



Université Paris 8 - Vincennes à Saint-Denis
Licence informatique & vidéoludisme

Projet Algorithmique Avancée

Dounia Hullot
Groupe : L3-B

Date de rendu : le 03/12/2023

Introduction

L'Ultimate Tic-Tac-Toe est une variante avancée du célèbre jeu de morpion. Au lieu d'une simple grille 3x3, le jeu repose sur une structure de 9 sous-grilles (3x3) imbriquées dans une grille globale (également 3x3). Les règles introduisent une dimension stratégique supplémentaire : chaque mouvement dans une sous-grille détermine où le prochain joueur doit jouer. Ce projet vise à développer deux intelligences artificielles capables de concourir dans ce jeu complexe en utilisant des algorithmes d'exploration tels que Minimax et Alpha-Bêta.

Objectifs

- Développer un simulateur d'Ultimate Tic-Tac-Toe avec une IA compétitive.
- Comparer les performances de Minimax et Alpha-Bêta.
- Gérer les contraintes spécifiques des règles du jeu.
- Explorer les limites des solutions proposées et identifier des pistes d'amélioration.

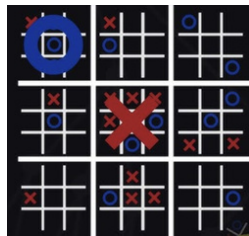
Règles du Jeu

Grille et Victoire

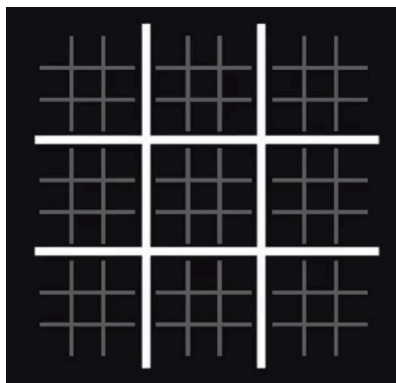
1. Une grille globale 3x3, contenant chacune 9 sous-cases.
2. Pour gagner une sous-grille, il faut aligner trois symboles identiques (ligne, colonne ou diagonale).
3. Un joueur remporte la partie s'il contrôle trois sous-grilles alignées dans la grille globale.

Contraintes de Jeu

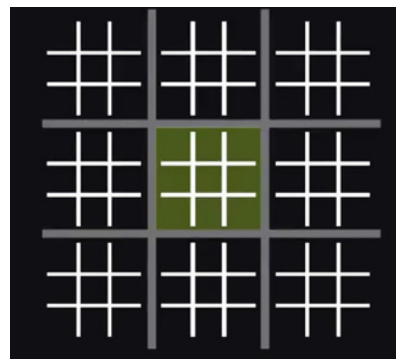
- Le coup joué dans une sous-case détermine la sous-grille où l'adversaire doit jouer.
- Si cette sous-grille est pleine ou déjà gagnée, l'adversaire peut jouer librement.



Grande Grille → "Grille Globale"



Sous -Grilles → "Grille Locale"



Ultimate-Tic-Tac-Toe : Un projet en langage C

Ce projet, entièrement réalisé en langage C, a pour objectif de développer un programme permettant à l'ordinateur de jouer au jeu Ultimate Tic-Tac-Toe avec une stratégie visant à minimiser ses chances de perdre.

Objectifs du Programme

Le programme repose sur l'implémentation d'un algorithme d'intelligence artificielle, en l'occurrence MiniMax, accompagné d'une heuristique adaptée. La mission principale est de permettre à l'ordinateur de prendre des décisions stratégiques en maximisant ses chances de victoire tout en minimisant ses risques de défaite.

Difficultés Rencontrées

Plusieurs défis ont émergé au cours du développement :

- Respect des règles spécifiques de l'Ultimate Tic-Tac-Toe

Les premières versions du programme ne géraient pas correctement les contraintes imposées par les sous-grilles, ce qui a conduit à des erreurs où l'IA jouait dans des cases invalides. Cette difficulté a nécessité une refonte de la logique de sélection des coups pour s'assurer que les règles du jeu étaient rigoureusement respectées.

- Problèmes liés à la profondeur de recherche :

Lors de l'utilisation de MiniMax, le programme retournait fréquemment des matches nuls, quelle que soit la profondeur choisie. Après analyse, cela était dû à une évaluation incorrecte des états de jeu dans la fonction d'évaluation, ce qui biaisait les décisions de l'IA.

