Master ANDROIDE

Développement d'une IA pour le jeu Schotten Totten

UE de projet M1

Lin Jie Wu & **Atena Amnache** & **Rim Hssain**

Encadrement : Olivier Spanjaard
Lien GitHub : `https://github.com/NeverDieTwyce/PROJET-ANDROIDE-SCHOTTEN-TOTTEN`

2024-2025

Table des matières

1 Introduction

Le jeu de stratégie Schotten Totten, créé par Reiner Knizia, se distingue par sa mécanique simple mais profonde, où deux joueurs s'affrontent pour prendre le contrôle de bornes en formant des combinaisons de cartes. Ce jeu, représente alors un terrain idéal pour implémenter une Intelligence Artificielle dans un contexte compétitif.

Le projet vise donc à implémenter une version numérique de Scotten Totten, intégrant une interface utilisateur et une IA capable de rivaliser efficacement contre un joueur humain, ou contre une autre IA tout en optimisant les performances pour des prises de décision rapides.

L'objectif principal est de créer un environnement de jeu en Python fidèle aux règles officielles avec une IA reposant sur des algorithmes de simulation Monte Carlo et des techniques d'optimisation via Numba. L'interface graphique quant à elle, sera développée avec PyGame.

Ce rapport détaillera l'architecture technique du projet, les choix algorithmiques, les optimisations de performance, et les résultats des tests de l'IA.

2 Présentation du jeu Schotten Totten

Schotten Totten est un jeu de cartes stratégique composé de 54 cartes. Chaque carte possède une valeur allant de 1 à 9, et chaque valeur est déclinée en 6 couleurs différentes. Le plateau de jeu est constitué de 9 bornes disposées en une seule ligne. L'objectif est de s'emparer d'une majorité de bornes : soit 5 bornes dispérsées, soit 3 bornes consécutives.

Pour revendiquer une borne, un joueur doit y poser une combinaison de 3 cartes. Ces combinaisons suivent un ordre de puissance décroissant :

— **Suite de même couleur** (ex. : 3-4-5 tous rouges)
— **Brelan** (trois cartes de même valeur, peu importe la couleur)
— **Couleur** (trois cartes de même couleur)
— **Suite** (trois cartes de valeurs consécutives, couleurs quelconques)
— **Hordes sauvages** (toute autre combinaison)

Une borne peut être revendiquée dans deux cas :

1. Les deux joueurs y ont posé 3 cartes.
2. Un des deux joueurs a posé 3 cartes et peut prouver, en analysant les cartes visibles sur le plateau seulement, qu'aucune combinaison possible pour l'adversaire ne peut surpasser la sienne.

Il existe également des cartes spéciales dans certaines variantes du jeu, introduisant des effets additionnels ou des règles spécifiques. Toutefois, dans le cadre de notre modélisation et du développement de l'IA, ces cartes spéciales ne seront pas prises en compte.

3 Stratégie de l'IA

Pour concevoir la stratégie de notre intelligence artificielle, une première approche a consisté à s'inspirer d'une IA existante pour le jeu Schotten Totten : *Deep Barca*, développée en 2017 par McCulloch, Bladow, Dobrow et Wright. L'idée principale de leur travail repose sur l'exploitation de la structure mathématique sous-jacente du jeu. Leur IA adopte une approche probabiliste, en choisissant l'action qui maximise la probabilité de victoire.

La stratégie s'articule en deux étapes principales : d'abord, le calcul de la probabilité de remporter chaque borne, puis l'évaluation de la probabilité de remporter la partie entière à partir de ces probabilités.

Évaluation des bornes

Pour estimer la probabilité de gagner une borne, Deep Barca distingue trois cas de figure :

1. **Aucun des deux joueurs n'a encore joué sur la borne** : la probabilité de remporter la borne est alors estimée à 50%, chaque joueur ayant les mêmes chances.
2. **Les deux joueurs ont posé au moins une carte sur la borne** : ils utilisent une méthode nommée **MFA** (Multiple Formation Approach), qui prend en compte plusieurs combinaisons possibles pour chaque joueur.
3. **Un seul joueur a posé au moins une carte sur la borne** : un mix des méthodes MFA et **BSFA** (Best Single Formation Approach) est alors utilisé pour affiner l'estimation.

