III. Beadandó

Habzda Fruzsina (XNUHTE)

Tartalomjegyzék

Feladat

4. feladat: Tron

Készítsünk programot, amellyel a Tronból ismert fény-motor párbajt játszhatjuk felülnézetből. Két játékos játszik egymás ellen egy-egy olyan motorral, amely fénycsíkot húz maga mögött a képernyőn. A motor minden másodpercben a legutoljára beállított irányba halad feltéve, hogy a játékos nem változtatja meg azt egy megfelelő billentyű lenyomásával. (WASD az első játékos, nyilak a második játékos.) Az a játékos veszít, aki előbb neki ütközik a másik játékos fénycsíkjának vagy a képernyő szélének. A játék elején kérjük el a játékosok nevét és engedjük meg, hogy maguk válasszák ki a fényük színét. A játék végekor a győztes játékos eredményét növeljük meg az adatbázisban. Ha a játékos még nem található meg az adatbázisban, úgy szúrjunk be egy új sort. Egy menüpontban legyen lehetőségünk a 10 legjobb eredménnyel rendelkező játékost megtekinteni, az elért pontszámukkal, továbbá lehessen bármikor új játékot indítani egy másik menüből.

Tervezés

Specifikáció

A program alapja egy frame (*GameFrame*), amihez hozzá van adva a játéktábla (*GameBoard*), egy információs felület (*GameInfoPanel*), és egy menüsor (*GameMenuBar*). A programoból a fő ablak bezárásával lehet kilépni. A menüsor két menüt tartalmaz. Az egyik új játékkezdésre szolgál (megnyit egy *NewGameDialog*-ot), míg a másikkal a felhasználó megtekintheti a toplistát (*Top10Frame*, lekéri az adatbázistól a legjobb 10-et). Az információs felület a játékosok neveit, és a játék kezdete óta eltelt időt jeleníti meg. A játéktábla sprite-okból áll, melyeket folyamatosan 1 másodpercenként frissítünk, eközben minden játékos csak egy egységnyit halad a felhasználók által megadott irányba (WASD, nyilak segítségével fordítható el a játékos). Ha az egyik játékos veszít, akkor feljön egy kis gratuláló ablak a nyertes nevével, aminek leOK-zása után felugrik az új játék ablak, és ezzel új játékot lehet kezdeni. A játéktábla vezérlését egy külön osztály (*GameLogic*) végzi, ez vizsgálja, hogy vége van-e a játéknak, tárolja a tábla adatait, lépteti a játékost.

Az adatbázis hozzáférést a *DBConnection* és *DBConnectionSource* biztosítja. Az adatbázis feldolgozásához/lekérdezéséhez, változtatásához még három további osztályt hoztunk létre: *HighScoreEntity*-t (egy sort reprezentál az adatbázisból), *HighScoreDao*-t (ezzel változtatható, kérhető le az adatbázis) és *DataSource*-ot.

