Izveštaj projekta “Domineering”

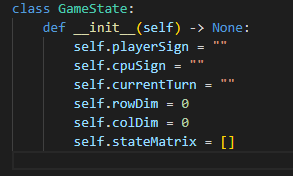
Nikola Sretković 17409

Darko Stošić 16413

*1.Faza – Formulacija problema i interfejs*

Stanje igre

Za definisanje stanja problem igre koristi se klasa **GameState** koja sadrži znakove koji predstavljaju igrače, ko je na potezu, dimenzije table, kao i matricu stanja igre.



Stanje (stateMatrix) je predstavljeno kao lista listi. Svaka podlista je jedan red table, a svaki element podliste predstavlja polje. Prazno polje je definisano sa “ “, a zauzeto polje je u zavisnosti “X” ili “O”.

Inicijalizacija igre

Igru pokrecemo iz fajla **main.py**. Direktno u kodu hardkodiramo maksimalne vrednosti za veličinu table koje možemo kasnije uneti kroz konzolu.

*main.py*

Text

Description automatically generated*GameInitializer.py*

**Funkcija initializeGame(...)** kao parametre prima maksimalni broj redova i kolona. Kolika će dimenzija table biti i ko igra prvi određuje korisnik kroz konzolu uz pomoć funkcija **getTableDimensions(...)** i **getWhoPlaysFirst(...)**.

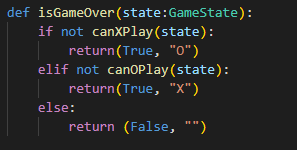
Ovi parametri se šalju u funkciju **initializeGameState(...)** gde se inicijalizuje promenljiva tipa **GameState** koja sadrži početne parametre i inicijalizovanu praznu tablu datih dimenzija spremnu za igru.

Provera Kraja Igre

Text

Description automatically generatedNakon inicijalizacije promenljiva **isGameFinished** se postavlja na „False“ i partija traje dok jedan od igrača u narednom potezu nema prostora da odigra potez na tabli kada se u promenljivu **isGameFinished** upisuje „True“ iz funkcije **isGameOver(...)** i pratija se završava.

Provera da li igrač može da odigra potez se vrši pomoću funkcija canXPlay(state) ili canOPlay(state) gde se na osnovu trenutnog stanja određuje da li postoje 2 horizontalna(O) ili vertikalna(X) mesta zaredom u matrici stanja.

*GameManager.py*

Funkcija **isGameOver(...)** u zavisnoti da li igra igrač „X“ ili „O“ koristi funkciju **canXPlay(...)** ili funkciju **canOPlay(...)** gde prolazi kroz moguće poteze igrača u zavisnoti od toga da li igrač postavlja uspravne ili položene domine i tako određuje da li postoji potez koji može da odigra u narednom potezu.

Odigravanje poteza

Text

Description automatically generated

Funkcija **playTurn(...) - *TurnController.py*** određuje da li je na potezu igrač „X“ ili igrač „O“, a zatim funkcije **getValidIntInput(...)** i **getValidCharToIntInput(...)** određujuju u kom redu i kojoj koloni želimo da postavimo dominu.

Text

Description automatically generated**getValidIntInput(...)** određuje red u koji postavljamo dominu i vraća grešku ukoliko je doslo do pogrešnog unosa u pogledu prekoračenja dimenzija table ili unosa nekog karaktera koji nije broj.

Text

Description automatically generated**getValidCharToIntInput(...)** određuje kolonu u koju postavljamo dominu i vraća gresku u slučaju pogrešnog unosa u pogledu prekoračenja dimenzija table ili unosa nekog karaktera koji nije oznaka kolone. Koristi pomoćnu funkciju **getNumberFromASCII(...) - *ASCIIConverter.py*** da bi pretvorila karakter koji unosi korisnik u broj koji predstavlja željenu kolonu.

Provera validnosti odigranog poteza

Text

Description automatically generated

Nakon unosa korisnika funkcijama **isVerticalMoveValid(...)** i **isHorizontalMoveValid(...)** proveravamo da li se potez koji je uneo korisnik može odigrati u smislu provere da li su na tabli 2 polja gde bi se potez odigrao slobodna.

U slučaju valjanih unosa od strane korisnika potez se završava, pamti se novo stanje i menja se koji igrač je na potezu.

Prikaz trenutnog stanja

Graphical user interface, website

Description automatically generatedFunkcija **printCurrentState(...)** *- StatePrinter.py* vrši štampu čitave table. Matrica koja predstavlja stanje se štampa od poslednjeg do prvog reda, takođe postoje pomoćne funkcije koje štampaju separatore između redova i između samih polja zbog bolje vizuelizacije stanja.

*2.Faza – Implementacija operatora promene stanja*

Unos početnih parametara igre

Text

Description automatically generated

Što se tiče unosa parametara igre za sada unosimo samo dimenzije table a podrazumevano je da igraju igrač protiv igrača a prvi igra „X“.

Unos poteza i provera da li je potez ispravan, Unos novog poteza sve dok on nije ispravan, Odigravanje novog poteza ako je ispravan i promena trenutnog stanja igre

Implementirano u fazi 1 u funkciji **playTurn(state)** *- TurnController.py.*

Izmena u odnosu na fazu 1 je da se sada playTurn funcija zove **playTurnWithInputs(state).** U odnosu na prvobitnu verziju, kao parametar prima trenutno stanje i na osnovu unosa sa tastature kreira i vraća novo izmenjeno stanje. Razlog preimenovanja funkcije je potreba da imamo jos jednu playTurn funkciju (**playValidTurnInstantly(state, row, col)**) u kojoj nije potreban unos sa tatature, već unos reda i kolona za potez prosleđujemo kao parametar funkcije.

