Dots and Boxes

Inteligentni sistemi

Anja Marković

0420/2017

Kratak prikaz implementiranih algoritama i njegovih funkcija

Za izradu naprednog i takmičarskog igrača korišćen je Minimax algoritam sa alfa – beta odsecanjem.

Cilj ovog algoritma je da odabere najbolji potez , na taj način se simulira razmišljanje unapred.

Za procenu kvaliteta mogućih poteza korišćena je statička funkcija procene, koja pokazuje trenutno stanje, koliko je igrač blizu pobede. Iskazuje se brojem poena iz opsega koji igrač može da dobije na kraju igre, odnosno predstavlja razliku osvojenih poena oba igrača. Funkcija procene takođe zavisi i od toga koji je igrač na potezu, MIN ili MAX. Pa se tako funkcija procene množi sa -1 u slučaju da je na potezu MIN igrač.

Dubinu pretrage kod naprednog igrača korisnik bira na početku igre, dok je ona uzeta na neku statičku vrednost kod takmičarskog igrača, s obzirom na to da su napravljene optimizacije, pa mu nije potrebna velika dubina kao kod naprednog igrača, pa se samim tim i brže izvršava.

Pošto je vreme izvršavanja veći problem od prostora, sam Minimax algoritam je optimizovan korišćenjem alfa – beta odsecanja, tako što se vrednost procene ograniči donjom i gornjom granicom.

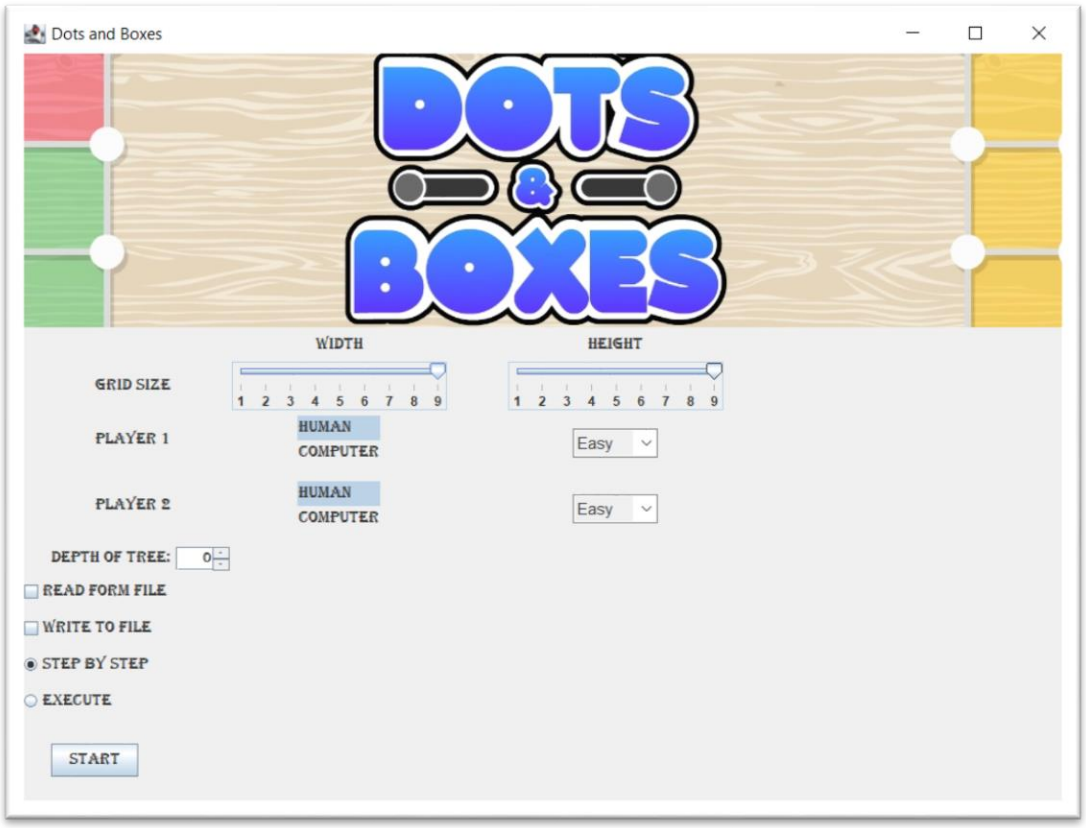
Alfa je vrednost najpovoljnijeg poteza po nas koji je do sada pronađen. Predstavlja donju granicu procene poteza koji možemo da prihvatimo.

