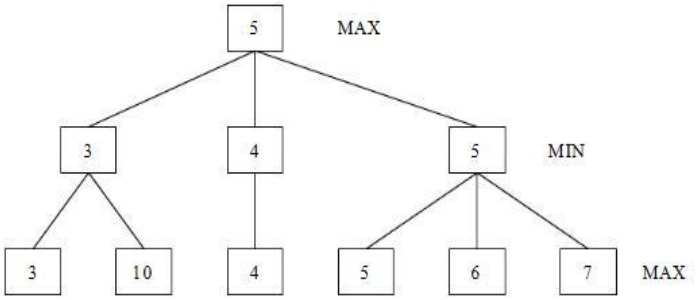
Izveštaj

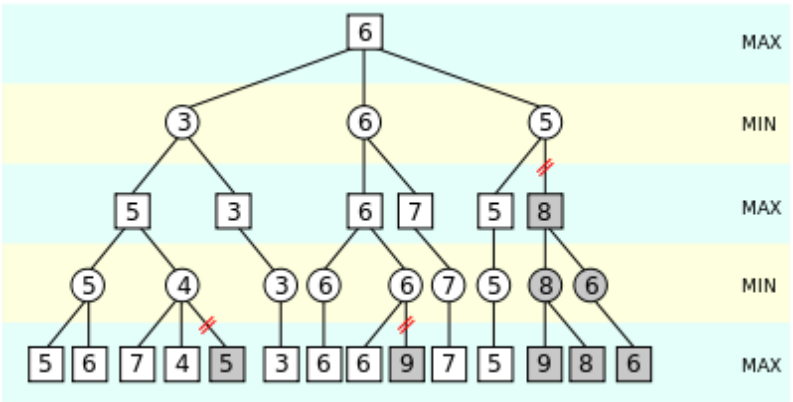
Prikaz koriscenog algoritma

Svaka potezna igra se moze predstaviti stablom igre. Svaki cvor stabla predstavlja jednu poziciju u igri a grane moguće poteze. Potezi preslikavaju jednu poziciju na tabli u drugu. Svakom igracu odgovara neki nivo stabla. Naime, ukoliko je prvom igracu odgovarao nivo N, onda će drugom igracu odgovarati nivo N+1. Neke od pozicija nemaju moguće poteze. Ove pozicije nazivaju se terminalne pozicije. U ovim pozicijama svaki igrac dobija odredjeni rezultat (na primer, za iks-oks igru, terminalna stanja su ona u kojima su sva polja popunjena ili je jedan od igraca spojio tri simbola). Broj grana od svakog cvora u stablu jednak je broju mogućih poteza u tom stanju i naziva se faktor grananja. Faktor grananja dobar je indikator koliko će sama igra biti komplikovana kompjuteru za igranje.

Ideja kod minimax algoritma je da moguci potez biramo pretragom unapred. Izaberemo jedan od mogucih poteza, pa zatim protivnikov odgovor na nas potez, pa nas odgovor i sve tako. Kada biramo za na sebe potez, biramo najbolji moguci. Kada nas protivnik bira, bira najbolji za sebe odnosno najgori za nas.



Postoji poboljsanje ovog algoritma, gde smanjujemo broj cvorova u grafu. To radimo tako sto ogranicimo vrednost procene donjom i gornjom granicom. Alfa je vrednost najpovoljnijeg poteza po nas koji je do sada pronadjen i predstavlja donju granicu procene poteza koji mozemo da prihvatimo. Nikada necemo odigati losiji potez od najboljeg do tada pronadjenog(alfa). Kasnije mozemo pronaci bolji potez, ali ako nas protivnik natera na losiji potez od alfa, mi ga necemo odigrati. Slicno tako i beta predstavlja gornju granicu, odnosno vrednost koju mozda mozemo da dostignemo. Sigurno ne mozemo vise od toga, tako da eliminisemo podstablo gde je vrednost moguceg poteza veca od beta.



Prikaz klasa

Od klasa imamo:

Easy-klasa koja predstavlja prostog AI igraca, koristi samo minimax algoritam

```
public int calculateFunction(int poles, int[] tolkens)-racunanje staticke funkcije procene
public void turn(int numPoles, int[] tolkens,GameActivity gameActivity)-odigravanje poteza, gde se formira root cvor, pronalazi najbolji sledeci potez i na kraju obavestava protivnik da je na njega red
private int minimax(Node node,int treeDepth,boolean isMaxPlayer)-implementacija minimax algoritma
private void constructTree(Node node)-konstruisanje dece root cvora i pozivanje minimax algoritma
```

