|  |
| --- |
| PMF Cirilica.wmf |
| Институт за математику и информатику  Природно-математички факултет  Универзитет у Крагујевцу |
| Семинарски Рад |
| **Игра шах**  **У програмском језику ЈАVA** |
| **Ментор Студент**  **др Ана Капларевић - Малишић Владимир Бараћ 59/2013**  **Лазар Стефановић 92/2013** |
| Септембар 2015. |

**Садржај**

[О пројекту 3](#_Toc429970028)

[Логички део пројекта 4](#_Toc429970029)

[Интерфејс GameConstants 5](#_Toc429970030)

[У овом интерфејсу се налазе све константе потребне за рад игре. 5](#_Toc429970031)

[*Класа Point* 6](#_Toc429970032)

[Класа Figure 7](#_Toc429970033)

[Класа Move 9](#_Toc429970034)

[Класа Engine 10](#_Toc429970035)

[Метод newGame 10](#_Toc429970036)

[Метод getMovableFigures 11](#_Toc429970037)

[Метод getPossibleMoves 15](#_Toc429970038)

[Метод playMove 18](#_Toc429970039)

[Метода interpretMove 21](#_Toc429970040)

[Графички део пројекта 22](#_Toc429970041)

[Прозор са странем за пријављивање 22](#_Toc429970042)

[Прозор са страном за регистровање 24](#_Toc429970043)

[Прозор са главном страном 25](#_Toc429970044)

[Прозор са игром у току 31](#_Toc429970045)

[Остали прозори који се појављују у току игре 32](#_Toc429970046)

[Серверски део пројекта 41](#_Toc429970047)

[Поруке које сервер обрађује 46](#_Toc429970048)

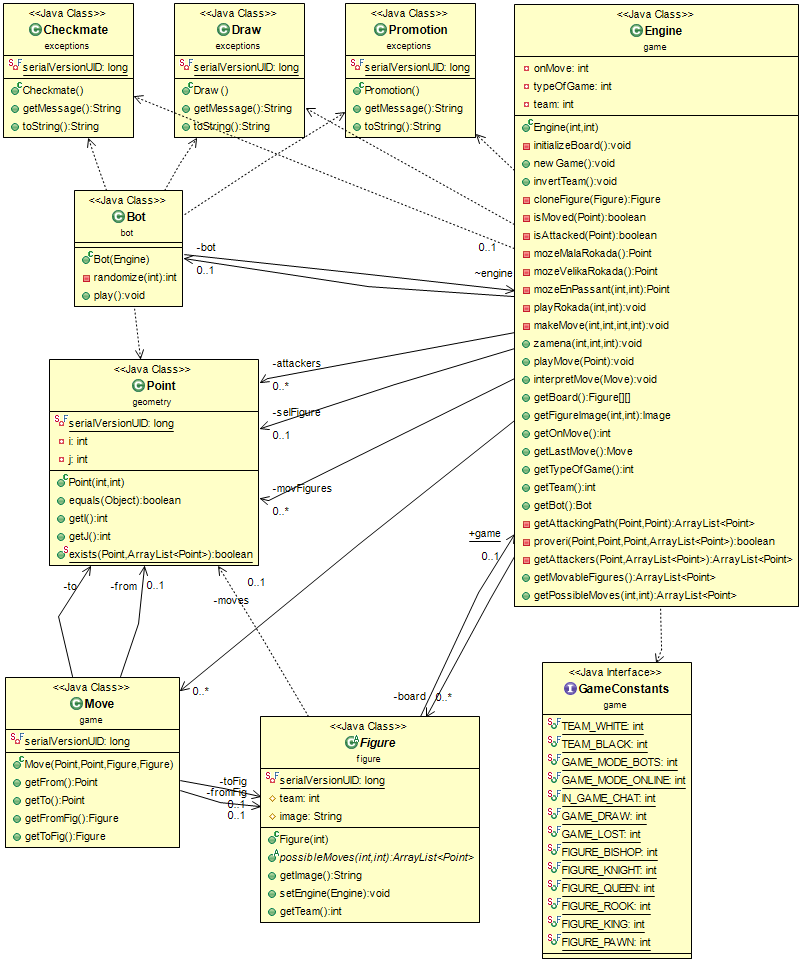
О пројекту

Програм који смо израдили омогућава корисницима да играју сах против компјутера, или других корисника, преко интернета. Пројекат се може поделити на три главне целине, логика, графика и серверски део.

Логички део пројекта

Задатак овог дела програма је да дефинише логику саме игре шах.

Основне класе логике су Point, Figure, Move, GameConstants, Engine, Checkmate, Draw, Promotion I Bot. њихову повезаност можете видети на слици 1.



Slika 1 UML дијаграм логике пројекта(основне класе)

Интерфејс GameConstants

У овом интерфејсу се налазе све константе потребне за рад игре.

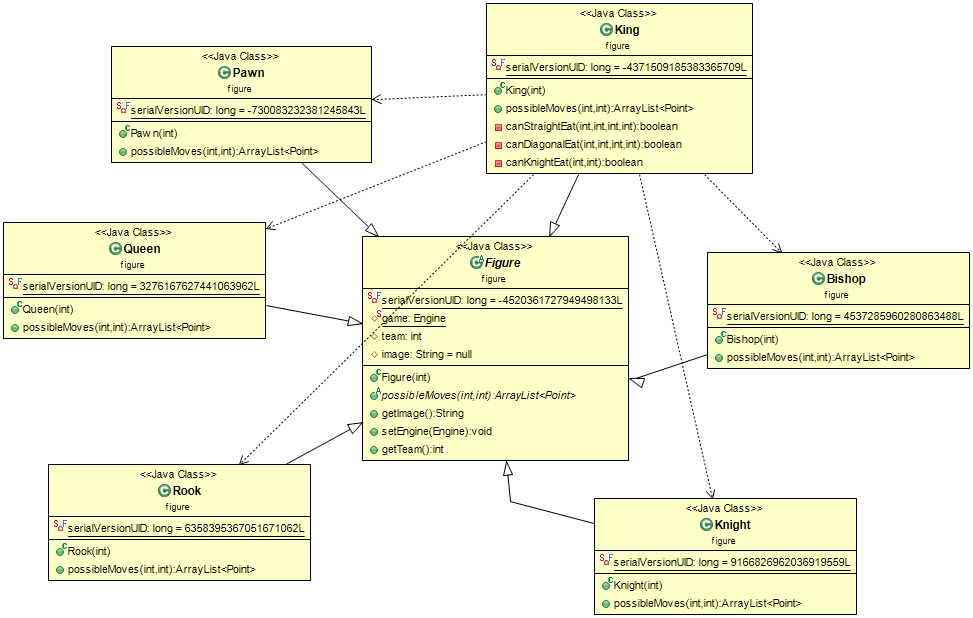
|  |
| --- |
| **Изворни код интерфејса: GameConstants.java** |
| 1. public interface GameConstants { 2. public static int TEAM\_WHITE = 1; 3. public static int TEAM\_BLACK = -1; 4. public static int GAME\_MODE\_BOTS = 2; 5. public static int GAME\_MODE\_ONLINE = 3; 6. public static int IN\_GAME\_CHAT = 4; 7. public static int GAME\_DRAW = 5; 8. public static int GAME\_LOST = 6; 9. public static int FIGURE\_BISHOP = 1; 10. public static int FIGURE\_KNIGHT = 2; 11. public static int FIGURE\_QUEEN = 3; 12. public static int FIGURE\_ROOK = 4; 13. public static int FIGURE\_KING = 5; 14. public static int FIGURE\_PAWN = 6; 15. } |

Класа Point  
Ова класа у себи има 2 прменљиве типа integer, које нам служе ко координате, како бисмо означили једно поље у матрици. Вредности променљивих i и ј се постављају у конструкору класе. Вредности координата могу се добити позивањем метода getI() и getJ(). Методом equals се проверава да ли су две тачке исте, тј. да ли имају исте координате. Exists метода омогућава да се провери постојање неке тачке у листи тачака које се прослеђују методи.

|  |
| --- |
| **Изворни код класе: Point.java** |
| 1. public class Point implements Serializable { 3. private static final long serialVersionUID = 7000736698499772936L; 4. private int i; 5. private int j; 7. public Point(int i, int j) { 8. this.i = i; 9. this.j = j; 10. } 12. @Override 13. public boolean equals(Object obj){ 14. if (((Point)obj).getI() == i && ((Point)obj).getJ() == j) 15. return true; 16. return false; 17. } 18. public int getI() { 19. return i; 20. } 21. public int getJ() { 22. return j; 23. } 25. public static boolean exists(Point p, ArrayList<Point> list) { 26. for (int ii = 0; ii < list.size(); ii++) 27. if (p.equals(list.get(ii))) 28. return true; 29. return false; 30. } 31. } |

Класа Figure

Класа Figure је абстрактна класа која означава шаховске фигуре. Из ње су изведене шест класа од којих свака представља по једни шаховску фигуру (слика 2).



Slika 2 УМЛ приказ повезаности класа фигура

Figure од променљивих има променљиву team типа integer која означава да ли фигура припада белом или црном играчу. Следећа променљива image типа String садрзи у себи путању до слике, одређене фигуре. Обе променљиве се сетују конструктору у подкласа класе Figure, а њихове вредонсти се могу добити позивањем метода getImage( ) и getTeam(). Статичка променљива ...

|  |
| --- |
| **Изворни код класе: Figure.java** |
| 1. public abstract class Figure implements Serializable{ 2. private static final long serialVersionUID = -4520361727949498133L; 3. Protected static Engine game; 4. Protected int team; 5. Protected String image = null; 7. public Figure(int team) { 8. this.team = team; 9. } 11. public abstract ArrayList<Point> possibleMoves(int i, int j); 13. public String getImage() { 14. return image; 15. } 17. public void setEngine(Engine game) { 18. Figure.game = game; 19. } 21. public int getTeam() { 22. return team; 23. } 24. } |

Абстрактни метод possibleMoves(int I, int j) имплементира свака фигура која наслеђује класу Figure, и даје све позиције на које та фигура тренутно може да скочи.

|  |
| --- |
| **Пример имплементације методе PossibleMoves(int I,int j) у класи: Rook.java** |
| 1. public ArrayList<Point> possibleMoves(int i, int j) { 2. ArrayList<Point> lista = new ArrayList<Point>(); 3. Figure[][] board = game.getBoard(); 4. for (int k = j+1; k < board.length; k++) { 5. if(board[i][k]==null){ 6. lista.add(new Point(i, k)); 7. } else { 8. if(board[i][k].getTeam()!=team) 9. lista.add(new Point(i, k)); 10. break; 11. } 12. } 13. for (int k = j-1; k > -1; k--) { 14. if(board[i][k]==null){ 15. lista.add(new Point(i, k)); 16. } else { 17. if(board[i][k]!=null && board[i][k].getTeam()!=team) 18. lista.add(new Point(i, k)); 19. break; 20. } 21. } 22. for (int k = i-1; k > -1; k--) { 23. if(board[k][j]==null){ 24. lista.add(new Point(k, j)); 25. } else { 26. if(board[k][j]!=null && board[k][j].getTeam()!=team) 27. lista.add(new Point(k, j)); 28. break; 29. } 30. } 31. for (int k = i+1; k < board.length; k++) { 32. if(board[k][j]==null){ 33. lista.add(new Point(k, j)); 34. } else { 35. if(board[k][j]!=null && board[k][j].getTeam()!=team) 36. lista.add(new Point(k, j)); 37. break; 38. } 39. } 41. return lista; 42. } |

Класа Move

Класу Move користимо да бисмо описали померање једне фигуре. Она у себи садржи четири промениљиве. Променљиве from и to koje су типа Point и означавају позиције одакле и где фигура скаче, респеткивно. FromFig и toFig су променљиве типа Figure и оне чувају типове фигуре која се помера, само примену ћемо објаснити мало касније када будемо причали о класи Engine. Све променљиве се постављају при позивању конструктора, а њихове вредности се могу добити позивањем одређеног гетера.

|  |
| --- |
| **Изворни код класе: Move.java** |
| 1. public class Move implements Serializable{ 2. private static final long serialVersionUID = -8042188616031818781L; 4. private Point from; 5. private Point to; 6. private Figure fromFig; 7. private Figure toFig; 9. public Move(Point from, Point to, Figure fromFig, Figure toFig) { 10. this.from = from; 11. this.to = to; 12. this.fromFig = fromFig; 13. this.toFig = toFig; 14. } 15. public Point getFrom() { 16. return from; 17. } 18. public Point getTo() { 19. return to; 20. } 21. public Figure getFromFig() {­­­–– 22. return fromFig; 23. } 24. public Figure getToFig() { 25. return toFig; 26. } 27. } |

Класа Engine

Класа Engineпредставља механизам кoји ради у пoзадини и покреће целу игру. За предстаљање шаховске табле коритстимо променљиву board, која је матрица фигура. Ако се на позицији board[i][j] не налази фигура, та ћелија матрице ће бити NULL, у супротном ће садржати инстанцу фигуре која се налази на тој позицији. Променљива onMove памти ко је на потезу, типа је integer и вредност добија од константи TEAM\_WHITE и TEAM\_BLACK из класе GameConstants.java. Team је променљива коју користимо када се игра игра преко интернета, и означава да ли је играч бели, или црни, ове вредности такође добија из константи. Променљива moves представља листу objekata Move, и користимо је као историју потеза. На крају имамо променљиву bot типа Bot, која чува инстанцу бота, у случају да играч жели да игра игру офлајн. То да ли је игра офлајн или онлајн брине се променљива typeOfGame, која такође своје вредности добија из GameConstants.java, те константе су GAME\_MODE\_BOTS и GAME\_MODE\_ONLINE. Свака од ових променљивих има свој гетер, како би им могли приступити.

