Rješavanje problema igre Poveži četiri primjenom umjetne inteligencije

Šimić, Leon

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

RJEŠAVANJE PROBLEMA IGRE POVEŽI ČETIRI PRIMJENOM UMJETNE INTELIGENCIJE

Završni rad

Leon Šimić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 29.08.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Leon Šimić
Studij, smjer:	Računalno inženjerstvo
Mat. br. Pristupnika, godina upisa:	R 4431, 13.10.2021.
OIB Pristupnika:	74640830814
Mentor:	izv. prof. dr. sc. Časlav Livada
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Rješavanje problema igre Poveži četiri primjenom umjetne inteligencije
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak završnog rad:	U radu je potrebno primjenom OpenGL biblioteke napraviti igru Poveži četiri koja omogućuje igranje protiv računala, ali i igranje dva računalno kontrolirana lika međusobno. Potrebno je omogućiti odabir različitih vrsta algoritama rješavanja kako bi se utvrdio optimalni algoritam. Tema rezervirana za: Leon Šimić
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	29.08.2023.
Datum potvrde ocjene od strane Odbora:	08.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 09.09.2023.

Ime i prezime studenta:	Leon Šimić
Studij:	Računalno inženjerstvo
Mat. br. studenta, godina upisa:	R 4431, 13.10.2021.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Rješavanje problema igre Poveži četiri primjenom umjetne inteligencije**

izrađen pod vodstvom mentora izv. prof. dr. sc. Časlav Livada

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Poveži četiri (engl. *Connect four*) je popularna misaona društvena igra koja je namijenjena za dva igrača. Sastoji se od ploče koja ima 6 redaka i 7 stupaca te od žetona koji dolaze u dvije boje. Cilj ove igre je prvi spojiti četiri iste boje žetona u okomitom, horizontalnom ili dijagonalnom smjeru. Igra se tako da igrači naizmjenično ubacuju svoje žetone iste boje u željene stupce te tako pokušavaju doći do pobjede, a ako se ploča potpuno popuni bez niza od 4 žetona iste boje, igra završava neriješeno.

U ovome će radu biti izložena igra napisana u programskom jeziku C#, koja koristi biblioteku *Open Graphics Library* (OpenGL). Ta biblioteka služi za vizualni prikaz i omogućuje interakciju s korisnikom kao što je klikanje mišem. Razvojno okruženje u kojem je pisana je Microsoftov *Visual Studio*.

Umjetna inteligencija (engl. *Artificial Intelligence* - AI) je široko područje računalne znanosti koje proučava i stvara inteligentne programe koji pokazuju ponašanje koje je slično čovjeku, prema [1]. Kako se s vremenom sve više razvija, AI je postala sveprisutna u raznim granama i područjima, uključujući i računalne igre. AI će u ovom radu predstavljati različite algoritme čija će implementacija biti prikazana i objašnjena.

Implementiranim algoritmima će se prvo mjeriti vrijeme potrebno za odigravanje poteza te će se oni tada međusobno natjecati. Pomoću ovih rezultata, analizirat će se međusobne brzine izvođenja svih algoritama i pobjednici igara te će se, naposljetku, odabrati optimalan algoritam, koji treba imati ravnotežu između dobrih poteza i brzine izvršavanja.

1.1. Zadatak završnog rada

Cilj ovog rada je primjenom OpenGL biblioteke napraviti igru *Poveži četiri* koja omogućuje igranje dva računalno kontrolirana lika međusobno i igranje protiv računala. Također, implementirati različite algoritme te usporediti njihove prednosti i mane. Konačno, na temelju usporedbi izabrati optimalni algoritam.

2. PREGLED PODRUČJA TEME

S obzirom na to da je *Poveži četiri* vrlo popularna i relativno stara (izdana je 1970-ih godina [2]) igra, na internetu je moguće pronaći mnogo različitih programa i stranica na kojima ju je moguće igrati. Na stranici *Boardgames.io* [3], koju tražilica *Google* prvu prikaže, moguće je igrati samo protiv drugih aktivnih igrača. Zatim, stranica *CBC Kids* [4] nudi samo igranje protiv 2. igrača kada se on priključi linkom. Najviše mogućnosti daje stranica *papergames.io* [5] gdje je moguće igrati protiv nasumičnih igrača, u dvoje i protiv računala. Vizualni prikaz najsličniji igri u ovom radu ima igra koja je dostupna na stranici *GitHub* [6], gdje je dostupan izvorni kod igre, napisan u programskom jeziku Python, čija implementacija podržava verziju igre s 2 igrača i verziju u kojoj korisnik igra protiv računala. U radu [7] je za razvoj umjetne inteligencije korišten isti algoritam kao i u ovom radu, pisan u Arduino integriranom razvojnom okruženju. Ova igra s više razina težine igranja napravljena je u radu [8], također u programskom jeziku C#, ali s jednostavnijim algoritmima.

3. ANALIZA ALGORITAMA

Kako bi se izradila umjetna inteligencija koja igra igru *Poveži četiri*, implementirani su različiti algoritmi koji omogućuju donošenje dobrih poteza. Svi ti algoritmi imaju neka zajednička obilježja i nekoliko ključnih elemenata koje koriste.

Jedan od tih ključnih elemenata je heuristička funkcija. Općenito govoreći, heuristika u programiranju je tehnika koja pokušava doći do optimalnog rješenja, prema [9]. U ovom slučaju, heuristička funkcija procjenjuje vrijednosti, tj. bodove trenutnog stanja igre analizirajući trenutni raspored žetona na ploči i postavljanje žetona u određene stupce. Kako se dodjeljuju bodovi od iznimne je važnosti, prema [10], tj. koliko bodova nosi određena pozicija žetona, zato što se bodovi koriste za usporedbu različitih poteza. Ova je heuristička funkcija, *GetScore*, pisana po uzoru na [6], a njen pseudokod prikazan je na slici 3.1.

```
Funkcija GetScore(žetoni, trenutniIgrač):
        ocjena = 0
        veličinaProzora = 4
        centar = svi žetoni u 4. stupcu
        veličinaCentra = broj žetona u centru koje je odigrao trenutniIgrač
        ocjena = ocjena + veličinaCentra * vrijednost
        ocjena = ocjena + GetScoreHorizontal(žetoni, veličinaProzora, trenutniIgrač)
        ocjena = ocjena + GetScoreVertical(žetoni, veličinaProzora, trenutniIgrač)
        ocjena = ocjena + GetScorePrimaryDiagonal(žetoni, veličinaProzora, trenutniIgrač)
        ocjena = ocjena + GetScoreSecondaryDiagonal(žetoni, veličinaProzora, trenutniIgrač)

        Vrati ocjena
```

Slika 3.1. Heuristička funkcija.

