UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

SPECIFICAŢIE PROIECT

PROGRAMARE ORIENTATĂ PE OBIECTE

AN ACADEMIC 2019-2020

TRUFIN RADU-SEBASTIAN – GRUPA 30223

ÎNDRUMĂTOR : NANEŞ ANA-MARIA

ABSTRACT

Acest document urmăreşte implementarea şi simularea în limbajul de programare Java a jocului Reversi/Othello cu ajutorul unei interfeţe grafice

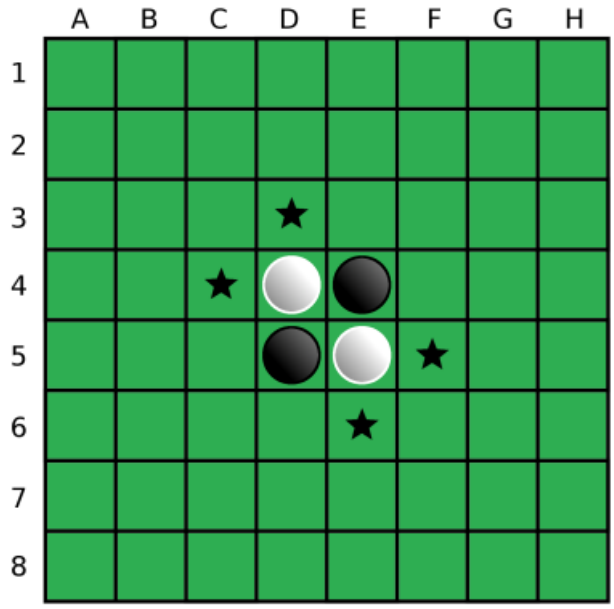
CUPRINS

1. *Introducere*

Reversi este unul dintre cele mai interesante jocuri logice (regulament simplu, partide dinamice, complexitate tactică ridicată, campionate, programe de calculator etc.). Inventat cu aproximativ un secol în urmă, el cunoaşte în ultimul timp o răspândire din ce în ce mai mare (este numit uneori şi *Othelo*).

Interesant este că de mai mulţi ani calculatorul s-a dovedit imbatabil în acest domeniu; mai exact, în 1982, campionul mondial la *Reversi* a fost învins de un program, şi acesta a fost primul joc raţional la care campionul mondial *en titre* a pierdut în faţa calculatorului.

2. *Reguli de joc*



Se joacă pe tabla de şah (pe orice caroiaj 8x8) cu 64 de piese discoidale, identice, toate având o faţă de o culoare şi cealaltă de o altă cu loare. Se începe fie cu tabla goală şi atunci primele patru mutări se fac în cele patru căsuţe centrale ale tablei.

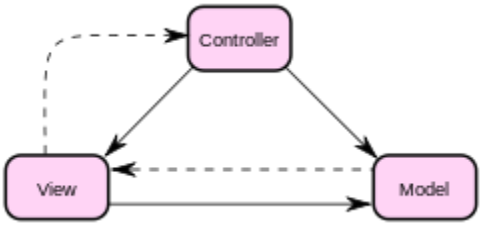
La fiecare mutare, jucătorul aflat la rând plasează pe tablă o piesă având în sus faţa a cărei culoare îi aparţine. La fiecare mutare (cu excepţia primelor patru, atunci când se pleacă cu tabla goală), piesa nou introdusă în joc trebuie să captureze măcar o piesă adversă, şi aceasta se face prin încadrare: piesa jucată trebuie să încadreze împreună cu o altă piesă, de aceeaşi culoare, piese adverse (de culoare opusă), pe o linie orizontală, verticală sau oblică. Piesele adverse sunt convertite în piese de culoare proprie şi intră în posesia celui care le-a capturat (ele nu părăsesc deci tabla). Pot fi încadrate simultan piese adverse pe mai multe direcţii, toate fiind capturate în urma mutării.

Un jucător care nu poate captura pierde rândul şi adversarul joacă încă o dată. Partida se încheie în momentul în care nici un jucător nu mai poate muta întrucât fie că s-a acoperit toată tabla, fie că nici un jucător nu poate captura. Cel care are mai multe piese cu culoarea proprie în sus la terminarea partidei este câştigătorul.

