Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Tramte

VIZUALIZACIJA ALGORITMA MINIMAX NA PRIMERU IGRE NIM

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Ljubljana 2013

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Tramte

VIZUALIZACIJA ALGORITMA MINIMAX NA PRIMERU IGRE NIM

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Marko Robnik Šikonja

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00508/2013
Datum: 23.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATJAŽ TRAMTE**

Naslov: **VIZUALIZACIJA PREISKOVANJA PO NAČELU MINIMAKS**
VISUALIZATION OF MINIMAX SEARCH

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Razumevanje in poučevanje preiskovalnih algoritmov nam olajšajo orodja za vizualizacijo njihovega delovanja. V igrah dveh igralcev se pogosto uporablja načelo minimaks, ki nam pomaga pri preiskovanju drevesa igre. Vozlišča pregledujemo od trenutnega vozlišča do izbrane globine, kjer stanja hevristično ocenimo, jih po načelu minimaks prenesemo navzgor do trenutnega položaja in na tej podlagi povlečemo potezo.
Na primeru nekaj inačic igre Nim prikažite delovanje preiskovanja po načelu minimaks tako, da prikazujete stanje igre in stanje drevesa igre. Ilustrirajte tudi alfa-beta rezanje. Izdelajte spletno interaktivno aplikacijo, ki bo uporabniku omogočila igranje igre in ga istočasno poučila o delovanju algoritma minimaks.

Mentor:
izr. prof. dr.Marko Robnik Šikonja

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matjaž Tramte, z vpisno številko **63070143**, sem avtor diplomskega dela z naslovom:

VIZUALIZACIJA ALGORITMA MINIMAX NA PRIMERU IGRE NIM

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marka Robnik Šikonje,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 2. julij 2013 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju prof. dr. Marku Robnik Šikonji za vso pomoč in nasvete pri izdelavi diplomske naloge. Prav tako se zahvaljujem bratu Primožu Tramtetu za njegove smernice in nasvete pri programiranju. Zahvaljujem se še staršema za njuno finančno podporo in pa Tadeji Deželan za moralno podporo in ker je vedno verjela vame.

Kazalo

Povzetek

V sklopu diplomskega dela je bila izdelana spletna aplikacija, ki vizualno prikaže delovanje načela minimax. Namen diplomske naloge je boljše razumevanje načela minimax in alfa-beta rezanja. Opisana je ideja za predstavitev algoritma in način izdelave. Algoritem je implementiran na treh različicah igre nim: Grundyjeva igra, igra odštevanja in požrešni nim. Vse tri različice so v delu podrobno predstavljene in razložene. Na eni izmed različic je implementirano in razloženo alfa-beta rezanje. Predstavljene so tehnologije in orodja s katerimi je bila aplikacija izdelana. Razložena je arhitektura aplikacije. Aplikacija je sestavljena iz dveh delov: same igre in drevesa možnih potez, na katerem je prikazano delovanje algoritma.

Ključne besede: igra nim, načelo minimax, alfa-beta rezanje, igre dveh igralcev, vizualizacija algoritmov

Abstract

We present a web application for minimax principle visualization. The purpose of it was to improve understanding of minimax algorithm and alpha-beta pruning. We describe idea for algorithm presentation and its implementation. Minimax algorithm was implemented on three variants of nim game: Grundy's game, Subtraction game and Greedy nim. All three variants are presented and their rules explained. On one of the variants we implemented and explained alpha-beta pruning. Technologies and tools we used in the application are described and the architecture of application is explained. Application consists of two parts: the game itself and a game tree where working of minimax algorithm is visualised.

Keywords: nim game, minimax principle, alpha-beta pruning, two player games, algorithm visualization

Poglavje 1

Uvod

Z razvojem računalniških tehnologij se razvijajo tudi igre. Sodobne igre se lahko pohvalijo z dobro grafiko, akcijo in igralnostjo. Za mnoge so še vedno najprivlačnejše igre za dva igralca, ki so obstajale že dolgo pred razvojem računalnikov. Te igre so na primer šah, go, štiri v vrsto, nim,... Zaradi popularnosti so te igre prenesli tudi na računalnike, kjer jih igramo sami ali proti računalniku.

Načelo minimax je mogoče uporabiti pri igrah, ob katerih sem odrastel. Kljub temu, da je imel z minimaxom opravka sleherni računalničar, pa je algoritem še vedno precej nepoznan. Minimax smo želeli širše predstaviti in ljudi seznaniti z njegovim delovanjem. Algoritem smo želeli vizualizirati, da bo lažje razumljiv, njegovo delovanje pa prikazano ter razloženo. V želji po čim boljši igri računalnika so se pojavili mnogi algoritmi, ki skušajo najti najboljšo možno potezo. Eden izmed teh algoritmov je tudi minimax. Princip minimaxa je, da poskuša igralec na potezi maksimirati svojo korist in minimizirati korist nasprotnika. Minimax preišče vsa možna stanja drevesa igre, dokler ne pride do listov, ki jih oceni. Listi dobijo vrednost 1, če zmaga računalnik, oziroma vrednost 0, če izgubi. Te vrednosti propagiramo navzgor in se na njihovi podlagi odločimo za najboljšo potezo v trenutni poziciji.

Algoritem smo implementirali na preprosti igri nim. Nim je starodavna strateška igra, ki naj bi izvirala iz Kitajske, vendar pa njene korenine niso

točno znane. Izraz nim prihaja iz nemške besede »nimm«, kar pomeni »vzemi«. Pri nimu igralca izmenično jemljeta predmete iz določenih kopic. Igralec, ki ne more več napraviti poteze, izgubi. Različic nima je mnogo. Minimax smo implementirali na treh: Grundyjeva igra, igra odštevanja in požrešni nim.

V drugem poglavju smo opisali uporabljena orodja in razložili arhitekturo aplikacije. V tretjem poglavju smo predstavili algoritem minimax in alfa-beta rezanje. Podrobno je opisano njuno delovanje in implementacija. V četrtem poglavju so opisane implementirane različice igre Nim. Opisana je tudi uporaba minimaxa in alfa-beta rezanja. V petem poglavju je predstavljena vizualizacija iger in dreves stanj. V sklepnem poglavju povzemamo ugotovitve in kritično analiziramo aplikacijo.

Poglavje 2

Opis orodij in arhitekture aplikacije

Vizualizacija igre nim je v obliki spletne aplikacije. Za vizualizacijo igre in algoritma sem uporabil javanski programček, spletna stran pa je v jeziku HTML. Za programski jezik java sem se odločil, ker ga dobro poznam. Javanski programček sem izbral zaradi njegove relativne preprostosti in ker vsebuje vse, kar sem za programiranje potreboval. Vse skupaj sem razvijal v razvojnem okolju NetBeans. V poglavju so najprej opisana uporabljena orodja, nato pa je razložena arhitektura aplikacije.

2.1 Orodja uporabljena pri razvoju

2.1.1 Javanski programček (applet)

Javanski programček [8, 9] je majhen program napisan v java bytecode, v obliki navodil, ki jih izvrši javanski virtualni stroj. Ponavadi se poganja na spletnih straneh. V našem primeru je to HTML stran. Javanski programček teče v procesu ločenem od spletnega brskalnika, vendar se vseeno lahko pojavi v okviru na spletni strani, kjer se izvaja. Javanski programčki so neodvisni od platforme, zato jih lahko poganjamo na brskalnikih za npr. Microsoft Windows, Linux, Mac OS,... Javanski programček je lahko napisan v mno-

gih programskih jezikih. Najpogosteje je to java, lahko pa tudi na primer Jython, Jruby, Eiffel in drugi. Prvič so bili javanski programčki predstavljeni s prvo verzijo programskega jezika java leta 1995.

Javanski programčki prinašajo spletnim stranem interaktivne možnosti, ki jih HTML sam ni zmožen nuditi. Zaznavajo lahko miško, tipkovnico ali vsebujejo različne kontrole, kot so recimo gumbi. V odgovor na uporabnikovo akcijo javanski programček ustrezno spremeni grafično vsebino. Zato je ta način primeren za vizualizacije, demonstracije in učenje. Hitrost programov, napisanih v obliki javanskega programčka, je nekoliko manjša od programov napisanih v C++ jeziku, od leta 2011 naprej pa je veliko večja od programov, napisanih v javaScript jeziku. Uporabljajo lahko tudi 3D strojno pospeševanje (hardware acceleration), ki je na voljo v javi. Na spletni strani se javanski programček izvaja v tako imenovanem »peskovniku«, kar mu preprečuje dostop do lokalnih podatkov, kot je odložišče. Programska koda javanskega programčka se prenese iz spletnega strežnika, brskalnik pa ga zažene znotraj spletne strani ali pa odpre novo okno, v katerem prikaže njegov uporabniški vmesnik.

Prednosti javanskega programčka so:

- enostavno jih je poganjati na različnih operacijskih sistemih, saj so neodvisni od platforme,
- enak javanski programček lahko teče na vseh nameščenih verzijah jave naenkrat, in ne le na zadnji,
- večina brskalnikov ima javanske programčke shranjene v pomnilniku, tako da se hitro spet naložijo, ko se vrnemo nazaj,
- lahko prenese delo iz strežnika do uporabnika in tako razbremeni strežnik pri velikem številu uporabnikov,
- javanski programček, ki mu ne zaupamo nima dostopa do lokalnega računalnika in lahko dostopa le do podatkov na strežniku,
- so razmeroma hitri.

Slabosti javanskega programčka pa so:

- zahteva javo,
- ne moremo jih poganjati na nekaterih mobilnih brskalnikih,
- varnostne omejitve otežijo ali onemogočijo nepreverjenemu javanskemu programčku dostop do željenih uporabnikov,
- nekateri javanski programčki zahtevajo točno določeno java okolje.