Uz heurističku funkciju, element pomoćne funkcije također ima važnu ulogu. Pomoćna funkcija, kao što je u kodu ove igre funkcija *EvaluateWindow* (Slika 3.2.), također napisana po uzoru na [6], služi za procjenu vrijednosti određenih dijelova ploče, tj. prozora veličine 4 unutar kojih se nalaze žetoni. Na temelju ove funkcije, algoritmi imaju više znanja i time donose bolje odluke te odigravaju bolje poteze. Pomoćna se funkcija poziva u svakoj funkciji koju poziva funkcija *GetScore*.

```
170     private int EvaluateWindow(List<int> window, int player)
171     {
172         int score = 0;
173 
174         int opponentPlayer = (int)PlayerType.PlayerA;
175         if (player == (int)PlayerType.PlayerA)
176             opponentPlayer = (int)PlayerType.PlayerB;
177 
178         if (window.Count(x => x == player) == 4)
179             score += 100;
180         else if (window.Count(x => x == player) == 3 && window.Count(x => x == 0) == 1)
181             score += 10;
182         else if (window.Count(x => x == player) == 2 && window.Count(x => x == 0) == 2)
183             score += 5;
184         if (window.Count(x => x == opponentPlayer) == 3 && window.Count(x => x == 0) == 1)
185             score -= 80;
186 
187         return score;
188     }
```

Slika 3.2. Pomoćna funkcija.

Kombinirajući heurističku i pomoćnu funkciju, algoritmi dobivaju jasnu sliku o trenutnom stanju igre, što im pomaže da bolje procjenjuju moguće poteze i odaberu najbolji mogući. Iz tog razloga, ovi su elementi ključni te pružaju temelj za spomenute algoritme.

U nastavku poglavlja, algoritmi će se analizirati i usporediti te će se predstaviti njihove prednosti i nedostatci te uspješnost u pronalaženju optimalnih poteza.

3.1. Osnovni algoritam

Najjednostavniji od svih implementiranih, osnovni algoritam temelji se na evaluaciji svih valjanih poteza koristeći heurističku funkciju. Ovaj algoritam, implementiran kao u [6], prvo prolazi kroz sve nepopunjene stupce, a zatim postavlja privremene žetone na ploču za svaki mogući potez. Tada koristi heurističku funkciju kako bi dobio vrijednost trenutnog stanja ploče nakon svakog poteza. Na kraju, uspoređuje sve vrijednosti i odabire stupac s najvišom vrijednosti, jer ona predstavlja najbolji potez. Funkcija koja vraća najbolji stupac dana je slikom 3.3.

Ovaj osnovni algoritam predstavlja samo osnovu na koju se nadograđuju složeniji algoritmi, koji su detaljnije opisani u nastavku rada, te je time ostvarena umjetna inteligencija koja odigrava bolje poteze.

```
17      protected int PickBestColumn(Board board, Coin coin)
18      {
19          Random rand = new Random();
20          List<int> validLocations = board.GetValidLocations();
21          int bestScore = -1000;
22          int bestColumn = validLocations[rand.Next(validLocations.Count)];
23
24          foreach (var column in validLocations)
25          {
26              int row = board.GetNextOpenRow(column);
27              Coin[,] tempCoins = Coin.GetCopyOfArray(board.GetCoins());
28              tempCoins[row, column] = coin;
29
30              int score = GetScore(tempCoins, (int)coin.Type);
31              if (score > bestScore)
32              {
33                  bestScore = score;
34                  bestColumn = column;
35              }
36          }
37
38          return bestColumn;
39      }
```

Slika 3.3. Funkcija osnovnog algoritma koja vraća najbolji stupac.

3.2. Minimax

Jedan od najpopularnijih algoritama za rješavanje igre *Poveži četiri* je minimax algoritam. Ovaj se algoritam temelji na rekurzivnoj pretrazi stabla igre kako bi pronašao najbolji potez za igrača. Algoritam radi tako što izvršava evaluaciju svih valjanih poteza kroz rekurzivne pozive. Za svaki nepopunjeni stupac, simulira potez postavljanjem privremenog žetona na ploču te se nakon toga opet poziva kako bi rekurzivno nastavio izvršavanje.

Minimax algoritam koristi funkciju *DoMiniMax* koja prima 3 argumenta: trenutno stanje ploče, dubinu pretrage stabla i vrijednost koja označuje je li trenutni igrač maksimizirajući ili minimizirajući. Na početku provjerava dubinu pretrage i trenutno stanje ploče, pomoću čega određuje je li došlo do kraja rekurzije. U slučaju da je, vraća vrijednost stanja ploče, tj. ocjenu ili odgovarajuće vrijednosti, ovisno koji je igrač pobijedio (Slika 3.4.). Zatim, ako je trenutni igrač maksimizirajući (Slika 3.5.), prolazi kroz sve nepopunjene stupce, simulira odigravanje poteza te rekurzivno poziva sebe, ali prije toga smanjuje dubinu pretrage za 1 i zamjenjuje igrača. Kada se vrati iz rekurzivnog poziva, algoritam provjerava povratnu vrijednost te ju ažurira zajedno s najboljim stupcem ako je pronađen bolji potez. U slučaju da je trenutni igrač minimizirajući, izvršava se sličan kod (Slika 3.6.). Na kraju se vraća najbolji stupac i najbolja ocjena. Ovaj je algoritam pisan po uzoru na [6]. Funkcija koju koristi poziva se na način koji je prikazan na slici 3.7.

```
14          List<int> validLocations = board.GetValidLocations();
15          int isTerminalNode = IsTerminalNode(board);
16
17          if (depth == 0 || isTerminalNode != 0)
18          {
19              if (isTerminalNode != 0)
20              {
21                  if (isTerminalNode == 2)
22                      return (0, 100000);
23                  else if (isTerminalNode == 1)
24                      return (0, -100000);
25                  else
26                      return (-1, 0);
27              }
28              else
29                  return (-1, GetScore(board.GetCoins(), (int)PlayerType.PlayerB));
30          }
```