Beta predstavlja gornju granicu koju možda možemo da dostignemo.

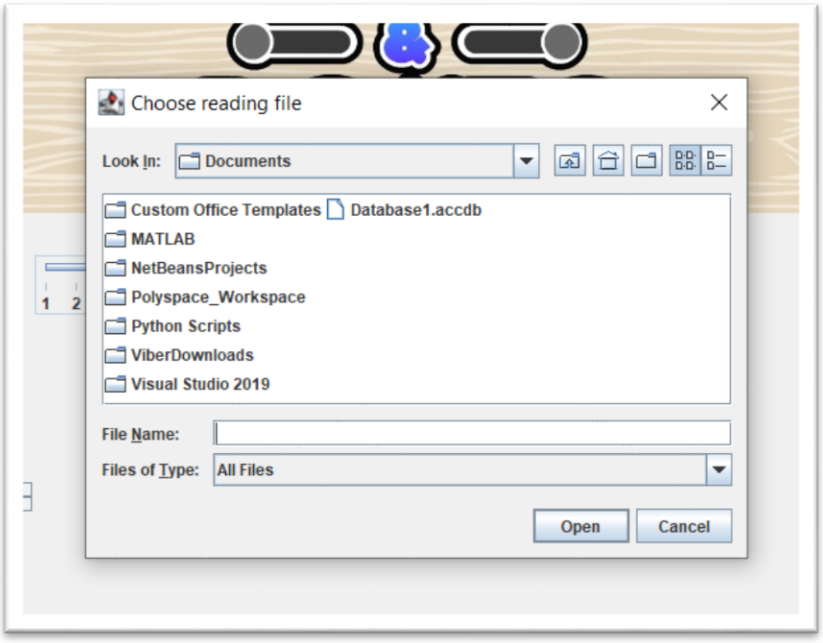
Na osnovu ovih vrednosti eliminišemo podstablo gde je vrednost mogućeg poteza manja od alfa i podstablo gde je vrednost mogućeg poteza veća od beta. Na taj način se stablo igre značajno smanji, a samim tim I vreme izvršavanja.

Pregled klasa i njenih metoda

```
//glavna klasa, ujedno i početni prozor, sadrži main funkciju
public class Main extends JFrame implements ActionListener {
        //konsturktor klase u kome se postavljaju vrednosti svih labela i dugmadi
        public Main();
        //metoda koja se poziva na pritisak dugmeta “Start”
        public void actionPerformed(ActionEvent ae);
        //glavna metoda koja pravi objekat ove klase
        public static void main(String[] args);
}
```



Početni prozor igre



Prozor u kome se bira fajl za čitanje (isto je i za čitanje)

```
//klasa koja predstavlja prozor igre
public class GamePlay extends JFrame {
        //privatna klasa koja predstavlja prozor u kome se ispisuju odigrani potezi
        private class TextAreaClass extends JFrame {
        //geter za broj osvojenih poena plavog igrača
        public int getBlueScore();
        //geter za broj osvojenih poena crvenog igrača
        public int getRedScore() ;
        //geter za prvog igrača
        public Player player1();
        //geter za drugog igrača
        public Player player2() ;
        //metoda koja postavlja na prozor komponente za prikaz stanja igre
        public void addComponents();
        //geter za objekat klase FileIO
        public FileIO getFileIO();
```

```
        //metoda koja postavlja vidljivost labela
        public void enableLables(boolean enable1) ;
        //metoda koja postavlja vrednost trenutnog rezultata na prozor
        public void setScore(boolean scored1, int num);
}

//klasa koja služi za iscrtavanje igre, predstavlja awt platno
public class Board extends Canvas {
        //privatna statička klasa koja predstavlja polje table
        //sadrži geterske i seterske metode boje i koordinata, i metodu za iscrtavanje na platnu
        private static class Tile {}
        //privatna statička klasa koja predstavlja linije koje se mogu povući na tabli igre
        //sadrži geterske i seterske metode, kao i metodu za iscrtavanje na platnu
        public static class Edge {}
        //geterska metoda koja dohvata boolean da li je prvi igra; na redu
        public boolean isTurn1();
        //geterska metoda za sve horizontalne linije
        public Edge[][] getHorizontal() ;
        //geterska metoda za sve vertikalne linije
        public Edge[][] getVertical() ;
        //geterska metoda koja dohvata trenutni objekat klase GamePlay
        public GamePlay getGamePlay() ;
        //metoda koja boji odredjeno polje na tabli i odredjuje broj osvojenih poena u potezu
        public int colorTile(Edge edge, int i, int j);
        //metoda koja odigra zadati potez
        public void makeMove(Edge edge, int i, int j);
        //metoda za iscrtavanje cele table igre
```

```
        public void paint(Graphics g);
}
```