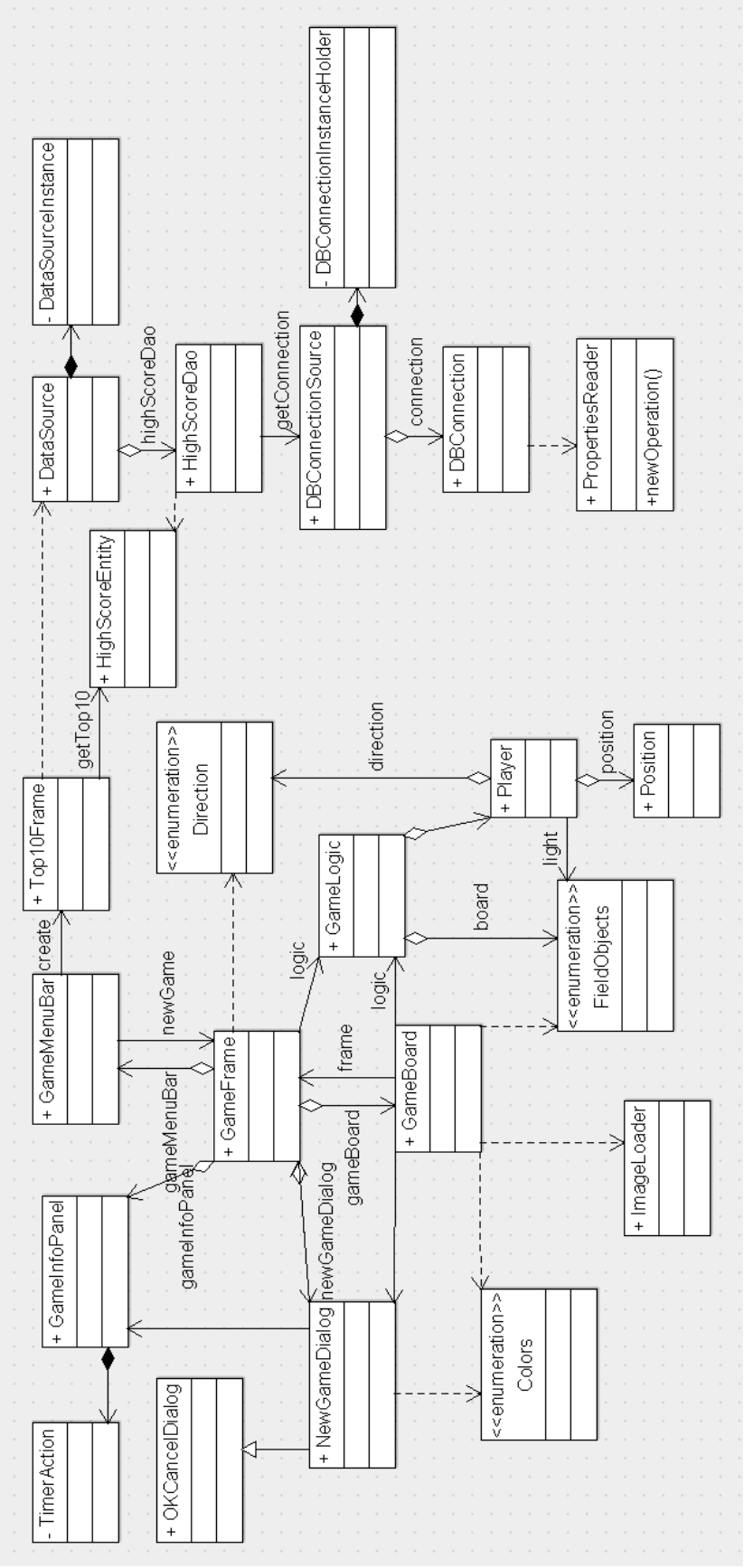
További kisebb osztályokra, enumerátorokra is szükség van a játék működéséhez, pl.: *Colors* (mely a választható színeket tárolja), *Direction*, *Position*, *Player* (egy játékos pozícióját, irányát tárolja), *FiledObject* (ezek az objektumok szerepelhetnek a logika tábla reprezentációjában).

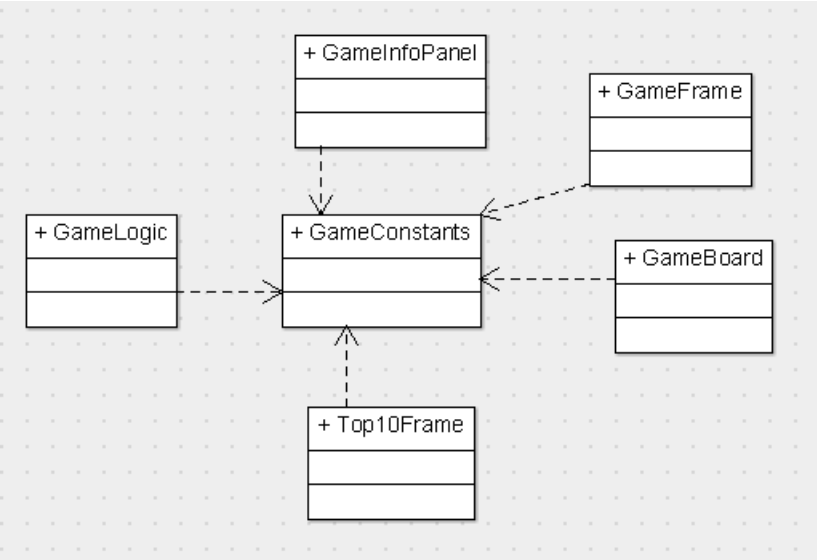
Beolvasásra két osztályt használunk: *ImageLoader*-t (képekhez), és *PropertiesReader*-t (property-khez).

A játék – legfőképp a gui – konstansait a *GameConstants* osztály tárolja.