Slika 3.4. Početni dio funkcije *DoMiniMax*.

```
32          if (maximizingPlayer)
33          {
34              double value = double.NegativeInfinity;
35              int bestColumn = validLocations[rand.Next(validLocations.Count)];
36
37              foreach (var column in validLocations)
38              {
39                  int row = board.GetNextOpenRow(column);
40                  Board boardCopy = board.GetCopy();
41                  boardCopy.PopulateAt(row, column, new Coin(PlayerType.PlayerB));
42
43                  var columnAndScore = DoMiniMax(boardCopy, depth - 1, false);
44                  double score = columnAndScore.Item2;
45
46                  if (score > value)
47                  {
48                      value = score;
49                      bestColumn = column;
50                  }
51              }
52
53              return (bestColumn, value);
54          }
```

Slika 3.5. Provjera je li trenutni igrač maksimizirajući u funkciji *DoMiniMax*.

```
55              else //Minimizing player
56              {
57                  double value = double.PositiveInfinity;
58                  int bestColumn = validLocations[rand.Next(validLocations.Count)];
59
60                  foreach (var column in validLocations)
61                  {
62                      int row = board.GetNextOpenRow(column);
63                      Board boardCopy = board.GetCopy();
64                      boardCopy.PopulateAt(row, column, new Coin(PlayerType.PlayerA));
65
66                      var columnAndScore = DoMiniMax(boardCopy, depth - 1, true);
67                      double score = columnAndScore.Item2;
68
69                      if (score < value)
70                      {
71                          value = score;
72                          bestColumn = column;
73                      }
74                  }
75
76                  return (bestColumn, value);
77              }
78          }
```

Slika 3.6. Provjera je li trenutni igrač minimizirajući u funkciji *DoMiniMax*.

```
(int bestColumn, double score) = DoMiniMax(board, depth, true);
```

Slika 3.7. Pozivanje funkcije *DoMiniMax*.

Bitno je istaknuti da dubina pretrage stabla igre ima važnu ulogu u minimax algoritmu. Ona određuje koliko duboko algoritam rekurzivno pretražuje stablo igre prije nego što donose odluku o najboljem potezu, prema [11]. Odabir prikladne dubine pretrage je važan faktor u postizanju ravnoteže između brzine izvršavanja algoritma i optimalnog poteza.

Dubina pretrage uvelike utječe na vremensku složenost algoritma. Veća dubina pretrage dopušta algoritmu da izvrši bolju i dužu analizu stanja igre, ali je zato algoritmu potrebno više vremena za izvršavanje. S druge strane, ako algoritam prima manju dubinu pretrage, brže će se izvesti, ali zbog toga može vratiti lošiji potez.

3.3. Minimax s alpha-beta rezanjem

Algoritam minimax s alpha-beta rezanjem (engl. *pruning*) predstavlja poboljšanje minimax algoritma i koristi se za učinkovitije pretraživanje stabla igre kako bi se pronašao najbolji potez.

U odnosu na minimax algoritam, ovaj algoritam uvodi dva dodatna argumenta: alpha i beta. Pomoću njih se odbacuju („režu“) nepotrebne grane stabla igre, što rezultira smanjenjem broja

evaluacija stanja igre i značajno bržim donošenjem odluka. Funkcija *DoMiniMaxPruning* koju koristi ovaj algoritam ima vrlo sličnu strukturu kao funkcija *DoMiniMax*. Jedina je razlika postojanje već spomenutih parametara alpha i beta, koji se koriste u kodu samo na dva mjesta (Slika 3.8. i Slika 3.9.). Poziv funkcije može se vidjeti na slici 3.10. Algoritam je implementiran po uzoru na [6].

```
52                  alpha = Math.Max(alpha, value);
53                  if (alpha >= beta)
54                      break;
55              }
```

Slika 3.8. Dio funkcije *DoMiniMaxPruning* u kojoj se koristi alpha.

```
79                  beta = Math.Min(beta, value);
80                  if (alpha >= beta)
81                      break;
82              }
```

Slika 3.9. Dio funkcije *DoMiniMaxPruning* u kojoj se koristi beta.

```
93          (int bestColumn, double score) = DoMiniMaxPruning(
94              board, depth, double.NegativeInfinity, double.PositiveInfinity, true);
```

Slika 3.10. Poziv funkcije *DoMiniMaxPruning*.

Odbacivanje nepotrebnih grana koje izvodi alpha-beta rezanje omogućuje ubrzanje izvođenja algoritma bez značajnog gubitka preciznosti u odabiru najboljeg poteza.

3.4. Negamax

Algoritam negamax predstavlja varijaciju minimax algoritma i koristi se jer je jednostavniji za implementaciju. Iz tog je razloga često korišten kao zamjena za minimax.

U odnosu na algoritam minimax, negamax koristi drugačiji koncept koji se temelji na negiranju. Njegova se implementacija temelji na pseudokodu iz [12]. Umjesto podjele igrača na maksimizirajućeg i minimizirajućeg, algoritam negamax koristi samo jednog igrača te negira vrijednosti ocjena za suprotnog igrača. Ovim se postiže jednostavniji kod s manje linija, zato što

se ovako maksimizacija i minimizacija obavljaju na isti način. Funkcija *DoNegaMax* koju koristi ovaj algoritam ima sličnu strukturu kao funkcija *DoMiniMax*. Razlika je u tome što ovaj algoritam, umjesto argumenta koji predstavlja igrača, koristi argument koji označava „boju“ igrača, koja može biti +1 ili -1. Na taj način, negamax omogućuje efikasno pretraživanje stabla igre, a samim time i pronalazak najboljeg poteza.

Početni je dio algoritma sličan kao početak minimax algoritma, a razlika je u tome što negamax ne vraća samo ocjenu, nego ju množi s vrijednosti boje igrača, kao što je vidljivo na slici 3.11.

```
14          List<int> validLocations = board.GetValidLocations();
15          int isTerminalNode = IsTerminalNode(board);
16
17          if (depth == 0 || isTerminalNode != 0)
18          {
19              if (isTerminalNode != 0)
20              {
21                  if (isTerminalNode == 2)
22                      return (0, 100000 * color);
23                  else if (isTerminalNode == 1)
24                      return (0, -100000 * color);
25                  else
26                      return (-1, 0);
27              }
28              else
29                  return (-1, color * GetScore(board.GetCoins(), (int)PlayerType.PlayerB));
30          }
```

Sika 3.11. Početni dio funkcije *DoNegaMax*.

