Proiect PA - Anul 2021
Echipa MEM

Membrii echipei:

- *Timpuriu Mircea*
- *Zaharov Evghenii*
- *Oprea Mihail*
- *Ghițan Răzvan*

ETAPA III

➔ *Instrucțiuni de compilare:*

Arhiva noastră a fost realizată în conformitate cu cerințele din enunț. De aceea, în rădăcina arhivei veți găsi un fișier Makefile care conține regulile ***build***, ***run***, ***clean***. Pentru a putea compila programul, este necesară compilarea surselor cu ajutorul comenzii ***make build***. După aceea, vom porni o partidă dintre botul nostru și botul pus la dispoziție de echipa PA, prin intermediul comenzii ***xboard -variant 3check -fcp "make run" -scp "pulsar2009-9b-64 mxT-depth" -tc 5 -inc 2 -autoCallFlag true -mg 4 -sgf partide.txt -reuseFirst false -debug***.

Pentru realizarea acestui proiect, ne-am hotărât să utilizăm ***limbajul de programare Java*** în detrimentul limbajului C++, deoarece am considerat mult mai ușoară folosirea acestuia, mai ales datorită cursului de POO din cadrul semestrului I.

➔ *Detalii despre structura proiectului:*

Structura proiectului nostru include următoarele clase:

- ***Main*** - în care citim și prelucrăm comenzile de la xboard;
- ***Logic*** - în care păstrăm informații generale despre starea jocului (e.g. culoarea care mută, numărul de mișcări efectuate);
- ***Board*** - reprezentarea internă a tablei;
- ***Piece*** - clasă abstractă care reprezintă o piesă;
- ***Pawn*** - una din clasele care moștenește clasa Piece și implementează comportamentul specific unui pion;
- ***Rook***, ***Knight***, ***Bishop***, ***King***, ***Queen*** - restul de clase care moștenesc clasa Piece și care descriu/realizează mutările posibile ale respectivelor piese;
- ***Coordonate*** - clasă pentru reprezentarea coordonatei;
- **Move** - clasa care stochează informațiile despre o mutare;
- **En passant** - clasă care stochează coordonata la care se poate captura pionul care realizează en passant;
- **Ray** - clasă care permite parcurgerea boardului pe o linie și determinarea amenințării căsuței inițiale de către celelalte piese.

➔ *Abordarea algoritmică a etapei:*

În timpul receptionarii comenzilor primite la stdin de la xboard, noi le vom analiza și vom putea interpreta comanda sau simula mișcarea primită în structura internă a programului, acest lucru fiind realizat în complexitate O(1).

Pentru a realiza o mișcare (calculul unei mișcării și trimiterea ei la stdout), noi vom genera lista de piese care pot fi mutate, acest lucru fiind realizat printr-o parcurgere a unei liste de piese aflate pe tablă, căutare ce se realizează în complexitate O(n), n fiind numărul de piese de pe tablă.

În implementarea noastră, piesa care urmează a fi mutată este aleasă în mod aleator dintre piesele disponibile. Această selecție este realizată în O(1), deci nu influențează complexitatea algoritmului.

Parcurgerea board-ului în implementarea clasei Ray pentru a determina dacă o piesă se află în șah sau în pericolul de a intra în șah este realizata în O(n), n fiind dimensiunea tablei.

Pentru etapa a III-a, am făcut în așa fel incat sa reprezentăm o structura de tip Board ca un nod dintr-un arbore și să creăm spațiul stărilor, un arbore prin care se reprezinta toate posibilele pozitii viitoare, determinate de mutările legale realizabile la un moment dat de către botul nostru.

Pentru reducerea spațiului stărilor și alegerea celei mai bune mutari la un moment dat, am folosit algoritmul alpha beta pruning, a cărui complexitate se poate exprima ca O(b^(d/2)), unde b = branching factor, în cazul nostru valoarea aproximativă este de 30 și d = depth, adică adâncimea arborelui de stări, în cazul nostru valoarea aceasta este 4.

Iar, pentru optimizarea strategiei botului nostru și antrenarea acestuia pentru varianta de joc 3-check chess, am implementat o serie de euristici: pentru început una care ia în considerare numărul, respectiv, valoarea pieselor rămase pe tablă la un anumit moment, una care ia în considerare distanța față de centrul tablei de joc la care se află regele, fiind mult mai avantajat cel care este mai departe de centru, una care ia în considerare numărul de piese ce înconjoară regele și îi oferă siguranța, iar, probabil cea mai importantă pentru modul actual de joc, cea care ia în considerare numărul de șahuri primite/date la un moment dat, și atribuirea unei importanțe acestora.

➔ *Surse de inspirație și responsabilitatea fiecărui membru:*

Deoarece toți pașii de implementare ai acestei etape sunt puternic dependenți unul de celălalt, am decis ca toți membrii echipei sa lucreze în același timp, în timp real și, de asemenea, pentru a putea discuta și împărtăși mai ușor idei de implementare și de euristici.

Pentru implementarea algoritmului alpha-beta pruning am folosit un pseudocod open-source.

➔ ***Bibliografie:***

https://makefiletutorial.com/ - pentru Makefile.
https://docs.oracle.com/en/java/ - pentru probleme / detalii despre metode si clase.
https://www.gnu.org/software/xboard/engine-intf.html - pentru a înțelege mai bine algoritmica din spatele acestei etape.
https://en.wikipedia.org/wiki/En_passant - pentru a înțelege toate regulile en passant-ului.
https://en.wikipedia.org/wiki/Castling - pentru a afla detalii despre condițiile rocadelor.
https://www.chess.com/article/view/3-check-chess-tips-for-beginners - pentru idei de implementare si euristici