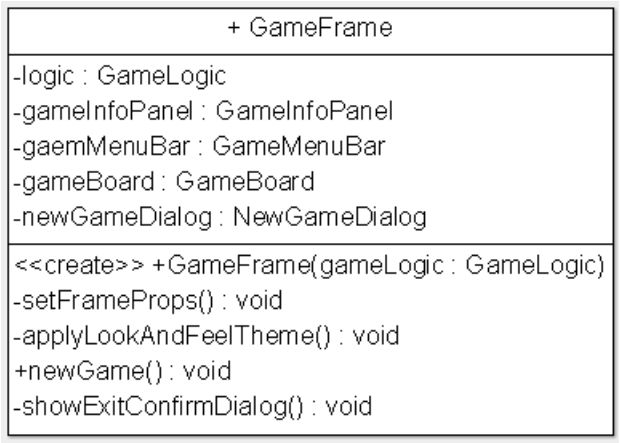
Osztály diagram

Kapcsolatok

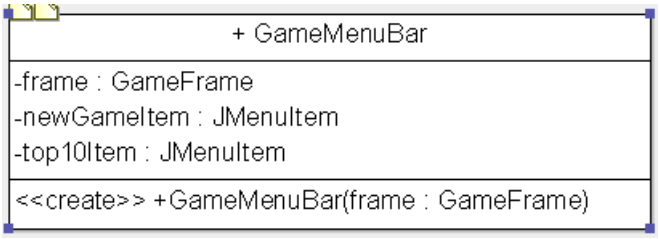


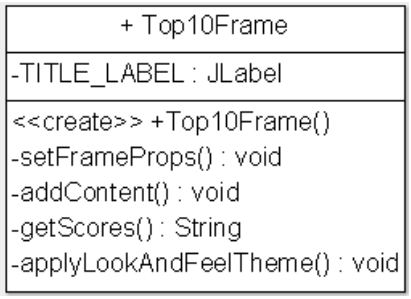


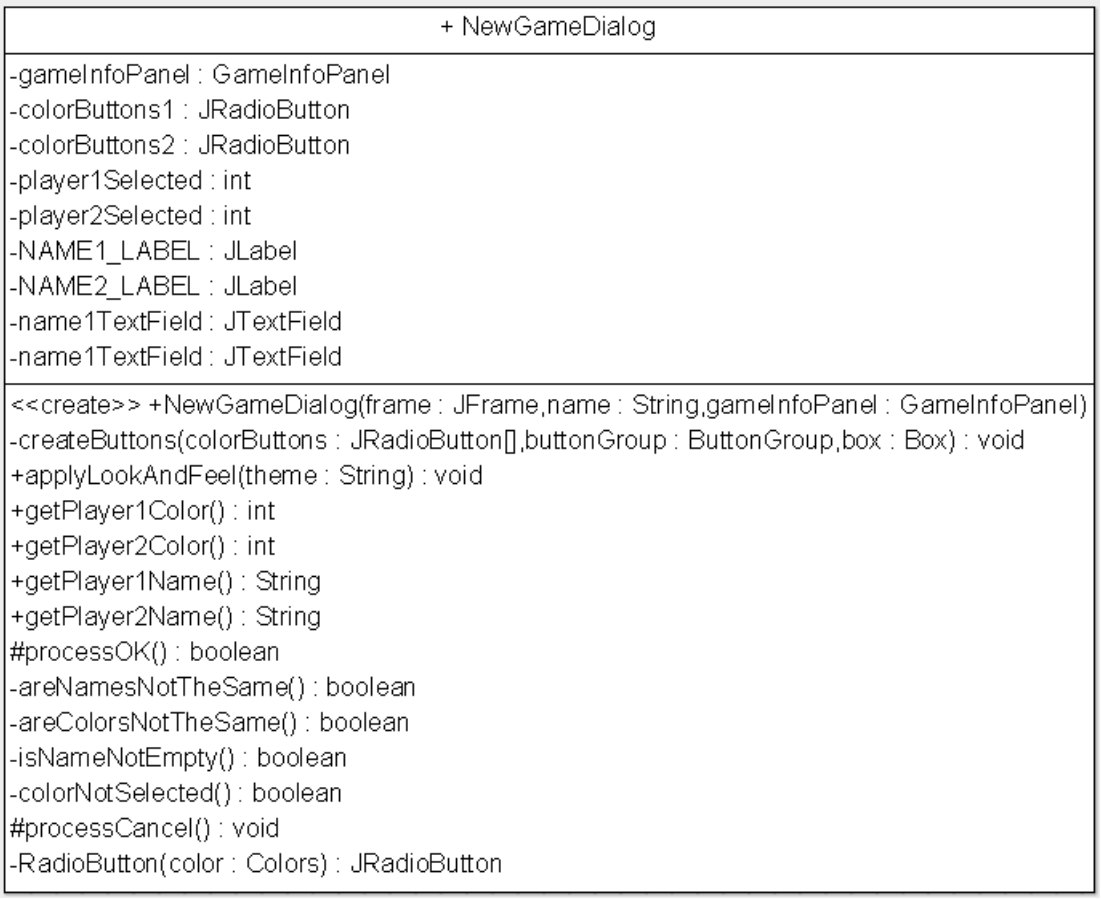
Részletezett UML diagrammok

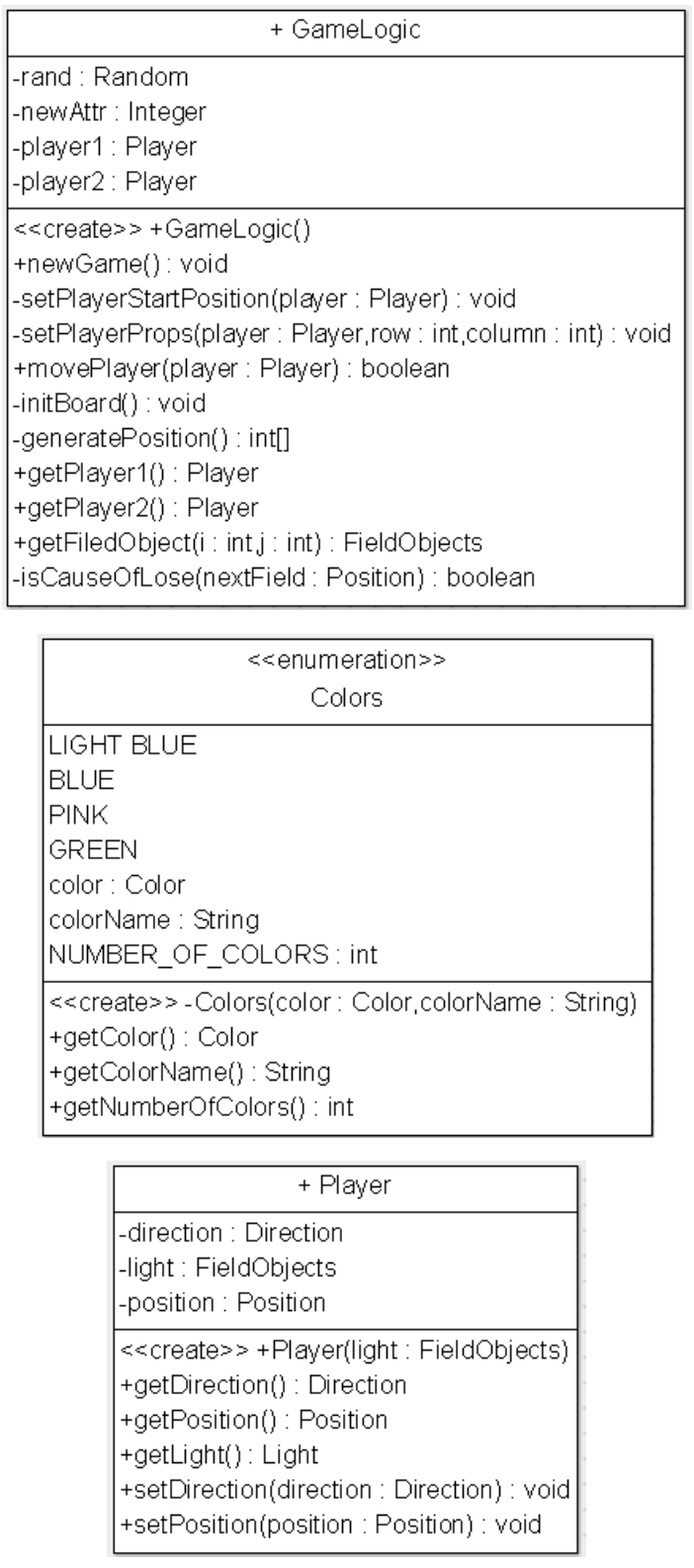


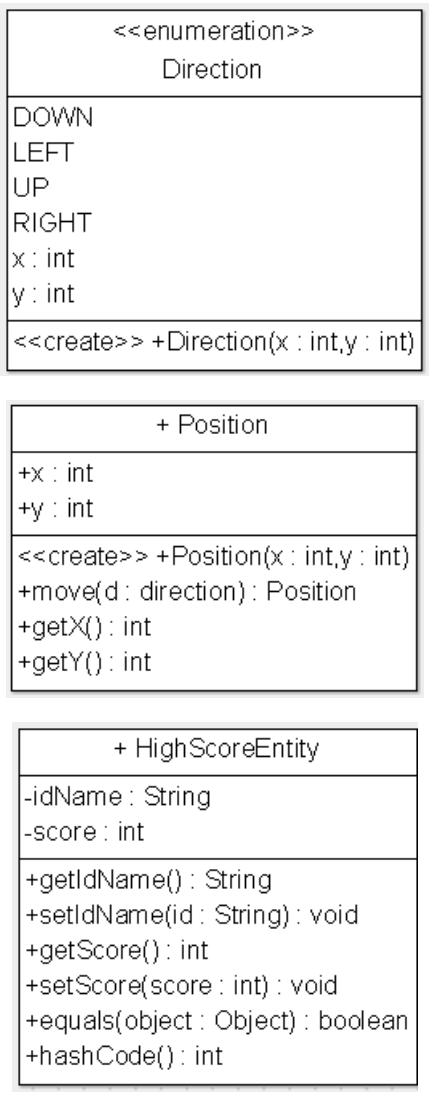

+ GameBoard
-frame : GameFrame
-newGameDialog : NewGameDialog
-logic : GameLogic
-floor : Image
-wall : Image
-player : Image
-coverImage : Image
-boardUnitInPixels : int
-light1 : Colors
-light2 : Colors
-timer : Timer
-gameHasStarted : boolean
-gameBoardReady : boolean
-light1Pos : ArrayList
-light2Pos : ArrayList
-timerAction : Action
<<create>> +GameBoard(frame : GameFrame,newGameDialog : NewGameDialog,logic : GameLogic)
+getGameHasStarted() : boolean
+loadImages() : void
+newGame() : void
-initLights() : void
-setupBoard(g : Graphics2D) : void
-updateLightsArray() : void
-paintLights(g : Graphics2D,light : Colors) : void
-paintPlayers(g : Graphics2D) : void
-updateBoard(g : Graphics2D) : void
-setupCover(g : Graphics2D) : void
#paintComponent(g : Graphics) : void
-showGameOverMessage(winner : String) : void
-endGame(winner : String) : void
+ GameInfoPanel
-timer : Timer
-newAttr : Integer
-timeLabel : JLabel
-player1NameLabel : JLabel
-player2NameLabel : JLabel
-VS_LABEL : JLabel
<<create>> +GameInfoPanel(logic : GameLogic)
+newGame(name1 : String,name2 : String) : void
-addLabels() : void
+stopTimer() : void
+continueTimer() : void
- TimerAction
-timerLabel : JLabel
-startTime : Instant