Tehnologije, s katerimi lahko dosežemo podobno so na primer javaScript, Flash ali Microsoft Silverlight.

2.1.2 Spletni jezik HTML

HyperText Markup Language (HTML) [7] je jezik za izdelavo spletnih strani, ki jih prikazuje spletni brskalnik. Napisan je v obliki HTML elementov, ki so sestavljeni iz značk, zapisanih v koničastih oklepajih (npr. ¡html¿) znotraj spletne strani. Običajno so značke zapisane v parih, ena odpre oblikovanje, druga ga zapre. Dobra stran tega je, da lahko HTML napišemo v vsakem urejevalniku besedil.

Brskalnik prebere HTML dokument in ga sestavi v vidno in zvočno stran. Brskalnik ne prikazuje značk, ampak jih uporabi za interpretacijo vsebine strani. Dovoljuje vgradnjo slik in objektov v strani, ki so zaradi tega lahko interaktivne. Prav tako lahko strani uporabljajo skripte, napisane v na primer jeziku javaScript, ki vplivajo na obnašanje spletne strani.

HTML je razvil Tim Berners-Lee v želji po skupni uporabi in deljenju dokumentov, njegovi začetki pa segajo v zgodnja devetdeseta leta prejšnjega stoletja,. Prvi javno dostopen HTML dokument je bil dokument »HTML značke«, ki ga je Berners-Lee prvič objavil na internetu konec leta 1991.

Zgodovina verzij:

- HTML 1.0 (1989 – 1994): prva verzija, ki je podpirala dodajanje slik in tekstovnih kontrol. Bila je precej omejena glede oblikovanja in predstavitve vsebine, zato so bile vse strani videti podobno.

- HTML 2.0 (1995): verzija je podpirala več brskalnikov. Tu smo že lahko spreminjali barvo ozadja in dodajali tabele.

- HTML 3.20 (1997): ta verzija je podpirala izdelavo tabel in imela več možnosti za oblikovanje elementov. Spletnim stranem je dovoljevala uporabo matematičnih enačb.

- HTML 4.01 (1999): Podpirala je stilske podloge in multimedijske elemente. Osredotočala se je na ločevanje stila od predstavitve podatkov na spletni strani z uporabo stilskih podlog.

Zadnja HTML verzija je HTML5. Je peta revizija HTML standarda. Njegovi glavni cilji so izboljšanje jezika in podpore za najnovejše multimedijske tehnologije. Je lahko razumljiv in naj bi zamenjal HTML 4, XHTML 1 in DOM Level 2 HTML dokumente. V HTML5 lahko vključujemo veliko novih elementov, vključno z vektorsko grafiko, izboljšan pa je tudi vnos matematičnih formul. HTML5 ima tudi izboljšano obvladovanje neveljavnih dokumentov, tako da brskalniki vse sintaktične napake obravnavajo enotno.

2.1.3 Razvojno okolje NetBeans

NetBeans [5] je odprtokodno integrirano razvojno okolje (IDE), namenjeno predvsem programskemu jeziku java. Podpira razvoj vseh tipov javanskih aplikacij kot so Java SE, Java ME, EJB, spletne in mobilne aplikacije. Omogoča tudi programiranje v drugih jezikih, kot so PHP, C, C++ ali HTML5. NetBeans okolje je napisano v javi in ga lahko poganjamo na operacijskih sistemih Windows, OS X, Linux, Solaris in drugih, ki podpirajo ustrezen javanski virtualni stroj (JVM). Platforma NetBeans dovoljuje razvoj aplikacij z nizom modularnih programskih komponent imenovanih moduli. Vsebuje vse module potrebne za razvoj v javi. Omogoča tudi namestitev modulov za razvoj v drugih programskih jezikih.

2.2 Arhitektura aplikacije

Aplikacijo smo sprogramirali v javanskem programčku, ki smo ga vgradili v HTML spletno stran. Na spletni strani je poleg javanskega programčka še opis vseh treh verzij igre, ki smo jih implementirali. Prav tako sta na kratko opisana tudi algoritem minimax in alfa-beta rezanje. Javanski programček je napisan z zbirko orodij AWT (Abstract Window Toolkit). To je zbirka orodij za okvirjanje, grafiko in uporabniške vmesnike. Zbirka orodij Swing je sicer sodobnejša, vendar pa je AWT preprostejši in vsebuje vse, kar smo potrebovali. Možno je združevanje AWT in Swing komponent, vendar se pogosto pojavijo nezaželeni učinki. To je glavni razlog, da je celoten javanski programček napisan z AWT komponentami.

Aplikacija je na pogled preprosta. Glavna stran vsebuje tri možnosti za izbiro igre. Ob izbiri se nam na ekranu izriše izbrana igra. Na vrhu javanskega programčka je postavljena igra nim, pod njo pa iskalno drevo, na katerem je prikazano delovanje algoritma minimax in alfa-beta rezanja. Vsa grafična vsebina se izrisuje na platno (canvas). Platno je komponenta AWT. Je pravokotna površina, na katero lahko aplikacija riše ali ujame uporabniške vnose. V nekaterih primerih lahko pride do velikega iskalnega drevesa, zato je bilo potrebno javanskemu programčku dodati drsnike. To smo dosegli z uporabo komponente drsniki (scrollpane). To je vrsta vsebovalnika, ki implementira horizontalne in vertikalne drsnike. Platnu na katerega smo risali smo dodali drsnike in tako enostavno dobili oba drsnika. Na vrhu javanskega programčka smo dodali še vsebovalnik plošča (panel), ki smo mu dodali dva gumba in kontrolno okno. Prvi gumb nas vrne na glavno stran, drugi pa ponovno zažene izbrano igro. S pomočjo kontrolnega okna lahko po želji prikazujemo delovanje minimaxa ali alfa-beta rezanja na iskalnem drevesu.

Koda javanskega programčka je napisana v treh razredih:

- MyApplet,
- MyCanvas,

- Node

2.2.1 Razred MyApplet

MyApplet je naš glavni razred, ne glede na to, da ima samo 53 vrstic kode. To je razred, ki razširja razred java.applet.Applet. Ta razred vsebuje metode, ki jih lahko uporabimo za pridobivanje raznih informacij o kontekstu brskalnika. To vključuje metode za:

- pridobitev parametrov javanskega programčka,
- izpis statusnih sporočil v brskalniku,
- pridobitev omrežne lokacije datoteke HTML, ki vsebuje javanski programček,
- spreminjanje velikosti javanskega programčka,
- pridobitev slike,
- pridobitev zvočne datoteke,...

V programih je glavna metoda init(). To je edina metoda, ki smo jo implementirali v razredu MyApplet. V tem razredu smo definirali oba gumba, kontrolno okno, platno, ploščo in drsnike. V metodi init(), smo vsem tem komponentam določili njihovo pozicijo in ostale lastnosti. Plošči smo tu dodali oba gumba in kontrolno okno, platnu pa smo dodali drsnike. Ob inicializaciji platna se kliče razred MyCanvas.

2.2.2 Razred MyCanvas

Razred MyCanvas razširja razred Canvas in implementira večje število vmesnikov:

- MouseListener,
- MouseMotionListener,

- Runnable,
- ActionListener,
- ItemListener

Delovanje komponente platno smo že opisali. Vse Listener komponente smo uporabili za zajem dogodkov, ki smo jih povzročili z miško. To so recimo klik na gumb, vklop in izklop kontrolnega stikala, opravljena poteza v igri,... Vmesnik Runnable smo uporabili za izvajanje programa v dveh ločenih nitih. Eno nit uporabljamo zato, da se izvajanje programa ob potezi računalnika ustavi za eno sekundo. Razlog je, da računalnik svoje poteze ne opravi takoj, ker to opravi prehitro in bi izgledalo, kot da je igralec opravil dve potezi naenkrat. Druga nit poskrbi za vse ostalo.

Razred MyCanvas vsebuje vse metode in funkcije, ki sestavljajo igro in vizualizacijo. Vsebuje metode za izris, funkciji minimax in alphaBeta, pomožne funkcije ter različne sezname in spremenljivke, ki smo jih uporabili pri implementaciji. Dolžina razreda je 898 vrstic in vsebuje 28 različnih metod in funkcij.

2.2.3 Razred Node

Node je objektni razred, ki definira vozlišče. Vsakemu ustvarjenemu vozlišču se dodelijo različne vrednosti. Vsebuje več get in set metod. Ta razred je enak za vse tri igre, čeprav se vozlišča v igrah razlikujejo. Glavni atributi objekta Node so:

- tabela values, ki nam pove velikost kopic v igri,
- depth, ki nam pove globino vozlišča v drevesu stanj,
- seznam children, ki vsebuje vse otroke vozlišča, tudi otroci so tipa Node.

Vsi ostali atributi so pomožni atributi, s katerimi smo si pomagali pri vizualizaciji drevesa. Razred Node vsebuje 314 vrstic kode. Pri vizualizaciji

igre stanje kopic na plošči prikazuje vrednost objekta Node. Celotno drevo stanj dobimo tako, da s pomočjo pomožne metode v seznam dodamo vse otroke začetnega vozlišča. Vsakemu vozlišču v seznamu tekom igre prirejamo določene atribute, ki nam pomagajo pri vizualizaciji.

Poglavje 3

Minimax in alfa-beta rezanje

Igre dveh igralcev lahko igramo proti računalniku. Za učinkovito igro potrebuje računalnik algoritme, ki mu uspejo določiti dobre poteze. Eden izmed njih je tudi minimax.