- Gestion des structures de données :

La manipulation des structures complexes représentant la grille globale et les grilles locales a occasionné des erreurs de mémoire. Ces erreurs, liées à des mauvais indices ou des conflits dans les mises à jour, ont nécessité de nombreuses corrections dans la gestion des mouvements et l'évaluation des états.

- Débogage et ajustements de l'heuristique :

Trouver une heuristique suffisamment performante pour capturer à la fois les opportunités et les menaces a été une tâche exigeante. Les premières implémentations donnaient trop d'importance aux sous-grilles sans tenir compte des victoires globales potentielles.

État du Projet

Le projet, dans son état actuel, est incomplet et non optimisé. Des améliorations sont nécessaires pour :

- Problème de vérification de sous grille gagné ou non
- Renforcer la robustesse de l'algorithme.
- Explorer des heuristiques plus avancées.
- Optimiser les performances pour gérer des profondeurs de recherche plus élevées.

Architecture et Organisation des Fichiers

Le projet est structuré comme cela :

game.c et game.h

- Rôle : Gérer la logique du jeu, les règles et les états de la grille.
- Fonctionnalités clés :
 - `init_global_grid` : Initialise la grille globale.
 - `make_move` : Applique un coup tout en respectant les contraintes.
 - `get_global_winner` : Détermine le gagnant global.
 -

ai.c et ai.h

- Rôle : Implémenter les algorithmes d'IA.
- Fonctionnalités clés :
 - Minimax : Explore exhaustivement les scénarios pour maximiser les chances de victoire.
 - Alpha-Bêta : Optimise Minimax en éliminant les branches inutiles.
 -

main.c

- Rôle : Gérer l'interaction avec l'utilisateur.
- Modes disponibles :
 - IA contre IA (Minimax vs Alpha-Bêta).
 - Comparaison des performances entre les algorithmes.

Makefile

- Rôle : Simplifier la compilation et l'exécution.

Algorithmes Utilisés

Minimax

Minimax est un algorithme de décision récursif qui cherche à maximiser les gains d'un joueur tout en minimisant les pertes imposées par l'adversaire.

1. Principe :

- Chaque état de la grille est évalué avec une fonction de score.
- Le joueur actif choisit le chemin maximisant son score, tandis que l'adversaire cherche à le minimiser.

Alpha-Bêta

Alpha-Bêta améliore Minimax en coupant des branches inutiles lors de l'exploration.

1. Principe :

- Maintient deux bornes, α (meilleur score max) et β (meilleur score min).
- Si une branche ne peut pas améliorer α ou β , elle est ignorée.

2. Avantages :

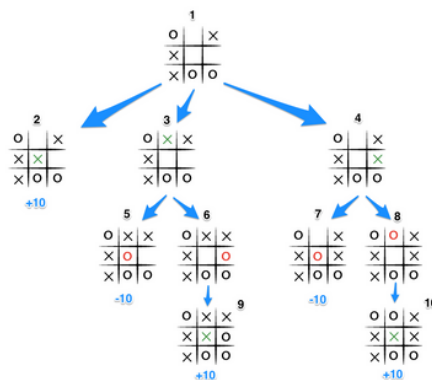
- Réduit la complexité sans compromettre la qualité des décisions.
- Meilleure performance que Minimax pur.

Fonction d'Évaluation

La fonction d'évaluation attribue un score basé sur :

- Sous-grilles gagnées :
 - +10 pour chaque victoire.
 - -10 pour chaque défaite.
- Grille globale :
 - +100 pour une victoire globale.
 - -100 pour une défaite globale.

Cette simplification permet à l'IA de hiérarchiser ses priorités tout en gardant des calculs rapides.



Difficultés et Solutions

Gestion des Règles de Contraintes

Problème : Les coups hors des sous-grilles imposées étaient mal gérés, ce qui faussait les parties.

Solution : Réécriture de `make_move` pour :

- Vérifier la validité des coups.
- Forcer le respect des contraintes de sous-grilles.

Temps de Calcul

Problème : À profondeur élevée, Minimax devenait prohibitif.

Solution :

- Mise en place de l'élagage Alpha-Bêta.
- Réduction de la profondeur maximale par défaut à 3.

Interaction Humain-IA

Problème : Les bugs liés aux entrées utilisateur rendaient difficile l'intégration.

