**Universitatea Tehnică ”Gheorghe Asachi” din Iași**

**Facultatea de Automatică și Calculatoare**

**Calculatoare și Tehnologia Informației Specializarea Tehnologia Informației**

**Proiect Inteligență Artificială**

Studenți:

Bejinaru Mihnea-George, Grupa 1408B

Chircă Radu-Iulian, Grupa 1408B

Ciocan Ștefan-Robert, Grupa 1408B

Ianuarie 2024

# Cuprins

**Capitolul 1. Descrierea problemei considerate 3**

**1.1 Enunțarea temei proiectului 3**

**1.2 Relevanța algoritmului 3**

**Capitolul 2. Aspecte teoretice privind algoritmul 3**

**Capitolul 3. Modalitatea de rezolvare 5**

**3.1 Implementare 5**

**3.2 Aspecte generale despre proiect 5**

**Capitolul 4. Părți semnificative ale codului sursă 5**

**4.1 Agentul MCTS 5**

**4.2 Logica de joc 8**

**Capitolul 5. Rezultate obținute prin rularea programului 10**

**Capitolul 6. Concluzii 11**

**Capitolul 7. Bibliografie 12**

**Capitolul 8. Lista contribuțiilor membrilor echipei 12**

# Capitolul 1. Descrierea problemei considerate

## 1.1 Enunțarea temei proiectului

Tema proiectului constă în implementarea algoritmului de căutare pe arbori Monte Carlo Tree Search (MCTS) pentru jocul Hex între un jucător si un agent AI. Hex este un joc de strategie abstract în care doi jucători încearcă să formeze o cale continuă de la una dintre marginile tablei la cealaltă, folosind hexagoanele dintr-un grid prestabilit.

## 1.2 Relevanța algoritmului

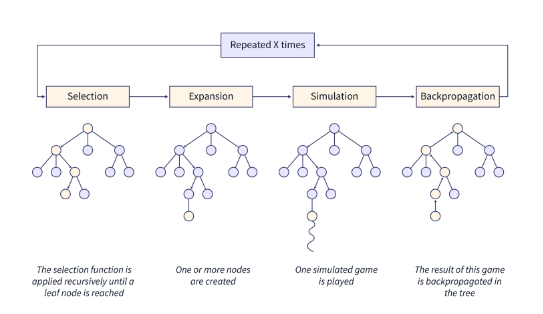
Relevanța MCTS în acest context constă în capacitatea sa de a explora eficient și de a lua decizii informate în medii de joc cu spații de stări mari și complexe. Algoritmul oferă avantaje semnificative peste alte metode precum alpha-beta pruning sau alți algoritmi similari care minimizează spațiul de cautare.

În mod particular, MCTS pur nu necesită o funcție explicită de evaluare. Simpla implementare a mecanicilor jocului este suficientă pentru a explora spațiul de căutare (de ex. generarea mișcărilor permise într-o anumită poziție și cunoasterea condițiilor de sfârșit de joc). Astfel, algoritmul poate fi utilizat în jocuri fără o teorie dezvoltată sau în cadrul general-game-playing.

Arborele de joc în MCTS crește asimetric, deoarece metoda se concentrează asupra subarborilor mai promițători. Astfel, se obțin rezultate mai bune decât cele ale algoritmilor clasici în jocurile cu un factor de ramificare mare.

# Capitolul 2. Aspecte teoretice privind algoritmul

Algoritmul Monte Carlo Tree Search este o metodă de căutare și luare a deciziilor bazată pe eșantionare stocastică. A fost inițial dezvoltat pentru jocuri de tip board, cum ar fi Go, dar a fost extins și aplicat cu succes într-o varietate de probleme de luare a deciziilor.

****

## Se începe cu un arbore de căutare format din noduri care reprezintă stările posibile ale jocului sau problemei. Algoritmul începe cu rădăcina arborelui și folosește un criteriu de selecție pentru a naviga în josul arborelui până la o frunză. Un criteriu comun este UCB (Upper Confidence Bound), care îmbină informațiile despre cât de bine a mers un nod până acum cu cât de puțin a fost explorat.

## Odată ajuns la o frunză, se decide dacă aceasta reprezintă o stare finală a jocului sau dacă poate fi extinsă. În cazul extinderii, se adaugă copii noi pentru toate acțiunile posibile din starea curentă.

## Se efectuează o simulare sau ‘playout’ a jocului începând de la starea extinsă pana la cea finală. Aceasta implică de obicei desfășurarea acțiunilor în mod aleatoriu sau prin alte metode simplificate.

## Se actualizează informațiile pentru toate nodurile de la nodul extins înapoi la rădăcină. Se actualizează numărul de vizite și funcția de evaluare a câștigului pentru fiecare nod.