Tijekom pretrage, algoritam prolazi kroz sve nepopunjene stupce, simulira odigravanje poteza određenog igrača i rekurzivno se poziva, nakon što promijeni vrijednost boje. Nakon toga, ocjene dobivene iz rekurzivnih poziva negiraju se. Ostatak koda za ovaj algoritam vidi se na slici 3.12., a poziv funkcije na slici 3.13.

```
32              double bestValue = double.NegativeInfinity;
33              int bestColumn = validLocations[rand.Next(validLocations.Count)];
34
35              foreach (var column in validLocations)
36              {
37                  int row = board.GetNextOpenRow(column);
38                  Board boardCopy = board.GetCopy();
39                  boardCopy.PopulateAt(row, column, new Coin(color == 1 ? PlayerType.PlayerB : PlayerType.PlayerA));
40
41                  var columnAndScore = DoNegaMax(boardCopy, depth - 1, -color);
42                  double score = -columnAndScore.Item2;
43
44                  if (score > bestValue)
45                  {
46                      bestValue = score;
47                      bestColumn = column;
48                  }
49              }
```

Slika 3.12. Ostatak funkcije *DoNegaMax*.

```
(int bestColumn, double score) = DoNegaMax(board, depth, 1);
```

Slika 3.13. Pozivanje funkcije *DoNegaMax*.

3.5. Negamax s alpha-beta rezanjem

Algoritam negamax s alpha-beta rezanjem je poboljšana verzija negamax algoritma i koristi se za efikasnije pretraživanje stabla igre. Ovaj je algoritam vrlo sličan negamax algoritmu, ali uvodi koncept alpha-beta rezanja, koji je već objašnjen kod algoritma minimax s alpha-beta rezanjem. Ovaj algoritam koristi isto načelo negiranja kao i negamax te također zamjenjuje dva igrača jednim. Dodani kod u ovom algoritmu, u odnosu na negamax, isti je kao i kod funkcije *DoMiniMaxPruning* (Slika 3.8.), a poziv funkcije se može vidjeti na slici 3.14.

```
(int bestColumn, double score) = DoNegaMaxPruning(
    board, depth, double.NegativeInfinity, double.PositiveInfinity, 1);
```

Slika 3.14. Pozivanje funkcije *DoNegaMaxPruning*.

4. ANALIZA REZULTATA ALGORITAMA

U ovom poglavlju izložit će se i analizirati vrijeme, koje je bilo potrebno da se algoritmi izvrše, te rezultati igara gdje su algoritmi međusobno igrali jedan protiv drugog. Mjerenja su izvršena više puta pri različitim dubinama pretrage stabla igre kako bi se odredila optimalna vrijednost i algoritam. Rezultati su prikazani u obliku tablica.

Potrebno je istaknuti da su algoritmi negamax i minimax dali identične rezultate tijekom igara, tj. oba algoritma su odigrala potpuno iste poteze, što pokazuje da su ekvivalentni u tom smislu. Isto tako, algoritmi negamax s alpha-beta rezanjem i minimax s alpha-beta rezanjem su odigrali iste poteze.

Iako su ti algoritmi dali iste rezultate u smislu odigranih poteza, nisu jednako vremenski učinkoviti. Te vremenske razlike bit će uspoređene i analizirane u nastavku.

4.1. Vremenska analiza algoritama

Vrlo važno svojstvo algoritma je vrijeme koje mu je potrebno da se izvrši. Iz tog je razloga izvršena vremenska analiza svih ranije spomenutih algoritama. Izvedena je tako što su se mjerila vremena potrebna da algoritam odigra potez za prvih 8 poteza, a pri tome su vrijednosti dubina pretrage stabla igre postavljane od 2 do 9. Sve vrijednosti vremena prikazane su u sekundama. Budući da osnovni algoritam ne ovisi o dubini pretrage stabla, mjerenja su izvršena samo jednom. Ona se mogu vidjeti u tablici 4.1. Kao što je vidljivo, sva vremena su iznimno mala, što je očekivano iz razloga što algoritam nema mnogobrojne rekurzivne pozive kao što imaju ostali algoritmi.

Tablica 4.1. Izmjerena vremena osnovnog algoritma.

Potez	Vrijeme [s]
1.	0,0073
2.	0,0006
3.	0,0004
4.	0,0005
5.	0,0006
6.	0,0004
7.	0,0004
8.	0,0003

U nastavku slijede tablice 4.2. ─ 4.9. u kojima su prikazana mjerenja svih ostalih algoritama za različite vrijednosti dubine pretrage stabla. U njima zadnji stupac predstavlja zbroj svih 8 poteza radi lakše usporedbe algoritama.

Tablica 4.2. Izmjerena vremena algoritama za dubinu pretrage 2.

Dubina = 2

Potez	Minimax	MinimaxRezanje	Negamax	NegamaxRezanje
1.	0,0143	0,0238	0,0315	0,0227
2.	0,0082	0,0063	0,0138	0,0062
3.	0,0058	0,0049	0,0123	0,0051
4.	0,0056	0,0068	0,0132	0,0061
5.	0,0061	0,0077	0,0122	0,0071
6.	0,0055	0,0076	0,0156	0,0065
7.	0,0067	0,0049	0,0083	0,0049
8.	0,0057	0,0054	0,0116	0,0051
Zbroj	*0,0579*	*0,0674*	*0,1185*	*0,0637*

Tablica 4.3. Izmjerena vremena algoritama za dubinu pretrage 3.

Dubina = 3

Potez	Minimax	MinimaxRezanje	Negamax	NegamaxRezanje
1.	0,0231	0,0683	0,0654	0,0492
2.	0,0292	0,0368	0,0934	0,0376
3.	0,0057	0,0333	0,0949	0,0333
4.	0,0273	0,0338	0,0774	0,0337
5.	0,0051	0,0334	0,0825	0,0341
6.	0,0342	0,0472	0,0879	0,0463
7.	0,0071	0,0362	0,0569	0,0292
8.	0,0305	0,0178	0,0599	0,0203
Zbroj	*0,1622*	*0,3068*	*0,6183*	*0,2837*

Tablica 4.4. Izmjerena vremena algoritama za dubinu pretrage 4.

