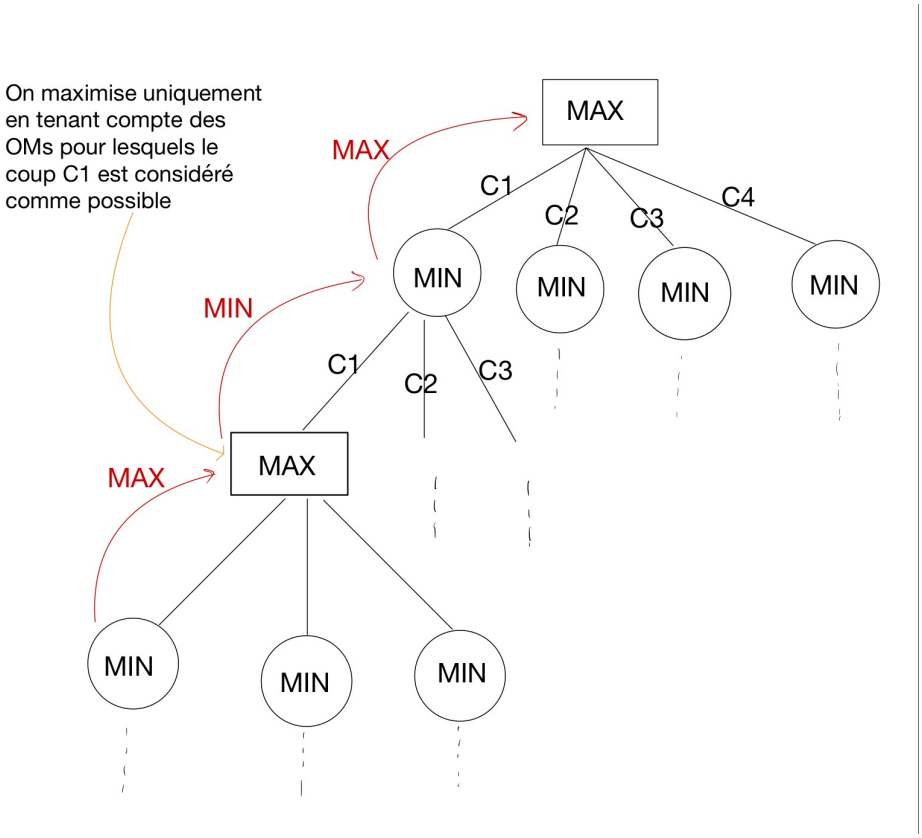

Figure 2 – Structure de l'algorithme MinMax réaliste avec filtrage dynamique des modèles crédibles

Étapes de l'algorithme :

- Le joueur MAX explore ses coups possibles (C1, C2, C3...).
- Pour chaque coup, le jeu passe au tour de l'adversaire (MIN).
- Pour simuler les réponses de l'adversaire, l'algorithme sélectionne uniquement les modèles crédibles capables d'avoir produit ce coup.
- À chaque niveau, les modèles incompatibles sont éliminés.
- L'évaluation finale est obtenue uniquement à partir des modèles encore crédibles, ce qui permet une meilleure anticipation.

Avantages et limites :

- **Avantage :** Le comportement du joueur s'adapte dynamiquement aux actions réelles de l'adversaire. Cela rend l'algorithme plus robuste, surtout dans les jeux à information incomplète.
- **Limite :** Cette méthode est plus coûteuse en calcul car elle doit filtrer les modèles et construire des arbres plus complexes.

4.3 Modèles d'adversaires

Pour évaluer le comportement de notre algorithme MinMax, nousavons défini plusieurs modèles d'adversaires. Chaque modèle représente une stratégie ou un style de jeu spécifique. Ces adversaires sont utilisés dans les deux versions de MinMax (prédictif et réaliste), et permettent d'étudier comment l'algorithme s'adapte à différents comportements.