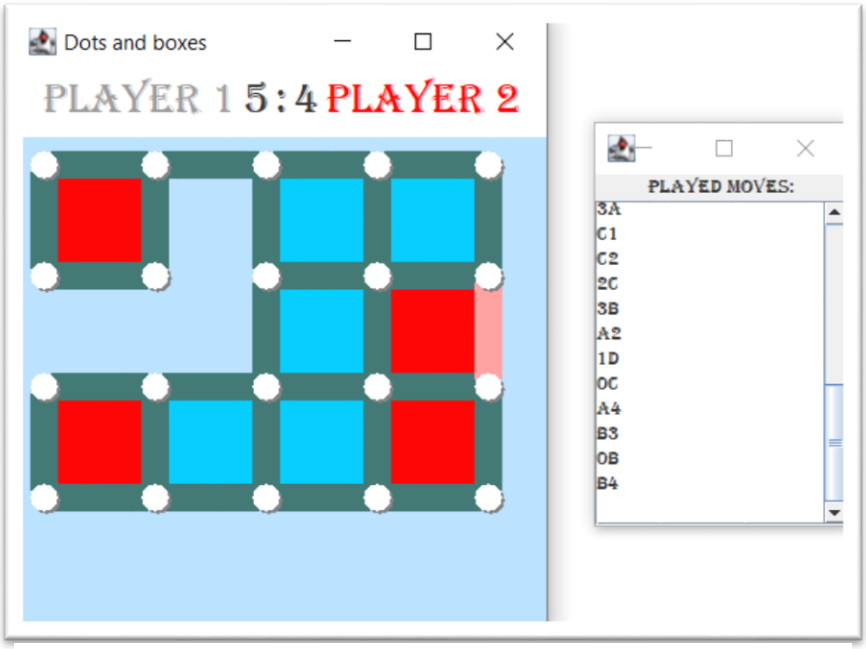
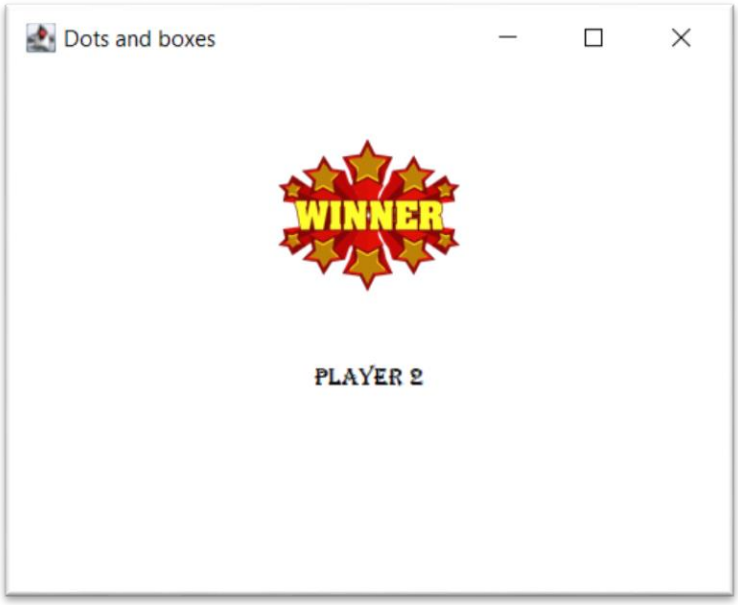


Tabla igre sa prikazanim trenutnim stanjem igre

```
//klasa koja predstavlja krajnji prozor igre I sadrži samo konstruktor
public class EndWindow extends JFrame {}
```