Inter-klasa koja predstavlja slozenijeg AI igraca sa poboljsanim perfomansama, koristi alfa-beta odsecanje

```
public int calculateFunction(int poles, int[] tolkens)- racunanje staticke funkcije procene

public void turn(int numPoles, int[] tolkens,GameActivity gameActivity) )-odigravanje poteza, gde se formira root cvor, pronalazi najbolji sledeci potez i na kraju obavestava protivnik da je na njega red
private int minimax(Node node,int treeDepth,boolean isMaxPlayer,int alpha, int beta)-implementacija minimax algoritma sa alfa-beta odsecanjem
private void constructTree(Node node) )-konstruisanje dece root cvora i pozivanje minimax algoritma
```

Pro-klasa koja predstavlja profesionalnog AI igraca, pokazuje se bolje od prethodna dva igraca

```
public int calculateFunction(int poles, int[] tolkens)- racunanje staticke funkcije procene
public void turn(int numPoles, int[] tolkens,GameActivity gameActivity) )-odigravanje poteza, gde se formira root cvor, pronalazi najbolji sledeci potez i na kraju obavestava protivnik da je na njega red
private int minimax(Node node,int treeDepth,boolean isMaxPlayer,int alpha, int beta)-implementacija minimax algoritma sa alfa-beta odsecanjem i sa dodatnim poboljsanjem
private void constructTree(Node node) )-konstruisanje dece root cvora i pozivanje minimax algoritma
```

Player-apstraktna klasa iz koje su izvedeni svi podrzani tipovi igraca

```
public int calculateFunction(int poles, int[] tolkens)-apstraktna funkcija
public void turn(int numPoles, int[] tolkens,GameActivity gameActivity)-apstraktna funkcija
protected void makeMove(Node next)-„odigravanje" poteza, tj. Prelazak iz jednog cvora u drugi
public void setOpponent(Player player)-setter
public void setTurn(GameActivity gameActivity)-setter
public Player getOpponent()-getter
public void ready()-setter
public boolean getReady()-getter
```

```
public void setTreeDepth()-setter
public void endTurn()-setter
public boolean getTurn()-getter
public int checkEnd(int numPoles, int[]tolkens)-provera da li je u pitanju
terminalno stanje igre
```

Human-predstavlja korisnikovog igraca

```
public int calculateFunction(int poles, int[] tolkens)-nikad se ne poziva za ovu
klasu
public void turn(int numPoles, int[] tolkens,GameActivity gameActivity)-nista ne
radi, potez odradjen preko listenera za dugme
```

Node-klasa koja reprezentuje cvor u grafu minimax algoritma

```
public Node getBestChild()-dohvata najboljes sina za dati cvor(this)
public int finalState()-provera da li je u pitanju terminalno stanje igre
public int getPoles()-getter
public int[] getTolkens()-getter
public List<Node> createChildren()-funkcija stvara sve cvorove koji, po pravilima
igre, mogu proizaci iz trenutnog cvora
public int getFunction()-getter
public boolean isMax()-dohvata tip igraca koji je u datom cvoru(min ili max)
public void setFunc_value(int value)-setter
```

StartActivity-pocetni “ekran”, sluzi za odabir stanja igre

```
protected void onCreate(Bundle savedInstanceState)-poziv pri stvaranju
public void leftClick(View view)-funkcija koja se poziva pritiskom na dugme „Begin
game“, stvara se intent i salju se svi potrebni podaci u Gameactivity
```

MyImage-klasa izvedena iz ImageView, sluzi pri prisanju zetona sa stubova

```
public void setMatrix(int i,int j)-datu sliku umece u staticku listu svih slika i
zadaje joj mesto u „matrici“-sluzi kao id
public static MyImage getMatrix(int i, int j)-dohvata sliku sa zadatim id-om
public int getI()-getter
public int getJ()-getter
```

GameActivity-“ekran” igrice

```
private void initialize(Intent intent)-prihvatanje podataka iz prethodnog activity-
ja
protected void onCreate(Bundle savedInstanceState)-poziv pri stvaranju
public static int checkMove(int pole, int number, int numPoles,int[] tolkens,Player
p1,Player p2)-proverava da li je korisnikov potez dozvoljen
public static void declareVictor(Context context)-proglasava pobednika partije
public static void repaint(int pole, int from, int to)-nakon odigranog poteza,
uklanjanje zetona koji su visak
public void switchColors()-menjanje boje kod teksta za „player1“ i „player2“,
govori nam koji igrac je na potezu
public static int checkState(int numPoles, int[] tolkens)-provera da li je zadato
stanje dozvoljeno, poziva se iz checkMove
private void illegalMove()-pruza informaciju da zeljeni potez nije po pravilima
igre
protected void onPause()-brisanje prethodnog stanje, poziva se kada activity vise
nije vidljiva
private void paint()-crtanje pocetne situacije
```

public void clickMachine(View view)-daje informaciju masini da je sledeca na potezu, resen problem oko crtanja
public static int[] getTolkens()-getter
public static int getNumPoles()-getter
public void leftClick(View view)-poziva se kada korisnik pritisne na dugme za odigravanje poteza