**Super Software Team**

## *3D Connect 4*

Studenti:

Stefan Marinković 16683

Tamara Marjanović 16685

Stefan Stojanović 16929

I FAZA

**1.Main**

**main ( game ) –** Glavna funkcija programa, argument je funkcija **initialization-game** u kojoj se vrsi inicijalizacija pocetnog stanja igre. Nadalje se u ovoj funkciji izdvajaju atributi potrebni za odvijanje same igre, stampanje trenutnog stanja igre, promena stanja (novi potezi) kao i provera samog kraja igre.

**2.Inicijalizacija**

**Initialization-game() –** U ovoj funkciji se kreira pocetno stanje igre tako sto se zadaje velicina (4 ili 6), mod igre (covek protiv coveka ili covek protiv racunara), atribut **state** koji ce biti lista listi u kojima ce se smestati vrednosti u zavisnosti od poteza igraca.

***Pomocne funkcije:***

**Table-size() –** odredjuje velicinu table (4 ili 6).

**Game-mode()-**odredjujemod igre, covek protiv coveka ili racunar protiv coveka.

**First-player()-** u slucaju da igraju covek i racunar, definise se ko igra prvi**.**

**Initial ( k n ) –** kreira inicijalnu tablu koja je prazna .

**Player-name() –** unos imena igracima preko konzole.

**3.Promena stanja ( novi potez )**

**Make-move ( state xo k name ) –**glavna funkcija koja se poziva sve dok se ne dodje ko kraja igre. U ovoj funkciji se unosi potez kroz konzolu, vrsi validacija poteza kao i promena stanja na tabli.Regularan potez je naziv kolone u hexa brojnom sistemu.

Pomocne funkcije:

**Read-move ( k ) –** funkcija koja sluzi da ucita potez sa konzole i proveri da li je potez regularno unet.

**Regular ( clist ) –** vrsi se provera regularnosti tako sto se proverava da li je uneti broj hexa broj u opsegu [0-f]. Ako jeste vraca True.

**Validp ( state move k ) –** nakon provere hexa broja, proverava se da li je uneti hexa broj (koji je konvertovan u decimalni broj) validan kao decimalni, to jest da li ne prelazi zadati broj stapica.Takodje u ovoj funkciji se poziva funkcija **stick-fullp**koja vrsi proveruda li ima slobodnih mesta na**move**-tom stapicu.

**New-move ( state xo move ) –**funkcija koja se pozivanakon validacije u glavnoj funkciji vrsi promenu stanja na tabli tako sto u **move**-ti stapic umece zadati karakter **xo**ukoliko ima slobodnih mesta na stapicu.

**Insert() –**direktno umece zadati karakter **xo**na slobodno mesto na stapicu.

**4.Stampanje**

**Print-state( state k m n ) –**Glavna funkcija pokrece stampanje trenutnog stanja igre na konzoli tako sto ispisuje najpre naziv kolona, telo table i na kraju ponovo nazive kolona (nazivi kolona su hexa cifre).

**Print-table ( state k i j ) –** stampa telo table.

**Print-state-end( state k m ) -**nakon stampanja tela table, ova funkcija stampa nazive kolona.

**Reverse-stick ( state br n ) -** okrece redosled stapica kako bi se na adekvatan nacin prikazali na tabli.

**Insert-stick ( stick br n ) –**u zavisnosti od parametara racuna broj praznih mesta sa obe strane stapica kako bi dobili odgovarajuci ’3D prikaz’ na konzoli.

**Empty-stick ( l ) -**vraca listu praznih mesta koji se dodaju na stapic.

**5.Kraj igre**

**Final-statep ( state ) –** nakon svakog poteza proverava da li je kraj igre tako sto se krece kroz trenutno stanje na tabli i proverava da li postoji prazno mesto. Ukoliko postoji , igra moze da se nastavi.

**Stick-fullp ( stick ) –** pomocna funkcija koja se poziva u okviru funkcije **Final-statep**koja ispituje na nivou stapica da li postoji prazno mesto. Ukoliko ne postoji, igra se zavrsava.