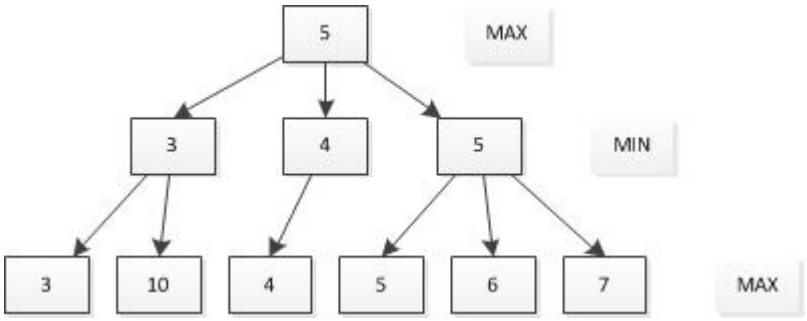
3.1 Minimax

Teorem minimaxa [1, 2, 3] je že leta 1928 dokazal ameriški matematik John Von Neumann. Minimax je namenjen igram dveh igralcev. Te igre so logične igre, kar pomeni, da imajo določena pravila in omejitve. Zaradi teh pravil vedno vemo, kakšne so naše naslednje možne poteze. Poznamo torej vse naše možne poteze, kot tudi vse nasprotnikove. Seveda pozna naše možne poteze tudi nasprotnik.

3.1.1 Iskalno drevo

Če hočemo poznati delovanje algoritma, moramo najprej vedeti kaj je iskalno drevo. Iskalno drevo je način, s katerim predstavimo iskanje. Primer drevesa stanj za neko igro je prikazan na sliki 3.1.

Kvadrati se imenujejo vozlišča in predstavljajo točke odločitve v iskanju. Vozlišča so med seboj povezana s povezavami. Iskanje se začne na vrhu slike, v korenu drevesa. V vsaki točki odločitve se ustvarijo nova vozlišča, ki so na



Slika 3.1: Primer drevesa stanj igre.

voljo, dokler ni možne več nobene odločitve. Vozlišča, ki predstavljajo konec iskanja se imenujejo listi.

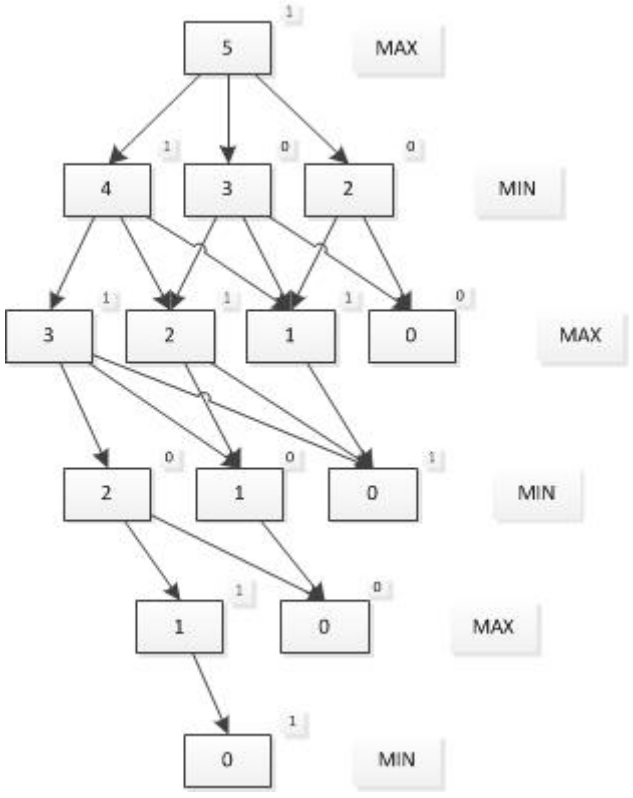
Pri minimaxu sta v igro vključena dva igralca, MAX in MIN. Najprej se ustvari iskalno drevo, najprej v globino. Prične se v trenutni poziciji in gre vse do listov. Slika 1 prikazuje, da se vrednosti končnih pozicij ocenjujejo s stališča igralca MAX. Za tem se notranja vozlišča od spodaj navzgor napolnijo z ocenjenimi vrednostmi. Vozlišča, ki pripadajo igralcu MAX, prejmejo največjo vrednost od svojih naslednikov. Vozlišča, ki pripadajo igralcu MIN pa med svojimi nasledniki izberejo najmanjšo vrednost. Psevdokoda algoritma minimax je prikazana spodaj.

```
double minimax (current_node) {
     if (is_leaf (current_node) || depth (current_node) == MaxDepth)
return heuristic_evaluation (current_node);
     if (is_min_node (current_node))
return min (minimax (children_of (current_node)));
     if (is_max_node (current_node))
return max (minimax (children_of (current_node)));
}
```

Izsek kode 3.1: Psevdokoda algoritma minimax.

Vrednosti torej predstavljajo, kakovost potez. Igralec MAX bo poskušal zase izbrati potezo z največjo vrednostjo, igralec MIN pa bo poskušal izbrati najboljšo možno potezo zase, torej bo poskušal minimizirati MAXov rezultat.

Na sliki 3.2 je prikazan primer delovanja algoritma minimax za igro odštevanja s petimi začetnimi predmeti in do k=3 vzetki. Igro prične igralec MAX. Vozlišča označimo z oznakami MIN in MAX po nivojih, glede na to kdo je na potezi. Algoritem preišče vsa vozlišča, dokler ne pride do listov. Tem minimax priredi vrednost 0, če zmaga MIN ali vrednost 1, kadar zmaga MAX. Te vrednosti propagiramo navzgor. Če je predhodnik tipa MAX dobi največjo minimax vrednost naslednikov (izbere najboljšo potezo zase), če pa je tipa MIN izbere najmanjšo minimax vrednost (izbere najslabšo potezo za nasprotnika, torej najboljšo zase).



Slika 3.2: delovanje algoritma minimax za igro odštevanja s 5 začetnimi predmeti in do 3 vzetki.

Ne obstaja veliko iger, pri katerih bi lahko ustvarili celotno iskalno drevo v kratkem času. Pri šahu je možnih potez v določeni situaciji veliko pa

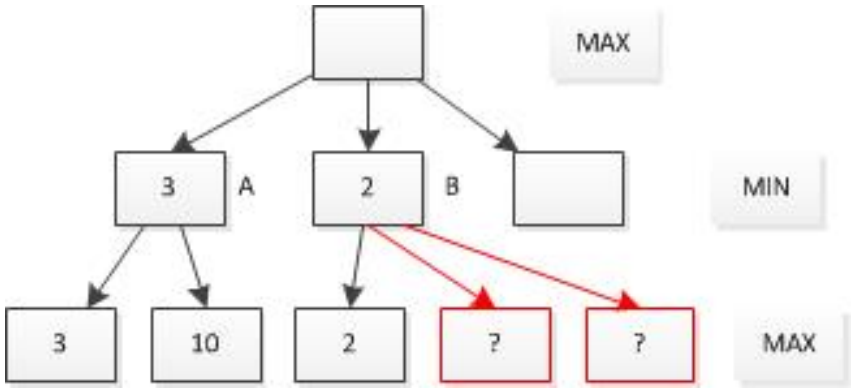
tudi globina drevesa bi bila ogromna. Generiranje takega drevesa bi nam vzelo preveč časa. Algoritem je torej potrebno poenostaviti. Z optimizacijo zamenjamo popolne informacije o potezah za bližnjice in verjetnost. Namesto da bi poznali točno pot do zmage, sedaj poznamo verjetno pot do zmage. Če je optimizacija slaba, so poteze slabe in bi bilo bolje, da bi naključno izbirali poteze.

Mogoča optimizacija je, da stanja pregledamo do določene globine. S tem se izognemo generiranju prevelikega drevesa. Vozlišča na tej globini hevristično ocenimo in vrednosti po načelu minimaxa propagiramo navzgor. Slabost minimaxa je, da preiskuje vsa poddrevesa, kljub temu da ve, da so nekatera neperspektivna. Tu se pojavi alfa-beta rezanje.

3.2 Rezanje alfa-beta

Hitrost iskanja najboljše poteze je pri umetni inteligenci zelo pomembna, saj če bi iskanje trajalo predolgo, algoritem ne bi bil primeren. Na primer, dober algoritem minimax lahko v sekundi preišče do 1000 potez. Na šahovskih turnirjih ima igralec za potezo na voljo okoli 150 sekund, torej bi bilo v tem času možno analizirati 150 000 potez. Problem je, ker ima pri šahu vsaka pozicija v povprečju na voljo 35 potez in bi tako lahko algoritem preiskoval le tri ali štiri poteze vnaprej. Tudi ljudje z zelo malo vaje so sposobni več od tega. Vendar lahko minimax precej pohitrimo. Vzemimo primer na sliki 3.4. Vrednost vozlišča A je 3. Prva najdena vrednost poddrevesa, ki se začne v vozlišču B je 2. Ker je vozlišče B na nivoju MIN, vemo, da bo končna vrednost vozlišča B manjša ali enaka 2. Vemo tudi, da ima vozlišče A vrednost 3 in da imata z vozliščem B istega starša na nivoju MAX. To pomeni, da pot, ki se začne pri vozlišču B ne bi bila izbrana, ker je vrednost 3 za vozlišče MAX boljša od 2. Zaradi tega otrok vozlišča B ni potrebno več preiskovati in jih lahko ignoriramo.

To pomeni, da lahko iskanje prekinemo, ker vemo da ne bomo našli boljše poteze. Ta način optimizacije se imenuje alfa-beta rezanje[2, 3, 4]. Potek



Slika 3.3: Drevo, ki prikazuje, katere veje bi lahko odrezali.

algoritma je sledeč:

- imamo dve vrednosti, ki se prenašata po vozliščih drevesa: alfa vrednost predstavlja najboljšo doslej najdeno vrednost za vozlišče MAX, beta vrednost pa prestavlja najslabšo doslej najdeno vrednost za vozlišče MIN,
- preiskujemo v globino,
- v vozliščih MAX zavržemo vrednosti manjše od alfa,
- v vozliščih MIN zavržemo vrednosti večje od beta.