## Acești pași (selecție, expandare, simulare, backpropagation) sunt repetati de un număr de ori sau până când un anumit criteriu de oprire este îndeplinit, cum ar fi expirarea unui anumit buget de timp.

## După terminarea iterărilor, se alege acțiunea care a avut cel mai mare câștig mediu în timpul simulărilor.

## Algoritmul MCTS este foarte eficient și se bazează pe ideea de a distribui resursele de eșantionare în mod adaptiv, concentrându-se pe părțile promițătoare ale spațiului de stări. A fost aplicat cu succes în jocuri complexe și în alte domenii care implică luarea deciziilor în medii nesigure.

# Capitolul 3. Modalitatea de rezolvare

**3.1. Implementare**

Funcționalitatea jocului este orientată în principal pe interacțiunea dintre utilizator și agentul AI prin intermediul protocolului Go Text Protocol (GTP). Acesta poate realiza acțiuni precum setarea dimensiunii tablei, plasarea unei piese pe tablă, interogarea stării jocului și altele.

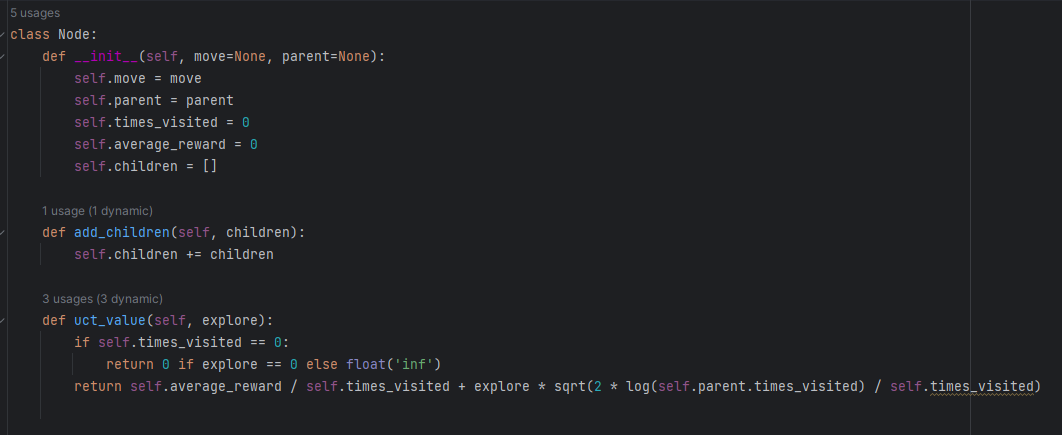
Regulile și opțiunile jocului, inclusiv informații detaliate despre comenzile GTP acceptate, sunt disponibile prin comanda About in meniul initial al programului.

**3.2. Aspecte generale despre proiect**

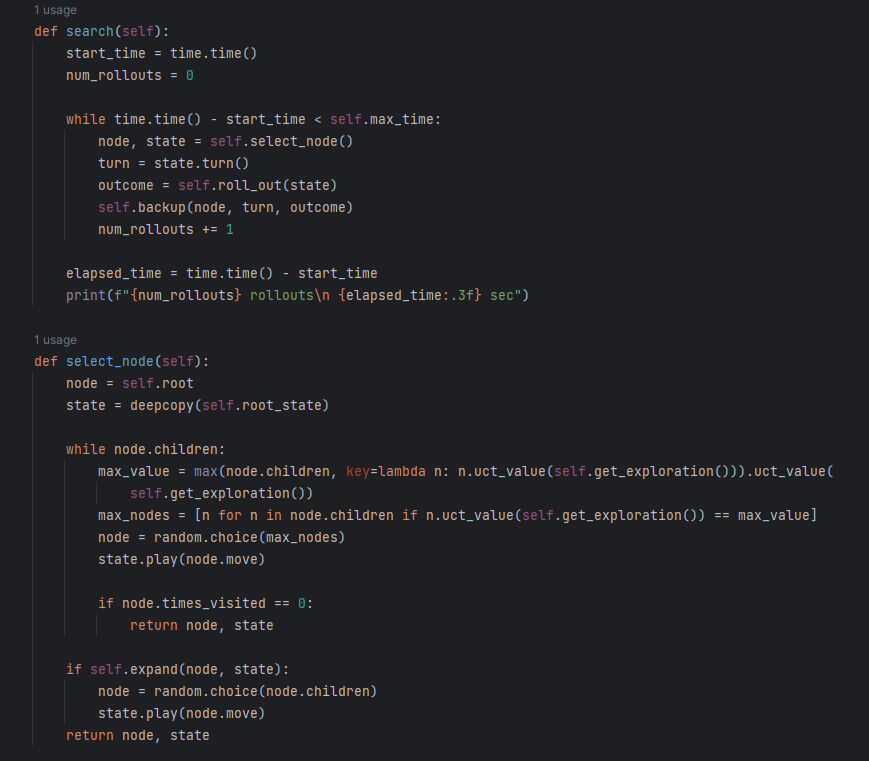
* gamestate.py: Gestionează logica jocului și oferă metode pentru efectuarea mișcărilor, verificarea stării victoriei și gestionarea schimbărilor de stare. Utilizeaza un algoritm depth-first search pentru detectarea conexiunilor intre celulele de joc.
* agent.py: Implementează agentul care utilizează algoritmul MCTS pentru a lua decizii în cadrul jocului. Acesta utilizează o structură de noduri de arbore pentru a gestiona explorarea și evaluarea posibilelor mișcări (clasa node.py).
* gtp.py: Implementează o interfață pentru comunicare folosind protocolul Go Text Protocol (GTP).
* game\_manager.py, console\_ui.py: Clase care asigura comunicarea intre utilizator si modulul GTP. Ofera metode helper pentru afisarea meniurilor si textului asociat comenzilor in timpul jocului (folosindu-se de pachetul utils).