+actionPerformed() : void
-formatDuration(duration : Duration) : String


+ GameMenuBar
-frame : GameFrame
-newGameItem : JMenuItem
-top10Item : JMenuItem
<<create>> +GameMenuBar(frame : GameFrame)


+ Top10Frame
-TITLE_LABEL : JLabel
<<create>> +Top10Frame()
-setFrameProps() : void
-addContent() : void
-getScores() : String
-applyLookAndFeelTheme() : void


+ NewGameDialog
-gameInfoPanel : GameInfoPanel
-colorButtons1 : JRadioButton
-colorButtons2 : JRadioButton
-player1Selected : int
-player2Selected : int
-NAME1_LABEL : JLabel
-NAME2_LABEL : JLabel
-name1TextField : JTextField
-name1TextField : JTextField
<<create>> +NewGameDialog(frame : JFrame,name : String,gameInfoPanel : GameInfoPanel)
-createButtons(colorButtons : JRadioButton[],buttonGroup : ButtonGroup,box : Box) : void
+applyLookAndFeel(theme : String) : void
+getPlayer1Color() : int
+getPlayer2Color() : int
+getPlayer1Name() : String
+getPlayer2Name() : String
#processOK() : boolean
-areNamesNotTheSame() : boolean
-areColorsNotTheSame() : boolean
-isNameNotEmpty() : boolean
-colorNotSelected() : boolean
#processCancel() : void
-RadioButton(color : Colors) : JRadioButton


+ GameLogic
-rand : Random
-newAttr : Integer
-player1 : Player
-player2 : Player
<<create>> +GameLogic()
+newGame() : void
-setPlayerStartPosition(player : Player) : void
-setPlayerProps(player : Player,row : int,column : int) : void
+movePlayer(player : Player) : boolean
-initBoard() : void
-generatePosition() : int[]
+getPlayer1() : Player
+getPlayer2() : Player
