**Logika implementacije I faze projekta Triggle**

Ovaj deo dokumenta obuhvata celokupno objašnjenje logike koja je primenjena za implementaciju I faze projekta Triggle. U prvoj fazi je bilo potrebno odraditi sledeće funkcionalnosti :

1. Omogućiti izbor ko igra prvi (čovek ili računar).

2. Omogućiti izbor simbola igrača koji igra prvi (X ili O).

3. Unos dimenzija table i validacija unosa.

4. Generisanje početnog stanja igre na osnovu unosa.

5. Prikaz trenutnog stanja table sa pozicijama čvorova i poteza.

6. Unos poteza sa validacijom unosa.

7. Provera kraja igre na osnovu popunjenosti table ili broja zauzetih trouglića.

8. Definisati način za predstavljanje stanja problema (igre)

* Predstavljanje pozicija razvučenih gumica i zauzetih trouglića.

9. Napisati funkciju za postavljanje početnog stanja

* Definiše se na osnovu zadate veličine table.

10. Napisati funkcije za proveru kraja igre

* Tabla je popunjena razvučenim gumicama ili je igrač zauzeo više od polovine trouglića.

Radi bolje organizovanosti i preglednosti koda sve funkcije za izradu prve faze su podeljene u sledeće fajlove:

* **board.py** : sadrži sve funkcije za rad sa tablom
* **game\_logic.py**: sadrži sve funkcije vezane za logiku same igre, poteza i provere validnosti, kraja igre…
* **utilities.py**: sadrži sve pomoćne funkcije
* **main.py**: glavni deo programa

U nastavku dokumenta sledi opis ideje koja je primenjena za svaku od funkcionalnosti u prvoj fazi.

Inicijalizacija table za igru Triggle

U našoj implementaciji, funkcija **draw\_hexagon(side\_length)** generiše matricu koja predstavlja heksagonalnu tablu.

* Korisnik na početku unosi dimenziju table kao veličinu stranice šestougla (side\_length) , nakon čega se kreira matrica sa tačkama (stubovima u igri) označenim kao '●' .
* računaju se visina i širina matrice i kreira se matrica sa praznim poljima
* popunjava se matrica tačkama uporedno i simetrično odozgo i odozdo ka sredini
* srednji red se popunjava u narednoj petlji
* svaka tačka se pamti u čvoru (nodes) koji je implementiran kao dictionary koji čuva par x,y koordinata
* obezbedjuje se da svaki red kao i kolona sadrži dovoljno prostora za potege i trougliće
* funkcija vraća iscrtanu tablu kao i sve čvorove table.

Prikaz table

Funkcija **print\_board(matrix, nodes)** prikazuje trenutno stanje table na osnovu matrice matrix.

* red first\_row omogućava da brojevi kolona počinju sa leve strane ali da budu poravnati sa stubovima matrice za gornju polovinu matrice, kao i last\_row za donji deo
* brojevi se zatim dodaju na svakoj šestoj poziciji i tako omogućujemo da se odgovarajuć broj kolone nadje u liniji sa odgovarajućom kolonom.
* prolazi se kroz svaki red matrice i ukoliko je on deljiv sa 3 dodaje oznaku reda kao veliko slovo azbuke (dobijeno pomoću ASCII vrednosti, dodavanje u svakom trećem redu jer su tu čvorovi). Pre ispisivanja smo sve elemente pretvorili u stringove zbog korišćenja join funkcije.

Unos poteza igrača i validacija poteza

Unos poteza igrača omogućujemo kroz funkciju **play\_move(matrix, nodes, start, direction)**. Potez uključuje početnu poziciju (npr. A1) i smer razvlačenja gumice (D - desno, DD - dijagonalno desno, DL - dijagonalno levo).