Метод newGame

Овај метод се позива како би започела нова игра, поставља да је на потезу бели играц, и позива методу initializeBoard која све фигуре поставља на њихова почетна места.

|  |
| --- |
| **Изворни код метода initializeBoard I newGame** |
| 1. private void initializeBoard() { 2. board = new Figure[8][8]; 4. for (int i = 2; i < 6; i++) 5. for (int j = 0; j < 8; j++) 6. board[i][j] = null; 8. for (int i = 0; i < 8; i++) { 9. board[1][i] = new Pawn(-1); 10. board[6][i] = new Pawn(1); 11. } 13. board[0][0] = new Rook(-1); 14. board[0][7] = new Rook(-1); 15. board[7][0] = new Rook(1); 16. board[7][7] = new Rook(1);  19. board[0][1] = new Knight(-1); 20. board[0][6] = new Knight(-1); 21. board[7][1] = new Knight(1); 22. board[7][6] = new Knight(1); 24. board[0][2] = new Bishop(-1); 25. board[0][5] = new Bishop(-1); 26. board[7][2] = new Bishop(1); 27. board[7][5] = new Bishop(1); 29. board[0][3] = new Queen(-1); 30. board[7][3] = new Queen(1); 31. board[0][4] = new King(-1); 32. board[7][4] = new King(1); 34. board[0][0].setEngine(this); 36. moves = new ArrayList<Move>(); 37. } 38. public void newGame(){ 39. initializeBoard(); 40. onMove = 1; 41. } |

Метод getMovableFigures

|  |
| --- |
| **Изворни код функције: getMovableFigures** |
| 1. public ArrayList<Point> getMovableFigures() throws Draw, Checkmate{ 2. if((typeOfGame == GAME\_MODE\_ONLINE && onMove == team) || typeOfGame != GAME\_MODE\_ONLINE){ 3. Point king = new Point(0, 0); 4. ArrayList<Point> mojeFigure = new ArrayList<>(); 5. ArrayList<Point> drugeFigure = new ArrayList<>(); 6. for (int i = 0; i < board.length; i++) 7. for (int j = 0; j < board.length; j++) 8. if(board[i][j] != null){ 9. if(board[i][j].getTeam()==onMove){ 10. if(board[i][j] instanceof King ) 11. king = new Point(i,j); 12. else 13. mojeFigure.add(new Point(i, j)); 14. } else 15. drugeFigure.add(new Point(i, j)); 16. } 18. attackers = new ArrayList<Point>(); 19. attackers = getAttackers(king,drugeFigure); 20. //Ako oba igraca imaju samo kraljeve nereseon je 21. movFigures = new ArrayList<Point>(); 22. //Ako ja imam samo kralja i drugi igrac ima samo kralja onda je nereseno\\ 23. if(mojeFigure.size()==0 && drugeFigure.size() == 1){ 24. throw new Draw(); 25. } 27. if(board[king.getI()][king.getJ()].possibleMoves(king.getI(), king.getJ()).size()==0){ 28. if(attackers.size()>1) 29. throw new Checkmate(); 30. } else { 31. //Dodajemo kralja jer ima gde da se pomeri :) 32. movFigures.add(new Point(king.getI(), king.getJ())); 33. } 35. if(attackers.size()==1){//Ako ga 1 figura napada, proveri da li moze neka da se ispreci ili pojede tu figuru 36. ArrayList<Point> attackingPath = getAttackingPath(attackers.get(0), king); 37. //Kreiram novu listu figura koje smem da pomeram 38. //movFigures = new ArrayList<Point>(); 39. ArrayList<Point> posMov; 40. for (int i = 0; i < mojeFigure.size(); i++) { 41. //Uzimam sve moguce pozicije na koje moja figura moze da stane 42. posMov = board[mojeFigure.get(i).getI()][mojeFigure.get(i).getJ()].possibleMoves(mojeFigure.get(i).getI(), mojeFigure.get(i).getJ()); 43. //Dodajem samo figure koje mogu da se isprece, ili pojedu figuru koja napada kralja\\ 44. //a njihovim pomeranjem se ne ugrozava kralj sa druge strane\\ 45. for (int j = 0; j < attackingPath.size(); j++) { 46. //Ako se pozicija nalazu i nizu mogucih poteza dodaj figuru u listu mogucih figura\\ 47. if(Point.exists(attackingPath.get(j), posMov)){ 48. movFigures.add(new Point(mojeFigure.get(i).getI(), mojeFigure.get(i).getJ())); 49. break; 50. } 51. } 52. } 53. //ako on ne moze da se pomeri kralj, i ni jedna figura ne moze da spreci sah baca se exception sahMat 54. if(board[king.getI()][king.getJ()].possibleMoves(king.getI(), king.getJ()).size()==0 && movFigures.size()==0) 55. throw new Checkmate(); 57. } else if(attackers.size()==0){ 58. //Ako ni 1 figura ne napada kralja, dodaj samo figure koje mogu da se pomeraju i njihovim pomeranjem ne moze da se ugrozi kralj 60. //proveri da li pomeranje na bilo koju poziciju kralj ostaje zasticen 61. //movFigures = new ArrayList<Point>(); 63. for (int i = 0; i < mojeFigure.size(); i++) { 64. ArrayList<Point> pos = board[mojeFigure.get(i).getI()][mojeFigure.get(i).getJ()].possibleMoves(mojeFigure.get(i).getI(), mojeFigure.get(i).getJ()); 65. if(pos.size()>0){ 66. //Ako bar jedno pomeranje ne izaziva sah, dodaj figuru u listu pomerajucih figura\\ 67. for (int j = 0; j < pos.size(); j++) { 68. if(proveri(king, new Point(mojeFigure.get(i).getI(),mojeFigure.get(i).getJ()), pos.get(j),drugeFigure)){ 69. movFigures.add(new Point(mojeFigure.get(i).getI(),mojeFigure.get(i).getJ())); 70. break; 71. } 72. } 73. } 74. } 75. //Ako kralj nema gde da se pomeri i ni 1 figura ne moze da se pomeri, a nije sah, onda je nereseno 76. if(board[king.getI()][king.getJ()].possibleMoves(king.getI(), king.getJ()).size()==0 && movFigures.size()==0) 77. throw new Draw(); 79. } else{//ako ga napada vise od 1 figure 80. if(movFigures.size()==0)//Ako kralj ne moze da se pomeri shamat je 81. throw new Checkmate(); 82. } 83. //Ako kralj ima vise napadaca, i moze da se pomeri ovim uslovima primoravamo da samo kraljem moze igrac da igra :) 84. return movFigures; 85. } else // ako nije na tebe red, onda vrati prazan 86. return new ArrayList<Point>(); 87. } |

Ова метода даје све фигуре које играч, који је на потез, може у том тренутку да помера. То су све фигуре које имају где да скоче (када позовемо фигурин possibleMoves() враћа не празну листу Point-а), а њиховим померањем се не би угрозио краљ, или ако је краљ у саху, да могу да га спасу. Ова метода баца два exception-а, Checkmate и Draw. Checkmate се баца када играч коме је краљ у нападнут, не може да помери ни једни фигуру да би прекинуо напад. Draw се баца када оба играча остану само са краљевима на пољу, или када играч нема где да помери краља, а краљ нију нападнут.

Функција ради тако сто прво распореди фигуре по тимовима, и узме позицију краља тренутног играча. Затим позива мотод getAttackers, којој се порслеђују позиција граља и противничке фигуре. Ова метод враћа листу свих фигура које нападају краља, у случају да таквих фигура нема, враћа празну листу. Ако постоје фигуре које нападју краљан требају се наћи фигуре које могу да се испрече исред ,или поједну фигуру која напада краља. Фигуре које нападају краља добијамо позивањем методе getAttackers, која пролази кроз све противничке фигуре и проверава да ли се позиција краља налази на неком од њихових могућих позиција на које могу да скоче, ако се налази, оне се додају у листу нападача, коју функција после враћа. Фигуре које могу да се спрече напад на краља налазимо тако сто прво нађемо позиције преко којих је краљ нападнут, позивањем методе getAttackingPath, којој се прослећује позиција краља и фигуре која га напада, и онда по томе која је фигура утврђује пут којим напада. Када смо нашли пут напада, пролазимо кроз све наше фигуре и проверавамо да ли се и један од њихових могућих скокова подудара са путом напада, ако се подудара, фигуру додајемо у листу могућих фигура. У случају да краљ није нападнут, онда узимамо све фигуре играча на потезу и проверавамо да ли постоји бар једна позиција на коју оне могу да се помере, а да не угоржавају краља тиме. То радимо тако што порђемо кроз листу играчевих фигура и за сваку проверимо да ли у њеним могућим потезима постоји позиција која је одговарајућа, за то нам помаже метода provera, која за прослеђену тренутну позицију краља, фигуре, позиције на коју фигура може да скичи и свих противницких фигура, одрећује да ли би померањем фигуре на прослеђену позицију досло до угожавања краља. Ако Постоји бар једна позиција која задовољава критеријум, фигура се додаје на листу могућих фигура, која се на крају методе враћа позиваоцу.

|  |
| --- |
| **Изворни код метода: getAttackers, getAttackingPath proveri** |
| 1. private ArrayList<Point> getAttackingPath(Point attackerPosition,Point figurePosition){ 2. ArrayList<Point> attackPath = new ArrayList<>(); 3. //Dodajemo poziciju napadaca 4. attackPath.add(attackerPosition); 5. if(!(board[attackerPosition.getI()][attackerPosition.getJ()] instanceof Knight)){ 6. //Ako nije konj odmah ima jos neko polje na koje moze da se stane, a da se blokira sah 7. if(attackerPosition.getI()==figurePosition.getI()){//Napada Horizontalno 8. if(attackerPosition.getJ()<figurePosition.getJ()){//napada sa leva na desno 9. for (int i = attackerPosition.getJ(); i < figurePosition.getJ(); i++) { 10. attackPath.add(new Point(attackerPosition.getI(), i)); 11. } 12. } else {//napada sa desna na levo 13. for (int i = figurePosition.getJ(); i < attackerPosition.getJ(); i++) { 14. attackPath.add(new Point(attackerPosition.getI(), i)); 15. } 16. } 17. } else if(attackerPosition.getJ()==figurePosition.getJ()){//Napada Vertikalno 18. if(attackerPosition.getI()<figurePosition.getI()){//Napada na dole 19. for (int i = attackerPosition.getJ(); i < figurePosition.getJ(); i++) { 20. attackPath.add(new Point(i, attackerPosition.getJ())); 21. } 22. } else {//Napada na gore 23. for (int i = figurePosition.getJ(); i < attackerPosition.getJ(); i++) { 24. attackPath.add(new Point(i, attackerPosition.getJ())); 25. } 26. } 27. } else {//Napada Dijagnoalno 28. if(attackerPosition.getI()<figurePosition.getI()){//Napada na dole 29. if(attackerPosition.getJ()<figurePosition.getJ()){ 30. for (int i = 0; i < (figurePosition.getI()-attackerPosition.getJ()); i++) {//Napada Desno 31. attackPath.add(new Point(attackerPosition.getI()+i, attackerPosition.getJ()+i)); 32. } 33. } else { 34. for (int i = 0; i < (attackerPosition.getJ()-figurePosition.getI()); i++) {//Napada Levo 35. attackPath.add(new Point(attackerPosition.getI()+i, attackerPosition.getJ()-i)); 36. } 37. } 38. } else {//Npada na gore 39. if(attackerPosition.getJ()<figurePosition.getJ()){ 40. for (int i = 0; i < (figurePosition.getI()-attackerPosition.getJ()); i++) {//Napada Desno 41. attackPath.add(new Point(attackerPosition.getI()-i, attackerPosition.getJ()+i)); 42. } 43. } else { 44. for (int i = 0; i < (attackerPosition.getJ()-figurePosition.getI()); i++) {//Napada Levo 45. attackPath.add(new Point(attackerPosition.getI()-i, attackerPosition.getJ()-i)); 46. } 47. } 48. } 49. } 50. } 51. return attackPath; 52. } 54. //Proverava da li bi se figurinim pomeranjem stvorio sah\\ 55. private boolean proveri(Point kingPos,Point sPos, Point nPos,ArrayList<Point> drugeFigure){ 56. boolean ind = false; 57. Figure fig = board[nPos.getI()][nPos.getJ()]; 58. board[nPos.getI()][nPos.getJ()] = board[sPos.getI()][sPos.getJ()]; 59. board[sPos.getI()][sPos.getJ()] = null; 60. //Ako kralj nema napdaca posle pomeranja figure, figura moze da stane na to mesto 61. if(getAttackers(kingPos,drugeFigure).size() == 0) 62. ind = true; 63. board[sPos.getI()][sPos.getJ()] = board[nPos.getI()][nPos.getJ()]; 64. board[nPos.getI()][nPos.getJ()] = fig; 65. return ind; 66. } 68. //Uzima sve figure koje napadaju prosledjenu figuru\\ 69. private ArrayList<Point> getAttackers(Point figurePostition,ArrayList<Point> FigureList){ 70. ArrayList<Point> attackers = new ArrayList<Point>(); 71. //provera da li je na udaru neke figure i ako jeste uzmi njenu poziciju// 72. for (int i = 0; i < FigureList.size(); i++) { 73. if(Point.exists(figurePostition,board[FigureList.get(i).getI()][FigureList.get(i).getJ()].possibleMoves(FigureList.get(i).getI(),FigureList.get(i).getJ()))) 74. attackers.add(new Point(FigureList.get(i).getI(),FigureList.get(i).getJ())); 75. } 76. return attackers; 77. } |

­­­­­­Метод getPossibleMoves

|  |
| --- |
| **Изворни код методе: getPossibleMoves** |
| 1. public ArrayList<Point> getPossibleMoves(int i, int j){ 2. if((typeOfGame == GAME\_MODE\_ONLINE && onMove == team) || typeOfGame != GAME\_MODE\_ONLINE) 3. if(board[i][j] != null && board[i][j].getTeam() == onMove){//Izabiramo figuru igraca koji je na potezu i vracamo njene moguce poteze 4. selFigure = new Point(i, j); 5. ArrayList<Point> lista = board[i][j].possibleMoves(i, j); 6. if(board[i][j] instanceof Pawn){//Dodajemo EnPasant 7. Point p = mozeEnPassant(i, j); 8. if(p!=null) 9. lista.add(p); 10. } 11. ArrayList<Point> possibleMovies = new ArrayList<Point>(); 13. if(board[i][j] instanceof King){//Ako smo kliknuli na kralja, samo uzimamo njegove moguce poteze i dodajemo rokade ako su moguce 14. Point malaRokada = mozeMalaRokada(); 15. if(malaRokada!=null) 16. lista.add(malaRokada); 18. Point velikaRokada = mozeVelikaRokada(); 19. if(velikaRokada!=null) 20. lista.add(velikaRokada); 22. return lista; 23. } else {//Ako je odabrana druga figura uzeti njene poteze\\ 25. //Uzimamo mesto kralja( i protivnicke figure?)\\ 26. Point king = null; 27. ArrayList<Point> drugeFigure = new ArrayList<>(); 28. for (int k = 0; k < board.length; k++) { 29. for (int k2 = 0; k2 < board.length; k2++) { 31. if(board[k][k2] != null && board[k][k2].getTeam()==onMove && board[k][k2] instanceof King){ 32. king = new Point(k, k2); 33. } else if(board[k][k2] != null && board[k][k2].getTeam()!=onMove) 34. drugeFigure.add(new Point(k, k2)); 36. } 37. } 39. if(attackers != null && attackers.size()==1) {//Ako je kralj napadnut, daj moguce poteze koji ce ga spasiti\\ 41. //Uzimamo put kojim je kralj napadnut\\ 42. ArrayList<Point> path = getAttackingPath(attackers.get(0), king); 43. //proci kroz sve moguce potexe figure i odabrati samo one koji se nalaze na putanji napada 44. for (int k = 0; k < path.size(); k++) { 45. if(Point.exists(path.get(k),lista) && proveri(king,new Point(i, j),path.get(k),drugeFigure)) 46. possibleMovies.add(path.get(k)); 47. } 48. } else { //ako kralj nije napadnut samo dodaj poteze koji ga nece ugroziti\\ 49. //Proci kroz sve poteze figure i odabrati one koje ne ugrozavaju kralja\\ 50. for (int k = 0; k < lista.size(); k++) { 51. if(proveri(king, new Point(i, j),lista.get(k) , drugeFigure)) 52. possibleMovies.add(lista.get(k)); 53. } 54. } 56. return possibleMovies; 57. } 58. } else //Figura koju smo odabrali nije ispravna \\ 59. return null; 60. else return new ArrayList<Point>(); 61. } |