Text

Description automatically generated

Prikaz novonastalog stanja igre nakon odigravanja poteza

Implementirano i objašnjeno u fazi 1 u funkciji **printCurrentState(...)** *- StatePrinter.py*

Pozivamo je u *main.py* nakon sto se zavrsi playTurn za trenutni potez.

Proveru kraja i određivanje pobednika u igri nakon odigravanja svakog poteza

Implementirano i objašnjeno u fazi 1 u funkcijama **canXPlay(state)**, **canOPlay(state), isGameOver(state).** *- GameManager.py*

Realizovati funkcije koje na osnovu zadatog poteza i zadatog stanja igre formiraju novo stanje igre

Funkcije **playValidTurnInstantly(state, row, col)** i **playTurnWithInputs(state).**

Kao argument dobijaju trenutno stanje i na osnovu poteza(red,kolona) kreiraju i vraćaju novo izmenjeno stanje.

Realizovati funkcije koje na osnovu zadatog igrača na potezu i zadatog stanja igre formiraju sva moguća stanja igre

Funkcija **getAllPossibleMoves(state)** – vraća listu svih mogućih validnih poteza za trenutnog igrača.

**Text

Description automatically generated**

Funkcija **getAllPossibleNextStates(currentState)** – koristi getAllPossibileMoves da dobije listu svih validnih poteza. Iteracijom kroz sve validne poteze i korišćenjem playValidTurnInstantly() koji vraća novo stanje, dodaje i vraća svako novo stanje u listu. Tako dobijamo svako moguće naredno stanje.

Text

Description automatically generated

Takodje su implementirane **printAllPossibleMoves i printAllPossibleNextStates** zbog testiranja funkcionalnosti gore objašnjenih funkcija.

*3.Faza – Min-max algoritam i heuristika*

Heuristika pomoćne funkcije *- HeuristicHelpers.py*

U ovom python fajlu se nalaze pomoćne funkcije koje koristi heuristika.

Provera da li se domina nalazi iznad/ispod za O ili sa leve/desne strane za X:

**areThereTwoFreeFieldsLeft, areThereTwoFreeFieldsRIght, areThereTwoFreeFieldsBelow, areThereTwoFreeFieldsAbove**

Filtriranje najboljih stanja – **filterBestStates** :

Koristimo je da ručno izaberemo stanja koja očigledno bolja od ostalih. Takva stanja su stanja gde potez nije odigran uz ivicu table, gde potez koji je odigran nije „bezbedan“ potez (bezbedan potez je potez koji protivnik ne može da spreči) zato što je cilj da se ona ostave na kraju kada nije moguće osvajanje novih delova table i osujećivanje protivnika.

Text

Description automatically generated

Korišćenjem ove funkcije dobijamo na brzini tako što se odreknemo određenog broja stanja pre samog odsecanja u min-max algoritmu.

Da li je „bezbedan“ potez – **isSafeTurn** :

Kao što je objašnjeno iznad, bezbedan potez je onaj potez koji je odigran na mesto gde protivnik nije mogao da spreči taj potez.

Ova funkcija, u zavisnosti čiji je potez, proverava polja oko sebe (za X proverava levo i desno dva polja a za O gore i dole dva polja). Na primer ako se oko X domine sa obe strane nalazi još po jedna X domina potez možemo da smatramo bezbednim.

Bezbedne poteze je najbolje ostaviti za kraj partije kada samo popunjavamo tablu a ne možemo da naštetimo protivniku.

Da li izbegavamo ivicu – **isEdgeAvoided** :

Za X proveravamo da li se domina nalazi uz levu ili desnu ivicu table.

Za O proveravamo da li se domina nalazi uz gornju ili donju ivicu table.

Heuristika *- Heuristics.py*

Ocenjivanje stanja ukoliko je bezbedan potez – **isSafeTurnEvaluation** :

Ukoliko potez nije bezbedan vraćamo 5 poena jer nam je cilj izbegavanje bezbednih poteza dokle god je to moguće. Ukoliko je potez bezbedan vraćamo 0 poena.

Ocenjivanje stanja po tome da li kreiramo bezbedne poteze – **isCreatingSafeTurnsEvaluation** :

Cilj nam je da u završnoj fazi partije imamo što više bezbednih poteza. To postižemo tako što prioritizujemo potez koji se nalazi na drugoj/pretposlednjoj koloni za X ili drugom/pretposlednjem redu za O. Takođe uzimamo u obzir da li se iznad ili ispod (O) ili levo ili desno (X) nalaze prazna polja i tako sebe dovodimo u situaciju da sledećim potezom kreiramo još bezbednih stanja.

Text

Description automatically generated

Ocenjivanje stanja po tome da li prekidamo protivnički zid – **isStoppingEnemyWallEvaluation** :

Ukoliko protivnik (npr. X) kreira zid po nekoj koloni – postavlja u jednoj koloni svoje domine on sebi povećava kontrolu nad delom table. Ova funkcija ocenjuje stanje po to tome da li se sprečava kreiranje zida.

Text

Description automatically generated

Evaluacija stanja – **evaluateState** :

Na osnovu gore nabrojenih funkcija evaluiramo broj poena za stanje. Ukoliko to radimo za X, multiplikator je 1 zato što za njega računa max funkciju u min-max-u, a ukolko to radmo za O multiplikator je -1 zbog min funkcije.

Text

Description automatically generated

Min-max algoritam *- MinMax.py*

Min-max – **minimax** :

Text

Description automatically generated

Maksimalno stanje – **maxState** :

A screenshot of a computer

Description automatically generated

Minimalno stanje – **minState** :

A screenshot of a computer

Description automatically generated with medium confidence