* na početku funkcije definisani su svi validni pravci i izvršena je provera da ukoliko nije unet dozvoljen pravac ili ukoliko stub oko kog želimo da krenemo razvlačenje gumice nije u nodes, potez je nevalidan.
* u positions listi se čuvaju sve pozicije matrice koje treba ažurirati potezom. U zavisnosti od direction-a u position listu se dodaju pozicije na kojima bi trebale da budu gumice.
* pre nego da se potez ažurira, vršimo proveru da li su sve pozicije validne (funkcija **is\_valid\_move(matrix, x, y)**, proverava da li je koordinata x,y unutar granica matrice i da li postoji čvor) i da li je na poslednjoj poziciji stub(gumice obuhvataju 4 stuba)
* ukoliko jesu, dodaje se poteg u matrici, a ukoliko nisu onda je nevalidan potez.

\*dodat **helper** parametar koji služi da dodaje sve čvorove poteza u listu zauzetih ako je potez validan i odigran

Zauzimanje trouglića (tj. x ili o) na tabli i provera za kraj igre

Ovaj deo zadatka smo implementirale pomoću funkcije **draw\_triangle(matrix, symbol, count)**.

* Prolazimo kroz svaku poziciju matrice pomoću dvostruke petlje i ako je trenutna pozicija u matrici prazna, onda se prvo vrši provera da li je moguće formirati trougao. Postoje dve opcije:
* “–” na dnu(pozicija gore (i+1, j) mora biti validna i sadržati "-", pozicija gore-desno (i-1, j+1) mora biti validna i sadržati "\", pozicija gore-levo (i-1, j-1) mora biti validna i sadržati "/",
* “–” na vrhu (pozicija dole (i-1, j) mora biti validna i sadržati "-", pozicija dole-levo(i+1, j-1) mora biti validna i sadržati "\", pozicija dole-desno (i+1, j+1) mora biti validna i sadržati "/", )
* i ako jeste dodajemo ga i označavamo simbolom koji predstavlja igrača (x ili o) i povećavamo count
* Funkcija vraća broj nacrtanih trouglova.

Promena igrača (+započet deo II faze)

Pomoću funkcije **switch\_player(current\_player)**, omogućeno je da se menja koji igrač je trenutno na potezu (računar ili čovek).

Izbor koji igrač igra prvi (čovek ili računar)

Ovu funkcionalnost smo implementirale u Main delu programa. Od korisnika se zahteva da unese ko igra prvi (čovek ili računar) i da ukoliko navede neku drugu opciju od te dve se ponovo vrati i izabere jednu od dve moguće.

Izbor simbola igrača koji igra prvi (X ili O)

I ovu funkcionalnost smo implementirale u Main delu programa. Od korisnika se zahteva da unese koji simbol bira i da ukoliko navede neku drugu opciju od te dve se ponovo vrati i izabere jednu od dve moguće ( X ili O). Nakon njegovog odabira za drugog igrača se postavlja suprotan simbol i pravi se dictionary symbols za čuvanje informacija o simbolima.

Prikaz trenutnog stanja tabele na osnovu prosledjenog poteza

Funkcija **parse\_matrix(matrix\_str)** pretvara stanje prosledjeno stringom u matricu sa tabelom koja sadrži tablu sa već odigranim potezima (priprema za obradu u narednoj fazi).

Kraj igre

Funkcija za kraj igre **end\_of\_game(matrix, count, side\_length)** koristi pomoćnu funkciju **max\_connections(side\_length)** koja računa koliko potega je moguće povući kroz tablu i završava igru ako:

* tabla popunjena (svi potezi izmedju čvorova povučeni)
* neki od igrača je popunio više od pola trouglića i tako pobedio.