Ова метода се позива да би добили све могуће позиције на које фигура, чије смо координате проследили, може да скочи. Метод ради тако сто прво од фигуре коју смо одабрали узима могуће потезе, позивањем њене методе possibleMoves, затим проверава да ли смо одабрали краља, или неку другу фигуру. Ако је одабран краљ, проверавамо да ли мозе да се одигра нека од рокада. То радимо позивањем фукнција mozeMalaRokada и mozeVelikaRokada. Ако је могуће одрадити рокада, свака од ових функција ће вратити позицију на коју краљ треба да скочи да би се одрадила рокада, и та позиција се додаје у listu possibleMoves figure.

|  |
| --- |
| **Изворни код метода: mozeMalaRokada I mozeVelikaRokada и метода које они користе** |
| 1. //Da li je figura pomerana od pocetka partije\\ 2. private boolean isMoved(Point figPosition){ 3. for (int i = 0; i < moves.size(); i++) { 4. if(moves.get(i).getFrom().getI() == figPosition.getI() && moves.get(i).getFrom().getJ() == figPosition.getJ()) 5. return true; 6. } 7. return false; 8. } 10. //Da li polju preti neka figura\\ 11. private boolean isAttacked(Point poistion){ 12. for (int i = 0; i < board.length; i++) { 13. for (int j = 0; j < board.length; j++) { 14. if(board[i][j] != null && board[i][j].getTeam() != onMove && Point.exists(poistion,board[i][j].possibleMoves(i, j))) 15. return true; 16. } 17. } 18. return false; 19. } 20. //Proverava da li moze da se odigra mala rokada, ako ne moze vraca NULL\\ 21. private Point mozeMalaRokada(){ 22. Point a = null; 23. if(onMove == 1){//Beli igrac 24. if(isAttacked(new Point(7, 4))) 25. return null; 26. if(!isMoved(new Point(7, 4)) && !isMoved(new Point(7, 7))){ 27. for(int j=5;j<7;j++) 28. if(board[7][j]!=null || isAttacked(new Point(7, j))) 29. return null; 30. a = new Point(7, 6); 31. } 32. } else {//Crni igrac 33. if(isAttacked(new Point(0, 4))) 34. return null; 35. if(!isMoved(new Point(0, 4)) && !isMoved(new Point(0, 7))){ 36. for(int j=5;j<7;j++) 37. if(board[0][j] !=null || isAttacked(new Point(0, j))) 38. return null; 39. a = new Point(0, 6); 40. } 41. } 43. return a; 44. } 45. //Proverava da li moze da se odigra velika rokada, ako ne moze vraca NULL\\ 46. private Point mozeVelikaRokada(){ 47. Point a = null; 48. if(onMove == 1){ 49. if(isAttacked(new Point(7, 4))) 50. return null; 51. if(!isMoved(new Point(7, 4)) && !isMoved(new Point(7, 0))){ 52. for(int j=1;j<4;j++) 53. if(board[7][j]!=null || isAttacked(new Point(7, j))) 54. return null; 55. a = new Point(7, 2); 56. } 57. } else { 58. if(isAttacked(new Point(0, 4))) 59. return null; 60. if(!isMoved(new Point(0, 4)) && !isMoved(new Point(0, 0))){ 61. for(int j=1;j<4;j++) 62. if(board[0][j] !=null || isAttacked(new Point(0, j))) 63. return null; 64. a = new Point(0, 2); 65. } 66. } 68. return a; 69. } |

Ако је одабрана било која друга фигура, прво се проверава да ли је одабран пијун и ако јесте проверава се да ли може да одигра потез енпасант. Провера могућности играња енпасант потеза се врши методом mozeEnPasant,која у колико је могуће да се одигра овај потез, враћа позицију на коју пијун треба да скочи да би се одиграо. Затим се проверава да ли је краљ нападну, позивањем методе getAttacckers. Ако је нападнут, узима се позиција фигуре која га напада, а затим позива метода getAttackingPath, како би добили пут којим је краљ нападнут. Затим се пролази кроз могуће потезе одабране фигуре, и за свако могуће место на које фигура може да скочи се проверава да ли се налази на путањи напада, a померањем на ту позицију не би угрозило краља са друге стране. Ако се налази, додаје се у листу могућих потеза. Ово проверавање такође користи методу proveri, као и претходна метода getMovableFigures. Ако краљ није нападнут онда се само пролази кроз могуће потезе фигуре и проверава да ли би померањем на одређено поље угрозило краља, и ако неби, додаје се у листу могућих потеза, која се на крају методе враћа. И у овом делу методе се користи метод proveri.

|  |
| --- |
| **Изворни код методе: mozeEnPasant** |
| 1. private Point mozeEnPassant(int i, int j){ 2. if ((board[i][j].getTeam() == 1 && i == 3) || (board[i][j].getTeam()==-1 && i == 4)){ 3. if((j-1 > -1 && board[i][j-1] instanceof Pawn) || (j+1<8 && board[i][j+1] instanceof Pawn)){ 4. Point from = getLastMove().getFrom(); 5. Point to = getLastMove().getTo(); 6. if(board[to.getI()][to.getJ()] instanceof Pawn && Math.abs(from.getI() - to.getI()) == 2) 7. if(Math.abs(to.getJ()-j)==1) 8. //ako je skocio dva polja i to je pored naseg pijuna 9. return new Point(to.getI()-board[i][j].getTeam(), to.getJ()); 11. } 12. } 13. return null; 14. } |

Метод playMove

­­­

|  |
| --- |
| **Изворни код методе: playMove и makeMove** |
| 1. public void playMove(Point position) throws Promotion{ 2. if(selFigure != null) 3. makeMove(new Point(selFigure.getI(), selFigure.getJ()), new Point (position.getI(), position.getJ())); 4. } 5. private void makeMove(Point from,Point to) throws Promotion { 6. int fromi = from.getI(),fromj = from.getJ(); 7. int toi = to.getI(), toj = to.getJ(); 8. if (board[fromi][fromj] instanceof King && Math.abs(fromj-toj)==2) { 9. playRokada(toi,toj); 10. } else { 11. //Provera za zamenu, da li je pijun stigao do kraja ;) 12. if(board[fromi][fromj] instanceof Pawn){ 13. if(onMove == 1 && (toi == 0)){ 14. throw new Promotion(); 15. } else if(onMove == -1 && (toi == 7)){ 16. throw new Promotion(); 17. } else {//Za EnPasant 18. Point p = mozeEnPassant(fromi, fromj); 19. if(p!=null){ 20. board[toi+onMove][toj] = null; 21. } 22. } 23. } 24. //kraj provere za zamenu 25. board[toi][toj] = board[fromi][fromj]; 26. board[fromi][fromj] = null; 27. moves.add(new Move(new Point(fromi, fromj), new Point(toi, toj),cloneFigure(board[fromi][fromj]),cloneFigure(board[toi][toj]))); 29. } 30. attackers.clear(); 31. onMove \*= -1; 32. } |

Ова метода се позива када играч кликне на поље на које хоће да му фигура скочи. Он позива методу makeMove, којој је потребно проследити позицију фигуре која се помера и место на које се помера. Метода makeMove ради тако што прво проверава да ли је фигура коју желимо дапомеримо краљ, ако јесте, и ако се место на које краљ жели да скочи подудара са местом за неку од рокада позива се функција playRokada, којој се шаље место скока, како би онда позвала одговарјуће методе које извршавају рокаду.

|  |
| --- |
| **Изворни код методе: PlayRokada** |
| 1. private void playRokada(int toi,int toj){ 2. if(toj==2) {//Velika rokada 3. //Top 4. board[toi][3] = board[toi][0]; 5. board[toi][0] = null; 6. //kralj 7. board[toi][2] = board[toi][4]; 8. board[toi][4] = null; 9. moves.add(new Move(new Point(toi, 4), new Point(toi, 2), new King(onMove), new King(onMove))); 10. moves.add(new Move(new Point(toi, 0), new Point(toi, 3), new Rook(onMove), new Rook(onMove))); 11. } else if(toj==6){//Mala rokada 12. board[toi][5] = board[toi][7]; 13. board[toi][7] = null; 14. board[toi][6] = board[toi][4]; 15. board[toi][4] = null; 16. moves.add(new Move(new Point(toi, 4), new Point(toi, 6), new King(onMove), new King(onMove))); 17. moves.add(new Move(new Point(toi, 7), new Point(toi, 5), new Rook(onMove), new Rook(onMove))); 18. } 19. //Ovde uneti za move kao specijalni potez// 20. } |

У случају да фигура није краљ, онда се проверава да ли је пијун. Ако јесте, онда се проверава да ли је стигао до краја табле. Ако је стигао до краја табле, баца се греска promotion које означава да треба да се кориснику да избор фигуре у коју жели да унапреди пијуна. Када корисник изабере једну од фигура позива се метода zamena којој се шаљу позиција фигуре коју желимо да унапредимо и тип фигуре у коју желимо да је унапредимо. Типове извлачимо из GameConstants.FIGURE\_<ime figure>.

|  |
| --- |
| **Изворни код методе: zamena** |
| 1. public void zamena(int i, int j,int figura){ 2. Figure fig = null; 3. switch(figura){ 4. case GameConstants.FIGURE\_BISHOP://Bishop 5. fig = new Bishop(onMove); 6. break; 7. case GameConstants.FIGURE\_KNIGHT://Knight 8. fig = new Knight(onMove); 9. break; 10. case GameConstants.FIGURE\_QUEEN://queen 11. fig = new Queen(onMove); 12. break; 13. case GameConstants.FIGURE\_ROOK://Rook 14. fig = new Rook(onMove); 15. break; 16. } 17. board[i][j] = fig; 18. board[selFigure.getI()][selFigure.getJ()] = null; 19. moves.add(new Move(new Point(selFigure.getI(), selFigure.getJ()), new Point(i, j), new Pawn(onMove), cloneFigure(board[i][j]))); 20. attackers.clear(); 21. onMove \*= -1; 22. } |

Ако фигура није пијун, онда се само пребацује на жељено поље и на њено претходно место се ставља null. На крају методе makeMove се мења ко је на потезу, а одиграни потез се додаје на листу одиграних потеза.

Метода interpretMove

|  |
| --- |
| **Изворни код методе: interpretMove** |
| 1. public void interpretMove(Move move){ 2. Point from = move.getFrom(); 3. Point to = move.getTo(); 4. Figure toFig = move.getToFig(); 5. try { 6. makeMove(new Point(from.getI(), from.getJ()), new Point(to.getI(), to.getJ())); 7. } catch (Promotion e) { 8. if(toFig instanceof Bishop) 9. zamena(to.getI(), to.getJ(), 1); 10. else if(toFig instanceof Knight) 11. zamena(to.getI(), to.getJ(), 2); 12. else if(toFig instanceof Queen) 13. zamena(to.getI(), to.getJ(), 3); 14. else if(toFig instanceof Rook) 15. zamena(to.getI(), to.getJ(), 4); 16. board[from.getI()][from.getJ()] = null; 17. } 18. } |

Ова метода је задужена да прослеђени потез припреми и проследи методи makeMove.

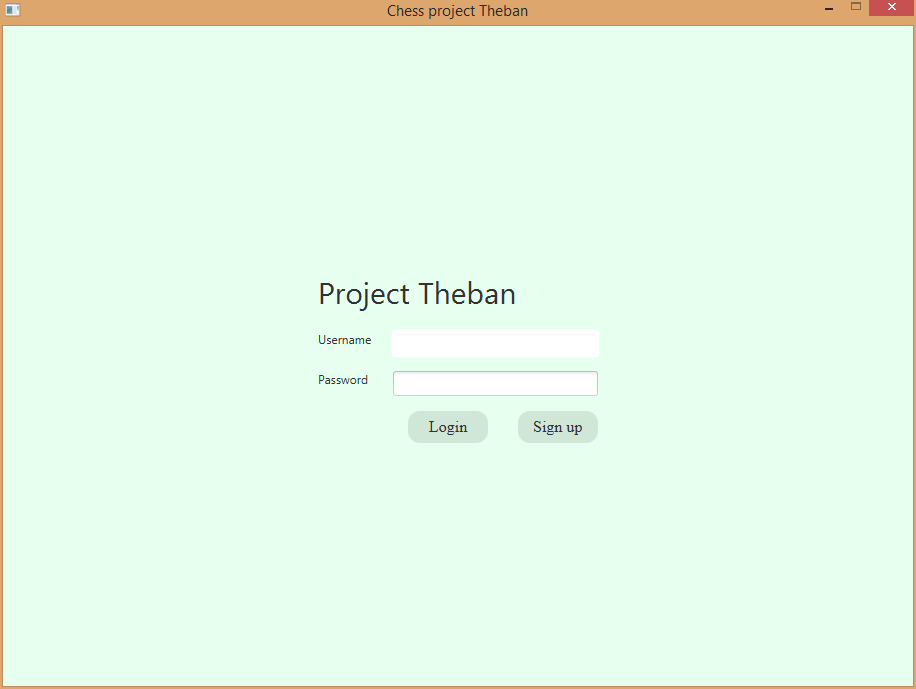
Класа Bot

|  |
| --- |
| **Изворни код класе: Bot** |
| 1. package bot; 2. import java.util.ArrayList; 3. import exceptions.Checkmate; 4. import exceptions.Draw; 5. import exceptions.Promotion; 6. import game.Engine; 7. import game.GameConstants; 8. import geometry.Point; 9. public class Bot { 11. Engine engine; 13. public Bot(Engine engine){ 14. this.engine = engine; 15. } 17. //Daje random broj 18. private int randomize(int max){ 19. return ((int) Math.floor(Math.random()\*max)); 20. } 22. //Odigra potez// 23. public void play() throws Draw, Checkmate { 24. Point move = null; 25. ArrayList<Point> moveableFigures; 26. Point figura = null; 27. try { 28. moveableFigures = engine.getMovableFigures();//Uzima neku figuru koja moze da se pomera// 29. figura = moveableFigures.get(randomize(moveableFigures.size())); 30. ArrayList<Point> posibleMoves = engine.getPossibleMoves(figura.getI(), figura.getJ()); 31. //Uzima neku od mogucih pozicija 32. move = posibleMoves.get(randomize(posibleMoves.size())); 33. try { 34. engine.playMove(move); 35. } catch (Promotion e) { 36. int rand = randomize(4); 37. switch (rand) { 38. case 0: 39. rand = GameConstants.FIGURE\_BISHOP; 40. break; 41. case 1: 42. rand = GameConstants.FIGURE\_KING; 43. break; 44. case 2: 45. rand = GameConstants.FIGURE\_QUEEN; 46. break; 47. case 3: 48. rand = GameConstants.FIGURE\_ROOK; 49. break; 50. } 51. engine.zamena(move.getI(), move.getJ(), rand); 52. } 53. } catch (Draw e1) { 54. throw new Draw(); 55. } catch (Checkmate e1) { 56. throw new Checkmate(); 57. } 59. } 61. } |

Ова класа симулира играње компјутера. Главна метода ове класе је play, она прво узима фигуре које могу да се померају, затим одабира насумично неку од тих фигура, и након тога поново насумично једну од могућих потеза те фигуре.