Krajnji prozor igre

```
//klasa koja upisuje i čita iz fajla
public class FileIO {
	//geter za dohvatanje fajla za čitanje
	public File getReadFile();
	//metoda koja služi za čitanje iz fajla
	public boolean read();
	//metoda koja služi za upis u fajl
	public void write(String text);
	//seter za TextArea za ispis odigranih poteza
	public void setTextArea(JTextArea ta);
	//geter za visinu table
	public int getM();
	//geter za širinu table
	public int getN();
	//geter koji dohvata listu svih poteza koji treba da se odigraju, pročitanih iz fajla
	public ArrayList<String> getMoves();
	//metoda koja pravi heš mapu za mapiranje stringova u stvarne vertikalne i horizontalne
	//linije i obrnuto
	public void makeHashMap(Edge[][] horizontal, Edge[][] vertical);
}

//klasa koja simulira rad igrača u zavisnosti od režima rada (korak po korak/sve odjednom)
public class Player extends Thread {
	public void run();
	//metoda za nastavak rada niti
	public synchronized void continueThread();
```

```
        //metoda za pauziranje niti
        public void pauseThread() ;
        //metoda za prekidanje niti
        public void stopThread() ;
        //geter koji dohvata da li je igrač bot ili čovek
        public boolean isBot();
}

//apstraktna klasa koja predstavlja strategiju igranja igrača
public abstract class GameSolver {
        //apstraktna metoda koja dohvata liniju koju treba da odigra igrač
        public abstract Edge getNextMove();
        //metoda koja formira dohvata granu kojom može da se formira kvadrat ukoliko
        //ona postoji
        public Edge formSquare()
}

//klasa koja predstavlja igrača početnika
public class Beginner extends GameSolver {
        //metoda koja dohvata granu kojom može da zatvori kvadrat ukoliko ona postoji, a ako
        //ne postoji, onda vraća bilo koju drugu granu koja moče da se odigra
        public Edge getNextMove();
}

//klasa koja predstavlja naprednog igrača
public class Medium extends GameSolver {
        //preklopljena metoda koja dohvata naredni potez na osnovu funkcije procene
```

```
        //koja se dobija pomoću Minimax algoritma sa alfa – beta odsecanjem
        public Edge getNextMove();
        //Minimax algoritam sa alfa – beta odsecanjem
        public int minimaxAlphaBeta(GameState currentState, int maxDepth, int currentDepth,
        int alpha, int beta);
}

//klasa koja predstavlja takmičarskog igrača
public class Competitive extends GameSolver {
        //preklopljena metoda za dohvatanje narednog poteza
        public Edge getNextMove();
        //minimax algoritam sa alfa – beta odsecanjem
        public int minimaxAlphaBeta(GameState currentState, int maxDepth, int currentDepth,
int alpha, int beta) {
        //metoda koja proverava da li prosledjena linija predstavlja dodavanje treće linije
        //kvadratu
        private boolean isThirdEdge(Edge e);
}

//klasa koja predstavlja trenutno stanje igre u određenom čvoru stabla Minimax algoritma
public class GameState {
        //metoda za kopiranje matrica vertikalnih i horziontalnih linija
        private Edge[][] copyMatrix(Edge[][] matrix, int m, int n);
        //privatna metoda koja proverava da li odigrana grana predstavlja igranje treće grane
        private boolean isThirdEdge(Edge e);
        //geter za max igrača
        public boolean isMaxPlayer();
        //geter za min igrača
```

```
        public boolean isMinPlayer();
        //metoda koja određuje da li je trenutno stanje krajnje stanje igre
        public boolean isTerminalState();
        //metoda koja proverava da li je u trenutnom čvoru sa odigranom granum igrač osvojio
        //poen
        public boolean isScored();
        //metoda koja postavlja rezultate za plavog I za crvenog igrača u skladu sa stanjem igre
        public void setScore();
        //metoda koja dohvata statičku funkciju procene za dati čvor i trenutno stanje igre
        public int heuristic();
        //metoda koja pravi čvorove decu u minimax stablu
        public GameState createChild();
        //metoda koja proverava da li postoji naredni potez
        public boolean hasNextMove();
        //metoda za ispis trenutnog stanja igre (ispisuje odigranu granu)
        public String toString();
}
```