Projekat Blockade – Izveštaj

Faza 1

Projekat se sastoji od 5 fajla:

* **Blockade.py** – sadrži kod za unos osnovnih podataka o igri (Izbor oznake, veličina tabele, broj zidova) kao i sama logika odigravanja igre. Ukoliko je uključen debug mode (debugMode = true), korisnik neće unositi podatke već će biti kreirana osnovna igra sa predefinisanim parametrima.

Funkcije:

* + requestInputForSign() – zahtevanje unosa validnog broja – 1 za X, 2 za O simbol (Izbor simbola igrača, drugim simbolom će igrati računar kasnije).
  + requestInputForTableSize() – zahtevanje unosa broja vrsta(n) I kolona(m) tabele, maksimalno vrsta 22, kolona 28 (po uslovu zadatka)
  + requestInputForWallsNumber() – zahtevanje unosa broja zidova (min 1, max 10).
* **Field.py** – Klasa Field predstavlja jedno polje u tabeli(matrici).

Funkcije:

* + getSymbol() – vraća simbol(string) u zavisnosti od tipa polja
  + isEmptyWallField() – provera da li je u pitanju slobodno polje za zid
  + isWall() – provera da li je u pitanju polje sa zidom
  + setWall(color) – postavljanje zida u zavisnosti od boje
  + areWallsCrossing() – provera da li se zidovi seku u zavisnosti od boje novog zida
  + isFieldForPlayer() – provera da li je u pitanju polje igrača (validno polje za pijuna) – parna vrsta I kolona – više o tome u objašnjenju Table.py fajla
  + changeType(type) – promena tipa na type
  + **getAllPossibleMovementFields()** – vraća sva polja na koja može da skoči imaginarni pijun od trenutnog polja ukoliko je u pitanju polje za igrača. Koristi se kod određivanja putanje od pijuna do cilja (protivničkog početnog polja). Prednost korišćenja imaginarnog pijuna je ta sto su kod pijuna implementirane sve funkcije za validaciju poteza što nama i treba u ovom slučaju, da oponaša skakanje pravog pijuna do ciljnog polja.
  + getNumberOfWallsTouching(color) – vraća broj zidova koji se dodiruju sa zidom koji bi se postavio na ovo polje. Koristi se prilikom provere da li postavljeni zid zatvara put jednom od pijuna do završnog polja. Ako je broj zidova koji veći ili jednak 2, onda će funkcija za proveru puta da se pozove. Bitna je kod optimizacije da se ne proverava da li postoji put kod svakog postavljanja zida.
* **Enums.py** – Fajl Enums sadrži sve enumeracije koje kod čine preglednijim i razumnijim.

Enumeracija FieldType se odnosi na tipove polja:

* + EMPTY – prazno polje,
  + X – polje sa X pijunom,
  + O – polje sa O pijunom,
  + VERTICAL\_WALL\_EMPTY/HORIZONTAL\_WALL\_EMPTY – slobodno polje za postavljanje vertikalnog/horizontalnog zida
  + VERTICAL\_WALL\_FULL /HORIZONTAL\_WALL\_FULL – polje koje sadrži vertikalni/horizontalni zid

Enumeracija PlayStatus se odnosi na status(fazu) igre jednog igrača.

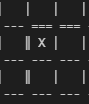
* **Table.py** – Klasa Table predstavlja tablu – matricu sa poljima. Elementi u parnim vrstama su prazno polje odnosno polje za igrača u slučaju parnih kolona ili vertikalni zid odnosno prazno mesto za vertikalni zid u slučaju neparnih kolona. Elementi u neparnim vrstama su horizontalni zidovi u slučaju parnih kolona ili prazna polja (nedostižne tačke) između zidova. Veličina matrice za obradu je 2\*n-1 vrsta i 2\*m-1 kolona (n broj vrsta table, m broj kolona table).