II FAZA

1. **Funkcije za operatore promene stanja problema (igre) u opštem slučaju:**
   1. **Na osnovu trenutne (proizvoljne) situacije u kocki (stanja) i zadatog (validnog) poteza formira novusituaciju na kocki (stanje). Ne menjati postojeće stanje već napraviti novo i na njemu odigrati potez**

**new-move ( state xo move ) –** Funkcija vraća novo stanje koje se dobije kada se nalazimo u **tekućem stanju (state)** odigra se **potez (move)**, a na redu je **xo (x ili o).** Tj. funkcija u **move**-ti štapić umeće zadati karakter **xo**. (Ova funkcija je već opisana u 1. fazi)

* 1. **Na osnovu trenutne (proizvoljne) situacije u kocki (stanja) i igrača koji je na potezu formira listu svih mogućih situacija u kocki (stanja) korišćenjem funkcije iz prethodne tačke**

**generate-all-states ( state xo k move ) –** Funkcija generiše listu svih mogućih stanja u koja se moze preći iz **trenutnog stanja(state)**, ako je na potezu**xo**. Argumenti funkcije su i**z** - broj štapića i **move**koji se pri pozivu ove funkcije postavlja na 0. Funkcija rekurzivno poziva sebe i u svakom koraku povećava **move** za 1, sve dok ne dođe do **k**, proverava da li je validno dodati **xo** u **move**-ti štapić, ako jeste poziva **new-move(state xo move)** i to novo stanje ubacuje u listu koju vraća. Za validaciju koristi funkciju **validp.**

**validp ( state move k ) –** proverava se da li je uneti potez **move** validan kao decimalni i da li ne prelazi zadati broj štapića **k**. Takodje u ovoj funkciji se poziva funkcija **stick-fullp**koja vrši proveru da li je štapić popunjen. Ako su ispunjena ova dva uslova i ako štapić nije popunjen funkcija **validp** vraća T.(Ova funkcija je već opisana u 1. fazi)

**stick-fullp (stick)** – proverava da li **stick** ima bar jedno "-" i ako ima vraca null. (Ova funkcija je već opisana u 1. fazi)

1. **Realizovati funkcije koje obezbeđuju odigravanje partije između dva igrača (dva čoveka, ne računara i čoveka)**

**2.1 unos poteza i provera da li je potez moguć**

**2.2 ukoliko nije moguć zahtevati unos novog poteza**

**2.3 ukoliko je moguć odigrati ga i promeniti trenutno stanje**

**2.4 prikazati novonastalo stanje sistema**

**2.1-2.4** ove stavke su opisane u 1. fazi.

**2.5 proveru kraja i određivanje pobednika u igri**

**final-statep ( state ) –** nakon svakog poteza proverava da li je kraj igre tako što se kreće kroz trenutno stanje na tabli i proverava da li postoji prazno mesto. Ukoliko postoji , igra može da se nastavi.

**stick-fullp ( stick ) –** pomoćna funkcija koja se poziva u okviru funkcije **final-statep**i koja ispituje na nivou stapića da li postoji prazno mesto. Ukoliko ne postoji, igra se završava.

**winner (state player1 player2 n k end)**– funkcija računa poene koje je osvojio **x** i poene koje je osvojio **o**, zatim ukoliko je **end** true, uporedjuje poene i štampa ko je pobedio **player1** ili **player2**ili da je nerešeno. Za računanje poena koristi funkcije **horizontal, vertical, diagonal-2d** i **diagonal-3d.**

**max-4 (l xo a b c) –** funkcija računa broj spojenih nizova od 4 iste kuglice **xo (x ili o)** u listi **l**, argumenti **a, b i c** (pri pozivu funkcije su **NULL)** se postavljaju na **T** ako je odgovarajuće mesto u listi **l** bilo ispunjeno kuglicom **xo.** Funkcija detektuje spojen niz kada je **a=T, b=T, c=T** i prvi element liste **l** =**xo**. Ovu funkciju koriste funkcije **horizontal, vertical, diagonal-2d** i **diagonal-3d** tako što joj prosleđuju izdvojenu listu kuglica u odgovarajucem pravcu.