Méthode MFA

La méthode MFA consiste à estimer les meilleures combinaisons qu'un joueur peut espérer former une fois les deux jeux partiellement posés sur une borne. Pour chaque joueur, les 4 meilleures combinaisons possibles sont générées, puis la probabilité de victoire pour chaque combinaison est estimée.

Cette probabilité est calculée comme le produit :

$$\text{Probabilité de victoire} = \mathbb{P}(\text{former la combinaison}) \times \mathbb{P}(\text{qu'elle ne soit pas battue})$$

La somme de ces quatre probabilités donne une estimation globale pour la borne.

Cependant, les auteurs ont observé que cette méthode devient moins efficace lorsque l'un des deux joueurs n'a pas encore posé de carte. En effet, le fait qu'une carte soit déjà posée restreint fortement les combinaisons réalisables, ce qui biaise l'évaluation. Pour pallier cela, la méthode BSFA a été introduite.

Méthode BSFA

La méthode BSFA (Best Single Formation Approach) reprend le principe de MFA, mais avec une approche plus ciblée. Lorsqu'un joueur n'a posé aucune carte sur une borne, BSFA génère la meilleure combinaison possible qu'il pourrait y jouer, et l'évalue face aux 4 meilleures combinaisons de l'adversaire ayant déjà posé une ou plusieurs cartes.

Probabilités de formation des combinaisons

Dans les méthodes MFA et BSFA, il est nécessaire de calculer la probabilité qu'un joueur puisse effectivement former une combinaison donnée. Cette probabilité est déterminée de manière systématique, selon la connaissance partielle du plateau :

— Si la carte est déjà posée sur la borne visée, sa probabilité d'y figurer dans la combinaison est de 100%.

— Si la carte est posée sur une autre borne, sa probabilité est de 0%.
— Si la carte est dans la main du joueur, sa probabilité d'être utilisée sur la borne est de 100%.
— Enfin, si la carte est encore non visible (ni sur le plateau ni dans la main), sa probabilité d'appartenir à la combinaison est estimée comme :

$$\frac{D}{2N}$$

où D est les nombre de cartes restantes dans la pioche et N est le nombre de cartes non vues (pioche + main de l'adversaire).

Ces estimations permettent d'évaluer toutes les possibilités sans avoir à simuler de manière exhaustive chaque distribution des cartes restantes.

Approche adoptée pour notre IA

Notre IA s'inspire largement des principes de *Deep Barca*, en particulier de son approche probabiliste fondée sur l'évaluation des combinaisons et des bornes. Toutefois, plusieurs adaptations ont été nécessaires pour répondre à des contraintes de simplicité de mise en œuvre, tout en assurant de bonnes performances en jeu.

1. Génération des coups

À chaque tour, l'IA commence par générer l'ensemble des coups possibles, c'est-à-dire toutes les actions légales consistant à poser l'une des cartes de sa main sur une borne encore accessible. Cela représente au maximum $6 \times 9 = 54$ actions possibles.

Chaque coup est ensuite évalué à l'aide d'une fonction de score, inspirée des probabilités de victoire estimées dans l'approche Deep Barca. Cette métrique prend en compte des facteurs comme la qualité potentielle de la combinaison formée, les cartes encore disponibles, et l'état actuel de la borne ciblée.

Afin de limiter le coût computationnel lié aux simulations, seules les k meilleures actions (selon ce score) sont retenues pour une analyse approfondie. L'hyperparamètre k, compris entre 1 et 54, permet ainsi de contrôler le compromis entre performance stratégique et temps de calcul.

2. Évaluation par simulation

Plutôt que de calculer les probabilités théoriques de victoire comme dans MFA/BSFA, notre IA utilise une approche par **simulation Monte Carlo** :

— Pour chaque coup considéré, l'IA simule un grand nombre de parties complètes à partir de la situation actuelle.
— Lors de chaque simulation, les cartes non connues (pioche et main adverse) sont tirées aléatoirement.
— L'IA joue ensuite le reste de la partie en appliquant une politique simple (comme jouer les cartes les plus fortes ou compléter les combinaisons les plus prometteuses).
— À la fin de chaque simulation, on détermine si le coup initial a conduit à une victoire.