Dubina = 4

Potez	Minimax	MinimaxRezanje	Negamax	NegamaxRezanje
1.	0,3312	0,0537	0,3751	0,0833
2.	0,4398	0,1012	0,3102	0,1372
3.	0,3272	0,0783	0,3985	0,0897
4.	0,4177	0,1134	0,4535	0,1231
5.	0,3268	0,0434	0,3789	0,0367
6.	0,3622	0,1278	0,3819	0,1142
7.	0,2975	0,1207	0,2895	0,1024
8.	0,5963	0,0943	0,4321	0,0995
Zbroj	*3,0987*	*0,7328*	*3,0197*	*0,7861*

Tablica 4.5. Izmjerena vremena algoritama za dubinu pretrage 5.

Dubina = 5

Potez	Minimax	MinimaxRezanje	Negamax	NegamaxRezanje
1.	2,0528	0,3574	2,1418	0,3646
2.	2,0594	0,3536	2,1464	0,3175
3.	1,9431	0,2896	2,1585	0,2399
4.	1,8983	0,2852	2,0005	0,2255
5.	1,7235	0,1971	1,6554	0,1551
6.	1,4967	0,2134	1,5683	0,1961
7.	0,852	0,2095	0,9072	0,2312
8.	0,3691	0,1419	0,4127	0,1232
Zbroj	*12,3949*	*2,0477*	*12,9908*	*1,8531*

Tablica 4.6. Izmjerena vremena algoritama za dubinu pretrage 6.

Dubina = 6

Potez	Minimax	MinimaxRezanje	Negamax	NegamaxRezanje
1.	12,9235	1,0078	12,8808	0,9915
2.	13,4664	1,2962	12,5256	1,377
3.	13,1109	0,8565	13,0135	0,9246
4.	13,1782	0,6353	13,2431	0,6535
5.	13,049	0,9544	12,7119	0,9106
6.	9,3179	0,6575	8,6706	0,6235
7.	4,371	0,4851	4,3087	0,4701
8.	4,749	0,4648	4,4118	0,3689
Zbroj	*84,1659*	*6,3576*	*81,766*	*6,3197*

Tablica 4.7. Izmjerena vremena algoritama za dubinu pretrage 7.

Dubina = 7

Potez	Minimax	MinimaxRezanje	Negamax	NegamaxRezanje
1.	89,7309	4,2379	86,7167	4,2922
2.	90,9662	4,0619	86,1499	4,2252
3.	95,2522	2,5171	89,9024	2,6766
4.	80,8018	2,0227	77,4719	1,8618
5.	69,3248	0,9349	68,2918	0,9319
6.	49,0009	1,6983	46,3592	1,9098
7.	49,9982	0,8689	19,9025	0,8371
8.	21,8594	1,3364	21,0432	1,4046
Zbroj	*546,9344*	*17,6781*	*495,8376*	*18,1392*

Za vrijednosti dubina pretrage stabla 8 i 9, algoritmima minimax i negamax je potrebno preko 2 minute za pojedinačni potez, stoga ta mjerenja nisu prikazana u tablicama.

Tablica 4.8. Izmjerena vremena algoritama za dubinu pretrage 8.

Dubina = 8

Potez	MinimaxRezanje	NegamaxRezanje
1.	12,0437	12,34
2.	17,5118	18,0937
3.	8,9816	9,3223
4.	8,3396	8,1153
5.	3,0097	3,3597
6.	3,524	3,551
7.	2,3991	2,2956
8.	1,5019	1,4129
Zbroj	*57,3114*	*58,4905*

Tablica 4.9. Izmjerena vremena algoritama za dubinu pretrage 9.

Dubina = 9

Potez	MinimaxRezanje	NegamaxRezanje
1.	60,4671	61,2952
2.	61,7067	61,9543
3.	32,5391	31,3551
4.	23,0366	23,3023
5.	10,7349	10,7684
6.	15,6941	15,334
7.	6,9627	6,9735
8.	9,3117	9,4621
Zbroj	*220,4529*	*220,4449*

Može se uočiti da za vrijednosti dubine pretrage 2, 3 i 4 svi algoritmi imaju poprilično dobru brzinu izvođenja jer im za svaki potez treba manje od pola sekunde, iako se vide neke manje razlike. Kako rastu vrijednosti dubine pretrage, tako je razlika između vremena sve veća. Tako vidimo da za vrijednost 5, algoritmima minimax i negamax početno treba oko 2 sekunde, a za vrijednost 6 treba i do 13 sekundi. Za vrijednost 7 treba im čak i do 90 sekundi. S druge strane, njihove verzije s alpha-beta rezanjem izvršavaju se značajno brže. Za vrijednost 5, njihova vremena ne iznose ni pola sekunde, a za vrijednost 6, tek prelaze jednu sekundu. Oko 4 sekunde im je potrebno za vrijednost 7, a za 8 i 9 izvršavaju se za 12 te 60 sekundi.

Pri usporedbi algoritama minimax i negamax, promatrajući redak *Zbroj*, može se vidjeti da je za vrijednosti dubine pretrage 2 i 3, minimax dva do četiri puta brži, a za vrijednosti 4 i 5 je neznatno

brži. Iako je uočljiva razlika, ta vremena su vrlo kratka, u smislu koliko igrač mora čekati na potez, stoga se razlika naoko niti ne primijeti. S druge strane, za preostale, veće vrijednosti negamax je brži za 3 ─ 9 %, što se u nekim slučajevima može bolje zamijetiti. Međutim, tada vremena prelaze preko 12 sekundi, stoga je optimalna dubina pretrage za ova dva algoritma 5, kada iznose oko 2 sekunde.

Uspoređujući algoritme minimax s alpha-beta rezanjem i negamax s alpha-beta rezanjem, može se primijetiti da imaju slična vremena za sve vrijednosti dubine pretrage stabla. Za sva mjerenja, prvi je 3 puta imao bolje vrijeme, dok je drugi bio brži 5 puta. Ta razlika iznosi oko 7 % za manje vrijednosti dubine, a nije veća od 2 % pri većim vrijednostima, što se u većini slučajeva ne primjećuje. Za ove je algoritme optimalna dubina pretrage 6, pri kojoj se oni izvršavaju za 1 sekundu.