1. **vertical**

**vertical (state xo k m)** – broji poene u štapićima (vertikalni su), **state** je stanje tabele, **xo**kaže da li brojimo poene za x ili za o, **k** je broj štapića, a **m** (0 pri pozivu funkcije) trenutni štapić u kojem brojimo (brojimo poene uz pomoć **max-4**), u svakom rekurzivnom pozivu **m** povećavamo za 1 sve dok ne bude jednak **k.**

1. **horizontal**

**horizontal (state xo n m)** - broji horizontalne poene, **state** je stanje tabele, **xo**kaže da li brojimo poene za x ili za o, **n** je veličina tabele (matrice), a **m** (0 pri pozivu funkcije) kaže u kojoj matrici brojimo poene, krece od nulte do n-1. (**m**-tu matricu čine svi **m**-ti elementi u štapićima), u svakom rekurzivnom pozivu **m** povećavamo za 1 sve dok ne bude jednak **n.** Za računanje poena koristi funkciju **vrste-kolone**.

**vrste-kolone (state xo n m br)** – broji poene u **m**-toj matrici, **state** je stanje tabele, **xo**kaže da li brojimo poene za x ili za o, **n** je veličina tabele (matrice), a **br** (0 pri pozivu funkcije) kaže u kojojvrsti/koloni brojimo poene, krece od nulte do n-1. , u svakom rekurzivnom pozivu **br** povećavamo za 1 sve dok ne bude jednak **n.** Za računanje poena u vrsti koristi najpre funkciju **vrsta** za izdvajanje liste a zatim **max-4** za brojanje poena u toj listi, za izdvajanje liste koja predstavlja odgovarajuću kolonu koristi funkciju **kolona**.

**vrsta (state n vr br m)** – izdvaja listu koja predstavlja vrstu **vr** unutar **m**-te matrice u stanju **state**čija je veličina **n**, **n\*vr+br**( **br** =0 pri pozivu funkcije, pa se u svakom rekurzivnom pozivu povećava za 1 sve dok ne bude jednak **n**) je trenutni element vrste i njega ubacujemo u rezultujuću listu.

**kolona (state n kol br m)** – izdvaja listu koja predstavlja kolonu **kol** unutar **m**-te matrice u stanju **state**čija je veličina **n**, **kol+br\*n** ( **br** =0 pri pozivu funkcije, pa se u svakom rekurzivnom pozivu povećava za 1 sve dok ne bude jednak **n**) je trenutni element kolone i njega ubacujemo u rezultujuću listu.

1. **diagonal-2d**

**diagonal-2d (state xo n k)** – funkcija koja sluzi da sabere broj poena za svakog igraca po dijagonalama u ravnima I to u tri slucaja. Matrice se izdvajaju tako da se sabiraju poeni iz matrica koje se kreiraju od donjih krajeva stapica pa sve do gornjih , zatim matrica od n levih stapica pa sve do desnih krajnjih I konacno matrica koje se kreiraju od stapica sa zdnje strane pa sve do onih prednjih.

**Diagonal-2dGore\_Dole (state xo n k)** – funkcija koja izdvaja matrice pocevsi od donjih krajeva stapica pa sve do gornjih I za takve matrice racuna broj poena.

**Napravi\_state\_za\_GoreDole (state n element k)** – rekombinuje state listu listi, tako da prilagodi I vrati listu listi koja odgovora izdvajanju elemenata za zadati slucaj opisan u prethodnoj funkciji.

**Napravi\_matricu\_za\_GoreDole(state n element)**–vraca listu listi, gde su te liste pozivi funkcija **napravi\_stapic\_prvi**I na taj nacin dobijamo zeljenu matricu rekombinovanih elemenata.

**Napravi\_stapic\_prvi (state n element)**–kreira jednu listu koja ide kroz n stapica I izdvaja elemente koji su zadati pozicijom element.