Natančnejša pravila alfa-beta rezanja so:

- odrežemo naslednike vozlišča MIN z vrednostjo beta manjšo ali enako vrednosti alfa njegovega predhodnika MAX,
- odrežemo naslednike vozlišča MAX z vrednostjo alfa večjo ali enako vrednosti beta njegovega predhodnika MIN.

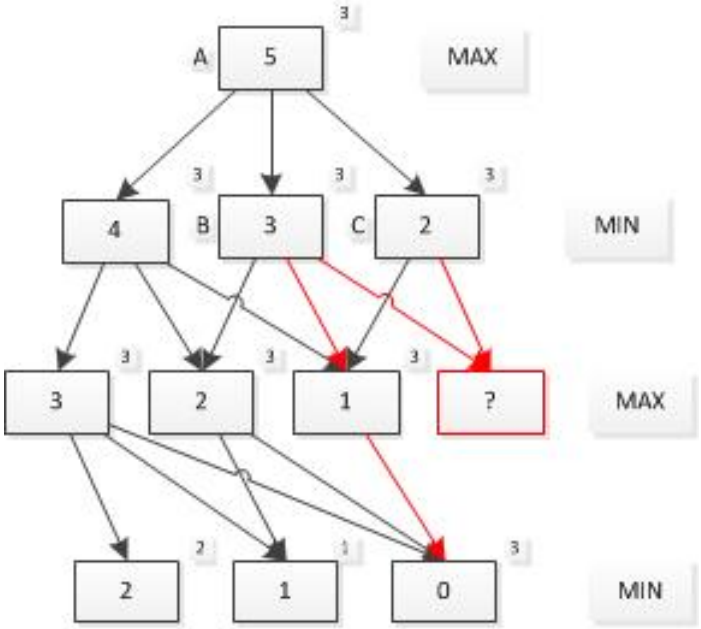
Za rezanje na nivoju m, tako primerjamo vrednosti na nivoju *(m - 1)* in *(m + 1)*. Doseganju vrednosti Alfa in Beta pri predhodnikih se izognemo tako, da implementacija te vrednosti pošlje kot parametre. Psevdokoda algoritma je prikazana spodaj.

```
//klic: alpha_beta (start_node, -infinity, infinity)
double alphaBeta (node, alpha, beta){
       if ( leaf (node)) {
              return heuristic_evaluation (node);
       if (max_node (node)){
              alpha = max (alpha, alphaBeta ( children, alpha, beta));
              if (alpha >= beta)
                     cut_off_search_below (node);
       }
        if (min_node (node)){
              beta = min (alpha, alphaBeta ( children, alpha, beta));
              if (beta <= alpha)
                     cut_off_search_below (node);
       }
}
}
```

Izsek kode 3.2: Psevdokoda alfa-beta rezanja.

Primer delovanja algoritma alphaBeta za enako situacijo, kot na sliki 3.2, je prikazan na sliki 3.4. Z rdečo črto so prikazane povezave, kjer nam ni potrebno preiskovati naprej. Algoritem pregleduje do globine 4, zato nižje od globine 4 vozlišča niso prikazana. Za lažjo razumljivost, so vozlišča na levi strani označena s črkami. V vozlišču A je alfa vrednost 3 in ker je A MAX, vrednost ne bo manjša od 3. Ko v vozlišču B, ki je na nivoju MIN, od levega sina dobimo vrednost 3, lahko oba desna naslednika odrežemo, saj bo njuna vrednost manjša ali enaka 3. Podobno lahko odrežemo desnega naslednika vozlišča C. Od levega naslednika dobi C vrednost 3 in ker je $3 = 3$ lahko desnega naslednika odrežemo.

Koliko je implementacija alfa-beta rezanja hitrejša, je odvisno od vrstnega reda vozlišč v drevesu. Če se vozlišča ustvarijo v vrstnem redu, da ne moremo izkoristiti prednosti alfa-beta rezanja, pohitritve ne bo. Če pa pozicije omogočijo veliko rezanja, so pohitritve ogromne.



Slika 3.4: Delovanje algoritma alphaBeta za enak primer kot na sliki 2.

Poglavje 4

Uporaba minimaxa in alfa-beta rezanja v igrah nim

Nim [10, 11] je strateška igra, v kateri dva igralca izmenično jemljeta ali premikata predmete iz kopic. Igralec na potezi mora premakniti ali odstraniti vsaj en predmet iz ene kopice. Različice nima so se igrale že v starodavnih časih. Igra naj bi izvirala iz Kitajske, vendar pa njen izvor ni točno znan. V Evropi se nim prvič omenja na začetku 16. stoletja. Trenutno ime ji je podelil Charles L. Bouton, zaposlen na Harvardski univerzi, ki je leta 1901 razvil tudi celotno teorijo igre, vendar izvor imena ni v celoti pojasnjen. Najverjetneje je ime izpeljanka iz nemške besede »nimm«, kar pomeni vzemi ali pa iz starega angleškega izraza »nim«, ki ima enak pomen. Lahko se igra na dva načina:

- igralec, ki naredi zadnjo možno potezo, zmaga,
- igralec, ki naredi zadnjo možno potezo, izgubi.

Različice, ki smo jih implementirali, se igrajo tako, da igralec, ki naredi zadnjo možno potezo, zmaga. Nim je ena izmed prvih računalniških iger, razvita leta 1952. Razvili so jo Herbert Koppel, Eugene Grant in Howard Bailer, inženirji podjetja W.L. Maxon. Stroj, ki so ga razvili, je tehtal 22 kilogramov. Igral je proti človeškemu nasprotniku in skoraj vedno zmagal.

Običajna igra Nim se igra tako, da imata igralca pred seboj tri poljubno velike kopice predmetov. Igralec na potezi iz poljubne kopice vzame poljubno število predmetov. Igralec, ki napravi zadnjo možno potezo, zmaga. Različic igre nim je zelo veliko, npr. igra odštevanja, igra 21, igra 100, krožni nim, požrešni nim, Grundyjeva igra in druge. Implementirali in vizualizirali smo tri različice:

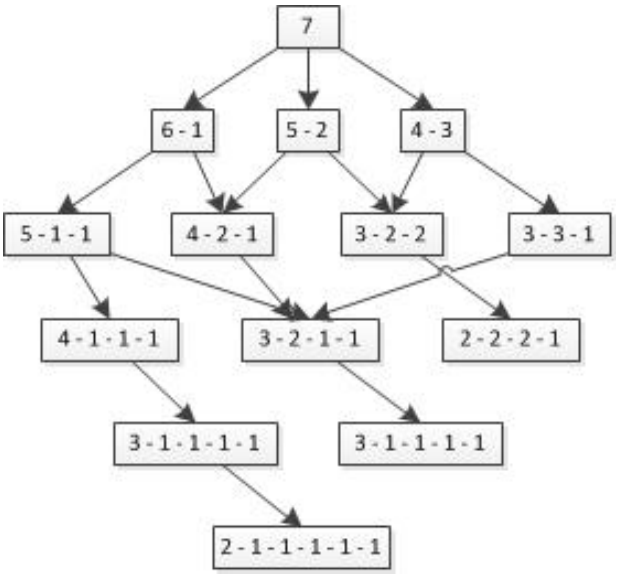
- Grundyjeva igra,
- igra odštevanja,
- požrešni nim.

4.1 Predstavitev implementiranih iger

4.1.1 Grundyjeva igra (Grundy's game)

Pri Grundyjevi igri predmetov ne jemljemo, ampak jih premikamo. Na začetku imamo na voljo eno kopico predmetov. Igralca kopice izmenično delita na dva neenaka dela. Igra se konča, ko na plošči ostanejo samo kopice z dvema ali enim predmetom, ker se teh ne da več razdeliti na dva neenaka dela. Igralec, ki naredi zadnjo potezo, zmaga.

Primer: Na plošči imamo kopico iz 7 predmetov. Igralec A kopico razdeli na 2 neenaki kopici, ki vsebujeta po 5 in 2 predmetov. Prva kopica ima 5 predmetov in druga 2 predmeta. Igralec B lahko razdeli prvo kopico (druge ne more razdeliti na 2 neenaka dela). Kopico razdeli na novi kopici, ki vsebujeta po 3 in 2 predmetov. Igralec A ima sedaj pred sabo 3 kopice, ki vsebujejo po 3, 2 in 2 predmetov. Razdeli lahko samo še kopico iz treh predmetov in sicer na kopici z 2 in 1 predmetom. Igralec B ima sedaj pred seboj kopice, ki vsebujejo 2, 2, 2 in 1 predmet. Nobene izmed kopic ne more več razdeliti na dva neenaka dela, zato je igro izgubil. Prostor vseh stanj za primer igre je prikazan na sliki 4.1.