Fonctionnement d'un adversaire : Un adversaire prend une décision en choisissant un coup parmi ceux qui sont possibles dans l'état actuel du jeu. Chaque classe d'adversaire implémente une méthode `choisir_coup()` avec une stratégie propre.

Liste des adversaires implémentés :

- **AdversaireRandom :** choisit un coup au hasard parmi les coups possibles. C'est un adversaire totalement imprévisible.
- **AdversairePremierCoup :** joue toujours le premier coup disponible dans la liste retournée par `jeu.coups_possibles()`. Il suit une stratégie fixe et déterministe.
- **AdversaireDernierCoup :** joue toujours le dernier coup de la liste des coups possibles. Comme le précédent, il est déterministe mais avec une logique différente.
- **AdversaireDominoFort (Domino uniquement) :** joue toujours le domino avec la plus grande somme de valeurs. Ce modèle simule un adversaire stratégique dans le jeu de Domino.

Utilisation dans notre projet :

- Les trois premiers modèles sont compatibles avec tous les jeux, comme le Morpion et le Domino.
- `AdversaireDominoFort` est spécifique au jeu de Domino.
- Ces adversaires sont utilisés à la fois pour simuler un joueur réel dans les parties, et comme base pour les modèles crédibles utilisés par MinMax.

Pourquoi plusieurs adversaires ? Utiliser plusieurs adversaires permet de tester les performances de MinMax dans des contextes variés. Chaque modèle aide à vérifier la capacité de l'algorithme à anticiper, s'adapter et prendre des décisions efficaces face à des comportements différents (aléatoires, rigides ou stratégiques).

5 Implémentation des l'algorithmes 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 : un jeu à information complète

Le Morpion est un jeu à information complète : l'ensemble de l'état du plateau est visible par tous les joueurs, et aucun élément n'est caché ni aléatoire. Cela permet une application directe de l'algorithme MinMax, sans incertitude liée au comportement adverse. Nous avons décliné deux variantes de MinMax dans notre projet : une version approximative (**MinMax1**) et une version réaliste (**MinMax2**).

Fonctionnement de l'algorithme : L'algorithme explore récursivement l'arbre des coups possibles jusqu'à une profondeur fixée. À chaque nœud :

- Le joueur MAX (l'agent IA) choisit le coup qui maximise son évaluation.
- Le joueur MIN (l'adversaire) choisit le coup qui minimise l'évaluation de MAX.

Les deux versions de l'algorithme diffèrent principalement dans leur manière de modéliser le comportement de l'adversaire :

- **MinMax1 (approximatif)** : suppose que tous les modèles d'adversaire crédibles sont également possibles à chaque coup. Le niveau MIN est donc évalué comme le pire cas sur tous les modèles (Max–Min–Expectimax).
- **MinMax2 (réaliste)** : sélectionne uniquement les modèles compatibles avec chaque coup adverse simulé. Cela permet une anticipation plus fine, en excluant les comportements adverses peu crédibles dans le contexte simulé.

Fonction d'évaluation utilisée : Une fonction d'évaluation commune est utilisée dans le Morpion pour quantifier la qualité des positions intermédiaires. Elle attribue les scores suivants :

- **+10** : Victoire de MAX (trois symboles alignés).
- **-10** : Victoire de MIN.
- **+3** : Deux symboles alignés de MAX sans blocage.
- **-3** : Deux symboles alignés de MIN sans blocage.
- **+1** : Un seul symbole de MAX aligné.
- **-1** : Un seul symbole de MIN aligné.
- **+2** : Contrôle de la case centrale.

Impact : Ces deux variantes donnent des comportements proches dans un jeu à information complète comme le Morpion. Cependant, MinMax2 présente une architecture plus robuste et mieux adaptée aux jeux complexes comme le Domino, où l'incertitude sur l'adversaire est significative.