Funkcije:

* + initFields() – inicijalizuje matricu po navedenim pravilima
  + printTable() – štampa elemente tablice kreirajući vizuelno tablu po ugledu na tablu sa slajdova.
  + createPlayers() – instancira X i O igrače
  + getFieldByRowAndColumn(row, column) – vraća polje u matrici izračunavajući indekse matrice na osnovu unetog reda i kolone. Polje (1, 1) na tabli predstavlja polje (0, 0) u matrici koja se obrađuje. Vrsta matrice se računa po formuli: 2\*(row – 1), oduzimamo 1 zato što indeksi matrice počinju od nule dok indeks na tabli(vizuelno) počinje od 1. Zatim množimo sa 2 zato što pijuni mogu da se nalaze samo na parnim vrstama i kolonama. Kolona matrice se računa na sličan način, s tim što se umesto reda koristi prosleđena kolona table.
  + getFieldsForWall() – u odnosu na boju vraća pozicije polja table na koje treba da se postavi zid. Ako je uneta boja ‘p’ onda treba postaviti zid horisontalno, ako je uneta boja ‘z’ onda treba postaviti zid vertikalno.
  + putWallOnPosition() – postavlja zid na unetu poziciju ukoliko je pozicija validna I ne sadrži druge zidove ili se kosi sa drugim zidovima.
  + isGameFinished() – proverava da li postoji pobednik u igri I ispusuje ga na standardnom izlazu.
  + **parseEnteredValueToTableIndex()** – provera da li je igrač uneo validan broj/slovo za vrstu/kolonu table. Unos je validan ukoliko je u pitanju broj ili slovo od A do Z (isključivo velika slova). Obzirom da vrsta I kolona počinju sa brojem 1, slovo A će predstavljati broj deset odnosno 10. vrstu/kolonu. Sva slova dobijaju vrednost u zavisnosti od svoje ASCII vrednosti. ASCII vrednost – 55. 55 oduzimamo zato što A ima vrednost 10, svako naredno slovo ima vrednost za jedan broj veći od prethodnog.
  + requestInputForPlayerPosition(sign) – zahteva unos vrste I kolone polja za poziciju izabranog pijuna od strane igrača sa znakom sign koji je na potezu
  + requestInputForWallPosition(sign) – zahteva unos boje, vrste I kolone polja za poziciju zida od strane igrača sa znakom sign koji je na potezu
  + getPlayerByType(type) – vraća igrača u zavisnosti od prosleđenog tipa
  + placeWallsInFields(fields, colors) – postavlja zidove u datim poljima date boje
  + **playMoveInNewState(playerType, figureNumber, newX, newY)** – odigrava potez u novom stanju i vraća dato stanje.
  + **placeWallInNewState(color, i, j)** – postavlja zid u novom stanju i vraća dato staje
  + **isWallClosingPath()** – proverava da li u trenutnom stanju postoji zatvoreni pijun (pijun kojem je put do cilja blokiran). Vrši proveru da li postoji put za svokog od pijuna do oba cilja. Vraća dužinu puta ako put postoji ili -1 ako put ne postoji.
  + **calculateDistanceBetweenNodes(start, end)** – provera da li postoji put od startnog do ciljnog čvora. Implementiran A\* algoritam.
  + calculateHeuristic(node, end) – osnovna heuristika potrebna za rad A\* algoritma. Heuristika je nadograđena u *Manhattan diagonal distance* algoritam. Računa ispravnu heuristiku za kretanje u 8 pravaca (vodoravno, horizontalno i dijagonalno).
  + getGameWinner() – vraća pobednika trenutne igre ukoliko postoji ili None u suprotnom.
  + **calculateNextMoveMinMax(depth, alpha, beta, maximizingPlayer, computerPlayer)** – pomoću minimax algoritma određuje najbolji naredni potez do dubine depth. Računa sledeći potez za oba pijuna igrača koji je na potezu i i vraća bolji potez od oba pijuna.
  + **calculateMinMaxHeuristic(maximizingPlayer, winner)** – osnovna heuristika za minimax. Prvo se proverava da li je jedan od pijuna na ciljnom polju funkcija vraća velike vrednosti (9999 za naš potezi i -9999 za protivnički potez). Nakon toga se za svakog pijuna određuje najkraća putanja do ciljnog polja i sabira se sa heuristikom za pijune igrača koji je trenutno na potezu, odnosno oduzima za pijune protivničkog igrača.
  + **getPossibleWallPositions()** – vraća sve moguće pozicije zidova za naredna stanja
  + getAllPossibleWallNextStates() – vraća odigrana naredna stanja za sve moguće pozicije zidova i samu poziciju
  + **chooseNextWallPosition**(maximizingPlayer) – koristeći heuristiku za minimax određujemo koji bi zid od svih mogućih zidova vratio najpovoljnije stanje za trenutnog igrača.
* **Pawn.py –** Klasa predstavlja pijuna. Sadrži atribute startingRow i startingColumn – početne pozicije pijuna koje su bitne radi provere kraja igre.