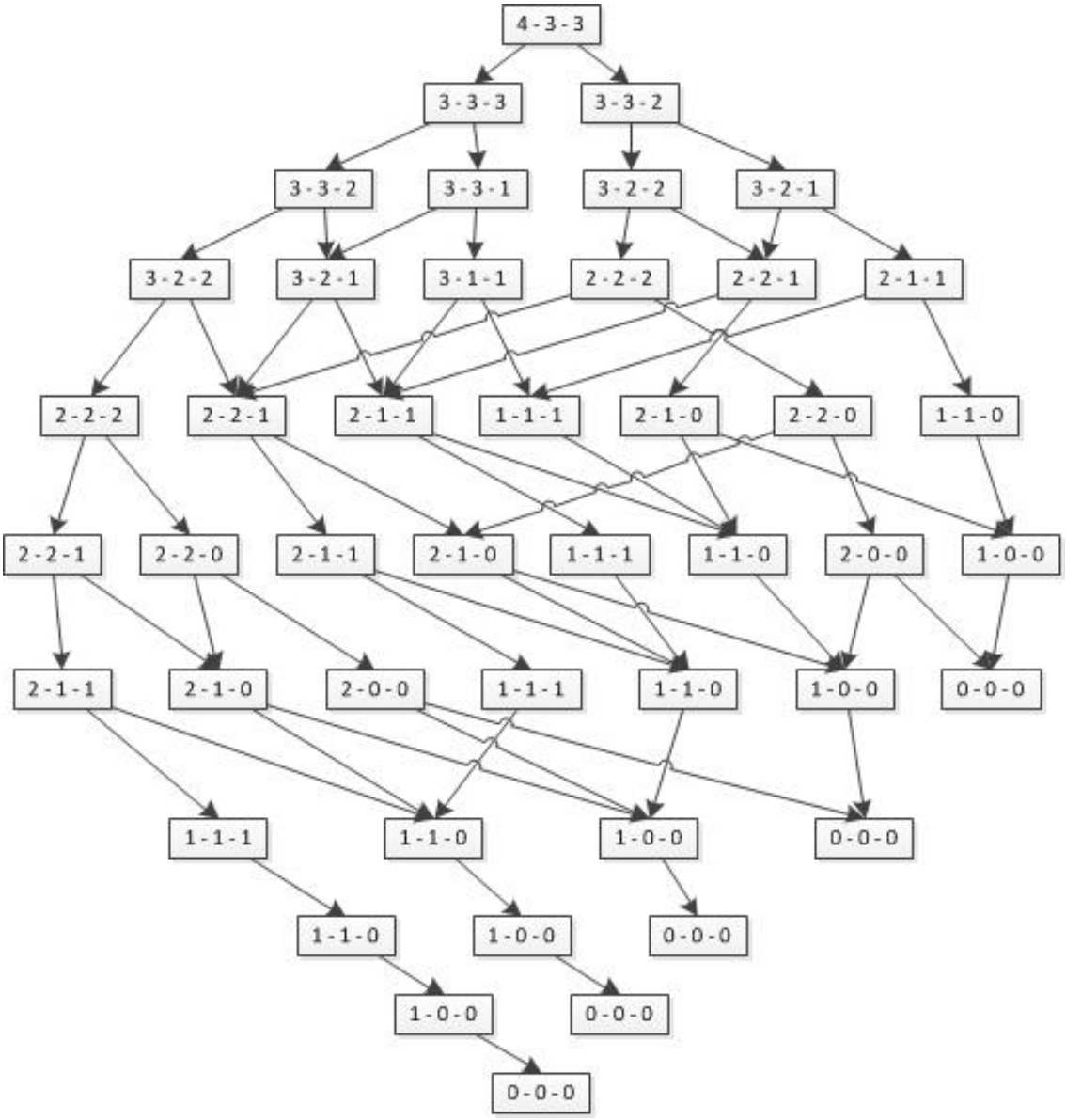
Slika 4.1: Prostor stanj za Grundyjevo igro s sedmimi začetnimi predmeti.

4.1.2 Požrešni nim (greedy nim)

Pohlepni nim je različica igre nim, kjer imamo na plošči določeno število kopic. Igralec, ki je na vrsti, lahko vzame 1, 2, ... k predmetov iz največje kopice. Največkrat je dovoljeno odstraniti 3 predmete naenkrat. Naša implementacija zaradi preglednosti iskalnega drevesa dovoljuje odstranitev do dveh predmetov naenkrat.

Primer: Na plošči imamo 3 kopice. Prva vsebuje 4 predmete, druga 3 in tretja 3. Igralec A vzame iz največje kopice 2 predmeta. Ostanejo kopice, ki vsebujejo po 3, 3 in 2 predmetov. Tudi igralec B vzame 2 predmeta iz največje kopice. Na plošči ostanejo kopice sestavljene iz 3, 2 in 1 predmeta. Na vrsti je igralec A, ki odstrani 1 predmet, tako da ostanejo kopice z 2, 2 in 1 predmetom. Igralec B odstrani 1 predmet. Ostanejo kopice, ki vsebujejo 2, 1 in 1 predmet. Igralec A odstrani 2 predmeta iz največje kopice, tako da ostanejo samo še 2 kopici. Vsaka vsebuje po 1 predmet. Igralec B nima izbire in lahko odstrani samo 1 predmet. Ostane 1 kopica z 1 predmetom, ki jo odstrani igralec A, in zmaga. Prostor vseh stanj za to igro je prikazan na

sliki 4.2.

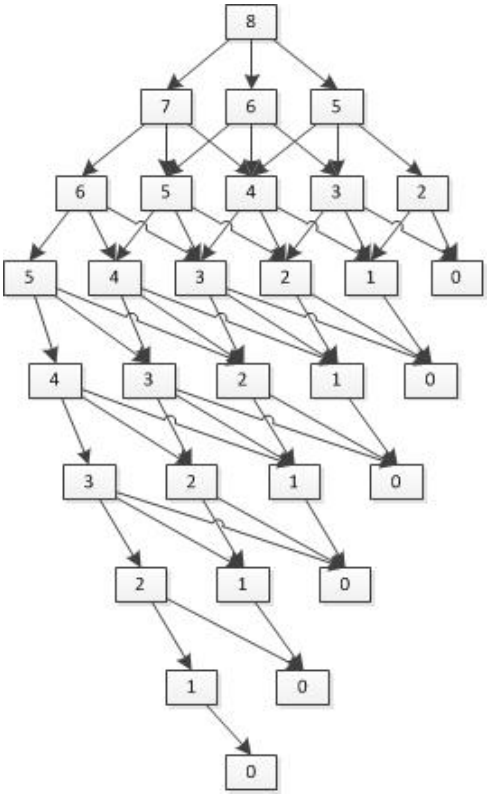


Slika 4.2: Prostor stanj za igro požrešni nim s 4 – 3 – 3 začetnimi kopicami in do dvema vzetkoma.

4.1.3 Igra odštevanja (subtraction game)

Tej igri pogostokrat pravijo tudi kar nim, vendar je natančnejše ime Subtraction Game ali »Igra odštevanja«. Na voljo imamo eno kopico. Igralca iz kopice izmenično odstranjujeta 1, 2,... do k predmetov. Najpogosteje je dovoljeno odstraniti do 3 predmete naenkrat.

Primer: Na plošči imamo kopico iz 8 predmetov. Igralec A najprej odstrani 1 predmet. Na plošči ostane 7 predmetov in igralec B odstrani 3 predmete. Na plošči zdaj ostanejo 4 predmeti. Igralec A odstrani 2 predmeta. Ostaneta 2 predmeta, ki ju odstrani igralec B, ki zato zmaga. Prostor stanj za igro odštevanja z 8 začetnimi predmeti je prikazan na sliki 4.3.



Slika 4.3: Prostor stanj za igro odštevanja z 8 začetnimi predmeti in do k=3 vzetki.

4.2 Uporaba minimaxa in alfa – beta rezanja

Algoritem minimax smo implementirali na vseh treh igrah. Alfa-beta rezanje smo implementirali le na igri požrešni nim. Algoritem minimax je enak za vse tri igre. Pri igri požrešni nim ima uporabnik na izbiro vizualizacijo minimaxa ali rezanja alfa-beta.

4.2.1 Uporaba minimaxa

Na izseku kode 4.1 je predstavljena naša implementacija Minimaxa, ki je enaka pri vseh treh igrah. Algoritem je napisan rekurzivno. Najprej preveri, ali je vozlišče »n«, podano kot parameter, list. V primeru, da je list, preveri ali leži na nivoju MIN ali MAX. Če leži na nivoju MIN, potem funkcija vrne vrednost 1, saj je MAX zmagal. Če je list in na nivoju MAX, vrne vrednost 0. V primeru, da vozlišče ni list, algoritem preveri, na katerem nivoju se vozlišče nahaja. V primeru, da se vozlišče nahaja na nivoju MIN, algoritem za vsakega naslednika vozlišča n kliče funkcijo minimax in mu dodeli vrednost, ki jo vrne. Nato vrne najmanjšo vrednost naslednikov. Če je vozlišče n na nivoju MAX, vrne največjo vrednost vseh naslednikov.

```
  public int minimax(Node n){
        int min, max;

        if(n.getChildren() == null){  //ce je n list se izvedejo notranje zanke
            if(n.isMinNode()) // ce je vozlisce n na nivoju MIN vrne 1, drugace vrne 0
                return 1;
            else
                return 0;
        }

        else if(n.isMinNode()){  //ce n ni list in je na nivoju MIN:
            int[] tabMin = new int[n.getChildren().size()]; // ustvarimo tabelo, v katero
//bomo shranili minimax vrednosti vseh naslednikov vozlisca n
            for(int i=0;i<n.getChildren().size();i++)
                tabMin[i] = minimax(n.getChildren().get(i)); // v tabelo vnesemo minimax
//vrednosti vseh naslednikov vozlisca n
            min=tabMin[0];
            for(int i=1;i<tabMin.length;i++) //poiscemo najmanjso vrednost v tabeli
                if(tabMin[i]<min) {
                min = tabMin[i];
```

```
            }
            return min;
        }
      //ce ni list in je na nivoju MAX:
          else{
            int[] tabMax = new int[n.getChildren().size()];// ustvarimo tabelo, v katero
//bomo shranili minimax vrednosti vseh naslednikov vozlisca n
            for(int i=0;i<n.getChildren().size();i++)
                tabMax[i] = minimax(n.getChildren().get(i)); // v tabelo vnesemo minimax
//vrednosti vseh naslednikov vozlisca n
            max=tabMax[0];
            for(int i=1;i<tabMax.length;i++) //poiscemo najvecjo vrednost v tabeli
                if(tabMax[i]>max)
                    max= tabMax[i];
            return max;
        }
    }

}
```

Izsek kode 4.1: Implementirana koda algoritma minimax.