**Logika implementacije II faze projekta Triggle**

Ovaj deo dokumenta obuhvata celokupno objašnjenje logike koja je primenjena za implementaciju II faze projekta Triggle. U drugoj fazi je bilo potrebno odraditi sledeće funkcionalnosti :

1. Realizovati funkcije koje obezbeđuju odigravanje partije između dva igrača (dva čoveka, ne računara i čoveka)
   * Unos početnih parametara igre
   * Ponavljanje unosa novog poteza sve dok se ne unese ispravan potez
   * Odigravanje novog ispravnog poteza sa promenom trenutnog stanja igre
   * Prikaz novonastalog stanja igre nakon odigravanja poteza
   * Proveru kraja i određivanje pobednika u igri nakon odigravanja svakog poteza, odnosno promene stanja igre
2. Realizovati funkcije koje implementiraju operator promene stanja problema (igre)
   * Realizovati funkcije koje na osnovu zadatog poteza i zadatog stanja igre formiraju novo stanje igre
   * Realizovati funkcije koje na osnovu zadatog igrača na potezu i zadatog stanje igre (table) formiraju sve moguće poteze
   * Realizovati funkcije koje na osnovu svih mogućih poteza formiranju sva moguća stanja igre, korišćenjem funkcija iz prethodne dve stavke
3. Napisati funkcije za proveru valjanosti poteza na osnovu konkretnog poteza i trenutnog stanja problema (igre).

U nastavku su opisane funkcionalnosti koje su realizovane u ovoj fazi, uz objašnjenje njihovih implementacija. Takođe, napravljene su neke izmene u odnosu na 1. Fazu:

* Funkcija **parse\_matrix(matrix\_str)** je izbačena, pa je taj deo koda sada implementiran kroz funkciju **pass\_board\_state(moves, matrix, yp, nodes, first\_player, second\_player, symbols, side\_length)**
* Funkciji **play\_move(matrix, nodes, start, direction)** je dodat parameter sa podrazumevanom vrednošću helper, dodatne provere za valjanost poteza...
* Dodatne izmene u main.py file-u.

Funkcije za proveru valjanosti poteza

Većina funkcionalnosti za ovaj deo II faze je bila već implementirana u I fazi. Dodatno je uvedena funkcija **is\_valid\_rubber\_band(positions)** (game\_logic.py) koja proverava da li su svi čvorovi koji čine potez razvlačenja gumice po tabli već zauzeti.

* Isti potez se ne može odigrati dva puta (globalni set zauzeti\_cvorovi sadrži sve već zauzete čvorove na tabli)
* Funkcija koristi all(...) da bi utvrdila da li su sve koordinate iz liste positions zauzete.
* U funkciji **play\_move** je u delu gde se proverava validnost poteza dodat i ovaj uslov. Ako su svi uslovi ispunjeni, onda se u set zauzeti\_cvorovi dodaju svi čvorovi u potezu (kako bi se onemogućilo kasnije igranje istog poteza).

Funkcije koje obezbeđuju odigravanje partije između dva igrača (dva čoveka, ne računara i čoveka)

Ove funkcionalnosti su implementirane u Main delu programa (deo iz prve faze nadograđen). Dodato je da igrač može da izabere da li će igrati protiv drugog čoveka (player vs player) ili protiv računara, čime se setuje mod igre. Takođe:

* igrač sada može da izabere da li će igrati prvi ili drugi.
* kada se setuju parametri za odigravanje u promenljivoj **first\_player** se pamti prvi igrač, u promenljivoj **second\_player** drugi igrač i **current\_playe**r promenljiva dobija vrednost first\_player.
* kroz while petlju se vrši naizmenično odigravanje poteza kao u prvoj fazi, s tim što se prvim if-om fokusiramo na player vs player igru, dok je u else deo logika za računar (biće odradjena u 3. Fazi, sada implemenitrana kao za običnog čoveka).
* izvršena mala promena u funkciji **switch\_player(current\_player, player1, player2)**, kako bi se korektno smenjivali igrači.

Funkcije koje implementiraju operator promene stanja problema (igre)

**pass\_board\_state(moves, matrix, yp, nodes, player1, player2, symbols, side\_length)** je funkcija koja omogućava:

* prikazivanje stanja na tabli
* pamćenje već unetih poteza svakog igrača kroz prosledjeni niz naizmeničnih poteza igrača.

**possible\_states(matrix, nodes)** na osnovu poslednjeg stanja table nalazi i ispisuje:

* sve moguće dalje poteze
* moguća stanja table za svaki potez

**print\_states(states,y)** je pomoćna funkcija za prikaz svih mogučih poteza i stanja table nakon odigranih mogućih poteza.