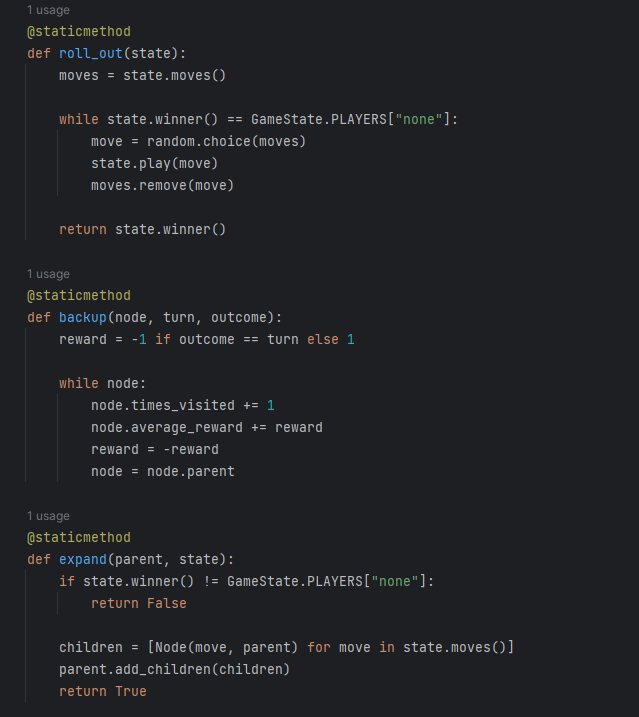
# Capitolul 4. Părți semnificative ale codului sursă

**4.1. Agentul MCTS**



Structura de tip nod asociata arborelui de joc. Se poate observa functia de calcul a valorii UCT.





Functiile specifice pasilor algoritmului Monte Carlo Tree Search.

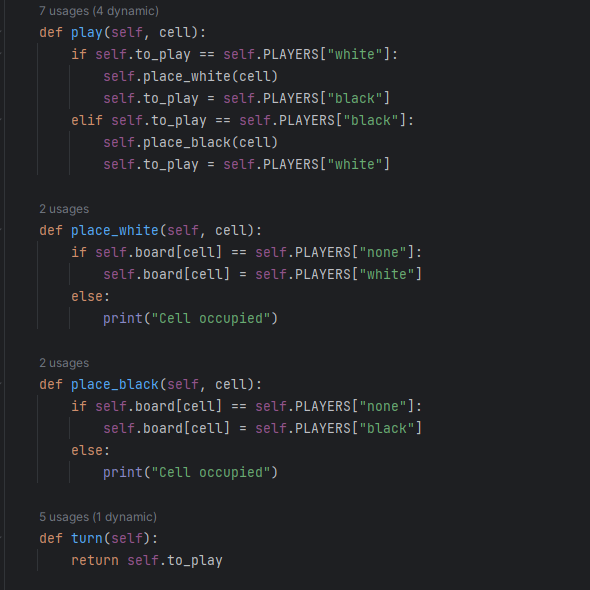
Timpul alocat miscarilor agentului poate fi setat prin intermediul dificultatii de joc, parametru care poate fi modificat de utilizator.

Se poate observa stransa legatura cu logica din gamestate, care dicteaza evenimentele ce au loc pe tabla de joc, cum ar fi plasarea de piese pentru fiecare jucator si detectarea castigatorului.