Le score d'un coup est alors défini comme le pourcentage de victoires obtenues parmi les simulations.

3. Sélection du meilleur coup

L'IA compare les scores obtenus pour tous les coups simulés, et joue celui qui maximise son taux de victoire estimé. Cette stratégie permet de prendre en compte à la fois la structure du jeu et l'incertitude liée aux cartes cachées, tout en restant flexible face à différentes situations de plateau.

4. Gestion des cartes non vues

Comme Deep Barca, notre IA tient un registre des cartes non vues (cartes restantes dans la pioche et cartes supposées dans la main de l'adversaire), afin de tirer des distributions réalistes lors des simulations. Cela permet d'estimer plus fidèlement les chances de former certaines combinaisons.

En résumé, notre IA remplace les calculs analytiques de Deep Barca par une approche empirique basée sur des simulations massives. Cette méthode se révèle plus simple à implémenter et plus adaptée à une IA jouant en temps limité, tout en obtenant de bons résultats en pratique.

4 Implémentation

Pour l'implémentation de notre IA, nous avons choisi le langage **Python**, en raison de sa simplicité d'utilisation, de sa richesse en bibliothèques et de la facilité d'intégration d'interfaces graphiques. Le projet s'articule autour de plusieurs fichiers Python organisés de manière modulaire.

- `player.py` : contient la classe `Player` représentant un joueur. Cette classe gère les cartes en main, les coups joués et les interactions avec le plateau.
- `board.py` : contient la classe `Board`, qui modélise l'état du plateau de jeu. Les bornes sont représentées par des listes imbriquées, et les cartes sont codées sous forme de tuples `(valeur, couleur)` pour simplifier les manipulations.
- `interface.py` : permet l'affichage graphique du jeu. L'interface améliore l'expérience utilisateur et facilite le test de l'IA en conditions réelles. Elle est construite à l'aide de `pygame`.
- `game.py` : contient les fonctions principales liées à la logique du jeu et à l'implémentation de l'IA. C'est dans ce fichier que sont définies les stratégies de sélection des coups, les simulations, et l'évaluation des parties.
- `main.py` : sert de point d'entrée pour exécuter le jeu. Il initialise les différents composants et lance une partie entre l'IA et un joueur humain (ou une autre IA).

D'autres fichiers annexes ont également été développés dans le but de conduire des expériences sur l'IA. Ces scripts permettent notamment de fixer et d'ajuster les hyperparamètres, de mesurer les performances, et de comparer différentes variantes de stratégie.

Graphique du jeu

L'interface à été enrichie pour inclure :
- Un menu principal avec séléction de difficulté.

FIGURE 1 – Menu principal de l'interface.

- l'interface de jeu :

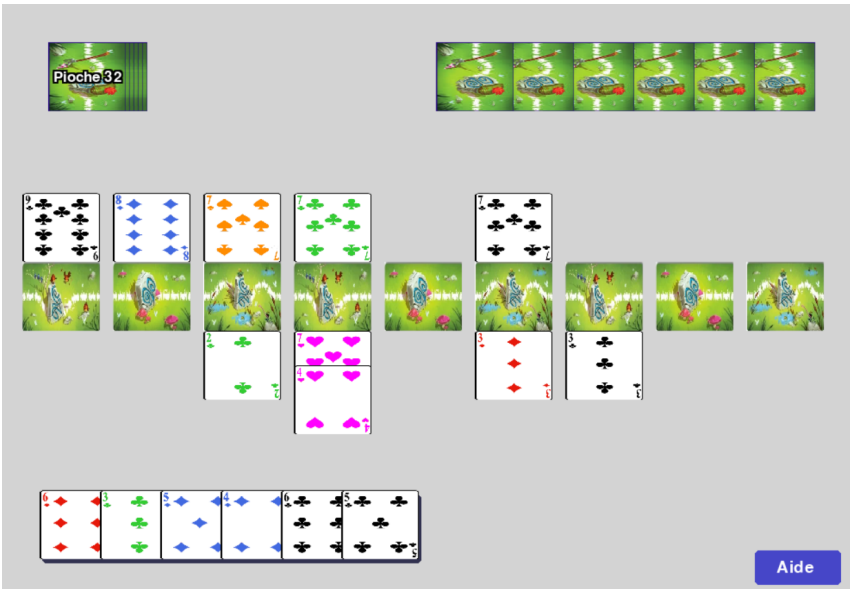

FIGURE 2 – Exemple d'interface du jeu.