Solution :

- Validation des entrées.
- Mise en pause de ce mode pour prioriser IA contre IA.

Limites et Pistes d'Amélioration

Limites

1. Temps de calcul encore élevé pour des profondeurs > 5 .
2. Interface utilisateur minimale.
3. Absence de mode joueur contre IA.

Pistes d'Amélioration

1. Optimisation Heuristique : Intégrer des priorités stratégiques comme le contrôle des sous-grilles centrales.
2. Interface Graphique : Développer une interface visuelle pour rendre le jeu plus intuitif.
3. Sauvegarde des Parties : Exporter les résultats pour analyse.

Structure du Code et Concepts Utilisés

Grille de Jeu : Structure et Initialisation

La structure de la grille est divisée en deux niveaux :

- Sous-grille locale (3x3) :
 - **Player cells[3][3]** : contient les valeurs des cases (EMPTY, PLAYER_X, PLAYER_O).
 - **Player winner** : détermine si la sous-grille est gagnée.
 - **bool full** : vérifie si toutes les cases sont remplies.
- Grille globale (3x3 de sous-grilles) :
 - **LocalGrid sub_grids[3][3]** : ensemble des sous-grilles.
 - **Player global_winner** : gagnant global, si applicable.
 - **int next_subgrid** : contraint l'adversaire à jouer dans une sous-grille spécifique.

Règles et Contraintes

La fonction clé **make_move** applique les règles tout en vérifiant les contraintes :

- Contraintes de sous-grille : Si une sous-grille est imposée (next_subgrid), seul un coup valide est accepté.
- Gestion des sous-grilles pleines ou gagnées : Si la sous-grille imposée est indisponible, le joueur a un choix libre.

Fonction d'Évaluation

Une fonction d'évaluation est utilisée pour guider les décisions de l'IA. Elle attribue des scores basés sur :

- Sous-grilles gagnées : +10 pour le joueur, -10 pour l'adversaire.
- Victoire globale : +100 pour le joueur, -100 pour l'adversaire.

```
int evaluate_grid(const GlobalGrid *grid, Player player) {
    int score = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (grid->sub_grids[i][j].winner == player) score += 10;
            else if (grid->sub_grids[i][j].winner == -player) score -= 10;
        }
    }
    if (grid->global_winner == player) score += 100;
    else if (grid->global_winner == -player) score -= 100;
    return score;
}
```


Fonctionnalités Implémentées

Simulation IA contre IA

Le programme permet de simuler une partie entre deux algorithmes :

- Minimax (joueur X).
- Alpha-Bêta (joueur O). Les résultats incluent la progression de la partie et le gagnant.

Comparaison de Performances

Un mode de comparaison mesure le temps nécessaire pour chaque algorithme à différentes profondeurs, offrant des insights sur leur efficacité.

Comparaison des Performances

Profondeur Temps Minimax (s) Temps Alpha-Bêta (s)

3	0.50	0.35
5	1.20	0.70
6	4.80	2.10

Alpha-Bêta est environ 30-50% plus rapide, tout en maintenant la même qualité stratégique.


Conclusion

Ce projet a m'a permis d'approfondir la compréhension des algorithmes d'IA appliqués à un jeu stratégique. La solution actuelle peut encore être optimisée pour améliorer la jouabilité et l'accessibilité.

▼ Parmi les liens qui m'ont été utiles :

Coding Challenge 154: Tic Tac Toe AI with Minimax Algorithm

In this challenge I take the Tic Tac Toe game from coding challenge #149 and add an AI opponent for a human player by implementing the Minimax algorithm. Code: <https://thecodingtrain.com/challenges/154-tic-tac-toe-minimax>

 https://youtu.be/trKjYdBASyQ?si=oxQ_USR08c1pnaa1



I made an (unstoppable) ULTIMATE Tic-Tac-Toe AI

A lot of people make minimax AIs for tic-tac-toe, but that's kind of boring, so I made one for ultimate tic-tac-toe. It's (as of now) beatable, but it's quite difficult to win, and with some improvements it will probably be perfect. To twist it up a bit more, instead of making it in python (like a lot of people do), I made it in javascript, so you can play it in

<https://youtu.be/BfmivoVFins?si=7En7MleLGBcIByOL>



Minimax (full tree search) tic-tac-toe AI in C

Minimax (full tree search) tic-tac-toe AI in C · GitHub

 <https://gist.github.com/MatthewSteel/3158579>