+getFiledObject(i : int,j : int) : FieldObjects
-isCauseOfLose(nextField : Position) : boolean
<<enumeration>>
Colors
LIGHT BLUE
BLUE
PINK
GREEN
color : Color
colorName : String
NUMBER_OF_COLORS : int
<<create>> -Colors(color : Color,colorName : String)
+getColor() : Color
+getColorName() : String
+getNumberOfColors() : int
+ Player
-direction : Direction
-light : FieldObjects
-position : Position
<<create>> +Player(light : FieldObjects)
+getDirection() : Direction
+getPosition() : Position
+getLight() : Light
+setDirection(direction : Direction) : void
+setPosition(position : Position) : void


<<enumeration>>
Direction
DOWN
LEFT
UP
RIGHT
x : int
y : int
<<create>> +Direction(x : int,y : int)
+ Position
+x : int
+y : int
<<create>> +Position(x : int,y : int)
+move(d : direction) : Position
+getX() : int
+getY() : int
+ HighScoreEntity
-idName : String
-score : int
+getIdName() : String
+setIdName(id : String) : void
+getScore() : int
+setScore(score : int) : void
+equals(object : Object) : boolean
+hashCode() : int
+ HighScoreDao
-tableName : String
-ID_ATTR_NAME : String
-SCORE_ATTR_NAME : String
-selectWholeSqlString : String
-selectTop10String : String
-updateScoreString1 : String
-updateScoreString2 : String
+getTop10() : ArrayList
-getScore(winner : String) : int
-updateScore(entity : HighScoreEntity) : void
+addEntity()
-newEntity()
+newOperation()
-getEntityAttributes(resultSet : ResultSet,entity : HighScoreEntity) : void
-setEntityAttributes(entity : HighScoreEntity,resultSet : ResultSet) : void