**Diagonal-2dLevo\_Desno (state xo n k)** -funkcija koja izdvaja matrice pocevsi od n levih stapica pa sve do n krajnjih desnih I za takve matrice racuna broj poena.

**Napravi\_state\_za\_LevoDesno (state n k br)**–kreira rekombinovanu listu state gde ce svaki n-ti stapic biti jedan pored drugog.

**Izdvoji\_svaki\_nti (state n element k)**–vraca listu gde je svaki n-ti stapic spojen jedan do drugog.

**Diagonal-2dNapred\_Nazad (state xo n k)** - funkcija koja izdvaja matrice pocevsi od n zadnjih stapica pa sve do n prednjih I za takve matrice racuna broj poena.

**Napred (state xo n k)**– funkcija koja racuna poene po dijagonalama iz matrice state.

**Skini\_n\_susednih (state n)** – vraca listu state bez n prvih elemenata.

**Iznad\_dijagonale (state xo n element)** –– pozivom funkcije **max-4** racuna I sabira poene koji se u matrici nalaze po dijagonalama iznad glavne dijagonale.

**Dijagonala\_ispod (state xo n)** – pozivom funkcije **max-4** racuna I sabira poene koji se u matrici nalaze po glavnoj dijagonali I dijagonalama ispod nje.

**Napravi\_stapic (state n element) –** pravi listu po dijagonali iz n susednih stapica.

1. **diagonal-3d**

**diagonal-3d (state xo n)**– sabira poene koje su funkcije**diagonal1** i **diagonal2** izračunale, **diagonal1**računa poene u glavnim 3d dijagonalama, a **diagonal2** računa poene u sporednim 3d dijagonalama, **state** je stanje tabele, **xo**kaže da li brojimo poene za x ili za o,a **n** je veličina tabele.

**diagonal1 (state n i xo)**– sabira poene u svim glavnim 3d dijagonalama, **i** (0 pri pozivu funkcije) se u svakom rekurzivnom pozivu povećava za 1 sve dok ne bude veći od **n-4,** uz pomoć **i** regulišemo koje dijagonale gledamo (nema potrebe da posmatramo dijagonale koje su veličine 3). Poziva funkciju **diagonal-13** za svako **i** i **n\*i** (sem za i=0, tad poziva samo jednom), gde **i** i **n\*i** predstavljaju startni broj štapića.

**diagonal13 (state n i br1 m xo)** – sabira poene u svim glavnim 3d dijagonalama počevši od**i**-tog štapića. Poziva funkciju **diagonal-11** za pronalaženje dijagonale koja ide od dna ka vrhu i funkciju **diagonal-12** za pronalaženje dijagonale koja ide od vrha ka dnu i nad tim listama poziva funkciju **max-4**kako biizračunala broj poena. Argument **m** ide od **(n-4)** do **-(n-4)** i on reguliše startni element u štapiću **i**, a argument **br1** reguliše dužinu dijagonale.

**diagonal11 (state n i br br1 m)** – pronalazi glavnu 3d dijagonalu koja ide od dna ka vrhu počevši od **m**-tog elementa u štapiću pod rednim brojem **i + n\*br + br** , gde se **br** (0 pri pozivu funkcije) u svakom rekurzivnom pozivu povećava za 1 sve dok ne bude jednak **br1** i na taj način računamo odgovarajući dijagonalni štapić, u svakom rekurzivnom pozivu povećavamo i **m** za 1 čime dobijamo 3d dijagonalu a ne 2d dijagonalu.

**diagonal12 (state n i br br1 m)** – pronalazi glavnu 3d dijagonalu koja ide od vrha ka dnu počevši od **m**-tog elementa u štapiću pod rednim brojem **i + n\*br + br** , gde se **br** (0 pri pozivu funkcije) u svakom rekurzivnom pozivu povećava za 1 sve dok ne bude jednak **br1** i na taj način računamo odgovarajući dijagonalni štapić, u svakom rekurzivnom pozivu smanjujemo **m** za 1 čime dobijamo 3d dijagonalu a ne 2d dijagonalu.

