ELEKTROTEHNIČKI FAKULTET
UNIVERZITET U BEOGRADU

INTELIGENTNI SISTEMI

Simulacija modifikovane igre Nim

Autor:
Milan Prokić

Broj indeksa:
695/2016

Odsek:
SI

Profesor:
Prof. dr Boško Nikolić

Asistent:
dipl. ing. Dragana Milovančević

Beograd
Januar 2018.

Predmet izveštaja je simulacija modifikovane igre Nim sa implementirane tri vrste računarskih igrača - jednostavni igrač, alfa beta igrač i takmičarski igrač.

Pravila modifikovane igre su sledeća:

- Postoji do 10 stubova sa žetonima
- Maksimalan broj žetona na stubu je 10
- Na svim stubovima mora postojati različit broj žetona u svakom stadijumu igre (osim 0)
- Na početku igre ne sme biti stubova bez žetona
- Potezi se povlače naizmenično, u svakom potezu se može skinuti određen broj žetona sa jednog stuba
- U prvom potezu se može skinuti neograničen broj žetona, u svim ostalim potezima broj skinutih žetona ne sme biti veći od dvostrukog broja žetona uzetih u prethodnom potezu od strane drugog igrača
- Igra se završava kada ni na jednom stubu nema preostalih žetona, pri čemu pobeđuje igrač koji je odigrao poslednji potez

Implementacija

Prema specifikaciji, postoje tri vrste računarskih igrača koji koriste različite algoritme bazirane na klasičnom minimax algoritmu, pri čemu za jednostavnog igrača i alfa beta igrača postoji mogućnost izbora različitih nivoa težine. Sva tri igrača su izvedena iz apstraktne klase *Igrac*. Pored toga, postoje dve klase koje se tiču same igre - *Game* i *Move*.

Klasa *Game*

Objekat klase *Game* pamti trenutno stanje na tabli - broj žetona na svakom stubu i broj žetona skinutih u prethodnom potezu.

Konstruktori:

```
public Game(int[] board)
```

board - niz u kom se nalaze informacije o broju žetona na svakom stubu
Inicijalizacija igre na samom početku, postavlja promenljivu *lastMove* na 1000, pošto se na početku igre može skinuti neograničen broj žetona. Parametar *board* predstavlja niz u kom se čuva trenutno stanje na tabli - na stubu *i* se nalazi *board[i]* žetona.

```
public Game(int[] board, int lastMove)
```

board - niz u kom se nalaze informacije o broju žetona na svakom stubu
lastMove - broj diskova skinutih u prethodnom potezu
Čuvanje trenutnog stanja igre nakon odigranog poteza. Niz se sortira pošto su sva stanja koja imaju iste brojeve žetona na stubovima i isti broj žetona skinut u prethodnom potezu identnična, nezavisno od rasporeda žetona.

Metode:

```
public int[] getBoard()
```

Vraća niz u kom se nalazi trenutno stanje igre.

```
public int getLastMove()
```

Vraća broj diskova skinutih u prethodnom potezu.

```
public boolean isOver()
```

Proverava da li je igra završena. Igra je završena kada ni na jednom stubu nema žetona. Vraća *true* ako je igri kraj i *false* ako nije.

```
public boolean isValid(Move m)
```

m - potez za koji treba proveriti validnost
Proverava da li je zadati potez dozvoljen. Potez je dozvoljen ako broj žetona koji se skida u datom potezu nije veći od dvostrukog broja žetona skinutih u prethodnom potezu i ako se skidanjem zadatog broja žetona ne dolazi do stanja na tabli u kom na dva stuba postoji

isti broj žetona.

public void makeMove(Move m)

m - potez koji treba odigrati
Proverava da li je zadati potez dozvoljen i ako jeste menja stanje na tabli shodno informaciji u potezu - skida odgovarajući broj žetona sa odgovarajućeg diska i postavlja promenljivu *lastMove* na broj diskova skinutih u datom potezu. Ukoliko potez nije validan, što je logička greška na strani programera, program se prekida sa statusom 10.

Klasa *Move*

Objekat klase *Move* sadrži kompletnu informaciju o jednom potezu - broj žetona koji treba skinuti i redni broj stuba sa kog treba skinuti žetone, kao i (u slučaju AI igrača) procenjenu vrednost datog poteza i informaciju o tome da li je potez odigrao maksimizer ili minimizer. Sam potez nema informaciju o trenutnom stanju na tabli, pa samim tim ne garantuje validnost.

Konstruktori:

public Move(**int** pos, **int** numDiscs)

numDiscs - broj žetona koje treba skinuti
pos - redni broj stuba sa kog treba skinuti žetone
Postavlja broj žetona koje treba skinuti i poziciju sa koje ih treba skinuti na *numDiscs* i *pos*, respektivno, i procenjenu vrednost poteza na -2000.

public Move(**int** pos, **int** numDiscs, **boolean** isMax)

numDiscs - broj žetona koje treba skinuti
pos - redni broj stuba sa kog treba skinuti žetone
isMax - informacija o tome da li je potez odigrao maksimizer ili minimizer
Postavlja broj žetona koje treba skinuti i poziciju sa koje ih treba skinuti na *numDiscs* i *pos*, respektivno i čuva informaciju o tome da li je potez odigrao minimizer ili maksimizer prema parametru *isMax*.