Megvalósítás

Program felépítés

- game
  - gui
    - dialog
      - newgame
        - NewGameDialog.java
        - OKCancelDialog.java
      - top10
        - Top10Frame.java
    - GameBoard.java
    - GameFrame.java
    - GameInfoPanel.java
    - GameMenuBar.java
  - io
    - ImageLoader.java
    - PropertiesReader.java
  - logic
    - player
      - Colors.java
      - Direction.java
      - Player.java
      - Position.java
    - FieldObjects.java
    - GameLogic.java
  - persistence
    - DataSource.java
    - DBConnection.java
    - DBConnectoinSource.java
    - HighScoreDao.java
    - HighScoreEntity.java
  - resources
    - floor.jpg
    - player.png
    - wall.jpg
    - icon.png
    - cover1.jpg
  - Boot.java
  - GameConstants
- gamedb

Főprogram

A főprogram (Boot) létrehoz egy új GameFrame-et, új GameLogic-kal, és láthatóvá teszi.

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

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                new GameFrame(new GameLogic()).setVisible(true);
            }catch(IOException e){
                System.err.println("Képbetöltési hiba");
            }
        }
    });
}
```

GameLogic

A játék belső számításait végző osztály a GameLogic. Ebből kiemelnék pár érdekesebb részletet:

newGame:

A tábla inicializálását végzi. A szélekre „falat” tesz, míg középre „üreset”.

```
private void initBoard(){
    board = new
FieldObjects[GameConstants.BOARD_SIZE][GameConstants.BOARD_SIZE];
    for (int row = 0; row < GameConstants.BOARD_SIZE; ++row) {
        for (int column = 0; column < GameConstants.BOARD_SIZE; ++column) {
            if (row==0 || column==0 || row==GameConstants.BOARD_SIZE-1 ||
column==GameConstants.BOARD_SIZE-1)
            {
                board[row][column] = FieldObjects.WALL;
            } else {
                board[row][column] = FieldObjects.EMPTY;
            }
        }
    }
}
```

setPlayerPosition:

Beállítja a paraméterül megkapott játékos kezdőpozícióját (játékost csak a pálya szélére lehet rakni). A randomRow segítségével eldöntjük, hogy a legenerált random számok közül melyik lesz a Position x, és melyik az y változója. Ellenőrzi, hogy azon a pozíción van-e már játékos, ha van, újrageneráltatja a pozíciót.

```
private void setPlayerStartPosition(Player player){
    int randomRow = rand.nextInt(2);
    int[] randomNums = generatePosition();

    if (randomRow == 1){
        while ((player1.getPosition().getX()==randomNums[0] &&
player1.getPosition().getY()==randomNums[1]) ||
                (player2.getPosition().getX()==randomNums[0] &&
player2.getPosition().getY()==randomNums[1]) ){
            randomNums = generatePosition();
        }
        setPlayerProps(player, randomNums[0], randomNums[1]);
```

```
    } else {
        while ((player1.getPosition().getX()==randomNums[1] &&
player1.getPosition().getY()==randomNums[0]) ||
                (player2.getPosition().getX()==randomNums[1] &&
player2.getPosition().getY()==randomNums[0]) ){
            randomNums = generatePosition();
        }
        setPlayerProps(player, randomNums[1], randomNums[0]);
    }
}
```

generatePosition:

Egy tömböt ad vissza két számmal, ami egy pozíciót reprezentál. Az első szám 1-től táblaméret mínusz 2-ig bármi lehet (mert 0-tól indexelünk), a második pedig 1 vagy táblaméret mínusz 2.

```
private int[] generatePosition(){
    int[] nums = new int[2];
    nums[0] = rand.nextInt(GameConstants.BOARD_SIZE-3)+1;
    nums[1] = rand.nextInt(2);
    if (nums[1] == 0)  nums[1] = (GameConstants.BOARD_SIZE-2);
    return nums;
}
```

setPlayerProps:

Beállítja a játékos kezdő pozícióját, és ez alapján az irányát, mindig a pálya közepe felé.

```
public void setPlayerProps(Player player, int row, int column){
    board[row][column] = player.getLight();
    player.setPosition(new Position(row, column));

    if (column == 1){player.setDirection(Direction.DOWN);}
    else if (column == GameConstants.BOARD_SIZE-2)
{player.setDirection(Direction.UP);}
    else if (row == 1) {player.setDirection(Direction.RIGHT);}
    else if (row == GameConstants.BOARD_SIZE-2)
{player.setDirection(Direction.LEFT);}
}
```

movePlayer:

Abban az esetben, ha lehetséges, mozgatja a játékost a táblán. Ha nem lehetséges false értéket ad vissza, ezzel jelezve a játék végét.

```
public boolean movePlayer(Player player){
    Position previous = player.getPosition();
    Position next = previous.move(player.getDirection());
    if (isCauseOfLose(next)) {
        return false;
    } else {
        player.setPosition(next);
        board[player.getPosition().getX()][player.getPosition().getY()] =
player.getLight();
        return true;
    }
}
```

isCauseOfLose:

Ha a következő pozíción fény, vagy fal áll, true-t ad vissza, ezzel jelezve a játék végét.