De reţinut că piesele sunt capturate numai în şiruri compacte şi numai prin încadrare cu ultima piesă depusă pe tablă. Dacă una dintre piesele tocmai capturate încadrează cu o altă piesă de aceeaşi culoare piese adverse, acestea nu se consideră capturate.

3. *Implementarea jocului*

Am utilizat (MVC) Model-View-Controller pentru a oferi acestuia o lizibilitate cât mai bună.



- **Model**

Această parte a controlatorului manipulează operațiunile logice și de utilizare de informație (trimisă dinainte de către rangul său superior) pentru a rezulta de o formă ușor de înțeles.

- **Vizualizare**

Acestui membru al familiei îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la controlor.

- **Controlor**

Cu acest element putem controla accesul la aplicația noastră. Pot fi fișiere, scripturi (eng. scripts) sau programe, în general orice tip de informație permisă de interfață. În acest fel putem diversifica conținutul nostru de o formă dinamică și statică, în același timp.

Modelul

Încapsulează datele specifice unei aplicații și definește logica și computațiile care manipulează și procesează datele respective.

In model este definită clasa **Board** a cărui câmp principal este o matrice de dimensiune 8x8 de tip player. Un player poate fi *NONE, BLACK, WHITE*.

Clasa **BoardCardinalIterator** iterează prin tablă în toate direcțiile cardinale.

```
{ -1, 0 }, // nord
{ -1, 1 }, // nord-est
{ 0, 1 }, // est
{ 1, 1 }, // sud-est
{ 1, 0 }, // sud
{ 1, -1 }, // sud-vest
{ 0, -1 }, // vest
{ -1, -1 } // nord-vest
```

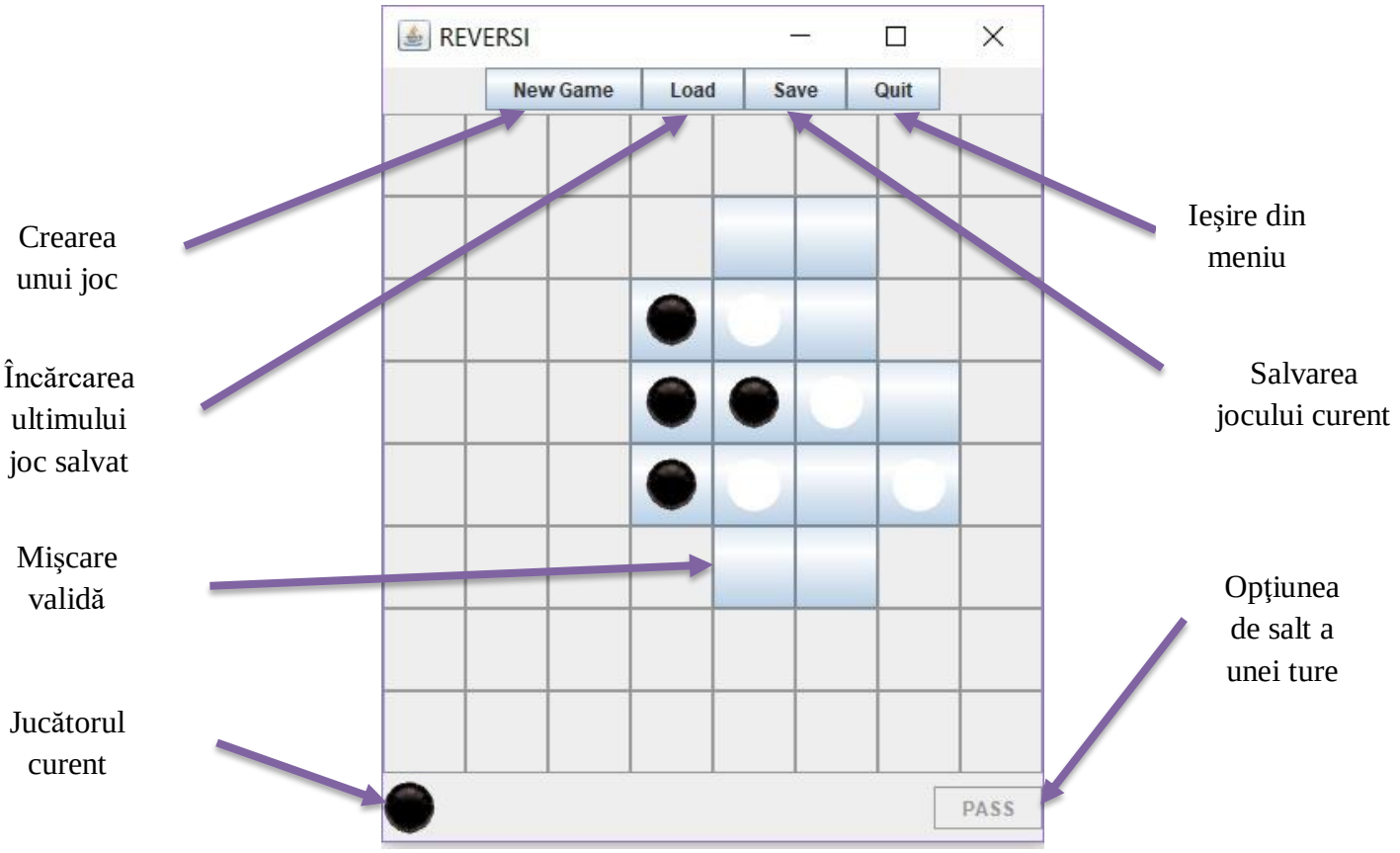
Clasa **BoardSquare** reprezinta o celula a tabelei, reprezentata prin linie-coloana-player.