**diagonal2 (state n i xo)**– sabira poene u svim sporednim 3d dijagonalama, **i** (n-1 pri pozivu funkcije) se u svakom rekurzivnom pozivu smanjuje za 1 sve dok ne bude manji od **n-1-(n-4),** uz pomoć **i** regulišemo koje dijagonale gledamo (nema potrebe da posmatramo dijagonale koje su veličine 3). Poziva funkciju **diagonal-23** za svako **i** i **n\*(n-i)-1**(sem za i=n-1, tad poziva samo jednom), gde **i** i **n\*(n-i)-1**predstavljaju startni broj štapića.

**diagonal23 (state n i br1 m xo)** – sabira poene u svim sporednim 3d dijagonalama počevši od**i**-tog štapića. Poziva funkciju **diagonal-21** za pronalaženje dijagonale koja ide od dna ka vrhu i funkciju **diagonal-22** za pronalaženje dijagonale koja ide od vrha ka dnu i nad tim listama poziva funkciju **max-4**kako biizračunala broj poena. Argument **m** ide od **(n-4)** do **-(n-4)** i on reguliše startni element u štapiću **i**, a argument **br1** reguliše dužinu dijagonale.

**diagonal21 (state n i br br1 m)** – pronalazi sporednu 3d dijagonalu koja ide od dna ka vrhu počevši od **m**-tog elementa u štapiću pod rednim brojem **i + n\*br -br**, gde se **br** (0 pri pozivu funkcije) u svakom rekurzivnom pozivu povećava za 1 sve dok ne bude jednak **br1** i na taj način računamo odgovarajući dijagonalni štapić, u svakom rekurzivnom pozivu povećavamo i **m** za 1 čime dobijamo 3d dijagonalu a ne 2d dijagonalu.

**diagonal22 (state n i br br1 m)** – pronalazi sporednu 3d dijagonalu koja ide od vrha ka dnu počevši od **m**-tog elementa u štapiću pod rednim brojem **i + n\*br -br** , gde se **br** (0 pri pozivu funkcije) u svakom rekurzivnom pozivu povećava za 1 sve dok ne bude jednak **br1** i na taj način računamo odgovarajući dijagonalni štapić, u svakom rekurzivnom pozivu smanjujemo **m** za 1 čime dobijamo 3d dijagonalu a ne 2d dijagonalu.

III FAZA

**min-max (state n depth alfa beta minmax player hplayer)** – Ova funkcija implementira Min-Max algoritam do dinamički odredjene dubine pomoću funkcije **depth** sa alfa-beta odsecanjem. Parametri su: **state** (trenutno stanje u nekom čvoru), **n**(veličina kocke), **depth** (dubina stable pretrage), **alfa** (koeficijent odsecanja, početna vrednost je **-999** I mora da bude oblika kao I heuristika ***'(() value)*** ), **beta** (koeficijent odsecanja, početna vrednost je **999** I mora da bude oblika kao i heuristika ***'(() value)*** ), **minmax** (odredjuje da li se na trenutnom nivou traži minimum ili maksimum heuristike potomaka, **1** je za max, **0** za min), **player** (igrač koji je na potezu na trenutnom nivou), **hplayer** (igrač za kojeg se odredjuje heuristika u terminalnim čvorovima). Ako je trenutno stanje na terminalnom nivou ili je finalno stanje poziva se funkcija **heuristika** za računanje heuristike, u suprotnom u zavisnosti od vrednosti **minmax** postavlja se početna vrednost **value**, generišu se svi potomci trenutnog stanja (pomocu funkcije **generate-all-states** opisanoj u 2. fazi), I poziva se funkcija **min-max2** za sve potomke.

**min-max2 (list-states n depth alfa beta minmax player value hplayer)** – Ova funkcija poziva min-max algoritam za svako stanje iz lista stanja **list-states** I implementira alfa-beta odsecanje. Parametar **value** prestavlja prosledjenu vrednost heuristike od roditelja, a ostali parametri su isti kao kod funkcije min-max. Ovde pozivamo I funkcije **greater-or-smaller** i **switch-player**.