4.2. Analiza uspješnosti algoritama

Na tablicama 4.10. ─ 4.14. prikazani su pobjednici međusobnih igara algoritama: osnovnog, negamax te negamax s alpha-beta rezanjem za različite vrijednosti dubine pretrage stabla. Odabrani su ovi algoritmi umjesto njihove minimax inačice, zato što su pokazali malo bolje vremenske rezultate, koji se mogu vidjeti u prošlom potpoglavlju.

U prvom su retku tablica algoritmi koji su prvi na potezu, dok su u prvom stupcu drugi.

Tablica 4.10. Pobjednici igara s dubinom pretrage 2.

Dubina = 2

1. igrač ->	Osnovni	Negamax	NegamaxRezanje
Osnovni	X	Negamax	NegamaxRezanje
Negamax	Osnovni	X	NegamaxRezanje
NegamaxRezanje	Osnovni	Izjednačeno	X

Tablica 4.11. Pobjednici igara s dubinom pretrage 3.

Dubina = 3

1. igrač ->	Osnovni	Negamax	NegamaxRezanje
Osnovni	X	Negamax	NegamaxRezanje
Negamax	Negamax	X	NegamaxRezanje
NegamaxRezanje	NegamaxRezanje	Negamax	X

Tablica 4.12. Pobjednici igara s dubinom pretrage 4.

Dubina = 4

1. igrač ->	Osnovni	Negamax	NegamaxRezanje
Osnovni	X	Negamax	NegamaxRezanje
Negamax	Negamax	X	NegamaxRezanje
NegamaxRezanje	MinimaxRezanje	NegamaxRezanje	X

Tablica 4.13. Pobjednici igara s dubinom pretrage 5.

Dubina = 5

1. igrač ->	Osnovni	Negamax	NegamaxRezanje
Osnovni	X	Negamax	NegamaxRezanje
Negamax	Negamax	X	NegamaxRezanje
NegamaxRezanje	NegamaxRezanje	Negamax	X

Tablica 4.14. Pobjednici igara s dubinom pretrage 6.

Dubina = 6

1. igrač ->	Osnovni	Negamax	NegamaxRezanje
Osnovni	X	Negamax	NegamaxRezanje
Negamax	Negamax	X	NegamaxRezanje
NegamaxRezanje	NegamaxRezanje	Negamax	X

Osnovni algoritam je pobijedio samo 2 puta, kada je igrao prvi i kada su algoritmi bili na dubini pretrage 2. Taj rezultat nije neočekivan s obzirom na to da je osnovni algoritam samo baza na koju

se nadograđuju ostali. Što se tiče ostala dva algoritma, vidljivo je da je uglavnom pobijedio onaj algoritam koji je igrao prvi. Jedina dva odstupanja od toga su u dubini pretrage 2, kada je igra završila izjednačeno te u dubini pretrage 4, kada je negamax s alpha-beta rezanjem pobijedio bez obzira na to što je negamax igrao prvi.

Kada bi se zbrojile pobjede ovih algoritama, rezultati bi bili: negamax – 12, negamax s alpha-beta rezanjem – 15. Ovi rezultati su prošli prema očekivanju jer, kao što je u radu već analizirano, algoritam negamax s alpha-beta rezanjem je poboljšanje negamax algoritma jer on odbacuje nepotrebna grananja. Kako bi se dodatno dokazalo da je on bolji, ova dva algoritma odigrala su još četiri igre, ali je ovaj put negamax igrao na dubini pretrage 6, dok je njegov protivnik igrao na 4 i 5. Pobjednici tih dvoboja mogu se vidjeti u tablicama 4.15. i 4.16.

Tablica 4.15. Pobjednici igara s dubinama pretrage 6 i 5.

1. igrač ->	Negamax, dubina = 6	NegamaxRezanje, dubina = 5
Negamax, dubina = 6	X	NegamaxRezanje
NegamaxRezanje, dubina = 5	NegamaxRezanje	X

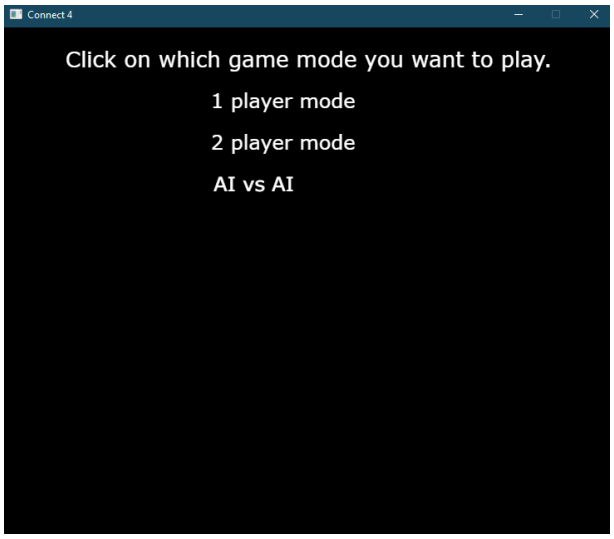
Tablica 4.16. Pobjednici igara s dubinama pretrage 6 i 4.

1. igrač ->	Negamax, dubina = 6	NegamaxRezanje, dubina = 4
Negamax, dubina = 6	X	Negamax
NegamaxRezanje, dubina = 4	Izjednačeno	X

Negamax je pobijedio samo jednom u ove četiri igre, što dokazuje da u većini slučajeva algoritam negamax s alpha-beta rezanjem odigrava bolje poteze, koji najčešće vode do pobjede.