**Logika implementacije III faze projekta Triggle**

Ovaj deo dokumenta obuhvata celokupno objašnjenje logike koja je primenjena za implementaciju III faze projekta Triggle.

U trećoj fazi projekta fokus je bio na implementaciji veštačke inteligencije za igru Triggle korišćenjem **Min-Max algoritma sa alfa-beta odsecanjem**,kao i **heurističke procene stanja igre** zasnovane na skupu pravila i zaključivanju. Takođe je realizovana potpuna interakcija između čoveka i računara u toku igre.

Implementacija Min-Max algoritma sa alfa-beta odsecanjem

U okviru **game\_logic.py**, implementirana je funkcija **minimax\_alpha\_beta(matrix, nodes, depth, alpha, beta, maximizing\_player, player\_symbol, opponent\_symbol)** koja omogućava optimalan izbor poteza za računarski protivnik:

* Funkcija rekurzivno pretražuje moguća stanja do određene dubine.
* Koristi se alfa-beta odsecanje za eliminaciju neperspektivnih grana i ubrzavanje pretrage.
* Za svako stanje poziva se heuristička funkcija **evaluate\_state(matrix, player\_symbol, opponent\_symbol)** koja ocenjuje poziciju u skladu sa nizom pravila.

Heuristička procena stanja igre

Heuristika je realizovana preko funkcije **evaluate\_state(...)** i zasniva se na sledećim pravilima:

* Broj osvojenih trouglova – osnovna vrednost pozicije.
* Broj povezanih trouglova – prednost se daje povezanim simbolima (**count\_connected\_triangles(matrix, symbol)**).
* Kontrola teritorije – centralna pozicija i slobodne susedne pozicije ocenjuju se dodatnim poenima (**evaluate\_territory\_control(matrix, symbol)**).
* Mogućnost daljih osvajanja – broj pozicija na kojima se može brzo formirati trougao (**count\_possible\_triangles(matrix, player\_symbol)**).

Ova pravila predstavljaju mehanizam zaključivanja koji je korišćen za definisanje heurističke vrednosti.

Automatska igra računara

U main.py, dodata je logika koja omogućava da se potez za računar odigra automatski:

* Kada je računar na potezu, koristi se **minimax\_alpha\_beta(...)** kako bi se pronašao najbolji potez.
* Dubina pretrage se određuje dinamički u zavisnosti od veličine table (**get\_depth\_based\_on\_board\_size(side\_length)**). Ova funkcija se nalazi u fajlu utilities.py i uvedena je zato što je dubina 3 na većim tablama dovodila do jako dugog čekanja da računar odigra potez. Sa manjim dubinama sve je radilo dosta brže i potezi su i dalje bili dobri. Koliko smo čitale, ovakvo prilagođavanje dubine se često koristi u ovakvim igrama da bi sve bilo brže i efikasnije.
* Potez koji računar odigra se automatski prikazuje korisniku uz odgovarajuće ažuriranje stanja.

Poboljšanje u interakciji

* Funkcija **play\_move(...)** je proširena dodatnim parametrom showMessage koji korisniku daje preciznije povratne informacije o tome zašto potez nije validan (van granica, već odigran, nedostatak završnog čvora i sl.).
* Prikaz trenutnog rezultata igrača je dodat nakon svakog poteza (ukupan broj trouglova).

Mašina za zaključivanje

U našoj implementaciji mašinu za zaključivanje predstavljaju funkcije koje vrše procenu stanja na osnovu skupa pravila. Konkretno, funkcija evaluate\_state koristi više podpravila (count\_connected\_triangles, evaluate\_territory\_control, count\_possible\_triangles) koja analiziraju trenutno stanje igre (činjenične informacije iz matrice), i na osnovu toga izračunavaju heurističku vrednost. Svako od tih pravila donosi zaključak u vidu dodeljivanja poena, čime funkcioniše kao jednostavna inferentna logika.