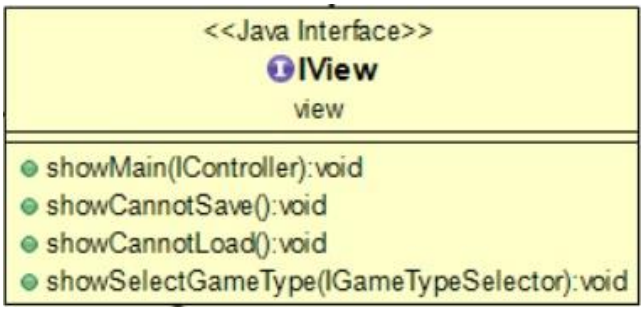
Clasa **GameState** ne indică situaţia în care se află meciul de Reversi/Othello. Starea poate fi *INTERACTIVE* (se așteaptă input-ul unui utilizator), *WAITING* (se aşteaptă input-ul calculatorului) sau *ENDED* (jocul este terminat).

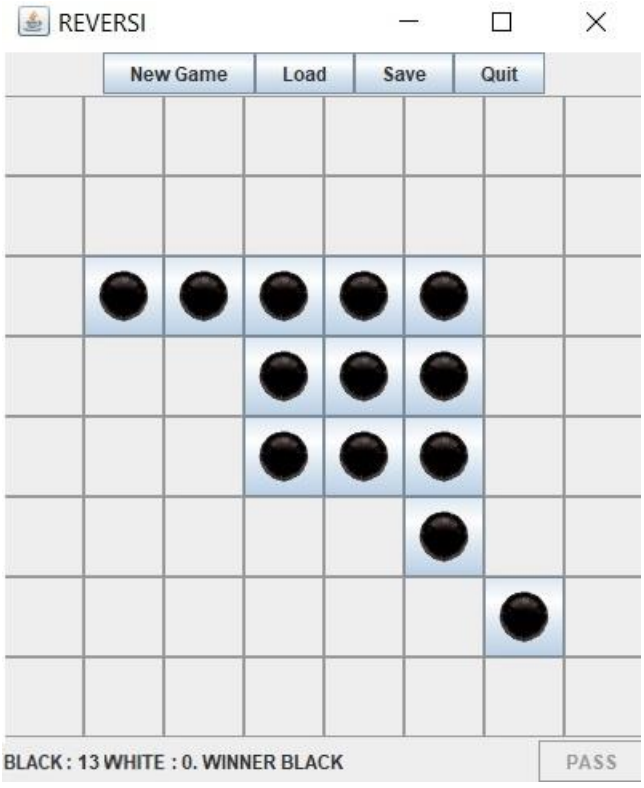
Clasa **LegalMovesIterator** iterează toate mișcările legale disponibile pentru jucător.

Vizualizarea

Redă conținutul modelului și specifică exact cum se prezintă utilizatorului datele din model. Dacă modelul se schimbă, vederea trebuie să-și actualizeze prezentarea după nevoi.