5.2 Implémentation pour le Domino : un jeu à information incomplète

Le Domino est un jeu à information incomplète : chaque joueur ne connaît que sa propre main, mais ignore les pièces restantes chez l'adversaire ou dans la pioche. Cette incertitude rend l'utilisation de l'algorithme MinMax plus complexe, car il devient nécessaire de modéliser le comportement potentiel de l'adversaire à partir d'hypothèses crédibles.

Nous avons concu deux variantes adaptées à ce contexte : une version approximative (**MinMax1**) et une version plus réaliste (**MinMax2**) qui intègre dynamiquement la cohérence entre les coups simulés et les modèles crédibles.

Fonctionnement de l'algorithme : À chaque étape, l'état du jeu est composé des éléments suivants :

- Les dominos placés sur le plateau.
- Les mains respectives des joueurs.
- Les dominos restants dans la pioche.
- Le joueur courant.
- Les scores actuels.

Lorsqu'un joueur ne peut effectuer aucun coup valide , il peut appeler la méthode `piocher()` pour récupérer un domino si la pioche est encore disponible. Une seule pioche est autorisée par tour. Si aucun coup n'est possible même après la pioche, le tour est simplement passé, ce qui augmente un compteur de tours sans action (`tours_sans_coup`). Le jeu se termine si ce compteur atteint 4 ou si un joueur vide sa main.

Fonction d'évaluation des coups : Lorsqu'un domino est joué, une méthode dédiée `evaluer_coup(coup)` attribue un score en fonction de critères stratégiques :

- **+5 points** : Pour chaque coup valide joué.
- **+2 × (a + b)** : Bonus proportionnel à la valeur totale du domino $(a|b)$ joué.
- **+10 points** : Pour avoir joué un double (ex. : (3|3)).
- **+15 points** : Si le domino connecte les deux extrémités du plateau.
- **+20 points** : Si le coup bloque l'adversaire (aucune pièce dans sa main ne peut suivre).

Fonction d'évaluation globale : L'évaluation globale d'un état de jeu, utilisée par MinMax, est donnée par la méthode `evaluation()`. Elle combine plusieurs critères pour estimer la qualité d'une position en cours de partie :

- **Différence de score** entre les deux joueurs (score du joueur actuel moins score de l'adversaire).
- **Pénalité par domino restant** : plus un joueur a de dominos en main, plus son score est réduit.
- **Bonus si le joueur a 2 dominos ou moins**, pour l'encourager à se rapprocher d'une victoire par vidage de main.

— **Bonus si les dominos du joueur sont connectables** aux extrémités du plateau.
— **Bonus si l'adversaire est bloqué** et que la pioche est vide (il ne peut plus jouer).

De plus, lorsqu'un état de jeu est terminal, une méthode spécifique `jeu_termine_score()` est appelée pour retourner une valeur finale :
— **+700 points** si le joueur actuel a gagné en vidant sa main.
— **-700 points** s'il a perdu parce que l'adversaire a vidé sa main.

Ce score élevé joue un rôle essentiel dans le fonctionnement de l'algorithme MinMax. Lorsqu'un état de jeu est terminal (la partie est finie), MinMax ne peut plus explorer d'autres coups. Il doit donc attribuer une valeur claire à cette situation pour décider si elle est bonne ou mauvaise. En donnant un score très fort en cas de victoire (+700) ou de défaite (-700), on s'assure que l'algorithme privilégie toujours les coups qui mènent vers une victoire assurée, même si les gains immédiats sont faibles. Cela permet de guider efficacement la recherche vers les scénarios gagnants, ce qui est crucial dans les derniers niveaux de l'arbre de décision.

Cette fonction cherche à maximiser les chances de victoire tout en minimisant les possibilités pour l'adversaire de continuer efficacement.

Variante 1 : MinMax1 (approche approximative) La première version, MinMax1, adopte une approche Max–Min–Expectimax :
— Tous les modèles crédibles de l'adversaire sont considérés de manière égale.
— Pour chaque coup simulé, l'algorithme évalue la pire réponse possible (niveau MIN), en utilisant soit l'espérance pondérée (Expectimax), soit le coup le plus probable selon le modèle.
— Le coup sélectionné par MAX est celui qui maximise cette évaluation minimale.