Графички део пројекта

Овај део пројекта је задужен да буде посредник имеђу корисника и логичког дела, и то је оно што корисник види при покретању апликације. Први прозор који се приказује кориснику, тражи од њега да се улогује, уношењем свог корисничког имена и шифре.

Slika 3 Прозор за пријављивање

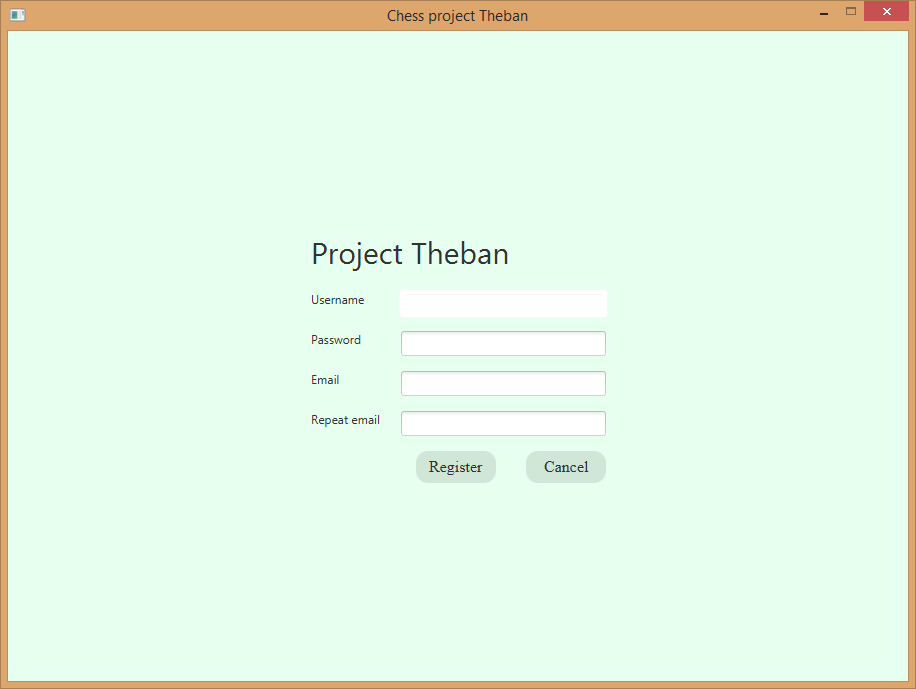
Прозор са странем за пријављивање

Страна за пријављивање се креира методом getLoginPage, а затим се додаје на примарни stage.

|  |
| --- |
| **Изворни код методе: getLoginPage** |
| 1. private Pane getLoginPage(Stage prStage) { 2. GridPane loginPagePane = new GridPane(); 3. loginPagePane.setAlignment(Pos.CENTER); 4. loginPagePane.setVgap(15); 5. loginPagePane.setStyle("-fx-background-color: " + backgroundColorValue); 7. Label gameNameLogin = new Label(graphicLogic.getLanguage().getGameTitleText()); 8. gameNameLogin.getStyleClass().add("game-title"); 9. loginPagePane.add(gameNameLogin, 0, 0); 11. Label usernameText = new Label(graphicLogic.getLanguage().getUsernameText()); 12. usernameText.setPrefWidth(55); 13. usernameText.getStyleClass().add("text-label"); 14. usernameLogin = new TextField(); 15. usernameLogin.getStyleClass().add("text-field"); 16. usernameLogin.setPrefColumnCount(17); 17. HBox usernameHBox = new HBox(20); 18. usernameHBox.getChildren().addAll(usernameText, usernameLogin); 20. loginPagePane.add(usernameHBox, 0, 1); 22. Label passwordText = new Label(graphicLogic.getLanguage().getPasswordText()); 23. passwordText.setPrefWidth(55); 24. passwordText.getStyleClass().add("text-label"); 25. passwordLogin = new PasswordField(); 26. passwordLogin.setPrefColumnCount(17); 27. HBox passwordHBox = new HBox(20); 28. passwordHBox.getChildren().addAll(passwordText, passwordLogin); 30. loginPagePane.add(passwordHBox, 0, 2); 32. HBox loginButtonsHBox = new HBox(30); 33. loginButton = new Button("Login"); 34. loginButton.setPrefWidth(80); 35. loginButton.getStyleClass().add("custom-button"); 36. loginButton.setOnAction(e -> { 37. LoginRequest loginRequest = new LoginRequest(usernameLogin.getText(), passwordLogin.getText()); 38. try { 39. toServer.writeObject(loginRequest); 40. } catch (Exception e1) { 41. e1.printStackTrace(); 42. } 44. }); 45. signUpButton = new Button(graphicLogic.getLanguage().getSignUpText()); 46. signUpButton.setPrefWidth(80); 47. signUpButton.getStyleClass().add("custom-button"); 48. signUpButton.setOnAction(e -> { 49. Scene scene = new Scene(getSignUpPage(prStage), 900, 650); 50. scene.getStylesheets().add("file:src/style.css"); 51. prStage.setScene(scene); 52. }); 53. loginButtonsHBox.getChildren().addAll(loginButton, signUpButton); 54. loginButtonsHBox.setAlignment(Pos.CENTER\_RIGHT); 56. loginPagePane.add(loginButtonsHBox, 0, 3); 57. return loginPagePane; 59. } |

Прозор са страном за регистровање

Уколико је корисник нов, и још увек није регистрован, то може урадити тако што ће кликнути на дугме Sign up, што ће га одвести на страну за регистровање. Да би се регистровао корисник мора да унесе жељено корисничко име, шифру и email адресу.



Slika 4 прозор за регистровање

Страна за регистровање се креира методом getRegisterPage

|  |
| --- |
| **Изворни код методе: getRegisterPage** |
| 1. private Pane getRegisterPage(Stage prStage) { 2. GridPane registerPane = new GridPane(); 3. registerPane.setAlignment(Pos.CENTER); 4. registerPane.setVgap(15); 5. registerPane.setStyle("-fx-background-color: " + backgroundColorValue); 7. textRegisterPage = new Label("Thank you for registering"); 9. registerPane.add(textRegisterPage, 0, 0); 11. okRegisterButton = new Button("OK"); 12. okRegisterButton.setPrefWidth(80); 13. okRegisterButton.getStyleClass().add("custom-button"); 14. okRegisterButton.setOnAction(e -> { 15. Scene scene = new Scene(getLoginPage(prStage), 900, 650); 16. scene.getStylesheets().add("file:src/style.css"); 17. prStage.setScene(scene); 18. }); 20. GridPane.setHalignment(okRegisterButton, HPos.CENTER); 21. registerPane.add(okRegisterButton, 0, 1); 23. return registerPane; 24. } |

Прозор са главном страном

Када се корисник пријави, појављује му сe главни прозор апликације. Са леве стране налази се мени који омогућава кориснику да одабере да игра игру са ботом, у средини је списак свих игара којим корисник може да се прикључи, и мени који му омогућава да креира нову партију одређене дужине кликом на дугме Search. Дружину бира кликом на одговарајући RadioButton. Са десне стране се налази списак корисникових пријатеља, а на врху инфомацијама о кориснику: његово корисничко име, рејтинг и број тренутно пријављених играча.



Slika 5 Главни прозор

Метод за креирање главне сцене је getMainPane. Главна сцена се састоји из четири дела. Горњег, централног, десног и левог. Горњи pane креира метод getTopBarPane, цетрални се креира у самој методи getMainPane, десни се креира методом getOnlineFriends, а леви се креира методом getBotsGamePane