- Un système de pop-ups en fin de partie proposant de : - Rejouer avec la même difficulté. - Retourner au menu principal.

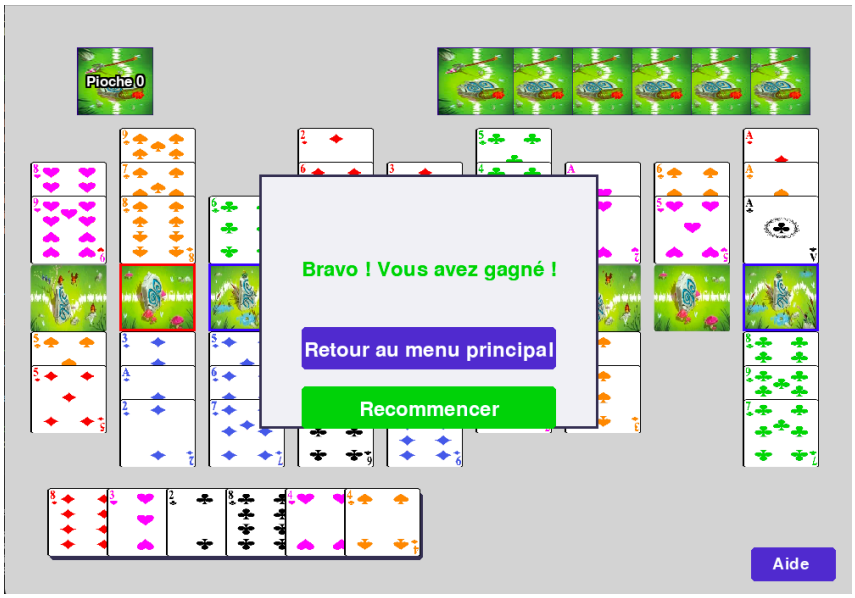

FIGURE 3 – Pop-up de fin de partie.

5 Optimisations et performances

Python étant un langage interprété, certaines opérations critiques peuvent s'avérer lentes, notamment dans le cadre d'un grand nombre de simulations ou de calculs combinatoires. Pour améliorer l'efficacité de notre IA, plusieurs optimisations ont été mises en place.

Compilation avec Numba. Certaines fonctions particulièrement coûteuses en temps de calcul, notamment celles impliquées dans l'évaluation des coups ou dans les simulations, ont été réécrites en utilisant la bibliothèque `Numba`. Cette dernière permet de « compiler » dynamiquement du code Python en code machine, ce qui offre des gains de performance significatifs pour les fonctions intensives en calcul.

Évaluation intelligente des meilleurs coups. L'évaluation des k meilleurs coups est l'une des étapes les plus coûteuses de l'algorithme. Pour limiter le nombre de calculs, les coups sont évalués dans un ordre heuristique décroissant : cela permet d'arrêter l'évaluation plus tôt lorsque les meilleurs coups ont déjà été identifiés, évitant ainsi d'analyser toutes les actions possibles.

Parallélisation des simulations Monte Carlo. Les simulations Monte Carlo étant indépendantes les unes des autres, elles se prêtent naturellement à une exécution parallèle. Nous avons donc parallélisé ces simulations à l'aide de la bibliothèque `multiprocessing` de Python, ce qui a permis de réduire considérablement le temps d'exécution total en exploitant les différents cœurs du processeur.

6 Évaluation de l'IA

Dans l'article présentant l'IA *Deep Barca*, l'évaluation a été réalisée en confrontant l'IA à l'un des meilleurs joueurs de *Schotten Totten* sur 10 parties. L'IA a remporté 3 parties sur 10, mais a également affronté d'autres joueurs avec un taux de victoires proche de 50%. Cependant, leur

méthode semble manquer de robustesse, car un échantillon de seulement 10 parties ne permet pas d'évaluer de manière fiable le niveau de l'IA.