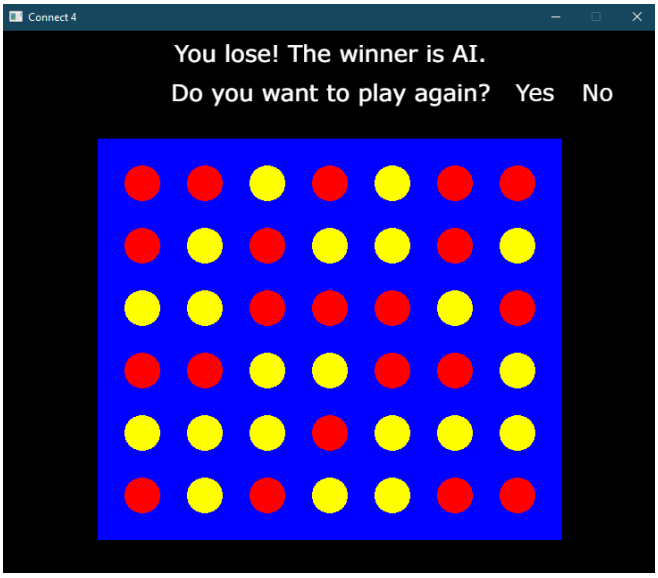
5. IZGLED SUČELJA IGRE

Nakon što se igra pokrene, prikazan je početni zaslon (Slika 5.1.) u kojemu korisnik dobiva uputu da odabere način igranja. Mogući izbori su: 1 igrač, 2 igrača i računalo protiv računala. Za navigaciju i igranje igre korisnik koristi miš te klikom odabire željeni način igranja i stupac u koji želi ubaciti svoj žeton, kada je to moguće. Prvi igrač na potezu uvijek igra žutim žetonom, a drugi crvenim.



Slika 5.1. Početni zaslon.

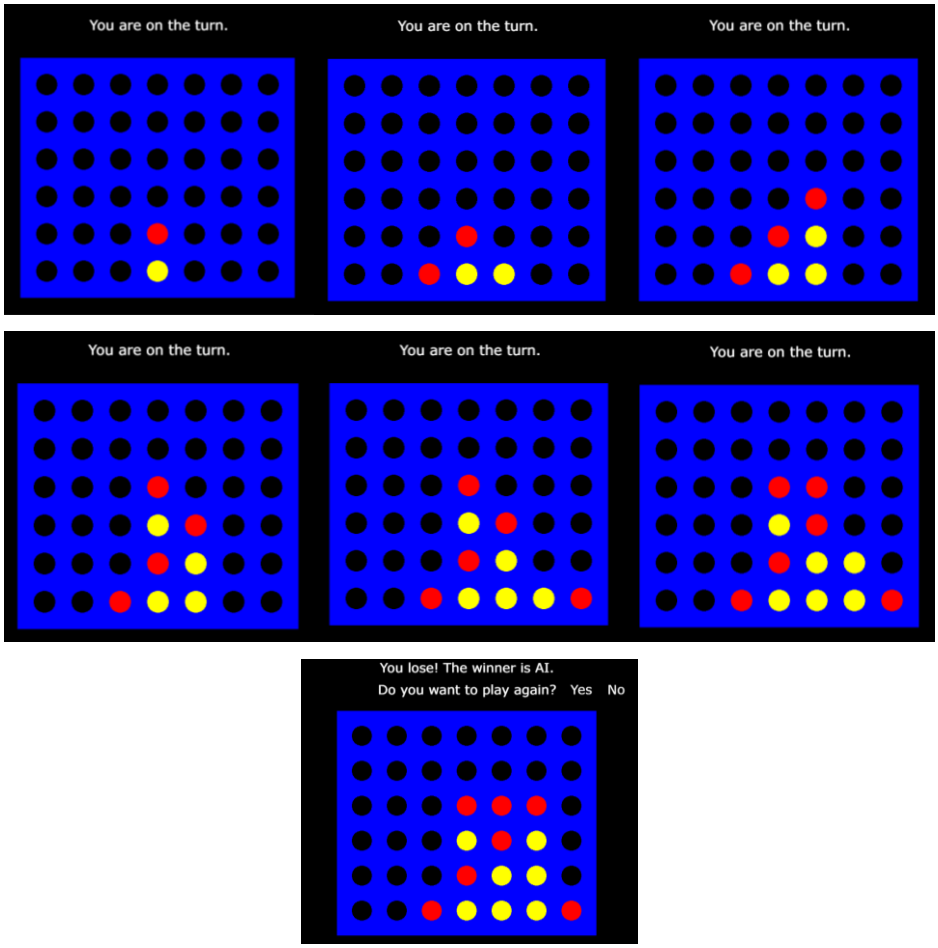
Ako je odabran prvi način, korisnik je prvi na potezu te igra protiv računala, tj. protiv jednog od algoritama koji su objašnjeni u radu. Kada se pojavi ploča na ekranu, korisnik može odigrati svoj potez te nakon toga čeka da računalo odigra. To je vrijeme kratko, zato što je odabran optimalan algoritam kojem ne treba više od par sekundi da se izvrši. Potezi koje je odigralo računalo (crveni žetoni) u jednom dvoboju protiv korisnika mogu se vidjeti na slici 5.3. Nakon što završi igra, pobjedom korisnika ili računala, ili neriješenim rezultatom, moguće je odabrati želi li korisnik igrati ponovno (Slika 5.2.). Ako odabere da ne želi, igra se isključi, a u suprotnom, prikazuje se početni zaslon.



Slika 5.2. Zaslon nakon završene igre.

Ukoliko korisnik odabere drugi način, počinje igra u kojoj mogu igrati dva igrača. Nakon što prvi igrač odigra, drugi je na potezu. Kada je igra završena, opet se pojavljuje opcija za ponovnu igru.

Ako se odabere treći način, pokreću se kombinacije kada različiti algoritmi igraju međusobno i izmjenjuju se u povlačenju prvog poteza. Ti algoritmi, koji su korišteni u ovom načinu, analizirani su i opisani u radu.



Slika 5.3. Svaki potez AI-ja protiv korisnika u jednoj igri.

6. ZAKLJUČAK

U ovom je radu predstavljena igra *Poveži četiri*, njena funkcionalnost, dizajn sučelja te različiti algoritmi koji predstavljaju AI. Oni su opisani i prikazana je njihova implementacija. Fokus rada je bio na analizi rezultata igara u kojima su ti algoritmi međusobno igrali jedan protiv drugog. Prvo je prikazana vremenska analiza algoritama, tj. koliko vremena im je potrebno da se izvrše pri različitim dubinama pretrage stabla igre, gdje su odabrane optimalne vrijednosti. Nakon nje, izloženi su pobjednici međusobnih dvoboja, koji su se tada analizirali. Na temelju ove dvije analize, zaključeno je da je algoritam negamax s alpha-beta rezanjem pri vrijednosti dubine pretrage 6 optimalan među navedenim te da on donosi najbolje poteze u većini slučajeva. Ovaj je algoritam poboljšanje nad osnovnim negamax algoritmom zbog korištenja tehnike alpha-beta rezanja. Također je primijećeno da su algoritmi negamax i minimax odigrali iste poteze tijekom igara, kao i algoritmi negamax s alpha-beta rezanjem i minimax s alpha-beta rezanjem.