|  |
| --- |
| **Изворни код методе: getMainPane, getTopBarPane, getOnlineFriends, getBotsGamePane** |
| 1. private Pane getMainPane(Stage prStage) { 2. BorderPane pane = new BorderPane(); 3. HBox hBox1 = new HBox(10); 4. RadioButton chose1min = new RadioButton("1 min"); 5. RadioButton chose5min = new RadioButton("5 min"); 6. RadioButton chose10min = new RadioButton("10 min"); 7. RadioButton chose20min = new RadioButton("20 min"); 8. ToggleGroup toggleGroup = new ToggleGroup(); 9. chose1min.setToggleGroup(toggleGroup); 10. chose5min.setToggleGroup(toggleGroup); 11. chose10min.setToggleGroup(toggleGroup); 12. chose20min.setToggleGroup(toggleGroup); 14. searchForGame = new Button(graphicLogic.getLanguage().getSearchText()); 15. searchForGame.setOnAction(e -> { 16. if (searchForGame.getText() == "Search") { 17. Platform.runLater(() -> { 18. searchForGame.setText("Cancel"); 19. }); 20. int time; 21. if (chose1min.isSelected()) 22. time = 1; 23. else if (chose5min.isSelected()) 24. time = 5; 25. else if (chose10min.isSelected()) 26. time = 10; 27. else 28. time = 20; 29. SearchForGame search = new SearchForGame(account.getId(), time, account.getUsername(), account.getRating()); 30. try { 31. toServer.writeObject(search); 32. } catch (Exception e1) { 33. e1.printStackTrace(); 34. } 35. } 36. else { 37. Platform.runLater(() -> { 38. searchForGame.setText("Search"); 39. }); 40. try { 41. toServer.writeObject(new SearchForGameCancel(account.getId())); 42. } catch (Exception e1) { 43. e1.printStackTrace(); 44. } 45. } 46. }); 47. hBox1.getChildren().addAll(chose1min, chose5min, chose10min, chose20min, searchForGame); 48. hBox1.setPadding(new Insets(5));  51. Pane toolbarPane = getTopBarPane(prStage); 53. pane.setTop(toolbarPane); 54. BorderPane.setAlignment(toolbarPane, Pos.CENTER); 56. Pane leftPlayPane = getBotsGamePane(prStage); 57. pane.setLeft(leftPlayPane); 59. GridPane paneCenter = new GridPane(); 61. Pane searchingPane = getOnlinePlayersReadyForGamePane(prStage); 62. paneCenter.add(searchingPane, 0, 0); 63. paneCenter.add(hBox1, 0, 1); 65. pane.setCenter(paneCenter); 66. BorderPane.setMargin(paneCenter, new Insets(15)); 68. Pane rightMainPane = getOnlineFriends(prStage); 69. rightMainPane.setPrefWidth(200); 70. pane.setRight(rightMainPane); 71. BorderPane.setMargin(rightMainPane, new Insets(10));  74. pane.setStyle("-fx-background-color: " + backgroundColorValue); 75. pane.getStyleClass().add("topBarBoxMain"); 76. return pane; 77. } 78. private Pane getTopBarPane(Stage prStage) { 79. HBox topHBox = new HBox(); 81. homeButton = new Button("Home"); 82. homeButton.setPrefHeight(30); 83. homeButton.getStyleClass().add("topBarText"); 84. homeButton.setPrefWidth(100); 85. homeButton.setAlignment(Pos.CENTER); 86. homeButton.setOnAction(e -> { 87. if (playingOnlineGame) { 88. Stage stage2 = new Stage(); 89. GridPane gridPane = new GridPane(); 90. Label lab = new Label("You will lose this game if you click yes!"); 91. Button yesBt = new Button("Yes"); 92. yesBt.setOnAction(e1 -> { 93. Scene scene = new Scene(getMainPane(prStage), 900, 650); 94. scene.getStylesheets().add("file:src/style.css"); 95. prStage.setScene(scene); 96. playingOnlineGame = false; 97. stage2.close();  100. //IZRACUNAJ GUBITKE POENA 102. try { 103. toServer.writeObject(new GameSessionEnded(account.getId())); 104. } catch (Exception e3) { 105. e3.printStackTrace(); 106. } 107. }); 108. Button noBt = new Button("No"); 109. noBt.setOnAction(e2 -> { 110. stage2.close(); 111. }); 112. gridPane.setAlignment(Pos.CENTER); 113. gridPane.add(lab, 0, 0, 2, 1); 114. gridPane.add(yesBt, 0, 1); 115. gridPane.add(noBt, 1, 1); 116. Scene scene2 = new Scene(gridPane); 117. stage2.setScene(scene2); 118. stage2.show(); 120. } 121. else { 122. Scene scene = new Scene(getMainPane(prStage), 900, 650); 123. scene.getStylesheets().add("file:src/style.css"); 124. prStage.setScene(scene); 125. } 126. }); 128. topHBox.getChildren().add(homeButton); 130. Image image = new Image("file:res/images/user.png"); 131. avatarImage = new ImageView(image); 132. avatarImage.setFitWidth(50); 133. avatarImage.setFitHeight(50); 135. topHBox.getChildren().add(avatarImage); 137. userNameMain = new Label(account.getUsername()); 138. userNameMain.setPrefHeight(30); 139. userNameMain.getStyleClass().add("topBarText"); 140. userNameMain.setPrefWidth(100); 141. userNameMain.setAlignment(Pos.CENTER); 143. topHBox.getChildren().add(userNameMain); 145. userRatingMain = new Label(Integer.toString(account.getRating())); 146. userRatingMain.setPrefHeight(30); 147. userRatingMain.getStyleClass().add("topBarText"); 148. userRatingMain.setPrefWidth(100); 149. userRatingMain.setAlignment(Pos.CENTER); 151. topHBox.getChildren().add(userRatingMain); 152. onlinePlayersMain = new Label(Integer.toString(onlinePlayersNo)); 153. onlinePlayersMain.setPrefHeight(30); 154. onlinePlayersMain.getStyleClass().add("topBarText"); 155. onlinePlayersMain.setPrefWidth(100); 156. onlinePlayersMain.setAlignment(Pos.CENTER); 158. topHBox.getChildren().add(onlinePlayersMain); 160. settingsMain = new Label(graphicLogic.getLanguage().getSettingsText()); 161. settingsMain.setPrefHeight(30); 162. settingsMain.getStyleClass().add("topBarText"); 163. settingsMain.setPrefWidth(100); 164. settingsMain.setAlignment(Pos.CENTER); 165. settingsMain.setOnMouseClicked(e -> { 166. Scene scene = new Scene(getSettingsPane(prStage), 900, 650); 167. scene.getStylesheets().add("file:src/style.css"); 168. prStage.setScene(scene); 169. }); 171. topHBox.getChildren().add(settingsMain); 173. signOutMain = new Label(graphicLogic.getLanguage().getSignOutText()); 174. signOutMain.setPrefHeight(30); 175. signOutMain.getStyleClass().add("topBarText"); 176. signOutMain.setPrefWidth(100); 177. signOutMain.setAlignment(Pos.CENTER); 178. signOutMain.setOnMousePressed(e -> { 179. try { 180. toServer.writeObject(new SignOutNotification(account.getId())); 181. } catch (Exception e1) { 182. e1.printStackTrace(); 183. } 184. account = null; 185. Scene scene = new Scene(getLoginPage(prStage), 900, 650); 186. scene.getStylesheets().add("file:src/style.css"); 187. prStage.setScene(scene); 188. }); 190. topHBox.getChildren().add(signOutMain); 191. HBox.setMargin(avatarImage, new Insets(15)); 192. HBox.setMargin(userNameMain, new Insets(15)); 193. HBox.setMargin(userRatingMain, new Insets(15)); 194. HBox.setMargin(onlinePlayersMain, new Insets(15)); 195. HBox.setMargin(settingsMain, new Insets(15)); 196. HBox.setMargin(signOutMain, new Insets(15)); 198. topHBox.getStyleClass().add("topBarBoxMain"); 199. topHBox.setAlignment(Pos.CENTER); 201. return topHBox; 202. } 203. private Pane getOnlineFriends(Stage prStage) { 204. try { 205. toServer.writeObject(new OnlineFriendsMessage(account.getId())); 206. } catch (IOException e) { 207. e.printStackTrace(); 208. } 209. GridPane pane = new GridPane(); 210. Label onlineFriendsLabel = new Label(graphicLogic.getLanguage().getOnlineFriends()); 211. onlineFriendsLabel.setAlignment(Pos.CENTER); 212. onlineFriendsLabel.setStyle("-fx-font-size: 18"); 213. onlineFriends = new VBox(10); 214. onlineFriends.getChildren().add(new Label("Not sign in")); 215. onlineFriends.setPrefWidth(150); 216. onlineFriends.setPrefHeight(300);  219. pane.add(onlineFriendsLabel, 0, 0); 220. pane.add(new ScrollPane(onlineFriends), 0, 1); 221. GridPane.setMargin(onlineFriendsLabel, new Insets(15)); 222. GridPane.setMargin(onlineFriends, new Insets(15)); 223. return pane; 224. } 225. private Pane getBotsGamePane(Stage prStage) { 226. GridPane gridPane = new GridPane(); 227. Label playWithBotsLabel = new Label(graphicLogic.getLanguage().getPlayAGameWithBotsText()); 228. playWithBotsLabel.setAlignment(Pos.CENTER); 229. playWithBotsLabel.setStyle("-fx-font-size: 18"); 230. gridPane.add(playWithBotsLabel, 0, 0); 232. buttonFor1min = new Button("1"); 233. buttonFor1min.setPrefSize(80, 80); 234. buttonFor1min.setStyle("-fx-background-radius: 50"); 235. buttonFor1min.setOnAction(e -> { 236. game = new Engine(GAME\_MODE\_BOTS, TEAM\_WHITE); 237. playerTimer = new PlayerTimer(1, TEAM\_WHITE); 238. opponentTimer = new PlayerTimer(1, TEAM\_BLACK); 239. startATimer(prStage); 240. Scene scene = new Scene(getGamePane(prStage, "Bot", 0), 900, 650); 241. scene.getStylesheets().add("file:src/style.css"); 242. prStage.setScene(scene); 244. }); 246. buttonFor5min = new Button("5"); 247. buttonFor5min.setPrefSize(80, 80); 248. buttonFor5min.setStyle("-fx-background-radius: 50"); 249. buttonFor5min.setOnAction(e -> { 250. game = new Engine(GAME\_MODE\_BOTS, TEAM\_WHITE); 251. playerTimer = new PlayerTimer(5, TEAM\_WHITE); 252. opponentTimer = new PlayerTimer(5, TEAM\_BLACK); 253. startATimer(prStage); 254. Scene scene = new Scene(getGamePane(prStage, "Bot", 0), 900, 650); 255. scene.getStylesheets().add("file:src/style.css"); 256. prStage.setScene(scene); 258. });  261. HBox hBox1 = new HBox(30); 262. hBox1.getChildren().addAll(buttonFor1min, buttonFor5min); 264. buttonFor10min = new Button("10"); 265. buttonFor10min.setPrefSize(80, 80); 266. buttonFor10min.setStyle("-fx-background-radius: 50"); 267. buttonFor10min.setOnAction(e -> { 268. game = new Engine(GAME\_MODE\_BOTS, TEAM\_WHITE); 269. playerTimer = new PlayerTimer(10, TEAM\_WHITE); 270. opponentTimer = new PlayerTimer(10, TEAM\_BLACK); 271. startATimer(prStage); 272. Scene scene = new Scene(getGamePane(prStage, "Bot", 0), 900, 650); 273. scene.getStylesheets().add("file:src/style.css"); 274. prStage.setScene(scene); 276. }); 278. buttonFor20min = new Button("20"); 279. buttonFor20min.setPrefSize(80, 80); 280. buttonFor20min.setStyle("-fx-background-radius: 50"); 281. buttonFor20min.setOnAction(e -> { 282. game = new Engine(GAME\_MODE\_BOTS, TEAM\_WHITE); 283. playerTimer = new PlayerTimer(20, TEAM\_WHITE); 284. opponentTimer = new PlayerTimer(20, TEAM\_BLACK); 285. startATimer(prStage); 286. Scene scene = new Scene(getGamePane(prStage, "Bot", 0), 900, 650); 287. scene.getStylesheets().add("file:src/style.css"); 288. prStage.setScene(scene); 290. }); 292. HBox hBox2 = new HBox(30); 293. hBox2.getChildren().addAll(buttonFor10min, buttonFor20min); 295. gridPane.add(hBox1, 0, 1); 296. gridPane.add(hBox2, 0, 2); 298. gridPane.setStyle("-fx-border-color: black; -fx-border-width: 1; -fx-border-radius: 10"); 299. GridPane.setHalignment(playWithBotsLabel, HPos.CENTER); 300. GridPane.setMargin(hBox1, new Insets(20)); 301. GridPane.setMargin(hBox2, new Insets(20)); 302. GridPane.setMargin(playWithBotsLabel, new Insets(20)); 304. return gridPane; 305. } |

Прозор са игром у току

Када корисник започне партију, позива се метода getGamePane која у левом делу прозора поставља саховску таблу, док у десном поставља део четовање и статус бар.

Статус бар приказује корисничка имена играча, њихове рејтинге и тајмере.



Slika 6 Изглед прозора са игром

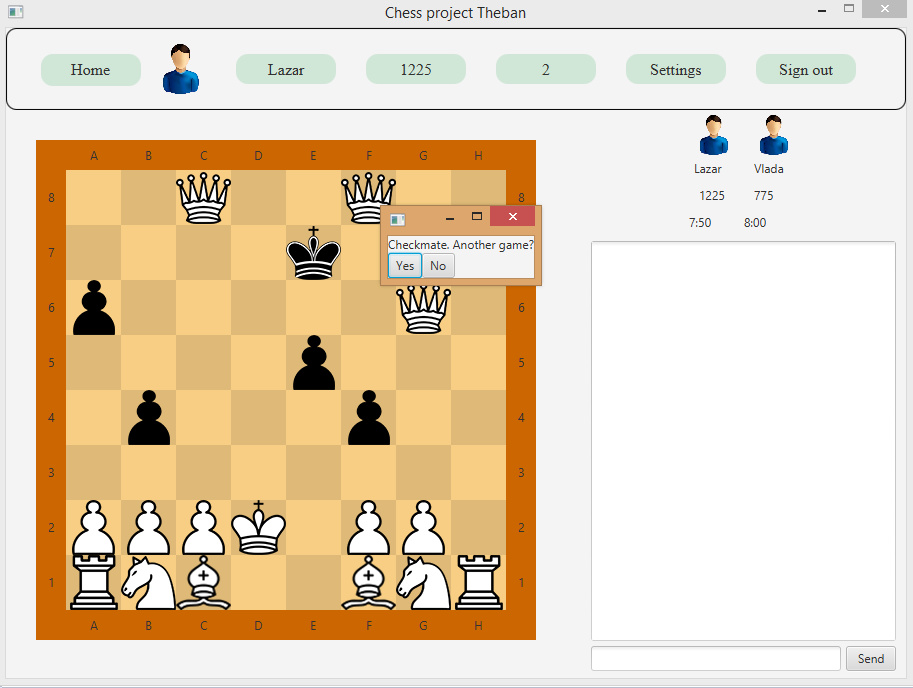
Остали прозори који се појављују у току игре

У току игре, ако у статус бару кликенте на корисничко име противника, добијате могућност да додате противника у листу пријатеља.



Slika 7 Додавање противника за пријатеља

У случају да се деси сахмат или да буде нерешено корисницима се приказује прозор који их пита да ли желе да играју нову партију. Овај прозор се такође приказује и при истеку времена неком од корисника.



Slika 8 Крај партије и порука за нову игур

|  |
| --- |
| **Пример кода за приказ упита за нову игру** |
| 1. GridPane gridPane = new GridPane(); 2. Stage endGameStage = new Stage(); 3. Label label = new Label("Draw" + ". Another game?"); 4. gridPane.add(label, 0, 0, 2, 1); 5. Button bt1 = new Button("Yes"); 6. bt1.setOnAction(e1 -> { 7. try { 8. toServer.writeObject(new AnswerRematch(account.getId(), true)); 9. } catch (Exception e2) { 10. e2.printStackTrace(); 11. } 12. endGameStage.close(); 13. }); 14. Button bt2 = new Button("No"); 15. bt2.setOnAction(e1 -> { 16. try { 17. toServer.writeObject(new GameSessionEnded(account.getId())); 18. playingOnlineGame = false; 19. } catch (Exception e2) { 20. e2.printStackTrace(); 21. } 22. Scene scene = new Scene(getMainPane(prStage), 900, 650); 23. scene.getStylesheets().add("file:src/style.css"); 24. prStage.setScene(scene); 25. endGameStage.close(); 26. }); 27. gridPane.add(bt1, 0, 1); 28. gridPane.add(bt2, 1, 1);   32. Scene sceneE = new Scene(gridPane); 33. endGameStage.setScene(sceneE); 34. endGameStage.show(); |