N'ayant pas accès à l'un des meilleurs joueurs de *Schotten Totten*, nous avons opté pour une autre approche. Nous avons choisi de faire affronter notre IA contre elle-même en modifiant ses hyperparamètres pour déterminer les meilleurs réglages. Cela nous permet également de créer une IA de difficulté variable.

Dans un premier temps, nous avons simplement fait affronter l'IA sans utiliser de simulations Monte Carlo. Elle choisissait aléatoirement parmi les k meilleurs coups. Nous avons fait varier la valeur de k entre 1 et 5 pour cette étude.

IA 1 / IA 2	**1**	**2**	**3**	**4**	**5**
1	487	597	637	683	758
2	450	538	579	648	684
3	407	465	516	565	606
4	312	399	473	544	548
5	305	370	375	448	529

TABLE 1 – Résultats des affrontements entre différentes versions de l'IA. Chaque case représente le nombre de victoires sur 1000 parties de l'IA en ligne contre celle en colonne.

En observant le tableau 1, on remarque que plus le nombre k est faible, plus l'IA est performante. En effet, l'IA qui joue le meilleur coup (c'est-à-dire $k = 1$) remporte un nombre de victoires beaucoup plus élevé que celles qui choisissent aléatoirement parmi 2 à 5 coups. Cette tendance se confirme sur l'ensemble des confrontations entre les différentes versions de l'IA.

Cette différence de performance peut s'expliquer par le fait que, lorsque l'IA choisit ses coups de manière aléatoire parmi plusieurs options (pour $k \geq 2$), elle n'applique pas réellement une stratégie réfléchie. En choisissant le meilleur coup, l'IA maximise ses chances de faire un choix optimal à chaque tour, ce qui lui permet de prendre l'avantage sur ses adversaires. En revanche, lorsque l'IA choisit aléatoirement parmi plusieurs options, elle court le risque de faire un choix moins optimal, ce qui se traduit par un nombre de victoires plus faible.

Cette observation met en lumière l'importance de la stratégie de sélection des coups dans les performances de l'IA. Une approche qui maximise systématiquement la qualité des coups semble plus avantageuse qu'une stratégie qui repose sur un choix aléatoire parmi plusieurs possibilités.

Analyse des Résultats avec Simulations Monte Carlo

Par la suite, nous avons fait jouer l'IA qui choisit toujours le meilleur coup contre une IA qui choisit parmi les k meilleurs coups ($k = 3$), en utilisant des simulations Monte Carlo avec 100 simulations par coup. Le résultat obtenu montre un taux de victoire de 93% pour l'IA utilisant les simulations sur 100 parties. Ce résultat démontre clairement que l'intégration des simulations apporte une amélioration significative à la qualité des coups, permettant à l'IA de prendre des décisions plus éclairées et efficaces.

Nous nous sommes ensuite intéressés au nombre de simulations nécessaires pour obtenir une bonne estimation de la qualité des coups. Pour cela, nous avons fait jouer des IA utilisant 50 simulations contre 500 simulations par coup. L'IA utilisant 500 simulations a remporté 64% des parties sur 100 confrontations, ce qui indique qu'un nombre de simulations plus élevé améliore les performances. Toutefois, cette augmentation du nombre de simulations engendre également un temps de calcul plus long. Il est donc essentiel de trouver un équilibre entre performance et temps de calcul.

Afin de mieux comprendre l'impact du nombre de simulations, nous avons également fait

jouer une IA effectuant 500 simulations contre une autre effectuant 1000 simulations. Cette fois, le taux de victoire était proche de 50% pour les deux IA. Cela suggère que dépasser 500 simulations n'apporte pas d'amélioration significative à la qualité des coups. En conséquence, nous avons déterminé qu'un nombre de simulations autour de 500 est suffisant pour obtenir une estimation précise de la qualité des coups, tout en limitant le temps de calcul nécessaire.

En conclusion, bien que l'augmentation du nombre de simulations améliore la performance de l'IA, il est crucial de choisir un nombre optimal de simulations qui équilibre bien la qualité des décisions et les contraintes de temps de calcul.

Temps de calcul

Actuellement, l'IA met moins de 15 secondes pour lancer 1500 simulations, soit 500 simulations pour chacun des trois meilleurs coups. Ce temps de calcul est tout à fait raisonnable, car il reste comparable au temps de réflexion qu'un joueur humain prendrait avant de jouer son coup dans une partie réelle.