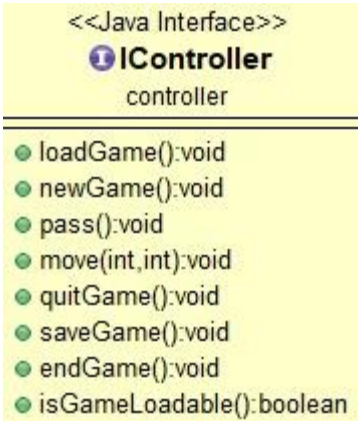




Frame-ul pricipal, interfața vederii și exemplu de victorie.

Controlorul

Utilizat pentru traducerea interacţiunilor utilizatorului, ce reprezinta click-uri pe butoane sau selecţii de meniu.



Interfaţa acestuia ilustrează toate metodele implementate în clasa Controller

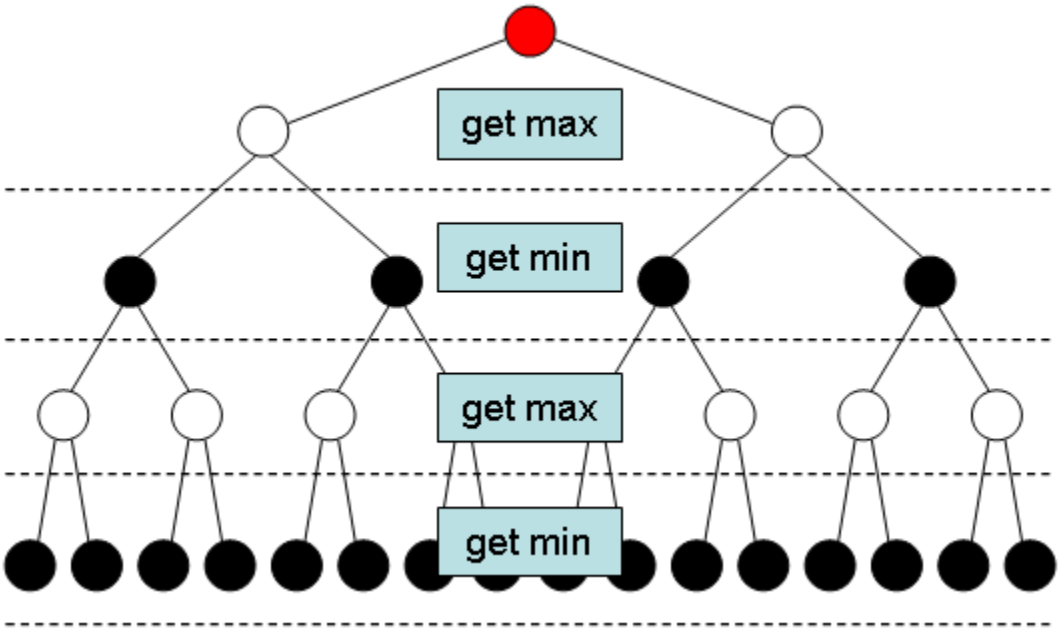
Metodele **loadGame()** și **saveGame()** lucrează cu obiecte de

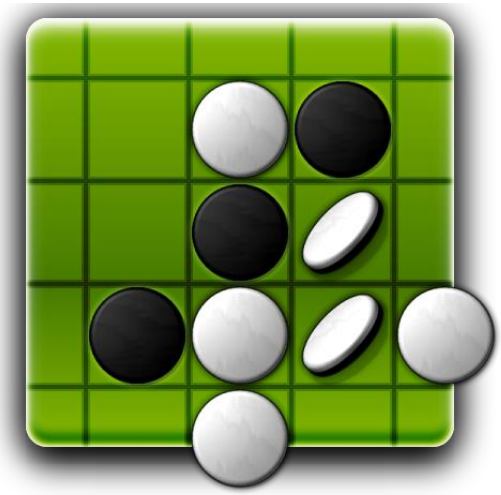
tipul ObjectInputStream, respectiv ObjectOutputStream care citesc

la load și scriu la load fișiere temporare pentru realizarea funcţiei dorite.

4. *Posibilitati de dezvoltare*

Algoritmul Minimax oferă o modalitate de a privi mai multe mișcări înainte, în timp ce maximizează scorul AI și minimizează scorul jucătorului uman. Definirea regiunilor strategice despre consiliu modelează capacitatea AI de a evalua riscurile de mișcare.





5. *Diagrama UML a claselor*