Metode

public int getPos()

Vraća poziciju sa koje treba skinuti žetone

public int getNumDiscs()

Vraća broj žetona koje treba skinuti

public int getValue()

Vraća procenjenu vrednost poteza

public boolean getIsMax()

Vraća *true* ako je potez odigrao maksimizer, u suprotnom *false*

public void setPos(**int** pos)

Postavlja poziciju sa koje treba skinuti diskove na *pos*

public void setValue(**int** value)

Postavlja procenjenu vrednost poteza na *value*

Klasa *Igrac*

Klasa *Igrac* je apstraktna klasa iz koje se izvode klase AI igrača. Sadrži informaciju o maksimalnoj dubini pretrage stabla, koja je vezana za izbor težine. Testiranjem je utvrđeno da su optimalne vrednosti za dubine sledeće: easy - 3, medium - 5, hard - 7.

Konstruktori:

public Igrac(**int** maxDepth)

maxDepth - maksimalna dubina pretrage stabla
Postavlja maksimalnu dubinu pretrage na *maxDepth*

Metode:

public int getMaxDepth()

Vraća maksimalnu dubinu pretrage stabla.

public abstract Move makeMove(Game g)

g - trenutno stanje na tabli
Određuje i vraća optimalan potez za dato stanje na tabli.

Klasa *JednostavanIgrac*

Objekat klase *JednostavanIgrac* određuje optimalan potez koristeći minimax algoritam i datu statičku funkciju procene $f = m_1 \oplus m_2 \oplus ... \oplus m_n$. Pobeda i poraz su kodirani kao ± 100. U idealnom slučaju, oba igrača žele da dovedu funkciju procene do nule u svom potezu, ali pošto to nije uvek moguće zbog ranije pomenutih ograničenja, smatra se da će oba igrača minimizovati funkciju procene. Kako je minimizovanje funkcije procene od strane suprotnog igrača nepovoljno, dobijena vrednost će se negirati ako ne predstavlja terminalno stanje (pobedu ili poraz). Ovim se omogućava da maksimizujući igrač dobija vrednosti u opsegu $[-100;0] \cup [+100]$, a da minimizujući igrač dobija vrednosti u opsegu $[-100] \cup [0;+100]$, nad kojima primenjuju klasičan minimax algoritam. Tokom određivanja optimalnog poteza u obzir se uzimaju samo validni potezi, pri čemu se u samom početku za optimalan potez uzima prvi validan na koji se naiđe, čime se garantuje vraćanje validnog poteza.

Algoritam na najvišem nivou prolazi kroz sve validne poteze za trenutno stanje na tabli i za svaki od tih poteza poziva metodu *minimax*, sa argumentom trenutne dubine 1, koja će rekurzivnim pozivima, sa inkrementiranjem dubine, odrediti vrednosti svih mogućih poteza koji mogu biti odigrani iz tog stanja, pri čemu će u slučaju da dostigne maksimalnu dubinu, a da tekuće stanje nije terminalno, sračunati vrednost stanja pomoću statičke funkcije procene. Ukoliko se dođe do terminalnog stanja, funkcija će vratiti odgovarajuću vrednost u zavisnosti od toga koji igrač pobeđuje. U slučaju da nije dostignuto terminalno stanje ni maksimalna dubina, vrednost optimalnog poteza se postavlja na ± 2000, u zavisnosti od toga koji igrač je poziva. Ova vrednost će se menjati pri izlascima iz rekurzivnih poziva funkcije, tako da će uvek imati optimalnu vrednost za zadatog igrača, i ta vrednost će prilikom izlaska iz najvišeg nivoa poziva biti vraćena metodi *makeMove*. U metodi *makeMove* se takođe prati vrednost optimalnog poteza, kao i sam potez koji je doveo do te vrednosti. Svaki put kada metoda *minimax* vrati neku vrednost, ona se upoređuje sa trenutno optimalnom, pa ukoliko je bolja, ta vrednost postaje trenutno optimalna, a potez koji je doveo do ove vrednosti postaje trenutni optimalan potez. Nakon što se odredi optimalan potez, on se vraća pozivaocu metode.

Konstruktori:

```
public JednostavanIgrac(int maxDepth)
```

maxDepth - maksimalna dubina pretrage stabla
Postavlja maksimalnu dubinu pretrage stabla na *maxDepth*.

Metode:

```
public Move makeMove(Game g)
```

g - trenutno stanje na tabli
Određuje optimalan potez za dato stanje na tabli.

```
private int minimax(Game g, int depth, int lastMove, boolean isMax)
```

g - trenutno stanje na tabli
depth - dubina na kojoj se nalazi trenutni poziv
lastMove - broj žetona skinutih u prethodnom potezu
isMax - da li metodu poziva maksimizer ili minimizer
Vraća vrednost optimalnog poteza za dato stanje na tabli.