Метод RefreshGUIBoard

|  |
| --- |
| **Изворни код методе: RefreshGUIBoard** |
| 1. private void refreshGUIBoard(Stage prStage) { 2. int i; 3. int j; 4. for (i = 0; i < 8; i++) 5. for (j = 0; j < 8; j++) { 6. buttonsChess[i][j].setGraphic(new ImageView(game.getFigureImage(i, j))); 7. if ((i \* 8 + j + i % 2) % 2 == 0) 8. buttonsChess[i][j].setBackground(new Background(new BackgroundFill(graphicLogic.getTheme().getWhiteFieldColor(), CornerRadii.EMPTY, Insets.EMPTY))); 9. else 10. buttonsChess[i][j].setBackground(new Background(new BackgroundFill(graphicLogic.getTheme().getBlackFieldColor(), CornerRadii.EMPTY, Insets.EMPTY))); 11. buttonsChess[i][j].setDisable(true); 12. buttonsChess[i][j].setOpacity(1); 13. } 14. try { 15. int k = 0; 16. ArrayList<Point> points = game.getMovableFigures(); 17. while (k < points.size()) { 18. buttonsChess[points.get(k).getI()][points.get(k).getJ()].setDisable(false); 19. k++; 20. } 21. } catch (Draw e) { 22. try { 23. toServer.writeObject(new GameEnd(account.getId(), GAME\_DRAW)); 24. } catch (IOException e3) { 25. e3.printStackTrace(); 26. } 27. GridPane gridPane = new GridPane(); 28. Stage endGameStage = new Stage(); 29. Label label = new Label(graphicLogic.getLanguage().getDrawAnotherGameText()); 30. gridPane.add(label, 0, 0, 2, 1); 31. Button bt1 = new Button(graphicLogic.getLanguage().getYesText()); 32. bt1.setOnAction(e1 -> { 33. try { 34. toServer.writeObject(new AnswerRematch(account.getId(), true)); 35. } catch (Exception e2) { 36. e2.printStackTrace(); 37. } 38. endGameStage.close(); 39. }); 40. Button bt2 = new Button(graphicLogic.getLanguage().getNoText()); 41. bt2.setOnAction(e1 -> { 42. try { 43. toServer.writeObject(new GameSessionEnded(account.getId())); 44. playingOnlineGame = false; 45. } catch (Exception e2) { 46. e2.printStackTrace(); 47. } 48. Scene scene = new Scene(getMainPane(prStage), 900, 650); 49. scene.getStylesheets().add("file:src/style.css"); 50. prStage.setScene(scene); 51. endGameStage.close(); 52. }); 53. gridPane.add(bt1, 0, 1); 54. gridPane.add(bt2, 1, 1);   58. Scene sceneE = new Scene(gridPane); 59. endGameStage.setScene(sceneE); 60. endGameStage.show(); 62. } catch (Checkmate e) { 63. try { 64. toServer.writeObject(new GameEnd(account.getId(), GAME\_LOST)); 65. } catch (IOException e3) { 66. e3.printStackTrace(); 67. } 68. GridPane gridPane = new GridPane(); 69. Stage endGameStage = new Stage(); 70. Label label = new Label(graphicLogic.getLanguage().getCheckmateAnotherGameText()); 71. gridPane.add(label, 0, 0, 2, 1); 72. Button bt1 = new Button(graphicLogic.getLanguage().getYesText()); 73. bt1.setOnAction(e1 -> { 74. try { 75. toServer.writeObject(new AnswerRematch(account.getId(), true)); 76. } catch (Exception e2) { 77. e2.printStackTrace(); 78. } 80. endGameStage.close(); 81. }); 82. Button bt2 = new Button(graphicLogic.getLanguage().getNoText()); 83. bt2.setOnAction(e1 -> { 84. try { 85. toServer.writeObject(new GameSessionEnded(account.getId())); 86. playingOnlineGame = false; 87. } catch (Exception e2) { 88. e2.printStackTrace(); 89. } 90. Scene scene = new Scene(getMainPane(prStage), 900, 650); 91. scene.getStylesheets().add("file:src/style.css"); 92. prStage.setScene(scene); 93. endGameStage.close(); 94. }); 95. gridPane.add(bt1, 0, 1); 96. gridPane.add(bt2, 1, 1);   100. Scene sceneE = new Scene(gridPane); 101. endGameStage.setScene(sceneE); 102. endGameStage.show(); 104. } 105. } 106. private void nextMove(Stage prStage){ 107. if(isPromotionPlayed) { 108. if(game.getTypeOfGame() == GAME\_MODE\_ONLINE){ 109. try { 110. toServer.writeObject(new PlayerMove(account.getId(), game.getLastMove())); 111. } catch (Exception e1) { 112. System.out.println(e1.getMessage()); 113. } 114. refreshGUIBoard(prStage); 115. } else if(game.getTypeOfGame() == GAME\_MODE\_BOTS){ 116. try{ 117. game.getBot().play(); 118. refreshGUIBoard(prStage); 119. } 120. catch (Exception e1) { 122. Label label = new Label(); 123. if(e1 instanceof Checkmate) 124. label.setText(graphicLogic.getLanguage().getCheckmateAnotherGameText()); 125. else if(e1 instanceof Draw) 126. label.setText(graphicLogic.getLanguage().getDrawAnotherGameText()); 128. GridPane gridPane = new GridPane(); 129. Stage endGameStage = new Stage(); 131. gridPane.add(label, 0, 0, 2, 1); 132. Button bt1 = new Button(graphicLogic.getLanguage().getYesText()); 133. bt1.setOnAction(e -> { 134. game.newGame(); 135. endGameStage.close(); 136. refreshGUIBoard(prStage); 137. }); 138. Button bt2 = new Button(graphicLogic.getLanguage().getNoText()); 139. bt2.setOnAction(e -> { 140. Scene scene = new Scene(getMainPane(prStage), 900, 650); 141. scene.getStylesheets().add("file:src/style.css"); 142. prStage.setScene(scene); 143. endGameStage.close(); 144. }); 145. gridPane.add(bt1, 0, 1); 146. gridPane.add(bt2, 1, 1);   150. Scene sceneE = new Scene(gridPane); 151. endGameStage.setScene(sceneE); 152. endGameStage.show(); 154. } 155. try { 156. Thread.sleep(500); 157. opponentTimer.reduce(); 158. } catch (Exception e1) { 159. e1.printStackTrace(); 160. } 161. } 162. } 163. } |

Функција RefreshGUIBoard освежава шаховску таблу, тако што псотавља боје на дефолтну вредност и искључује сва поља на која корисник не може да кликне. Позиције на које корисник може да кликне добија од методе getPosslibeMoves из енџина.

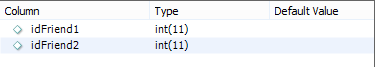
Померање фигура

|  |
| --- |
| **Изворни код: ActionListener I nextMove** |
| 1. ArrayList<Point> list = new ArrayList<Point>(); 2. int ii = ((ChessButton)e.getSource()).getI(); 3. int jj = ((ChessButton)e.getSource()).getJ(); 5. list = game.getPossibleMoves(ii, jj); 6. if(list==null){ 7. try { 8. game.playMove(new Point(ii, jj)); 9. } catch (Promotion e1) { 10. isPromotionPlayed = false; 11. Stage choseStage = new Stage(); 12. GridPane paneChoser = new GridPane(); 13. paneChoser.setAlignment(Pos.CENTER); 14. FigureChoseButton bishopChose = new FigureChoseButton(game.getOnMove(), "Bishop"); 15. bishopChose.setOnAction(e2 -> { 16. game.zamena(ii, jj, GameConstants.FIGURE\_BISHOP); 17. isPromotionPlayed = true; 18. refreshGUIBoard(prStage); 19. nextMove(prStage); 20. choseStage.close(); 21. }); 22. FigureChoseButton knightChose = new FigureChoseButton(game.getOnMove(), "Knight"); 23. knightChose.setOnAction(e2 -> { 24. game.zamena(ii, jj, GameConstants.FIGURE\_KNIGHT); 25. isPromotionPlayed = true; 26. refreshGUIBoard(prStage); 27. nextMove(prStage); 28. choseStage.close(); 29. }); 30. FigureChoseButton queenChose = new FigureChoseButton(game.getOnMove(), "Queen"); 31. queenChose.setOnAction(e2 -> { 32. game.zamena(ii, jj, GameConstants.FIGURE\_QUEEN); 33. isPromotionPlayed = true; 34. refreshGUIBoard(prStage); 35. nextMove(prStage); 36. choseStage.close(); 37. }); 38. FigureChoseButton rookChose = new FigureChoseButton(game.getOnMove(), "Rook"); 39. rookChose.setOnAction(e2 -> { 40. game.zamena(ii, jj, FIGURE\_ROOK); 41. isPromotionPlayed = true; 42. refreshGUIBoard(prStage); 43. nextMove(prStage); 44. choseStage.close(); 45. }); 46. Label choseFigureLabel = new Label(graphicLogic.getLanguage().getChoseAFigureText()); 47. paneChoser.add(choseFigureLabel, 0, 0, 4, 1); 48. paneChoser.add(bishopChose, 0, 1); 49. paneChoser.add(knightChose, 1, 1); 50. paneChoser.add(queenChose, 2, 1); 51. paneChoser.add(rookChose, 3, 1); 53. Scene sceneChoser = new Scene(paneChoser, 300, 120); 55. choseStage.setScene(sceneChoser); 56. choseStage.setTitle(graphicLogic.getLanguage().getAlertText()); 57. choseStage.show(); 58. } 59. nextMove(prStage); 60. private void nextMove(Stage prStage){ 61. if(isPromotionPlayed) { 62. if(game.getTypeOfGame() == GAME\_MODE\_ONLINE){ 63. try { 64. toServer.writeObject(new PlayerMove(account.getId(), game.getLastMove())); 65. } catch (Exception e1) { 66. System.out.println(e1.getMessage()); 67. } 68. refreshGUIBoard(prStage); 69. } else if(game.getTypeOfGame() == GAME\_MODE\_BOTS){ 70. try{ 71. game.getBot().play(); 72. refreshGUIBoard(prStage); 73. } 74. catch (Exception e1) { 76. Label label = new Label(); 77. if(e1 instanceof Checkmate) 78. label.setText(graphicLogic.getLanguage().getCheckmateAnotherGameText()); 79. else if(e1 instanceof Draw) 80. label.setText(graphicLogic.getLanguage().getDrawAnotherGameText()); 82. GridPane gridPane = new GridPane(); 83. Stage endGameStage = new Stage(); 85. gridPane.add(label, 0, 0, 2, 1); 86. Button bt1 = new Button(graphicLogic.getLanguage().getYesText()); 87. bt1.setOnAction(e -> { 88. game.newGame(); 89. endGameStage.close(); 90. refreshGUIBoard(prStage); 91. }); 92. Button bt2 = new Button(graphicLogic.getLanguage().getNoText()); 93. bt2.setOnAction(e -> { 94. Scene scene = new Scene(getMainPane(prStage), 900, 650); 95. scene.getStylesheets().add("file:src/style.css"); 96. prStage.setScene(scene); 97. endGameStage.close(); 98. }); 99. gridPane.add(bt1, 0, 1); 100. gridPane.add(bt2, 1, 1);   104. Scene sceneE = new Scene(gridPane); 105. endGameStage.setScene(sceneE); 106. endGameStage.show(); 108. } 109. try { 110. Thread.sleep(500); 111. opponentTimer.reduce(); 112. } catch (Exception e1) { 113. e1.printStackTrace(); 114. } 115. }   } |

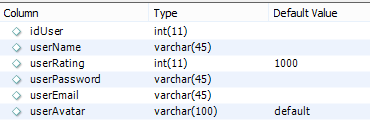
Код изнад је задужен да обради померање фигуре са графичке стране. Када корисник кликне на неко поље, позива се метод getPossibleMoves, који смо претходно објаснили. Ако метода врати листу, сва поља из те листе се фарбају зеленом бојом, и предтављају поља на која фигура може да скочи. Када корисник кликне на неко од зелених поља позива се функција playMove, коју смо такође већ описали. Када се ово одради, позива се метод nextMove, која притвнику шаље потез и позива методу refreshGUIBoard. Ако играч игра против бота, онда ће функција nextMove покренути бота, како би он одиграо свој потез.

Серверски део пројекта

Серверски део се брине о размени података измећу корисника, и омогућава приступ бази, како би корисници могли да се региструју и пројаве у систем. У својој бази чува податке о корисницима и који корисници су пријатељи међусобно.



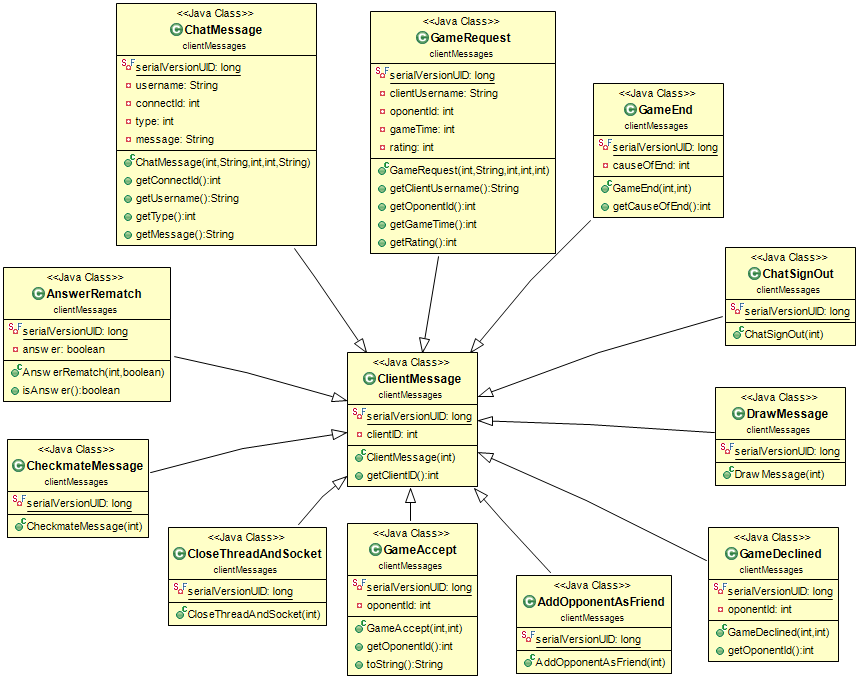
Slika 9 Изглед табеле која памти пријатеље



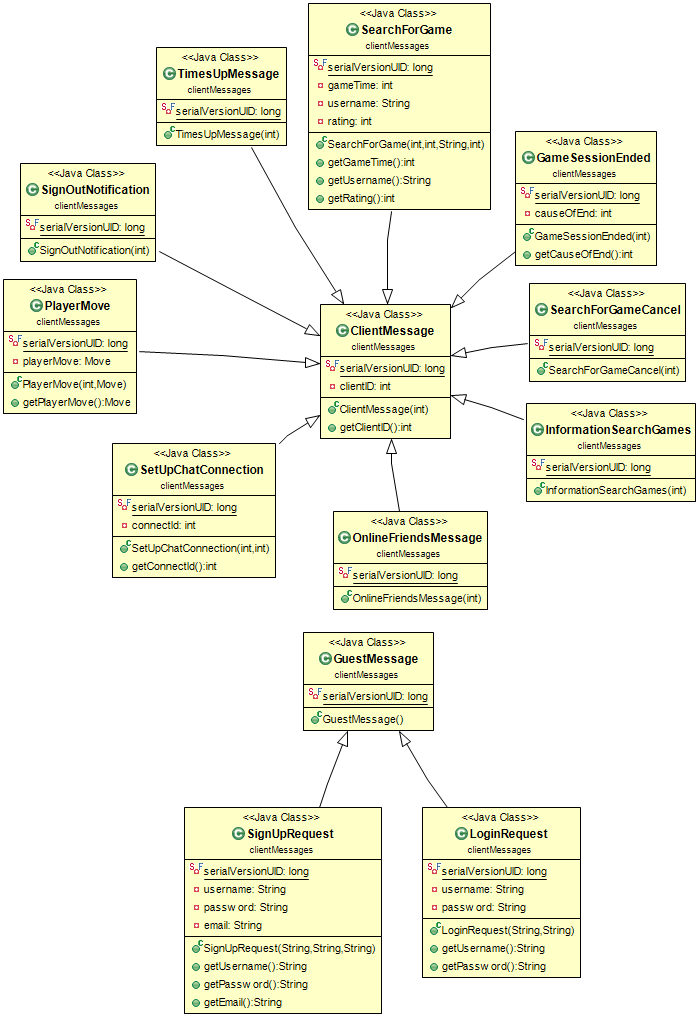
Slika 10 Изглед табеле која чува корисникове податке

Сваку комуникацију између сервера и корисника, сервер исписује на свој екран. Сервер са корисницима разговара преко сокета слањем одређених порука. Постоје две главне врсте порука, клијентске и серверске. Клијентске поруке су поруке које клијент шаље серверу, док су серверске поруке поруке које сервер шаљe кориснику. Све клијентске поруке, сем оних које се користе за пријављивање и регистровање налсеђују класу ClientMessage. Поруке које служе за регистровање и пријављивање на сервер наслеђују класу GuestMessage.