Cette approche, bien que fonctionnelle, peut conduire à des sous-estimations du comportement réel de l'adversaire, en ne tenant pas compte de la cohérence entre les modèles et les coups joués.

Variante 2 : MinMax2 (approche réaliste avec filtrage) La seconde version, MinMax2, introduit un filtrage dynamique :
— Pour chaque coup adverse simulé, seuls les modèles qui attribuent une probabilité non nulle à ce coup sont conservés.
— Cela permet de ne pas biaiser les évaluations par des modèles incompatibles.
— L'évaluation du coup est ensuite effectuée avec ce sous-ensemble cohérent de modèles.

Cette approche plus fine améliore la précision stratégique, notamment dans des situations où certains coups ne peuvent être justifiés que par un sous-groupe restreint de modèles.

Impact : Dans des jeux à forte incertitude comme le Domino, MinMax2 offre de meilleures performances en anticipant plus finement le comportement de l'adversaire(logiquement mais on va voir avec les resultats). MinMax1 reste néanmoins une base fonctionnelle, plus simple mais moins rigoureuse. À noter qu'en cas d'égalité entre plusieurs coups ayant la même évaluation, les deux algorithmes sélectionnent par défaut le premier de la liste. Une amélioration potentielle serait d'introduire un choix aléatoire parmi ces coups équivalents, pour diversifier les décisions de l'agent.

5.3 Résumé des performances

L'étude comparative des deux variantes de l'algorithme MinMax dans des contextes de jeux différents a permis de dégager plusieurs constats.

Dans le Morpion (information complète) :
— Les deux versions de l'algorithme (MinMax1 et MinMax2) sont performantes et mènent à des stratégies optimales ou quasi-optimales.
— Le Morpion, en tant que jeu déterministe sans incertitude, ne met pas en évidence les différences entre les deux variantes.

Dans le Domino (information incomplète) :

- **MinMax1** a montré une bonne capacité à simuler des coups adverses en utilisant tous les modèles crédibles, mais au prix d'une approximation parfois trop large.
- **MinMax2**, en filtrant dynamiquement les modèles incompatibles avec chaque coup simulé, a permis une évaluation plus réaliste et précise des états de jeu.
- Les deux variantes ont su adapter leur stratégie pour maximiser le score, en exploitant des coups qui bloquent l'adversaire, connectent les deux extrémités ou permettent de vider rapidement la main.

6 Structure du projet

6.1 Arborescence des fichiers

Voici l'organisation actuelle des fichiers dans notre projet. Chaque dossier regroupe des composants liés à un role bien precis (adversaires, jeux, joueurs, etc.). Cette structure modulaire facilite la maintenance, la lecture du code et l'ajout de nouvelles fonctionnalités.

```
.
|-- Adversaire
|   |-- Adversaire.py
|   |-- AdversaireDernierCoup.py
|   |-- AdversaireDominoFort.py
|   |-- AdversairePremierCoup.py
|   |-- AdversaireRandom.py
|   |__ __init__.py
|-- Jeu
|   |-- Domino.py
|   |-- Jeu.py
|   |-- JeuInformationComplete.py
|   |-- JeuInformationIncomplete.py
|   |-- Morpion.py
|   |__ __init__.py
|-- Joueur
|   |-- Joueur.py
|   |-- JoueurAdversaire.py
|   |-- JoueurMinMax.py
|   |-- JoueurMinMax2.py
|   |__ __init__.py
|-- Main.py
|-- README.md
|-- Rapport.pdf
```

Listing 1 – Arborescence des fichiers

6.2 Description des fichiers et dossiers

Le projet est organisé de manière modulaire, avec une séparation claire entre les adversaires, les jeux, les joueurs et les fichiers principaux. Cela facilite la lecture, la modification et l'extension du code.