Klasa *AlfaBetaIgrac*

Objekat klase *AlfaBetaIgrac* određuje optimalan potez koristeći isti algoritam kao klasa *JednostavanIgrac*, sa poboljšanjem u vidu alfa beta odsecanja. Pri pozivu metode *makeMove*, vrednosti parametara *alpha* i *beta* se postavljaju na ∓ 2000, respektivno. Parametar *alpha* predstavlja vrednost ispod koje maksimizujući igrač neće ići, a paramtetar *beta* predstavlja vrednost iznad koje minimizujući igrač neće ići. Ove vrednosti će se propagirati i menjati tokom poziva metode *minimax*, pri čemu će maksimizujući igrač menjati vrednost parametra *alpha*, a minimizujući igrač parametra *beta*. Ukoliko u

nekom trenutku dođe do preklapanja parametara *alpha* i *beta*, sprovešće se odsecanje i neće se ulaziti u ostale grane podstabla u kom je sprovedeno odsecanje.

Konstruktori:

```
public AlfaBetaIgrac(int maxDepth)
```

maxDepth - maksimalna dubina pretrage stabla
Postavlja maksimalnu dubinu pretrage stabla na *maxDepth*.

Metode:

```
public Move makeMove(Game g)
```

g - trenutno stanje na tabli
Koristi minimax algoritam sa alfa beta odsecanjem da odredi optimalan potez za dato stanje na tabli.

```
private int minimax(Game g, int lastMove, int depth,
                    int alpha, int beta, boolean isMax)
```

g - trenutno stanje na tabli
lastMove - broj žetona skinutih u prethodnom potezu
depth - dubina na kojoj se nalazi trenutni poziv
alpha - vrednost ispod koje maksimizer neće ići
beta - vrednost iznad koje minimizer neće ići
isMax - da li metodu poziva maksimizer ili minimizer
Vraća vrednost optimalnog poteza za dato stanje na tabli koristeći minimax algoritam sa alfa beta odsecanjem.

Klasa *Takmicar*

Objekat klase *Takmicar* određuje optimalan potez koristeći činjenicu da su stanja koja imaju iste brojeve žetona na stubovima i isti broj žetona skinut u prethodnom potezu identična, nezavisno od rasporeda žetona, što omogućava primenu dinamičkog programiranja u vidu pretrage stabla samo za stanja koja do tada nisu obrađena, pri čemu se po otkriću optimalnog poteza to stanje i njemu pridružen optimalni potez čuvaju u odgovarajućoj strukturi. Pre nego što se pristupi pretrazi stabla, prvo se proverava da li je dato stanje već obrađeno, i ako jeste, uzima se njemu pridružen optimalni potez. Ovim se drastično smanjuje prostor pretrage. Pretraga se dodatno ubrzava ranim zaustavljanjem - ako se u nekom trenutku do korena stabla pretrage propagira pobeda za pozivaoca, nema potrebe pretraživati ostala podstabla jer se u najboljem slučaju može pronaći samo putanja koja u manjem broju koraka vodi do pobede, nikako bolje rešenje od već postojećeg. Zbog svega navedenog, uvođenje nivoa težine bi bilo kontraproduktivno za ovog igrača, pošto se efekti različitih nivoa težine ne bi mnogo razlikovali od efekata pri korišćenju alfa beta igrača. Pošto se stanja čuvaju u sortiranom poretku, pridruženi potezi su specifični za sortirano stanje, pa je neophodno pronaći odgovarajući stub u nesortiranom stanju nad kojim treba odigrati potez. Ovo se radi tako što se iz sortiranog stanja dobije broj žetona na stubu

na koji ukazuje potez, a onda se u nesortiranom stanju pronađe pozicija stuba sa istim brojem žetona. Rezultat je jednoznačan jer, prema pravilima igre, ne mogu postojati dva stuba sa istim brojem žetona.

Konstruktori:

```
public Takmicar()
```

Metode:

```
public Move makeMove(Game g)
```

g - trenutno stanje na tabli
Određuje optimalan potez koristeći činjenicu da su stanja sa istim brojem žetona na stubovima i istim brojem žetona skinutim u prethodnom potezu identična, nezavisno od rasporeda žetona.

```
private int minimax(Game g, int lastMove, boolean isMax)
```

g - trenutno stanje na tabli
lastMove - broj žetona skinutih u prethodnom potezu
isMax - da li metodu poziva maksimizer ili minimizer
Vraća vrednost optimalnog poteza za dato stanje na tabli koristeći činjenicu da su stanja sa istim brojem žetona na stubovima i istim brojem žetona skinutim u prethodnom potezu identična, nezavisno od rasporeda žetona.

```
private int findSortedPos(int discsOnPile, Game state)
```

discsOnPile - broj diskova na stubu
state - stanje na tabli
Vraća poziciju stuba u sortiranom poretku