4.2.2 Uporaba alfa-beta rezanja

Alfa-beta rezanje smo, implementirali na igri požrešni nim. Klic algoritma je *alphaBeta (Node n, int alpha, int beta)*. Vrednosti *alpha* in *beta*, zaradi učinkovitejše implementacije, pošljemo kot parametra, da se izognemo dostopanju do vrednosti predhodnikov. Najprej funkcija preveri, ali je vozlišče n list in ali je na globini, ki je za 4 večja od globine vozlišča, ki prikazuje trenutno stanje igre. Če je, potem funkcija vozlišče n hevristično oceni in to vrednost vrne. Obiskanemu vozlišču nastavimo vrednost *alphaBeta* na *true*. Ta korak glede alfa-beta rezanja ni pomemben, vendar smo si z njim pomagali pri vizualizaciji iskalnega drevesa. Če n ni list, funkcija preveri, na katerem nivoju je n. V primeru, da je na nivoju MAX, najprej vzame prvega naslednika in mu dodeli *alphaBeta* vrednost. To vrednost primerja s parametrom *alpha* in shrani večjo izmed obeh vrednosti. Če je *alpha* po tem koraku večja ali enaka vrednosti *beta*, prekine nadaljnje iskanje alfa-beta vrednosti otrok in vrne vrednost *alpha*. Podobno se zgodi, če je n na nivoju

MIN, le da tu dobljeno vrednost primerja z *beta*, kamor shrani minimum obeh vrednosti. V primeru da je *beta* manjša ali enaka vrednosti *alfa*, se iskanje *alphaBeta* vrednosti naslednikov konča in funkcija vrne vrednost *beta*.

Hevristično vrednost posameznega vozlišča kličemo s *heuristic_evaluation (Node n)*. V primeru, da je vozlišče list, funkcija preveri na katerem nivoju se nahaja. Če je na nivoju MAX, potem MAX izgubi, torej vrne vrednost 0. Če pa je na nivoju MIN, igralec MAX zmaga, zato mu dodeli vrednost 3, ker sta vrednosti 1 in 2 rezervirani za oceno vozlišč, ki niso listi. Problem nastane, ko vozlišče ni list, pa mu je vseeno potrebno določiti hevristično vrednost, saj se nahaja na globini, do katere preiskujemo. Vrednost mu zato določimo po nekih hevrističnih načelih. Po preučevanju igre smo ugotovili, da se MAXu najbolj izplača, da nasprotniku ustvari situacijo, ko sta prvi dve kopici enako veliki, tretja pa je manjša. Tako mu namreč najlažje prepreči zmago. Ko je vozlišče torej na nivoju MIN, je na vrsti nasprotnik. Če kopice ustrezajo zgornjemu pogoju, dodeli vozlišču vrednost 2, sicer mu dodeli vrednost 1. Razlog, da mu dodeli vrednost 2 in 1, ne pa 3 in 0, je v tem, da so pri teh situacijah to le ocene in ne moremo zagotovo vedeti, da je situacija za nas zmagovalna. Ocenitev torej ni stoodstotno natančna, zato so vrednosti manjše. Naša implementacija alfa-beta rezanja je predstavljena na izseku kode 4.2.

```
public int alphaBeta (Node n, int alpha, int beta) {
     int vrednost;  // v to spremenljivko bomo shranjevali vrednosti alphaBeta algoritma
     if ( n.getChildren() == null || n.getDepth() == nova.getDepth() + 4) { //ta zanka se
//izvede, ce je n list ali pa je na globini, ki je vsaj 4 vecja od trenutnega vozlisca
          n.setEval (heuristic_evaluation(n)); //vozliscu nastavimo njegovo vrednost
          n.setAlphaBeta(true); //s to vrednostjo povemo, da smo vozlisce obiskali
          return heuristic_evaluation(n);
     }
     else if (n.isMaxNode()) { //zanka se izvede, ce je n na nivoju MAX
          for (Node c : n.getChildren()) {
               vrednost = alphaBeta (c, alpha, beta); //shranimo alphaBeta vrednost
//trenutnega otroka
               alpha = max (alpha, vrednost); //shranimo najvecjo vrednost med
//alpha in vrednost
               c.setAlphaBeta(true);
               if(alpha >= beta) //ce je alpha vecji od beta, prekinemo izvajanje
                    break;
```

```
            }
            setEval(alpha); //vozliscu nastavimo vrednost
            return alpha;
      }
      else { //zanka se izvede, ce je n na nivoju MIN
            for (Node c : n.getChildren()) {
                  vrednost = alphaBeta (c, alpha, beta); //shranimo alphaBeta vrednost otroka
                  beta = min (beta, vrednost); //shranimo najmanjso vrednost med
//alpha in vrednost
                  c.setAlphaBeta(true);
                  if(beta <= alpha) //ce je beta manjsi ali enak alpha, prekinemo izvajanje
                        break;
            }
            n.setEval(beta); //nastavimo vrednost
            return beta;
      }
}
```

Izsek kode 4.2: Implementirana koda algoritma alphaBeta.

Poglavje 5

Vizualizacija igre

Ob zagonu aplikacije se nam prikaže glavni meni (slika 5.1). Uporabnik ima na izbiro tri igre. Ob izbiri ene, se na zaslonu izriše izbrana igra.

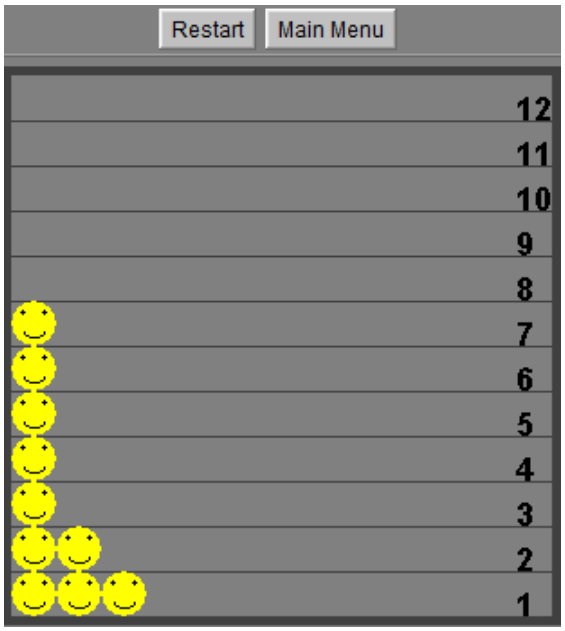
Slika 5.1: Prva stran aplikacije.

Struktura vizualizacije je pri vseh treh igrah enaka. Igra se izriše na vrhu okna, pod njo pa imamo prikazano drevo stanj, na katerem je prikazano delo-

vanje algoritma minimax. Predmeti, s katerimi igramo, so v obliki smejočih obrazov. Igralno polje je veliko 300 x 300 točk. Vodoravne črte in številke na desni strani nam pomagajo razbrati velikost kopice. Vsaka igra ima nad sabo gumba za ponovni začetek in vrnitev na glavno stran.

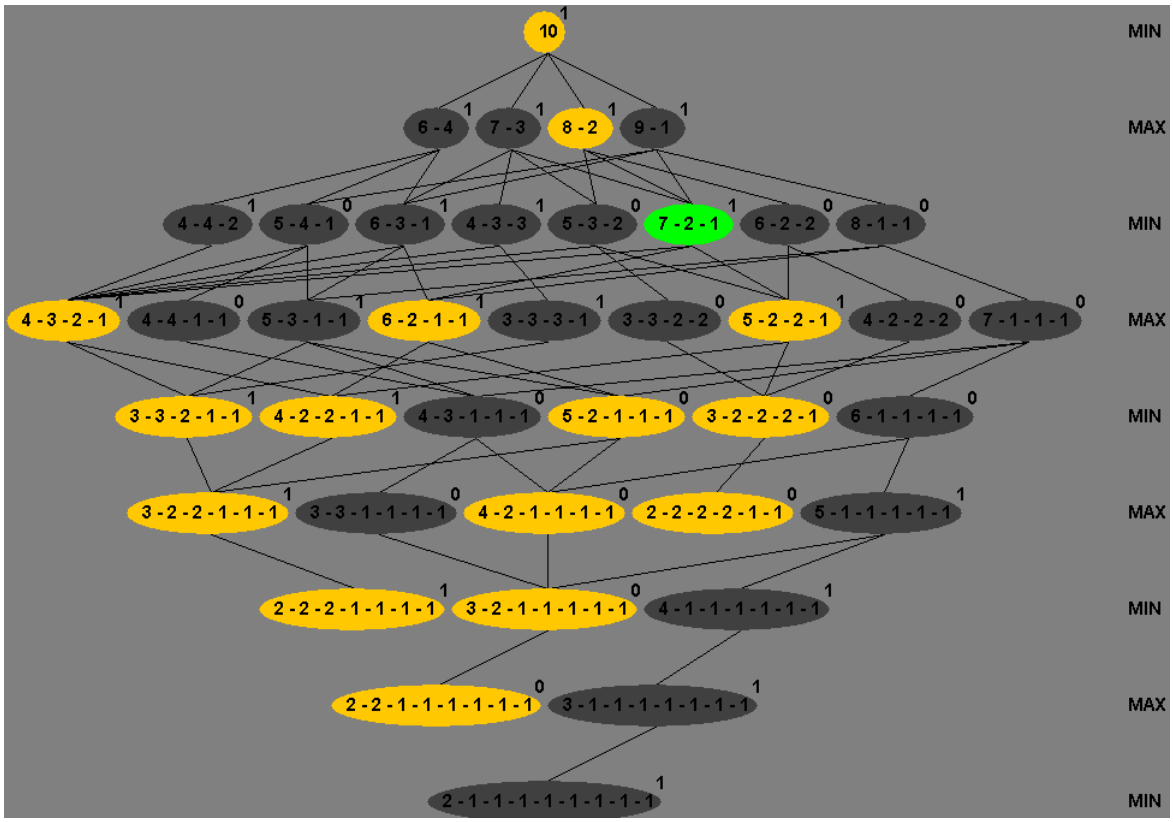
5.1 Grundyjeva igra

Na sliki 5.2 je prikazan izgled Grundyjeve igre. Igra se naključno prične z od 7 do 10 predmeti. Predmeti so razporejeni v eni kopici na levi strani polja. Uporabnik na vrsti klikne na predmet, kjer želi razdeliti kopico na dva dela. Vsi predmeti nad izbranim se premaknejo v novo kopico. Zaradi preglednosti se kopice uredijo po vrsti od največje do najmanjše. To nam olajša vpogled v drevo stanj. Po uporabnikovi potezi računalnik po sekundnem »premisleku« naredi svojo potezo. Če bi radi naredili neveljavno potezo, recimo kopico razdelili na dva enaka dela, se ne zgodi nič.