|  |
| --- |
| **Изворни код класе: clientMessage и LoginRequest** |
| 1. package clientMessages; 2. import game.GameConstants; 3. import java.io.Serializable; 4. public class ClientMessage implements Serializable, GameConstants { 5. private static final long serialVersionUID = 4404802185294150865L; 7. private int clientID; 9. public ClientMessage(int clientID) { 10. this.clientID = clientID; 11. } 12. public int getClientID() { 13. return clientID; 14. } 15. } 16. public class LoginRequest extends GuestMessage { 17. private static final long serialVersionUID = 5169802647953397087L; 18. private String username; 19. private String password; 21. public LoginRequest(String username, String password) { 22. this.username = username; 23. this.password = password; 24. } 25. public String getUsername() { 26. return username; 27. } 28. public String getPassword() { 29. return password; 30. } 31. } |

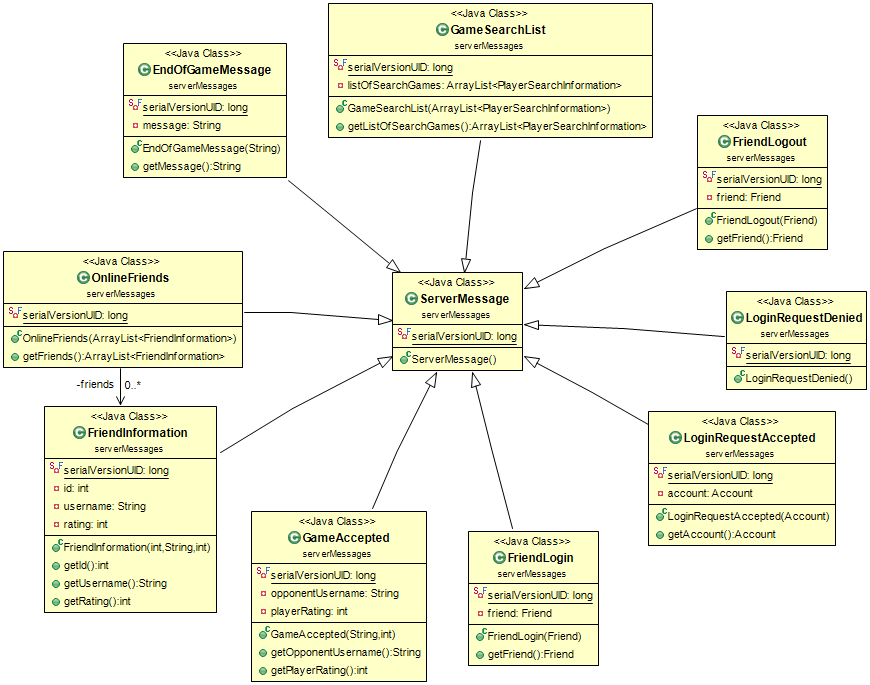


Slika 11 Неке од клијентских порука

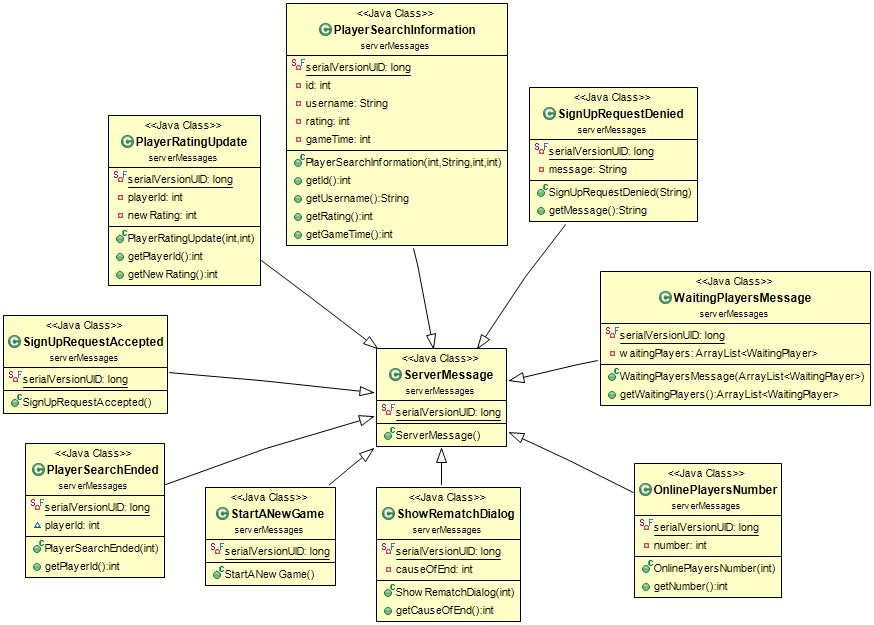


Slika 12 Остатак клијентских порука

Серверксе поруке населђују класу serverMessage.



Slika 13 Део серверских порука



Slika 14 Други део серверских порука

На једном треду сервер креира сервер сокет на порту 8000. Ту у бесконачној петљи прихвата све конекције од клијената и тиме креира сокет за сваког од њих. Затим креира објекат GuestConnection који прослеђује методи HandleAConnection, која креира посебан тред за сваку конекцију између сервера и корисника.

|  |
| --- |
| **Изворни код за серверски нит која прихвата конекције** |
| 1. new Thread(() -> { 2. try { 3. serverSocket = new ServerSocket(8000); 4. Class.forName("com.mysql.jdbc.Driver"); 5. connection = DriverManager.getConnection("jdbc:mysql://localhost/chessbase","chessUsername","chessPassword"); 6. while (true) { 7. Socket connectedPlayer = serverSocket.accept(); 8. InetAddress address = connectedPlayer.getInetAddress(); 9. Platform.runLater(() -> { 10. taLog.appendText(address.getHostAddress() + " connected to server\n"); 11. }); 12. GuestConnection guestConnection = new GuestConnection(connectedPlayer, new ObjectOutputStream(connectedPlayer.getOutputStream()), 13. new ObjectInputStream(connectedPlayer.getInputStream())); 14. new Thread(new HandleAConnection(guestConnection)).start(); 15. } 16. } catch (Exception e) { 17. e.printStackTrace(); 18. } 19. }).start(); |

Такав тред се састоји из бесконачне петље, тј. врти све док индикатор runAThread је true. Тред се искључује једино када корисник искључи свој програм. У овој петљи сервер чита поруке са сокета и у зависности од типа поруке извршава одређену радњу.

Објекти који су потреби сервеу су: userConnection, која садржи id корисника, његов username, сокет и ObjectOutputStream ка и од корисника. Сервер садржи ArrayList оваквих објеката који представљају кориснике који су онлајн. Помоћу ове листе сервер може да шаље поруке одређеним корисницима. За тренутне игре сервер користи ArrayList објеката Game. Објекат Game садржи две референце на објекте UserConnection. Захваљујући овоме сервер зна које се партије тренутно играју. И поруку за потезе ће послати само другом кориснику. На овај начин сервер може имати висе сесија игара. Још једна ArrayLista-а служи за сместање објеката RematchConformation, која пружа корисницима да наставе са игром по завршетку партије, тј. да одговоре да ли желе да одиграју нову игру са истим протувником. Такође постоји ArrayList-а са играма које се тренутно траже, тј. референце на GameSearch. Сваки корисник може затражити игру одређеног времена, оваква класа служи за то. Сервер такође садржи један integer који означава тренутни број пријављених играча. При свакој промени овог броја, он се шаље свим пријављеним корисницима.

Поруке које сервер обрађује

LoginRequest

|  |
| --- |
| **Изворни код обраде поруке: LoginRequest** |
| 1. LoginRequest loginRequest = (LoginRequest) object; 2. try { 3. preparedStatement = connection.prepareStatement(queryLogin); 4. preparedStatement.setString(1, loginRequest.getUsername()); 5. preparedStatement.setString(2, loginRequest.getPassword()); 7. ResultSet resSet = preparedStatement.executeQuery(); 8. if (resSet.next()) { 9. int id = Integer.parseInt(resSet.getString(1)); 10. String username = resSet.getString(2); 11. int rating = Integer.parseInt(resSet.getString(4)); 12. String avatar = resSet.getString(6); 14. Account account = new Account(id, username, rating, avatar); 16. Platform.runLater(() -> { 17. taLog.appendText(username + " with id " + id + " logged in\n"); 18. }); 20. guestConnection.getToGuest().writeObject(new LoginRequestAccepted(account)); 22. UserConnection userConn = new UserConnection(id, username, guestConnection.getSocket(), guestConnection.getToGuest(), guestConnection.getFromGuest()); 23. userConnection = new UserConnection(id, username, guestConnection.getSocket(), guestConnection.getToGuest(), guestConnection.getFromGuest()); 24. connections.add(userConn); 26. onlinePlayers++; 28. for (int i = 0; i < connections.size(); i++) { 29. connections.get(i).getToClient().writeObject(new OnlinePlayersNumber(onlinePlayers)); 30. }  33. preparedStatement = connection.prepareStatement(showFriendsQuery); 34. preparedStatement.setInt(1, account.getId()); 35. preparedStatement.setInt(2, account.getId()); 37. ResultSet resSetKome = preparedStatement.executeQuery(); 38. while (resSetKome.next()) { 39. int id1 = resSetKome.getInt(1); 40. int id2 = resSetKome.getInt(2); 41. int friendId; 42. if (id1 == account.getId()) 43. friendId = id2; 44. else 45. friendId = id1;  48. for (int g = 0; g < connections.size(); g++) 49. if (connections.get(g).getId() == friendId) { 50. connections.get(g).getToClient().writeObject(new FriendInformation(account.getId(), account.getUsername(), account.getRating())); 51. } 53. }  56. break; 57. } 58. else { 59. guestConnection.getToGuest().writeObject(new LoginRequestDenied()); 60. } 61. } catch (SQLException e1) { 62. e1.printStackTrace(); 63. } |

Ову поруку сервер обрађује тако што прво проверава да ли корисник са прослеђеним корисничким именом и сифром постоји у бази. Ако постоји креира објекат типа Account и порслеђује га кориснику. Такође креира објекат UserConnection и додаје га у листу пријављених корисника. Осталим корисницима прослеђује нов број пријављених корисника. А свим његовим пријатељима који су онлајн шаље да се пријавио. Ако корисник не постоји у бази послаће му се порука да је одбијен захтев за пријављивање.

SignUpRequest

|  |
| --- |
| **Изворни код обраде поруке: SignUpRequest** |
| 1. SignUpRequest signUpRequest = (SignUpRequest) object; 2. try { 3. preparedStatement = connection.prepareStatement(queryLogin); 4. preparedStatement.setString(1, signUpRequest.getUsername()); 5. preparedStatement.setString(2, signUpRequest.getPassword()); 7. ResultSet rSet = preparedStatement.executeQuery(); 9. if (rSet.next()) { 10. guestConnection.getToGuest().writeObject(new SignUpRequestDenied("I will do this later")); 11. } 12. else { 13. preparedStatement = connection.prepareStatement(querrySignUp); 14. preparedStatement.setString(1, signUpRequest.getUsername()); 15. preparedStatement.setString(2, signUpRequest.getPassword()); 16. preparedStatement.setString(3, signUpRequest.getEmail()); 18. preparedStatement.executeUpdate(); 20. Platform.runLater(() -> { 21. taLog.appendText("Account with " + signUpRequest.getUsername() + " was just created\n"); 22. }); 24. guestConnection.getToGuest().writeObject(new SignUpRequestAccepted()); 25. }  28. } catch (SQLException e) { 29. e.printStackTrace(); 30. } |

Прво из базе проверава да ли корисник са корисничким именом и шифром већ постоји у бази. Ако постоји, сервер ће му и послати му поруку SignUpRequestDenied. У упротном ће додати новог кросиника у бази и послати кориснику поруку SignUpRequestAccepted.

PlayerMove

|  |
| --- |
| **Изворни код обраде поруке: PlayerMove** |
| 1. PlayerMove pMove = (PlayerMove) object; 2. for (int i = 0; i < games.size(); i++) { 3. if (games.get(i).getUser1().getId() == pMove.getClientID() || games.get(i).getUser2().getId() == pMove.getClientID()) { 4. if (games.get(i).getUser1().getId() == pMove.getClientID()) { 5. games.get(i).getUser2().getToClient().writeObject(pMove); 6. } 7. else { 8. games.get(i).getUser1().getToClient().writeObject(pMove); 9. } 10. } 11. } |

Сервер прима поруку од играча, проналази којој игри припада, и противнику шаље потез.

ChatMessage

|  |
| --- |
| **Изворни код обраде поруке: ChatMessage** |
| 1. ChatMessage chatMessage = (ChatMessage) object; 2. if (chatMessage.getType() == IN\_GAME\_CHAT) { 3. for (int i = 0; i < games.size(); i++) { 4. if (games.get(i).getUser1().getId() == chatMessage.getClientID()) { 5. games.get(i).getUser2().getToClient().writeObject(chatMessage); 6. } 7. else if (games.get(i).getUser2().getId() == chatMessage.getClientID()) { 8. games.get(i).getUser1().getToClient().writeObject(chatMessage); 9. } 10. } 11. } |

Сервер прима поруку од играча, проналази којој игри припада и противнику шаље поруку са чета.

GameRequest

|  |
| --- |
| **Изворни код обраде поруке: GameRequest** |
| 1. Game game = new Game(null, null); 2. for (int i = 0; i < connections.size(); i++) { 3. if (gRequest.getOponentId() == connections.get(i).getId()) { 4. connections.get(i).getToClient().writeObject(gRequest); 5. } 6. if (gRequest.getClientID() == connections.get(i).getId()) { 7. game.setUser1(connections.get(i)); 8. } 9. } 10. games.add(game); |

Када корисник зели да са неким игра тражену игру, шаље серверу ову поруку. Сервер проверава са којим корисником жели да игра игру, и том кориснику прослеђује захтев и креира објекат типа Game, налази UserConnectio и ставља референцу тог корисника у Game.