**4.2. Logica de joc**

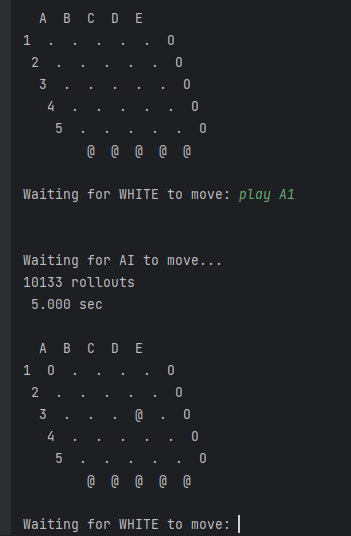
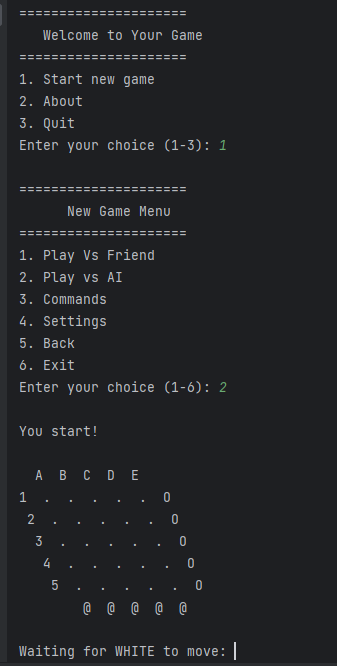
****

Algoritmul de detectare a unui lant neintrerupt intre capetele tablei de joc, fapt ce dicteaza daca un jucator a castigat meciul.

****

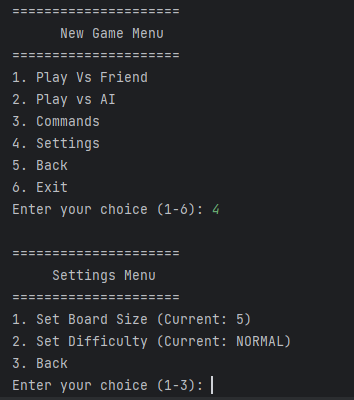
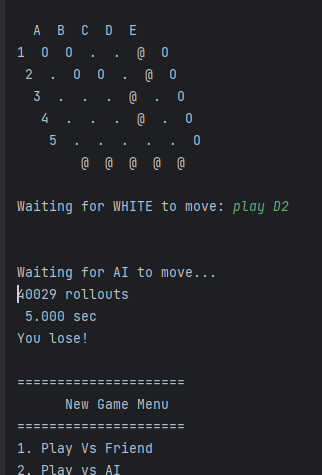
Logica de plasare a pieselor is alternarea turelor de joc.

# Capitolul 5. Rezultate obținute prin rularea programului



Initierea unui joc impotriva agentului MCTS si flow-ul unui astfel de meci. Se pot observa statisticile miscarii plasate de calculator, cele 5 secunde acordate fiind legate de dificultatea aleasa pentru joc.

Tabla de joc este afisata dupa fiecare miscare.



Sfarsitul meciului, agentul AI a castigat plasand cel mai probabil piesa neagra pe D5. Imediat dupa se poate alege inceperea unui joc nou.

In dreapta se poate observa meniul de setari, unde poate fi aleasa dimensiunea tablei de joc si dificultatea.

# Capitolul 6. Concluzii

Pe scurt, principalele avantaje ale MCTS, care diferențiază această metodă de majoritatea altor algoritmi de căutare, pot fi sintetizate astfel:

În primul rând, arborele problemei este construit asimetric, concentrându-se în principal pe liniile de acțiune cele mai promițătoare și explorând opțiunile care ar putea deveni cel mai probabil promițătoare.

În al doilea rând, formula UCT reprezintă cea mai populară metodă de asigurare a echilibrului între explorare și exploatare. În practică, metoda s-a dovedit eficientă în diverse probleme de căutare (chiar dacă nu în toate). Un avantaj suplimentar este justificarea teoretică a convergenței. Mai mult, metoda este inherent ușor de paralelizat.

Și în cele din urmă, MCTS este cunoscut sub denumirea de "anytime algorithm", adică poate fi oprit în orice moment și totuși va returna o soluție rezonabil de bună (cea mai bună găsită până în acel moment).

# Capitolul 7. Bibliografie

* <https://en.wikipedia.org/wiki/Monte_Carlo_tree_search>
* <https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/>
* <http://florinleon.byethost24.com/curs_ia.html>
* <https://www.maths.ed.ac.uk/~csangwin/hex/index.html>
* <https://www.kaggle.com/code/kmader/hex-board-game-statistical-approach>
* <https://towardsdatascience.com/monte-carlo-tree-search-implementing-reinforcement-learning-in-real-time-game-player-a9c412ebeff5>

# Capitolul 8. Lista contribuțiilor membrilor echipei

Documentația, clasa gamestate au fost realizate de Chircă Radu.

Clasele gtp, node au fost implementate de Ciocan Robert.

Clasa game\_manager, console\_ui, pachetul utils au fost implementate de Bejinaru Mihnea.

Clasa specifica agentului MCTS a fost realizata prin contributie comuna.