**heuristika (state n k xo)** – računa heuristiku kao razliku poena izmedju dva igrača u zavisnosti od parametra **xo**.

**greater-or-smaller (op h1 h2)** – uporedjuje heuristike u zavisnosti od parametra **op**. Ostavlja stanje **h1** (nulti član liste **h1**) i samo može da joj promeni vrednost heuristike (prvi član liste **h1**).

**switch-player (player)** – menja igrača, ako je **playerx**, vraća **o**i obrnuto.

**depth (state n num)** – odredjuje dinamički dubinu pretrage u zavisnosti od **n** i procenta popunjenosti kocke.

IV FAZA

**=add,=add3,=add4,=add5** – Funkcije koje se koriste u pravilima zaključivanja, prosleđuje im se 2,3,4 ili 5 parametara i računa se zbir tih parametara

**!eq,!ne**- Predikati koji se koriste u pravilima zaključivanja, vraćaju da li su dva parametra jednaka ili različita

**\*T1-RULES\* -** Definisana pravila na osnovu kojih mašina za zaključivanje utvrđuje postojanje 4 uzastopna ‘x ili ‘o u svim mogućim pravcima i smerovima. Ukoliko se određuje broj poena za ‘x izvode se činjenice bez M, a ukoliko je ‘o onda se generišu činjenice sa M u svom nazivu na kraju.

**stick-to-facts (stick i j)-** Funkcija koja na osnovu jednog štapića generiše činjenice za bazu znanja. Činjenica **('on "x" i j)** kaže da se u štapiću **i**, na mestu **j** nalazi **x**.

**state-to-facts (state n i)-** Funkcija koja stanje razdvaja u štapiće, štapiće prosleđuje funkciji **stick-to-facts** i generisane činjenice spaja u jednu listu. Takodje, na osnovu veličine kocke u listu dodaje činjenicu **(Veličina n)** koja se koristi u pravilima zaključivanja, i činjenice **(kraj i)** gde **i** predstavlja granične vrednosti štapića u kocki i zavisi od **n**. Činjenice **(kraj i)** se koriste za odbacivanje nevalidnih zaključaka o 4 uzastopna ista znaka u određenom pravcu i smeru (2d dijagonale, 3d dijagonale, horizontale)

**heuristika (state n k xo)-** Predefinisana funkcija iz 3. faze koja se poziva kada min-max algoritam dostigne maksimalnu dubinu i koristi se za procenu stanja. Definišu se parametri **\*T1-FACTS\*** u kome se smešta polazna baza činjenica, **knowledge** u kome se smešta rezultat poziva funkcija **prepare-knowledge**, i onda se računa heuristika za x i o igrača na sledeći način:

**Vertikala(M)** - (broj izvedenih zaključaka \*1 poena)

**Horizontala2(M)** - (broj izvedenih zaključaka \*6 poena)

**Horizontala1(M)** - (broj izvedenih zaključaka \*11 poena)

**Dijagonala3(M)**, **Dijagonala4(M)** - (broj izvedenih zaključaka \*16 poena)

**Dijagonala5(M)**, **Dijagonala6(M)** - (broj izvedenih zaključaka \*21 poena)

**Dijagonala1(M),Dijagonala2(M)** - (broj izvedenih zaključaka \*26 poena)

**Dijagonala7(M), Dijagonala8(M), Dijagonala9(M), Dijagonala10(M)** - (broj izvedenih zaključaka \*31 poena)

Broj poena se dobija pozivom funkcije **count-results** za svaku od iznad navedenih činjenica, množenjem tog rezultata odredjenim brojem (1,6,11,16,21,26,31) i sabiranjem svih tih rezultata. Heuristika se dobija oduzimanjem poena **(x-o**) ili **(o-x)** u zavisnosti od toga da li je kompjuter **x** ili **o** igrac.