GameAccept

|  |
| --- |
| **Изворни код обраде поруке: GameRequest** |
| 1. GameAccept gameAcc = (GameAccept) object; 2. for (int i = 0; i < games.size(); i++) { 3. if (games.get(i).getUser1().getId() == gameAcc.getOponentId()) { 4. for (int j = 0; j < connections.size(); j++) { 5. if (connections.get(j).getId() == gameAcc.getClientID()) { 6. games.get(i).setUser2(connections.get(j)); 7. int rating = 0; 8. try { 9. preparedStatement = connection.prepareStatement(getPlayerRatingQuery); 10. preparedStatement.setString(1, games.get(i).getUser2().getUsername()); 12. ResultSet resSet = preparedStatement.executeQuery(); 13. if (resSet.next()) { 14. rating = resSet.getInt(1); 15. } 16. } catch (SQLException e) { 17. e.printStackTrace(); 18. }   22. games.get(i).getUser1().getToClient().writeObject(new GameAccepted(games.get(i).getUser2().getUsername(), rating)); 24. // Posalji da ovaj igrac ne trazi vise igru 26. for (int g = 0; g < connections.size(); g++) { 27. connections.get(g).getToClient().writeObject(new PlayerSearchEnded(games.get(i).getUser1().getId())); 28. connections.get(g).getToClient().writeObject(new PlayerSearchEnded(games.get(i).getUser2().getId())); 29. }  32. for (int g = 0; g < gameSearch.size(); g++) { 33. if (gameSearch.get(g).getPlayerId() == games.get(i).getUser1().getId()) 34. gameSearch.remove(g); 35. } 36. for (int g = 0; g < gameSearch.size(); g++) { 37. if (gameSearch.get(g).getPlayerId() == games.get(i).getUser2().getId()) 38. gameSearch.remove(g); 39. } 40. } 41. } 42. } 43. } |

Допуљава Game објекат са другим корисником који је прихватио захтев. Затим из базе вати његов реитинг, и све то пакује у објекат типа GameAccepted, који шаље првом кориснику. Све остале кориснике обавештава да ови играчи више не траже игру.

GameDeclined

|  |
| --- |
| **Изворни код обраде поруке: GameDeclined** |
| 1. GameDeclined gameDec = (GameDeclined) object; 2. for (int i = 0; i < games.size(); i++) { 3. if (games.get(i).getUser1().getId() == gameDec.getOponentId()) { 4. games.remove(i); 5. break; 6. } 7. } |

Ако се деси да корисник коме је послат захтев за игру, одбије захтев, послаће се ова порука серверу. Сервер ће изацити објекат Game из листе.

GameSesionEnded

|  |
| --- |
| **Изворни код обраде поруке: GameSesionEnded** |
| 1. GameSessionEnded gameSE = (GameSessionEnded) object; 2. int user1Id = 0; 3. int user2Id = 0; 4. for (int i = 0; i < games.size(); i++) { 5. if (games.get(i).getUser1().getId() == gameSE.getClientID()) { 6. user1Id = games.get(i).getUser1().getId(); 7. user2Id = games.get(i).getUser2().getId(); 8. } 9. if (games.get(i).getUser2().getId() == gameSE.getClientID()) { 10. user1Id = games.get(i).getUser2().getId(); 11. user2Id = games.get(i).getUser1().getId(); 12. } 13. } 14. for (int i = 0; i < games.size(); i++) { 15. if (games.get(i).getUser1().getId() == gameSE.getClientID() || games.get(i).getUser2().getId() == gameSE.getClientID()) { 16. if (games.get(i).getUser1().getId() == gameSE.getClientID()) { 17. games.get(i).getUser2().getToClient().writeObject(gameSE); 18. } 19. else { 20. games.get(i).getUser1().getToClient().writeObject(gameSE); 21. } 22. } 23. } 24. for (int i = 0; i < games.size(); i++) { 25. if ((games.get(i).getUser1().getId() == user1Id && games.get(i).getUser2().getId() == user2Id) 26. || (games.get(i).getUser1().getId() == user2Id && games.get(i).getUser2().getId() == user1Id)) { 27. games.remove(i); 28. } 29. } |

Када корисник пошаље ову поруку серверу, он ће је проследити другом кориснику, тако да ђе и он занти да је дошло до крекида сесије игре. Из листе games се избацује њихова игра.

GameEnd

|  |
| --- |
| **Изворни код обраде поруке: GameEnd** |
| 1. GameEnd gameEnd = (GameEnd) object; 2. String userName1 = null; 3. nt user1Id = 0; 4. String userName2 = null; 5. int user2Id = 0; 6. for (int i = 0; i < games.size(); i++) { 7. if (games.get(i).getUser1().getId() == gameEnd.getClientID()) { 8. userName1 = games.get(i).getUser1().getUsername(); 9. user1Id = games.get(i).getUser1().getId(); 10. userName2 = games.get(i).getUser2().getUsername(); 11. user2Id = games.get(i).getUser2().getId(); 12. if (gameEnd.getCauseOfEnd() == GAME\_LOST) 13. games.get(i).getUser2().getToClient().writeObject(new ShowRematchDialog(GAME\_LOST)); 14. else 15. games.get(i).getUser2().getToClient().writeObject(new ShowRematchDialog(GAME\_DRAW)); 16. } 17. if (games.get(i).getUser2().getId() == gameEnd.getClientID()) { 18. userName1 = games.get(i).getUser2().getUsername(); 19. user1Id = games.get(i).getUser2().getId(); 20. userName2 = games.get(i).getUser1().getUsername(); 21. user2Id = games.get(i).getUser1().getId(); 22. if (gameEnd.getCauseOfEnd() == GAME\_LOST) 23. games.get(i).getUser1().getToClient().writeObject(new ShowRematchDialog(GAME\_LOST)); 24. else 25. games.get(i).getUser1().getToClient().writeObject(new ShowRematchDialog(GAME\_DRAW)); 26. } 27. } 28. rematchConformation.add(new RematchConformation(user1Id, user2Id)); 29. if (gameEnd.getCauseOfEnd() == GAME\_LOST) { 30. try { 31. preparedStatement = connection.prepareStatement(updateUserRating); 32. preparedStatement.setInt(1, -25); 33. preparedStatement.setString(2, userName1); 34. preparedStatement.executeUpdate(); 36. preparedStatement = connection.prepareStatement(updateUserRating); 37. preparedStatement.setInt(1, 25); 38. preparedStatement.setString(2, userName2); 39. preparedStatement.executeUpdate();   43. } catch (SQLException e) { 44. e.printStackTrace(); 45. } 47. try { 48. preparedStatement = connection.prepareStatement(getPlayerRatingQuery); 49. preparedStatement.setString(1, userName1); 51. ResultSet resSet = preparedStatement.executeQuery(); 52. if (resSet.next()) { 53. int rating = resSet.getInt(1); 54. for (int i = 0; i < connections.size(); i++) 55. if (connections.get(i).getId() == user1Id) { 56. connections.get(i).getToClient().writeObject(new PlayerRatingUpdate(user1Id, rating)); 57. } 58. } 60. preparedStatement = connection.prepareStatement(getPlayerRatingQuery); 61. preparedStatement.setString(1, userName2); 63. resSet = preparedStatement.executeQuery(); 64. if (resSet.next()) { 65. int rating = resSet.getInt(1); 66. for (int i = 0; i < connections.size(); i++) 67. if (connections.get(i).getId() == user2Id) { 68. connections.get(i).getToClient().writeObject(new PlayerRatingUpdate(user2Id, rating)); 69. } 70. } 72. } catch (SQLException e) { 73. e.printStackTrace(); 74. } |

Сервер ће пронаћи игру, а затим у бази ажурирати рејтинге играча те игре. Њима ће их затим проследити. Такође се креира објекат типа RematchConformation ће даље служити серверу како би могао да се одигра римеч.

AnswerRematch

|  |
| --- |
| **Изворни код обраде поруке: AnswerRematch** |
| 1. AnswerRematch ans = (AnswerRematch) object; 2. for (int i = 0; i < rematchConformation.size(); i++) { 3. if (ans.getClientID() == rematchConformation.get(i).getUserId1() || 4. ans.getClientID() == rematchConformation.get(i).getUserId2()) { 5. if (ans.getClientID() == rematchConformation.get(i).getUserId1()) { 6. rematchConformation.get(i).setAnswer1(ans); 7. if (ans.isAnswer()) { 8. if (rematchConformation.get(i).getAnswer2() != null && rematchConformation.get(i).getAnswer2().isAnswer()) { 9. for (int l = 0; l < games.size(); l++) { 10. if (games.get(l).getUser1().getId() == ans.getClientID() || games.get(l).getUser2().getId() == ans.getClientID()) { 12. games.get(l).getUser1().getToClient().writeObject(new StartANewGame()); 13. games.get(l).getUser2().getToClient().writeObject(new StartANewGame()); 14. } 15. } 16. } 17. } 18. else { 19. rematchConformation.remove(i); 20. for (int g = 0; g < connections.size(); g++) { 21. if (connections.get(g).getId() == rematchConformation.get(i).getUserId2()) { 22. connections.get(g).getToClient().writeObject(new GameSessionEnded(ans.getClientID())); 23. } 24. } 25. } 26. } 27. else { 28. rematchConformation.get(i).setAnswer2(ans); 29. if (ans.isAnswer()) { 30. if (rematchConformation.get(i).getAnswer1() != null && rematchConformation.get(i).getAnswer1().isAnswer()) { 31. for (int l = 0; l < games.size(); l++) { 32. if (games.get(l).getUser1().getId() == ans.getClientID() || games.get(l).getUser2().getId() == ans.getClientID()) { 33. games.get(l).getUser1().getToClient().writeObject(new StartANewGame()); 34. games.get(l).getUser2().getToClient().writeObject(new StartANewGame()); 35. } 36. } 37. } 38. } 39. else { 40. rematchConformation.remove(i); 41. for (int g = 0; g < connections.size(); g++) { 42. if (connections.get(g).getId() == rematchConformation.get(i).getUserId1()) { 43. connections.get(g).getToClient().writeObject(new GameSessionEnded(ans.getClientID())); 44. } 45. } 46. } 47. } 48. } 49. } |

Ову поруку корисници шаљу да би обавестили сервер да ли желе да одиграју римеч. Ако желе, сервер ће им послати поруку StartANewGame, у супротном ће овавестити корисника да је други одустао од игре.

AddOpponentAsFriend

|  |
| --- |
| **Изворни код обраде поруке: AddOpponentAsFriend** |
| 1. AddOpponentAsFriend addObj = (AddOpponentAsFriend) object; 2. for (int i = 0; i < games.size(); i++) { 3. if (games.get(i).getUser1().getId() == addObj.getClientID() || 4. games.get(i).getUser2().getId() == addObj.getClientID()) {   try {   1. preparedStatement = connection.prepareStatement(queryFriends); 2. preparedStatement.setInt(1, games.get(i).getUser1().getId()); 3. preparedStatement.setInt(2, games.get(i).getUser2().getId()); 4. preparedStatement.setInt(3, games.get(i).getUser2().getId()); 5. preparedStatement.setInt(4, games.get(i).getUser1().getId()); 7. ResultSet resSet = preparedStatement.executeQuery(); 9. if (resSet.next() == false) { 10. try { 11. preparedStatement = connection.prepareStatement(insertFriendsQuery); 12. preparedStatement.setInt(1, games.get(i).getUser1().getId()); 13. preparedStatement.setInt(2, games.get(i).getUser2().getId()); 15. preparedStatement.executeUpdate(); 16. } catch (SQLException e) { 17. e.printStackTrace(); 18. } 19. } 20. } catch (SQLException e1) { 21. e1.printStackTrace(); 22. } 24. } 25. } |

Када сервер прими ову поруку, он ће у базу убацити кориснике које тренутно играју игру за пријатеље.

CloseThreadAndSocket

|  |
| --- |
| **Изворни код обраде поруке: CloseThreadAndSocket** |
| 1. CloseThreadAndSocket clo = (CloseThreadAndSocket) object; 2. runAThread = false; 3. String userName1 = null; 4. int userForUpdate = 0; 5. String userName2 = null; 6. for (int i = 0; i < games.size(); i++) { 7. if (clo.getClientID() == games.get(i).getUser1().getId() || 8. clo.getClientID() == games.get(i).getUser2().getId()) { 9. if (games.get(i).getUser1().getId() == clo.getClientID()) { 10. userName1 = games.get(i).getUser1().getUsername(); 11. userName2 = games.get(i).getUser2().getUsername(); 12. userForUpdate = games.get(i).getUser2().getId(); 13. games.get(i).getUser2().getToClient().writeObject(new GameSessionEnded(clo.getClientID())); 14. } 15. if (games.get(i).getUser2().getId() == clo.getClientID()) { 16. userName1 = games.get(i).getUser2().getUsername(); 17. userForUpdate = games.get(i).getUser1().getId(); 18. userName2 = games.get(i).getUser1().getUsername(); 19. games.get(i).getUser1().getToClient().writeObject(new GameSessionEnded(clo.getClientID())); 20. }  23. try { 24. preparedStatement = connection.prepareStatement(updateUserRating); 25. preparedStatement.setInt(1, -25); 26. preparedStatement.setString(2, userName1); 27. preparedStatement.executeUpdate(); 29. preparedStatement = connection.prepareStatement(updateUserRating); 30. preparedStatement.setInt(1, 25); 31. preparedStatement.setString(2, userName2); 32. preparedStatement.executeUpdate(); 33. } catch (SQLException e) { 34. e.printStackTrace(); 35. } 37. try { 38. preparedStatement = connection.prepareStatement(getPlayerRatingQuery); 39. preparedStatement.setString(1, userName2);  42. ResultSet resSet = preparedStatement.executeQuery(); 43. if (resSet.next()) { 44. int rating = resSet.getInt(1); 45. for (int g = 0; g < connections.size(); g++) 46. if (connections.get(g).getId() == userForUpdate) { 47. connections.get(g).getToClient().writeObject(new PlayerRatingUpdate(userForUpdate, rating)); 48. } 49. } 50. } catch (SQLException e) { 51. e.printStackTrace(); 52. }   56. games.remove(i); 57. } 58. } 59. for (int i = 0; i < connections.size(); i++) { 60. if (clo.getClientID() == connections.get(i).getId()) { 61. int broj = i; 62. Platform.runLater(() -> { 63. taLog.appendText(connections.get(broj).getUsername() + " with id " + connections.get(broj).getId() + " log out\n"); 64. connections.remove(broj); 65. }); 67. } 68. } 69. for (int i = 0; i < gameSearch.size(); i++) { 70. if (clo.getClientID() == gameSearch.get(i).getPlayerId()) { 71. gameSearch.remove(i); 72. } 73. } 74. guestConnection.getSocket().close(); |

Онај играч који пошаље ову поруку серверу, сервер ће угасити његов тред и сокет. А ако неки корисник игра са њим, њему ће послати поруку GameSessionEnded и ажурирати базу новим рејтинзима. Другом кориснику ће послати рејтинг. Такође ће корисника који послао поруку избацити из листе пријављених корисника.

Постоје и друге поруке које корисници могу слати серверу и којим сервер одрађује неке мање важније радње.