Dossier `Adversaire`

Ce dossier contient tous les modèles d'adversaires utilisés dans les jeux.

- **Adversaire.py** : Classe de base (abstraite) définissant l'interface commune pour tous les adversaires.
- **AdversaireRandom.py** : Adversaire jouant un coup au hasard parmi les coups possibles.
- **AdversairePremierCoup.py** : Adversaire qui joue toujours le premier coup disponible dans la liste.
- **AdversaireDernierCoup.py** : Adversaire qui joue toujours le dernier coup disponible dans la liste.
- **AdversaireDominoFort.py** : Adversaire spécifique au Domino, jouant toujours la pièce avec la plus grande valeur.

— **__init__.py** : Rend le dossier importable en tant que module Python.

Dossier `Jeu`

Ce dossier regroupe toutes les classes liées aux jeux (logique, règles, structure du plateau, etc.).
— **Jeu.py** : Classe abstraite définissant les fonctions communes à tous les jeux (ex : `jouer()`, `est_termine()`, `evaluation()`).
— **JeuInformationComplete.py** : Spécialise `Jeu.py` pour les jeux à information complète (ex : Morpion).
— **JeuInformationIncomplete.py** : Spécialise `Jeu.py` pour les jeux à information incomplète (ex : Domino).
— **Morpion.py** : Implémente les règles du Morpion (plateau, victoire, alternance des joueurs...).
— **Domino.py** : Implémente les règles du Domino, y compris le plateau, les mains, la pioche et le score.
— **__init__.py** : Rend le dossier importable comme module Python.

Dossier `Joueur`

Ce dossier contient les différentes classes de joueurs contrôlés par un algorithme.
— **Joueur.py** : Classe de base abstraite commune à tous les joueurs.
— **JoueurMinMax.py** : Contient l'implémentation du joueur utilisant l'algorithme MinMax prédictif.
— **JoueurMinMax2.py** : Variante plus réaliste de MinMax avec gestion dynamique des modèles d'adversaires.
— **JoueurAdversaire.py** : Adaptateur pour permettre à un adversaire d'être utilisé comme un joueur dans le système.
— **__init__.py** : Module Python d'initialisation.

Fichiers principaux

Fichiers à la racine du projet qui assurent le lancement et la documentation.
— **Main.py** : Point d'entrée du projet. Permet de lancer une partie ou d'exécuter les expériences.
— **README.md** : Présente brièvement le projet, comment l'exécuter et son organisation.
— **Rapport.pdf** : Contient le rapport complet du projet (celui-ci).

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

7.1.1 Résultats avec MinMax prédictif

Nous avons réalisé 10 000 parties entre l'algorithme MinMax prédictif (`JoueurMinMax`) et différents modèles d'adversaires dans le jeu du Morpion. Voici les résultats :

Table 1 – Résultats du MinMax prédictif contre différents adversaires (Morpion)

Adversaire	Victoire MinMax	Match nul	Victoire Adversaire
Aléatoire	9902 (99.0%)	98 (1.0%)	0 (0.0%)
Premier coup	10000 (100.0%)	0 (0.0%)	0 (0.0%)
Dernier coup	10000 (100.0%)	0 (0.0%)	0 (0.0%)

Analyse : L'algorithme MinMax prédictif montre de très bonnes performances dans le Morpion. Il gagne systématiquement contre les adversaires simples et déterministes comme « Premier coup » ou « Dernier coup ». Contre l'adversaire aléatoire, il remporte 99 % des parties, avec seulement 1 % de matchs nuls. Cela montre qu'il gère bien l'incertitude même s'il ne s'adapte pas au comportement réel.

7.1.2 Résultats avec MinMax réaliste

Nous avons répété les mêmes expériences avec la version MinMax réaliste (`JoueurMinMax2`), qui met à jour les modèles crédibles à chaque tour en fonction des coups réellement joués par l'adversaire.