Slika 5.2: Grundyjeva igra z začetnimi desetimi predmeti po dveh potezah.

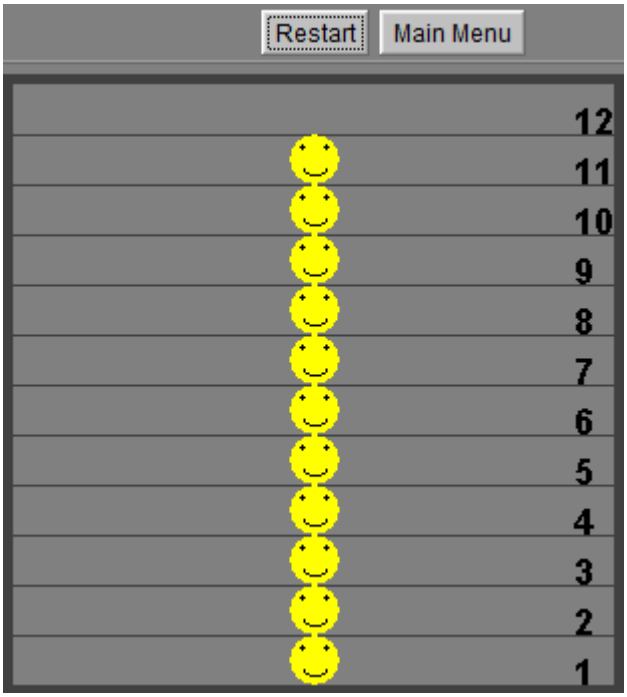
Drevo stanj za primer na sliki 5.2 je prikazano na sliki 5.3. Vozlišča so prikazana z obarvanimi elipsami, v vsaki je napisana vrednost stanja. Elipse so oranžne barve, če smo jih v prejšnjih potezah obiskali, ali če jih lahko še obiščemo. Vozlišča, ki jih nismo in jih ne moremo obiskati, so obarvana sivo. Vozlišče v katerem se trenutno nahajamo, je obarvano z zeleno barvo. Vsako vozlišče ima nad svojim desnim robom napisano vrednost, ki mu jo dodeli minimax. Te vrednosti so lahko 1 ali 0. Vozlišča so povezana s črnimi povezavami. Skrajno desno od drevesa nam oznake »MIN« in »MAX« prikazujeta, na katerem nivoju se vozlišče nahaja. Velikost elips se dinamično spreminja, glede na velikost besedila v njih. Ob vsaki potezi se drevo ustrezno osveži glede na spremembo pozicije.



Slika 5.3: Vizualizacija drevesa za primer na sliki 5.2.

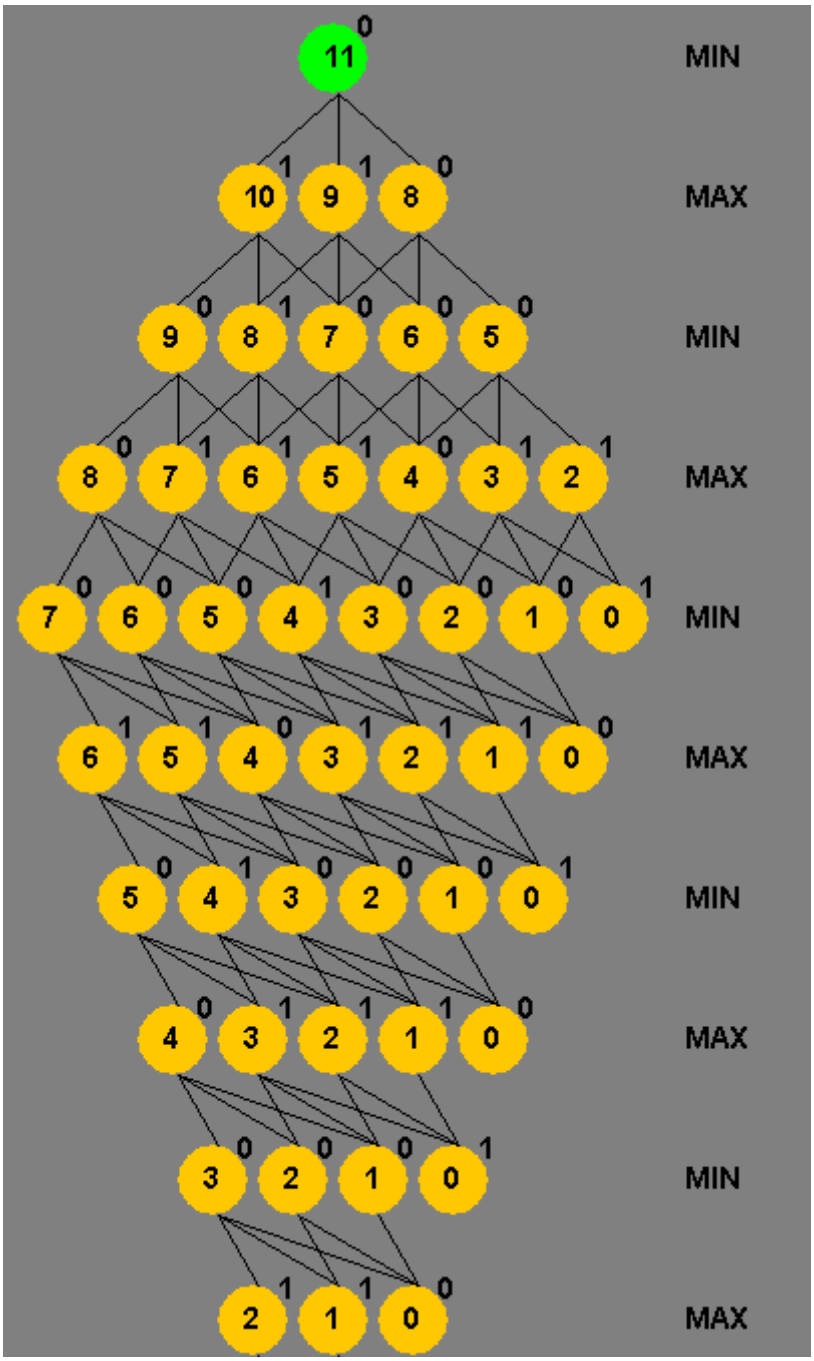
5.2 Igra odštevanja

Izgled igre odštevanja je prikazan na sliki 5.4. Število začetnih figur je izbrano naključno, od 9 do 12 predmetov. Igralna plošča je enaka tisti iz Grundyjeve igre. Kopica se postavi na sredini. Igralec lahko odstrani do 3 figure z eno potezo. To naredi tako, da klikne na enega od zgornjih treh predmetov. Če uporabnik poskuša napraviti neveljavno potezo, se ne zgodi nič.



Slika 5.4: Začetek igre odštevanja z 11 začetnimi predmeti.

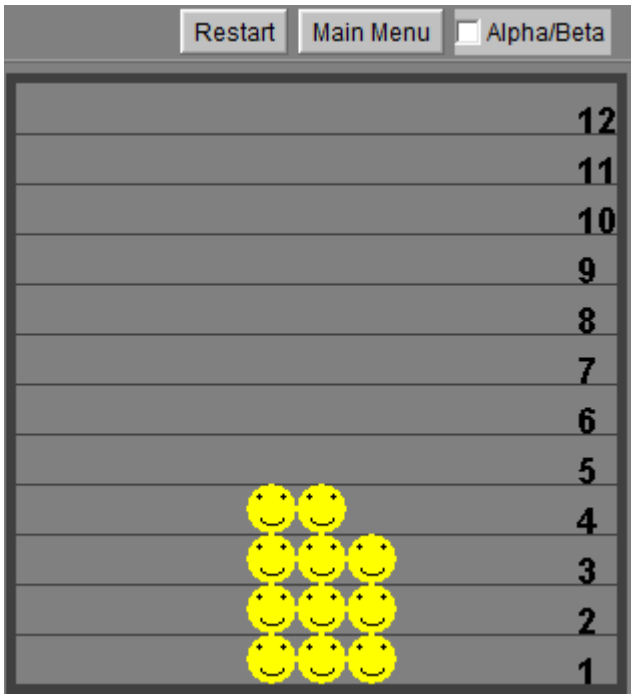
Drevo stanj je skoraj identično drevesu stanj za Grundyjevo igro. Vizualizacija dela drevesa za situacijo na sliki 5.4 je prikazana na sliki 5.5. Ker gre za začetni položaj, nimamo sivih vozlišč, saj so nam še vedno vsa dostopna. Prvo vozlišče je obarvano zeleno, ostala pa so oranžna. Širina vozlišč je vedno enaka, saj je zapis vrednosti vedno približno enak. Ob vsakem vozlišču, je desno njihova minimax vrednost. Drevo se ob vsaki potezi osveži.



