Izveštaj

Klasa **TheGame**: glavna klasa iz koje se pokreće igrica.

```
public static void main(String[] args);
```

Klasa **Board**:

`public Board()` – glavni konstruktor

`public Board(Board board)` – kopirajući konstruktor potreban za minimax/alfa-beta

`public void makeMove(Move move)` – poziva se samo iz minimaxa i alfa-beta algoritma; poteže zadati potez

`public void makeMove_justMove(Move move)` – poteže samo prvi deo poteza (pomeranje); razdvojeno zbog prikaza i Human igraca

`public void makeMove_justBuild(Move move)` – poteže samo drugi deo poteza (gradnju)

`public void initPosition(Move move)` – postavlja jednu figuricu na zadatu pocetnu poziciju na tabli

`public int isTheEnd()` – provera kraja igre; vraca 0 ako nije kraj, 1 ako je pobedio prvi igrac, 2 ako je pobedio drugi igrac

`public void loadFromFile()` – ucitava stanje igre iz tekstualnog fajla

`private void writeToFile(Move move, Position prevPos)` – upisuje potez u izlazni fajl; poziva se nakon svakog povucenog poteza

`private void writeToFileInit(Move move)` – upise u izlazni fajl inicijalnu poziciju figurice

`private String PosToString(Position p)` – konvertuje poziciju u string format koji se upisuje u izlazni fajl

`private Position StringToPos(String str)` – za citanje iz fajla, obrnuta konverzija

Klasa **Player**:

public Player(Board board) – pravi novog igraca koji ce citati stanja sa zadate table

public Player(Player player) – kopirajuci konstruktor koji se poziva pri kopiranju table

public void run() – sablonska metoda koja postavlja inicijalne pozicije igraca, prati koji je igrac na potezu, povlaci izracunate poteze

public abstract void setInitPositions() – racunanje pocetnih pozicija

protected abstract Move nextMove_toBuild() – vraca drugi deo narednog poteza

protected abstract Move nextMove_toMove() – vraca prvi deo narednog poteza

Klasa **Human**: implementira apstraktne metode klase Player vracajuci vrednosti koje joj MainFrame dostavlja nakon sto ih korisnik upise

public void setMove_toMove(Move move) – MainFrame upisuje sledeci potez, tj. prvi deo poteza

public void setMove_toBuild(Move move) - MainFrame upisuje drugi deo sledeceg poteza

Klasa **AI**:

public Move nextMove_toMove() – poziva computeMove() svoje zadate strategije (Strategy), ceka 1.5s radi preglednijeg prikaza i vraca dobijenu vrednost

public Move nextMove_toBuild() – potez je vec izgracunat ali se na tabli povlaci iz dva dela pa zato postoji i ova metoda; ceka 1.5s i vraca drugi deo poteza

public void setStrategy(Strategy strategy) – postavlja se procitana strategija

Klasa **Minimax**: implementira interfejs Strategy, izgracunava sledeci potez koristeci minimax algoritam i zadatu heuristicku funkciju

public Minimax(**int** depth, Player player) – pri stvaranju se zadaje maksimalna dubina izracunavanja

public Move computeMove() – metoda interfejsa Strategy, ovu metodu AI poziva

public double heuristicValue(Board currBoard) – racuna h. funkciju za trenutno stanje, zadato agrumentom currBoard

private double minimax(Board currBoard, **int** depth, **boolean** maxPlayer) – implementacija osnovnog minimax algoritma

public void setInitPositions() – pocetne pozicije su slucajne

Klasa **AlphaBeta**: implementira interfejs Strategy, izgracunava sledeci potez koristeci minimax algoritam sa alfa-beta odsecanjem i zadatu heuristicku funkciju

public AlphaBeta(**int** depth, Player player) – pri stvaranju se zadaje maksimalna dubina izracunavanja

public Move computeMove() – metoda interfejsa Strategy, ovu metodu AI poziva

public double heuristicValue(Board currBoard) – racuna h. funkciju za trenutno stanje, zadato agrumentom currBoard

private double alphaBetaPruning(Board currBoard, **int** depth, **boolean** maxPlayer) – implementacija minimax algoritma sa alfa-beta odsecanjem

public void setInitPositions() – pocetne pozicije su slucajne

Klasa **Advanced**: implementira interfejs Strategy, izgracunava sledeci potez koristeci minimax algoritam sa alfa-beta odsecanjem i unapredjenu heuristicku funkciju

public AlphaBeta(**int** depth, Player player) – pri stvaranju se zadaje maksimalna dubina izracunavanja

public Move computeMove() – metoda interfejsa Strategy, ovu metodu AI poziva

public double heuristicValue(Board currBoard) – racuna h. funkciju za trenutno stanje, zadato agrumentom currBoard

private double alphaBetaPruning(Board currBoard, **int** depth, **boolean** maxPlayer) – implementacija minimax algoritma sa alfa-beta odsecanjem

public void setInitPositions() – napredan nacin racunanja pocetnih pozicija; jedna figurica blizu centra (napadac), druga u sto izolovanijem cosku (graditelj)

Klasa **Move:**

public Move(Board board, Position posToMove, Position posToBuild, **int** player, **int** piece) – Human igrac koristi ovaj konstruktor

public Move(Board board, **int** player, **int** piece, **int** field, **int** build) **throws** CantMoveException – poziva se iz strategija; pozicije su kodovane tako da odgovoraju prolasku kroz sve moguce pozicije koji se obavlja u algoritmima; baca izuzetak ako neka od zadatih pozicija izlazi iz opsega table

public boolean isMovePossible() – provera validnosti celokupnog poteza; poziva se iz stragegija

public boolean isMovePossible_justMove() - – proverava da li je za stanje zadato u konstruktoru moguce pomeriti se na zeljeno mesto

public boolean isMovePossible_justBuild() – proverava da li je za stanje zadato u konstruktoru moguce izgraditi na zeljenom mesto

*navedene su samo glavne klase, relevantne za srž zadatog problema