Table 2 – Résultats du MinMax réaliste contre différents adversaires (Morpion)

Adversaire	Victoire MinMax	Match nul	Victoire Adversaire
Aléatoire	9909 (99.1%)	91 (0.9%)	0 (0.0%)
Premier coup	10000 (100.0%)	0 (0.0%)	0 (0.0%)
Dernier coup	10000 (100.0%)	0 (0.0%)	0 (0.0%)

Analyse : Le MinMax réaliste obtient des résultats similaires, voire légèrement meilleurs contre l'adversaire aléatoire (99.1 % de victoires, 0.9 % de nuls). Cela montre que la mise à jour dynamique des modèles permet à l'algorithme d'être un peu plus précis face à des comportements imprévisibles. Dans un jeu à information complète comme le Morpion, la différence entre les deux versions reste cependant très faible, car toutes les informations sont connues dès le départ.

7.2 Domino

7.2.1 Résultats avec MinMax prédictif

Nous avons effectué 10 000 parties de Domino entre MinMax prédictif (`JoueurMinMax`) et plusieurs adversaires. Les résultats sont les suivants :

Table 3 – Résultats du MinMax prédictif contre différents adversaires (Domino)

Adversaire	Victoire MinMax	Match nul	Victoire Adversaire
Aléatoire	6982 (69.8%)	9 (0.1%)	3009 (30.1%)
Premier coup	6941 (69.4%)	18 (0.2%)	3041 (30.4%)
Dernier coup	7092 (70.9%)	13 (0.1%)	2895 (28.9%)
Domino fort	7042 (70.4%)	6 (0.1%)	2952 (29.5%)

Analyse : Contrairement au Morpion, les performances de MinMax dans le Domino sont plus modestes. Même s'il gagne environ 70 % des parties, il perd encore près de 30 % contre tous les adversaires. Cela s'explique par le fait que le Domino est un jeu à information incomplète : MinMax ne connaît pas les dominos dans la main de l'adversaire ni ceux restants à piocher. Cette incertitude rend ses décisions plus difficiles.

De plus, la stratégie fixe de l'algorithme (comme choisir toujours le premier meilleur coup) peut limiter sa flexibilité. L'adversaire, meme s'il joue de manière simple, peut parfois profiter de situations favorables ou de la chance dans la pioche.

7.2.2 Résultats avec MinMax réaliste

Nous avons réalisé les mêmes tests avec la version réaliste de MinMax (`JoueurMinMax2`), qui met à jour les modèles crédibles à chaque tour.

Table 4 – Résultats du MinMax réaliste contre différents adversaires (Domino)

Adversaire	Victoire MinMax	Match nul	Victoire Adversaire
Aléatoire	6990 (69.9%)	16 (0.2%)	2994 (29.9%)
Premier coup	6872 (68.7%)	12 (0.1%)	3116 (31.2%)
Dernier coup	7076 (70.8%)	12 (0.1%)	2912 (29.1%)
Domino fort	6934 (69.3%)	14 (0.1%)	3052 (30.5%)

Analyse : Les résultats sont très proches de ceux obtenus avec MinMax prédictif. On observe une légère amélioration dans certains cas (contre aléatoire et dernier coup), mais aussi de légères baisses contre d'autres (comme contre Premier coup).

Cela montre que, même avec une gestion plus fine des modèles adverses, le Domino reste un jeu difficile pour MinMax. L'incertitude liée aux pièces cachées et la pioche limite l'efficacité de la stratégie, surtout si plusieurs coups sont évalués de manière équivalente.

Néanmoins, le MinMax réaliste a l'avantage de mieux s'adapter en cours de partie, ce qui pourrait le rendre plus robuste dans des contextes plus complexes ou contre des adversaires plus stratégiques.

Conclusion sur le Domino : Dans un jeu à information incomplète comme le Domino, l'algorithme MinMax montre une bonne capacité à gagner, mais reste limité par l'incertitude et la chance. La fonction d'évaluation, la profondeur de recherche et la gestion des modèles jouent un rôle essentiel dans les performances.