```
private boolean isCauseOfLose(Position nextField){
    if (board[nextField.getX()][nextField.getY()] == FieldObjects.WALL ||
            board[nextField.getX()][nextField.getY()] ==
FieldObjects.LIGHT1 ||
            board[nextField.getX()][nextField.getY()] ==
FieldObjects.LIGHT2 ) {
        return true;
    } else return false;
}
```

GameBoard

A GameBoard végzi a játéktábla megjelenítését GridLayout és sprite-ok segítségével. JPanel-ből lett származtatva.

Eltárolja a játékosok pozícióit két tömbben, és ennek segítségével rajzolja fel a fény csíkokat.

paintCompontent:

A tábla felrajzolását végzi. Ha még nem kezdődött el a játék, borítót rajzol, ha már folyik a játék, akkor táblát ezen kívül mozgatja a játékosokat a logic.movePlayer()-rel, és vizsgálja annak visszatérési értékét, tehát hogy véget ért-e a játék.

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

    if(gameHasStarted) {
        if (gameBoardReady==false){
            updateBoard(g2);
        } else {
            if (logic.movePlayer(logic.getPlayer1())==false){
                endGame(newGameDialog.getPlayer2Name());
            }
            if (logic.movePlayer(logic.getPlayer2())==false){
                endGame(newGameDialog.getPlayer1Name());
            }
            updateBoard(g2);
        }
    }else{
        setupCover(g2);
    }
}
```

paintCompontent:

Megjeleníti az egész táblát, játékosokkal, fényekkel.

```
private void updateBoard(Graphics2D g){
    setupBoard(g);

    updateLightsArray();
    paintLights(g, light1);
```

```
    paintLights(g, light2);

    paintPlayers(g);

    g.dispose();
}
```

paintPlayers:

Felrajzolja a játékosokat a helyükre.

```
private void paintPlayers(Graphics2D g){

g.drawImage(player,logic.getPlayer1().getPosition().getY()*boardUnitInPixel
s,
            logic.getPlayer1().getPosition().getX()*boardUnitInPixels,
            boardUnitInPixels, boardUnitInPixels, null);


g.drawImage(player,logic.getPlayer2().getPosition().getY()*boardUnitInPixel
s,
            logic.getPlayer2().getPosition().getX()*boardUnitInPixels,
            boardUnitInPixels, boardUnitInPixels, null);
}
```

setupBoard:

Felrajzolja a tábla alapját.

```
private void setupBoard(Graphics2D g){
    Image image = null;
    for (int row = 0; row < GameConstants.BOARD_SIZE; ++row) {
        for (int column = 0; column < GameConstants.BOARD_SIZE; ++column) {
            FieldObjects fieldObjects = logic.getFiledObject(column, row);
            switch (fieldObjects){
                case WALL: image = wall; break;
                case LIGHT1: image = floor; break;
                case LIGHT2: image = floor; break;
                case EMPTY: image = floor; break;
            }
            g.drawImage(image, column * boardUnitInPixels, row *
boardUnitInPixels, boardUnitInPixels, boardUnitInPixels, this);
        }
    }
    gameBoardReady = true;
}
```

updateLightsArray:

Hozzáfűzi a játékosok újabb pozícióját a szín listákhoz.

```
private void updateLightsArray(){
    Position player1Position = new
Position(logic.getPlayer1().getPosition().getY()*boardUnitInPixels +
boardUnitInPixels/2,
            logic.getPlayer1().getPosition().getX()*boardUnitInPixels +
boardUnitInPixels/2);
    light1Pos.add(player1Position);

    Position player2Position = new
Position(logic.getPlayer2().getPosition().getY()*boardUnitInPixels +
```

```
boardUnitInPixels/2,
            logic.getPlayer2().getPosition().getX()*boardUnitInPixels +
boardUnitInPixels/2);
    light2Pos.add(player2Position);
}
```

paintLights:

Attól függően, hogy milyen színt kapott paraméterül, felrajzolja a hozzá kapcsolódó tömb alapján a vonalakat.

```
private void paintLights(Graphics2D g, Colors light){
    BasicStroke lightStroke = new BasicStroke(8,
            BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
    g.setStroke(lightStroke);
    g.setColor(light.getColor());

    ArrayList<Position> lightPos;
    if (light.equals(light1)){
        lightPos = light1Pos;
    } else {
        lightPos = light2Pos;
    }

    for(int i=0; i<lightPos.size()-1; i++){
        g.drawLine(lightPos.get(i).getX(), lightPos.get(i).getY(),
lightPos.get(i+1).getX(), lightPos.get(i+1).getY());
    }
}
```

endGame:

A játék befejezéséért felelős. Elmenti a győztest az adatbázisba, és felkínálja az új játék lehetőségét.

```
private void endGame(String winner){
    timer.stop();
    gameHasStarted = false;
    gameBoardReady = false;
    DataSource.getInstance().getHighScoreDao().addEntity(winner);
    showGameOverMessage(winner);
    frame.newGame();
}
```