U konačnici, prema iznesenim podacima, AI u igri *Poveži četiri* predstavlja algoritam negamax s alpha-beta rezanjem pri vrijednosti dubine pretrage 6, zato što bira optimalne poteze pri vrlo prihvatljivom vremenu izvršavanja.

Iako je najbolji od ostalih predstavljenih algoritama, moguće je razviti i isprogramirati još bolji algoritam, koji bi odigravao najbolje moguće poteze. Ovo bi se moglo postići, primjerice, metodom strojnog učenja, koja bi omogućila, između ostalog, razvoj savršene heurističke funkcije za ovu igru.

LITERATURA

[1] N., Kok, E., J. W. Boers, W., A. Kosters, P., van der Putten, M., Poel, „Artificial Intelligence: Definition, Trends, Techniques and Cases“, dostupno na: https://www.eolss.net/Sample-Chapters/C15/E6-44.pdf [22.7.2023.]

[2] R., Stellabotte, „Toy Story: Catching Up with Howard Wexler, Inventor of the Classic Game Connect 4“, *Fordham News*. 2018., dostupno na: https://news.fordham.edu/fordham-magazine/toy-story-catching-up-with-howard-wexler-inventor-of-the-classic-game-connect-4/ [23.7.2023.]

[3] „Boardgames.io“. 2023., dostupno na: https://boardgames.io/en/connect4 [23.7.2023.]

[4] „CBC Kids“. 2023., dostupno na: https://www.cbc.ca/kids/games/play/connect-4 [23.7.2023.]

[5] „papergames.io“. 2023., dostupno na: https://papergames.io/en/connect4 [23.7.2023.]

[6] K., Galli, „Connect4-Python“, 2019., dostupno na: https://github.com/KeithGalli/Connect4-Python [10.4.2023.]

[7] I., Šišić, „Automat za igranje igre Poveži 4“, Diplomski rad, Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, 2020., dostupno na: https://urn.nsk.hr/urn:nbn:hr:200:376726 [20.8.2023.]

[8] D., Trputec, „Logička igra Četiri u nizu s više razina težine u programskom jeziku C#“, Diplomski rad, Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, 2018., dostupno na: https://urn.nsk.hr/urn:nbn:hr:200:791426 [22.8.2023.]

[9] M., H. J. Romanycia, F. J., Pelletier, „What is a heuristic?“, str. 49–50., 1985., dostupno na: https://www.researchgate.net/publication/227733871_What_is_a_heuristic [13.6.2023.]

[10] G., Chu, P., J. Stuckey, „Learning Value Heuristics for Constraint Programming“, dostupno na: https://people.eng.unimelb.edu.au/pstuckey/papers/autopt.pdf [13.6.2023.]

[11] R. E., Korf, D. M., Chickering, „Best-first minimax search“, str. 299–302, 1996., dostupno na: https://www.sciencedirect.com/science/article/pii/0004370295000968 [2.5.2023.]

[12] A., Elnaggar, M. E., Gadallah, A. A. M., Mostafa, H., Eldeeb, „A Comparative Study of Game Tree Searching Methods“, izd. 5., sv. 5., str. 70, 2014., dostupno na: https://www.researchgate.net/publication/262672371_A_Comparative_Study_of_Game_Tree_Searching_Methods [30.4.2023.]

RJEŠAVANJE PROBLEMA IGRE POVEŽI ČETIRI PRIMJENOM UMJETNE INTELIGENCIJE

SAŽETAK

Zadatak ovog rada bio je primjenom *OpenGL* biblioteke napraviti igru *Poveži četiri* koja omogućuje igranje 2 igrača i protiv računala. Također, implementirani su različiti algoritmi koji predstavljaju umjetnu inteligenciju: osnovni algoritam, minimax, minimax s alpha-beta rezanjem, negamax i negamax s alpha-beta rezanjem. Od tih 5 algoritama trebao se odabrati najbolji. Objašnjena je njihova funkcionalnost i prikazan je implementirani kod. Ključna komponenta u svim ovim algoritmima je heuristička funkcija, koja je važna jer se pomoću nje odlučuje koliko je svaki potez dobar te se onda vrše usporedbe tih poteza. Kako ovi algoritmi imaju različit kod, njihovo vrijeme izvršavanja nije jednako. Na tu vremensku učinkovitost također utječe i vrijednost dubine pretrage stabla igre. Što je ona veća, algoritmi dublje pretražuju grananja u stablu igre te im je potrebno više vremena za izvršavanje. Osim ove vremenske analize, prikazani su pobjednici međusobnih dvoboja algoritama. Iz svih rezultata zaključeno je da je negamax s alpha-beta rezanjem pri dubini pretrage 6 optimalan algoritam.

Ključne riječi: algoritmi, C#, *OpenGL*, *Poveži četiri*, umjetna inteligencija

SOLVING THE CONNECT FOUR GAME PROBLEM USING ARTIFICIAL INTELLIGENCE

ABSTRACT

The task of this paper was to create a Connect Four game using the OpenGL library, enabling gameplay between two players and against a computer opponent. Additionally, various algorithms representing artificial intelligence were implemented: basic algorithm, minimax, minimax with alpha-beta pruning, negamax and negamax with alpha-beta pruning. Among these five algorithms, the goal was to determine the most effective one. Their functionality was explained and their implemented code was presented. A key component in all these algorithms is the heuristic function, which plays a crucial role in evaluating the quality of each move, leading to comparisons between these moves. Due to the different code structures of these algorithms, their execution times vary. The efficiency of these algorithms is also influenced by the depth value of the game tree search. A higher depth value leads to deeper exploration of the game tree branches, thereby increasing the execution time. Alongside this time analysis, the winners of algorithm duels were displayed. Based on all the results, it can be concluded that negamax with alpha-beta pruning at a search depth of 6 is the optimal algorithm.

Key words: algorithms, artificial intelligence, Connect four, C#, OpenGL