8 Comparaison entre MinMax prédictif et MinMax réaliste

Apres avoir développé et testé deux versions de l'algorithme MinMax, nous allons faire une comparaison globale basée sur leur fonctionnement, leur structure et leur coût de calcul.

1. Logique des deux approches

- **MinMax prédictif (`JoueurMinMax`)** : utilise un seul modèle d'adversaire fixé. Il prédit les coups adverses en fonction de ce modèle, et calcule une évaluation par espérance ou en choisissant le coup le plus probable.
- **MinMax réaliste (`JoueurMinMax2`)** : prend en compte plusieurs modèles d'adversaires et les met à jour dynamiquement. À chaque coup simulé, seuls les modèles compatibles avec ce coup sont conservés. Cela permet à l'algorithme de s'adapter progressivement au comportement réel de l'adversaire.

2. Structure et coût algorithmique

- Dans `JoueurMinMax`, le modèle d'adversaire reste constant pendant toute la simulation. Le traitement est simple, rapide et linéaire par rapport au nombre de coups.
- Dans `JoueurMinMax2`, chaque coup simulé entraîne un filtrage des modèles crédibles. Ce traitement est plus coûteux, car il faut vérifier la compatibilité de chaque modèle pour chaque coup possible.
- **Complexité** :
  - MinMax prédictif : environ $O(b^d)$, avec b le facteur de branchement et d la profondeur.
  - MinMax réaliste : environ $O(b^d \cdot m)$, où m est le nombre de modèles crédibles (4 maximum dans notre projet).

3. Points forts et limites

- Le MinMax prédictif est plus rapide et simple à implémenter. Il fonctionne très bien si le modèle d'adversaire est proche de la réalité.
- Le MinMax réaliste est plus adapté à l'incertitude. Il est capable d'éliminer les modèles qui ne correspondent pas aux coups observés. Cela le rend plus fiable dans des jeux où l'adversaire peut changer de comportement.
- Le choix entre les deux dépend du jeu : pour un jeu entièrement observable (comme le Morpion), MinMax prédictif suffit. Pour un jeu avec des informations cachées (comme le Domino), la version réaliste a un intérêt plus stratégique meme si avec nos résultats nous avons pas vu une grande différence entre les deux algorithmes.

Conclusion

Ce projet nous a permis de concevoir, implémenter et tester deux versions de l'algorithme MinMax dans des jeux à information complète et incomplète . Nous avons développé une architecture modulaire permettant de gérer différents jeux, adversaires et stratégies, tout en respectant une API commune.

Nous avons commencé par une version prédictive de MinMax, dans laquelle un seul modèle d'adversaire est utilisé pour simuler les réponses. Cette version s'est révélée très efficace dans le Morpion, avec des performances quasi parfaites face à tous les adversaires testés.

Ensuite, nous avons implémenté une version plus réaliste, capable de gérer plusieurs modèles d'adversaires en parallèle. Cette version met à jour dynamiquement les modèles crédibles à chaque coup jouéce qui permet de mieux s'adapter à des situations incertaines, notamment dans le Domino.

Les résultats obtenus montrent que dans un jeu à information complète, comme le Morpion les deux versions de MinMax donnent des résultats similaires. En revanche, dans le Domino, qui contient beaucoup plus d'incertitude, les limites du raisonnement prédictif apparaissent plus clairement meme si la version realiste reste également confrontée e des difficultés liees à l'information cachée.

Le projet nous a aussi permis de comprendre les limites du raisonnement déterministe dans les jeux, comme le fait de toujours choisir le premier coup évalué. Nous avons constaté que cela pouvait nuire à la variété stratégique de MinMax.

En résumé ce projet nous a permis d'explorer en profondeur les problématiques liées à la prise de décision dans les jeux adversariaux, en particulier dans des contextes d'incertitude. Les fondations sont solides et peuvent être enrichies par des stratégies plus avancées dans la suite du travail.