Funkcije:

* setPawnOnTable() – postavlja pijuna na tabli.
* removePawnFromCurrentPosition() – briše pijuna sa trenutne pozicije. Koristi se kod uspešnog pomeranja pijuna, kada treba da se obriše pijun sa prethodne pozicije.
* updatePawnCordinates() – koristi funkcije removePawnFromCurrentPosition() i setPawnOnTable() kako bi se kroz jednu funkciju izvršio kompletan prelaz pijuna sa jednog polja na drugo.
* **movePawn(x, y)** – proverava sve validacije i ako su sve validacije ispravne pomeri pijuna na zadato polje
* validateMoveForBoardDimensions(x, y) – vrši validaciju unetih kordinata x i y u odnosu na ivice table, da li je izabrana pozicija izvan granica table.
* validateMoveForOneFieldMove(x, y) – validacija da li je player uneo potez koji je dužine 1 i da li je to polje na koje želi da ode ciljno polje.
* validateMoveDirection(x, y) – vrši validaciju da li pravilno krecemo u pravcima gore, dole, levo, desno za jednu ili dve pozicije i da li se ispravno krecemo po dijagonali za jednu poziciju
* validateMoveForWalls(x, y) – vrši validaciju da li se zid nalazi na putu do odredišne pozicije. Prvi deo funkcije vrši validaciju za pravce: gore, dole, levo i desno. Drugi deo funkcije u odnosu u koji pravac ide ( poziva funkciju za gore levo - diagonalMoveUpLeft(x, y), itd. ).
* **diagonalMoveUpLeft(x, y)** – vrši proveru četiri različita slučajeva pozicije zidova.

* validateMoveForOtherPawns(x, y) – vrši validaciju da li pijun može da se pomeri na polje koje je već zauzeo drugi pijun. Funkcija ignoriše slučaj kada se protivnički pijun nalazi na njegovoj startnoj poziciji.
* validateMoveIfPawnOnNeighborField(x, y)– vrši validaciju da li se pijun nalazi na polju izmedju sadašnjeg i polja na koje želimo da odemo.
* **validateMove(x, y)** – provera da li je potez validan bez štampanja poruka.
* **getPossibleMoves()** – vraća sve validne naredne poteze za datog pijuna
* **getAllPossibleNextStates()** – vraća sva moguća naredna stanja(odigrane poteze) za datog pijuna
* **Player.py** – Klasa predstavlja igrača (X ili O). Sadrži atribut *type* koji predstavlja tip (X ili O). Takođe, sadrži listu pijuna.

Funkcije:

* + createPawn(i, j) – kreira novog pijuna za datog igrača na zadatim pozicijama
  + isWinner() – proverava da li je naš pijun došao do protivničke startne pozicije.
  + choosePawn() – zahteva unos rednog broja pijuna igrača.
  + **play()** – odigrava potez jednog igrača. Prvo zahteva unos rednog broja pijuna, zatim unos nove pozicije, nakon toga, ukoliko igrač ima preostale zidove, zahteva unos pozicije zida, kao i izbor boje zida.
  + getFigureByNumber(num)– vraća pijuna datog igrača sa rednim brojem num
  + computerPlay() – potez za datog igrača odigrava(određuje) kompjuter
  + humanPlay() – potez za datog igrača unosi čovek (sa tastature)