HighScoreDao

Ez az osztály tudja lekérdezni, megváltoztatni az adatbázist.

getTop10:

Lekéri az adatbázisból a legjobb tíz játékost (akinek legtöbb pontszáma van), és entitások formájában adja vissza. Ha hiba lép fel, null-t ad értékül.

```
public ArrayList<HighScoreEntity> getTop10() {
    try(Connection connection =
DBConnectionSource.getInstance().getConnection();
        Statement statement =
connection.createStatement(ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_READ_ONLY);
        ResultSet rs = statement.executeQuery(selectTop10String)){
```

```
            final ArrayList<HighScoreEntity> entities = new ArrayList<>();
            while (rs.next()) {
                final HighScoreEntity entity = new HighScoreEntity();
                setEntityAttributes(entity, rs);
                entities.add(entity);
            }
        return entities;
    } catch(SQLException e){
        System.err.println(e.getMessage());
        return null;
    }
}
```

getScore:

A paraméterül kapott név alapján lekéri az adatbázisból a pontot. Abban az esetben, ha hiba lép fel (például nincs benne az adott név az adatbázisban), 0-t ad vissza.

```
private int getScore(String winner){
    String statementString = "SELECT * FROM " + tableName + " WHERE "+
ID_ATTR_NAME + " = '" + winner + "'";
    try(Connection connection =
DBConnectionSource.getInstance().getConnection();
        Statement statement =
connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = statement.executeQuery(statementString)){
        rs.next();
        final HighScoreEntity entity = new HighScoreEntity();
        setEntityAttributes(entity, rs);
        return entity.getScore();
    }catch(SQLException e){
        System.err.println(e.getMessage());
        return 0;
    }
}
```

updateScore:

A paraméterül kapott entitás alapján megváltoztatja az adatbázis egyik sorának a pontszámát.

```
private void updateScore(HighScoreEntity entity){
    try(Connection connection =
DBConnectionSource.getInstance().getConnection();
        Statement statement =
connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);) {
        statement.executeUpdate(updateScoreString1 + entity.getScore() +
updateScoreString2 + "'" + entity.getIdName() + "'");
        } catch (SQLException e){
        System.err.println(e.getMessage());
    }
}
```

addEntity:

Eldönti, hogy a paraméterül kapott játékos szerepel-e az adatbázisban, ha nem beleteszi getScore() (ami ezesetben nulla) + 1-es pontszámmal, ha igen, meghívja az updateScore()-t eggyel növelt getScore()-os entitással, ezáltal eggyel növekszik az adatbázisban a játékos pontszáma).

```
public void addEntity(String winner){
    try(Connection connection =
DBConnectionSource.getInstance().getConnection();
        Statement statement =
connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = statement.executeQuery(selectWholeSqlString)){
        HighScoreEntity entity = newEntity(winner,(getScore(winner)+1));
        if (entity.getScore()==1){
            rs.moveToInsertRow();
            getEntityAttributes(rs, entity);
            rs.insertRow();
        } else {
            updateScore(entity);
        }
    } catch(SQLException e){
        System.err.println(e.getMessage());
    }
}
```

Tesztelés

Elvégzendő tesztesetek

- Menügombok működése
  - új játék ablak megnyílása
  - toplista megnyitásánál megnyílik az új ablak
- Bezárás
  - kérdező ablak megnyílása, igenre program bezárása
- Új játék ablak
  - ellenőrzi az inputot
    - a név mező kitöltését
    - a megegyező neveket
    - a szín kiválasztását
    - hogy ugyanolyan szín van e kiválasztva
  - OK lenyomásra új játékot indít
    - inicializálja a játéktáblát
      - felrajzolja a táblát
    - inicializálja az infópanelt
      - beállítódnak/megváltoznak a nevek
      - lenullázódik az idő, majd elkezdődik
    - inicializálja a logikát
- Toplista
  - a 10 legnagyobb pontszámú játékost írja ki, sorrendben
- Játékosok irányításának helyes működése
- Játéktábla helyes felrajzolása
  - helyükön vannak a játékosok
  - a maguk után húzott fénycsík helyesen, jó irányba rajzolódik ki
- Ha az egyik játékos falnak megy, vagy fénycsíkba megy, veszít, játék vége
- Játék vége
  - felugrik a nyertest kiíró ablak
  - újul a pontokat tartalmazó adatbázis
    - ha nem létezett a nyertes játékos, létrehozza
    - ha létezett a nyertes játékos, megnöveli a pontjait 1-el
  - felhozza az új játék ablakot

Továbbfejlesztési ötletek

- Súgó beépítése, a játék használatának leírásával
- Grafikai elemek fejlesztése
- Infó panel átrendezése
- Akadályok beiktatása a játékba
- A felhasználó lekérdezhesse a saját pontjainak mennyiségét