Slika 5.5: Vizualizacija dela drevesa z začetnimi 11 predmeti za igro odštevanja.

5.3 Požrešni nim

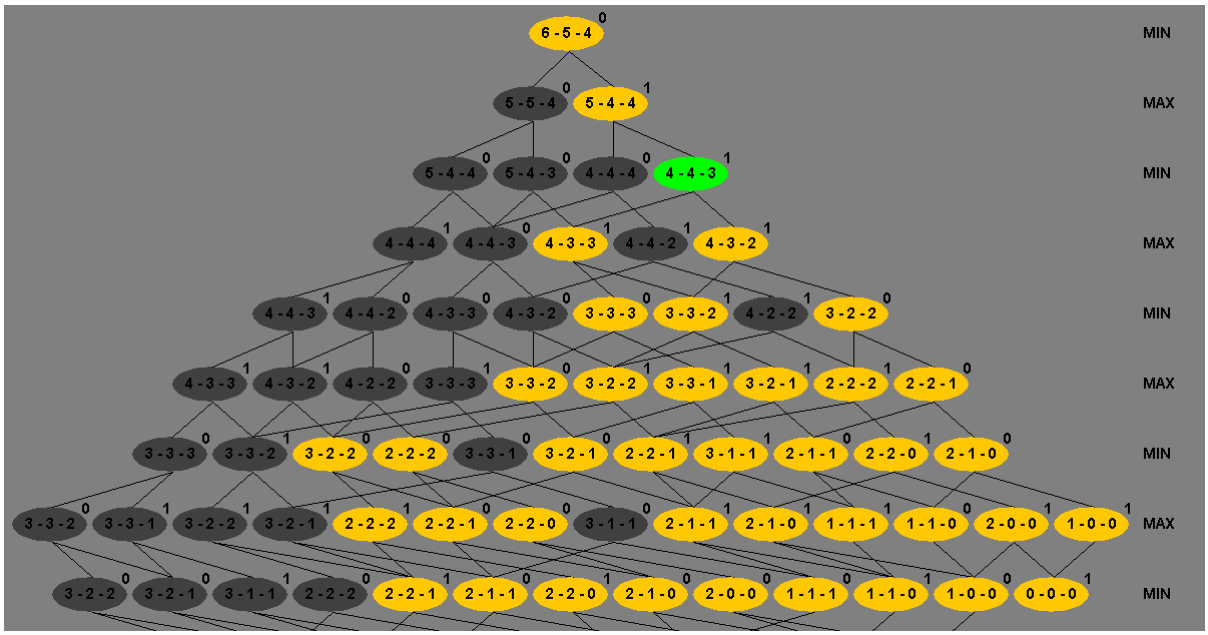
Igra požrešni nim se prične s tremi kopicami. V vsaki je lahko od 3 do 6 predmetov. Kopice so postavljene druga ob drugi na sredini igralne plošče. Uporabnik lahko odstrani do dva predmeta iz največje kopice. Odstranjevanje poteka enako kot pri igri odštevanja. Poleg gumbov za ponovni zagon in vrnitev na glavno stran imamo na voljo še kontrolno okno, ki nam prikaže delovanje alfa-beta rezanja. Kopice se zaradi preglednosti uredijo po velikosti od največje do najmanjše. Na sliki 5.6 je prikazana igra s kopicami, ki vsebujejo 4, 4 in 3 predmete.



Slika 5.6: Igra požrešni nim z začetnimi kopicami, ki vsebujejo 4, 4 in 3 predmete.

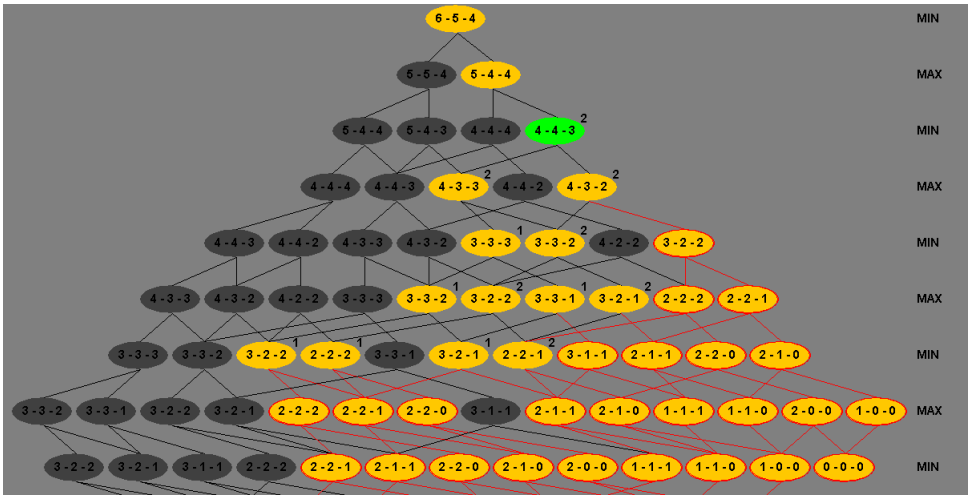
Dokler je kontrolno okno Alpha/Beta izključeno je drevo stanj enako, kot pri prejšnjih igrah (barve, povezave, napisi, širina vozlišč). Primer dela drevesa s kopicami, ki vsebujejo 6, 5 in 4 predmete, je prikazan na sliki 5.7.

Slika je bila zajeta po dveh potezah.



Slika 5.7: Vizualizacija dela drevesa za požrešni nim, s kopicami po 6, 5 in 4 predmetov.

V primeru, da je kontrolno okno Alpha/Beta vključeno se vizualizacija spremeni v delovanje alfa-beta rezanja. V desnem zgornjem kotu vozlišč niso napisane vrednosti minimax, ampak alfa-beta vrednosti vozlišča. Vozlišča so ocenjena z vrednostmi od 0 do 3 (in ne z 0 in 1). Vozlišča, ki smo jih z alfa-beta rezanjem odrezali so obrobljena z rdečo barvo. Povezave do njih so rdeče namesto črne. Vozlišča, ki so na globini, ki je za vsaj 5 nižja od globine trenutnega vozlišča, so obrobljena z rdečo, saj algoritem alfa-beta do njih nikoli ne pride. Z rdečo so torej obrobljena vozlišča, do katerih smo dostopali z algoritmom minimax, z alfa-beta rezanjem pa ne bomo. Vidimo, da nam alfa-beta rezanje prostor stanj, ki ga je potrebno pregledati, bistveno zmanjša. Primer alfa-beta drevesa za stanje identično tistemu iz slike 5.7, je prikazan na sliki 5.8.



Slika 5.8: Vizualizacija alfa-beta dela drevesa za enak primer kot na sliki 16.

Poglavje 6

Sklep

V diplomskem delu smo razložili delovanje načela minimax in alfa-beta rezanja. Opisali smo uporabljena orodja in arhitekturo aplikacije. Aplikacija vizualno predstavi delovanje obeh algoritmov. Namenjena je učenju in boljšemu razumevanju njunega delovanja, v izobraževalnih ustanovah, kjer poučujejo te algoritme. Prav bi prišla tudi razvijalcem iger, ki bi jih zanimalo njuno delovanje ali implementacija poteznih iger. Javanski programček je sestavljen iz 1265 vrstic kode, v treh razredih, ki vsebujejo 55 funkcij.

Programiranje grafike mi je bilo pred izdelavo diplomske naloge tuje, saj z njim nisem imel praktičnih izkušenj. Zaradi tega bi lahko vizualizacijo izboljšal. Smiselno bi bilo uporabiti orodja, ki bi olajšala delo ali izboljšala izgled igre. Tudi način igranja bi lahko poenostavili.

V prihodnosti bi lahko algoritem implementirali tudi na drugih poteznih igrah in primerjali učinkovitost delovanja med igrami. Lahko bi delovanje podobnih algoritmov primerjali z delovanjem minimaxa.

Verjamem, da bo diplomsko delo koristno za izobraževanje in vizualizacijo minimaxa in alfa-beta rezanja ter da bo marsikomu olajšalo razumevanje umetne inteligence v poteznih igrah.

Literatura

[1] (2013) Minimax. http://en.wikipedia.org/wiki/Minimax (Dostop 20.6.2013)

[2] (2002) Minimax Explained. http://ai-depot.com/articles/minimax-explained/ (Dostop 20.6.2013)

[3] Igor Kononenko, Marko Robnik Šikonja: Inteligentni sistemi. Založba FE in FRI, Ljubljana, 2010.

[4] (2013) Alpha-Beta pruning. http://en.wikipedia.org/wiki/Alpha-beta_pruning (Dostop 20.6.2013)

[5] NetBeans. https://en.wikipedia.org/wiki/NetBeans (Dostop 20.6.2013)

[6] (2011) Java^{TM} Platform, Standard Edition 6 API Specification. http://docs.oracle.com/javase/6/docs/api/overview-summary.html (Dostop 20.6.2013)

[7] (2013) HTML. https://en.wikipedia.org/wiki/HTML (Dostop 20.6.2013)

[8] (2012) Java Applet Tutorial. http://www.javakode.com/applets/ (Dostop 20.6.2013)

[9] (2013) Java applet. http://en.wikipedia.org/wiki/Java_applet (Dostop 20.6. 2013)

[10] (2013) NIM History. http://www.archimedes-lab.org/game_nim/nim.html (Dostop 20.6.2013)

[11] (2013) Nim. https://en.wikipedia.org/wiki/Nim (Dostop 20.6.2013)