7 Niveaux de difficulté

Pour adapter l'IA à différents profils de joueurs, trois niveaux de difficulté ont été implémentés :

1. Facile : L'IA sélectionne aléatoirement un coup parmi les k meilleurs avec $k > 1$ évalués par la fonction heuristique (ici $k = 10$, mais on peut augmenter k pour faciliter encore plus). Cette approche permet des décisions rapides mais moins optimales, simulant un joueur débutant.

2. Moyen : L'IA choisit systématiquement le meilleur coup à chauque tour selon l'évaluation heuristique (sans simulations Monte Carlo). Cette stratégie équivaut à une IA "parfaite" dans un contexte à information complète.

3. Difficile : L'IA utilise des simulations de Monte Carlo (500-1000 simulations/coup) pour affiner ses décisions, comme décrit en section 3.3. Ce niveau reproduit le comportement d'un joueur expert tenant compte des probabilités.

Ces niveaux s'appuient sur les résultats de la Table 1, montrant que le choix déterministre ($k=1$) surpasse les stratégies aléatoires ($k>1$). La version "Difficile" combine cette riguer avec une estimation probabiliste via Monte Carlo.

8 Discussion

Le développement de cette IA a permis de relever plusieurs défis techniques et conceptuels, en mettant en lumière les spécificités d'un jeu combinant stratégie et hasard.

En effet, le caractère aléatoire de la distribution des cartes et l'information partielle disponible compliquent considérablement la conception d'une IA optimale. Bien que les simulations Monte Carlo permettent de mieux appréhender les probabilités de succès, elles ne peuvent totalement éliminer l'incertitude liée aux cartes cachées. Cette limitation souligne la difficulté de modéliser des jeux où les résultats dépendent fortement de facteurs incontrôlables. Donc malgré ses points forts, l'approche adoptée reste perfectible. Par exemple, les simulations Monte Carlo reposent sur des hypothèses probabilistes qui ne prennent pas toujours en compte les subtilités des décisions humaines, telles que le bluff ou les choix imprévisibles. Cela limite la capacité de l'IA à simuler fidèlement le comportement d'un joueur humain.

9 Conclusion

Ce projet a permis de mettre en œuvre une intelligence artificielle compétitive pour le jeu stratégique Schotten Totten. Grâce à l'intégration d'algorithmes de simulation Monte Carlo et de techniques d'optimisation comme Numba, l'IA parvient à rivaliser efficacement avec des joueurs humains, tout en offrant une interface utilisateur accessible et fluide. Cependant, le caractère intrinsèquement aléatoire du jeu représente un défi majeur dans le développement de l'IA. L'information incomplète et la distribution aléatoire des cartes introduisent une part importante d'incertitude, limitant la capacité de l'IA à prendre des décisions parfaitement optimales. Ces contraintes ont nécessité une approche hybride, combinant heuristiques et simulations, pour équilibrer la gestion du hasard et la performance stratégique.

Les résultats démontrent non seulement la faisabilité d'une IA compétitive, mais aussi l'importance des compromis entre performance algorithmique et expérience utilisateur. Bien que l'IA ait démontré de solides performances, l'introduction de techniques adaptatives ou d'apprentissage pourrait permettre de mieux gérer les incertitudes liées au hasard et de s'ajuster dynamiquement aux stratégies adverses.

En conclusion, ce projet a constitué une exploration enrichissante des défis liés à l'implémentation d'une IA dans un environnement de jeu aléatoire, tout en posant des bases solides pour des développements futurs.

A Annexes

A.1 Captures d'écran

Fig.1 : Menu principal avec sélection de difficulté.

Fig.2 : Etat d'une partie en cours.

Fig.3 : Pop-up de fin de partie avec option de rejeu.

A.2 Code source

Dossier Projet : Contient tout les fichiers .py permettant de coder le jeu et sa structure ainsi que les fichiers de test.

B Références

Deep Barca : A probabilistic Agent to Play the Game Battle Line :
`https://ceur-ws.org/Vol-1964/PU1.pdf`

Algorithme Monte Carlo pour faire un moteur IA jeu :
`https://www.youtube.com/watch?v=UFqaCkTZ65w&authuser=0`