Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Программирование

Отчет по курсовой работе
Игра "Мельница"

Работу выполнила:
Власова А.В.
Группа: 23501/4
Преподаватель:
Вылегжанина К.Д.

Санкт-Петербург
2017

Содержание

1 Проектирование приложения для игры в Мельницу

Мельница - логическая настольная игра для двух игроков, сохраняющая свою популярность уже более 3000 лет. Считается, что именно Мельница стала прародителем знаменитой игры "Крестики-нолики"из-за некоторого сходства в правилах. В наши дни Мельница остается известной среди любителей настольных игр. Традиционно для игры в Мельницу используют деревянную доску, однако в настоящее время для этой цели широко используются специальные приложения для компьютеров, смартфонов и других устройств.

1.1 Задание

Разработать Android-приложение, позволяющее двум игрокам играть в Мельницу.

1.2 Концепция

Готовое приложение дает возможность двум пользователям играть в Мельницу на Android-устройствах.

1.3 Минимально работоспособный продукт

Android-приложение, предназначенное для игры двух соперников в Мельницу.

1.4 Правила

Игра начинается с того, что оба игрока по очереди размещают по одной своей фишке на свободных кружках игрового поля. Каждый игрок имеет по 9 фишек. Если игроку удалось выставить три фишки в ряд на одной линии, он забирает любую фишку противника. Ряд из трех фишек называется мельницей.

После того, как все фишки будут размещены на поле, игроки начинают по очереди передвигать свои фишки на соседние кружки с целью построить мельницу. Перемещать фишки можно только по линиям, расположенным на поле. Когда у игрока остается только три фишки, он может перемещать их на любые свободные кружки не зависимо от линий. В это время его соперник продолжает ходить по старым правилам, пока у него тоже не останется три фишки.

Игра считается выигранной, когда соперник не может построить мельницу. Это может произойти, когда у противника осталось всего две фишки, или у него нет возможности сделать ход.

1.5 Выделенные подпроекты

В процессе проектирования приложения было выделено два подпроекта.

- **Core**

  Библиотека, представляющая бизнес-логику приложения.

- **app**

  Android-приложение, обеспечивающее взаимодействие пользователей с ядром.

1.6 Описание интерфейса библиотеки

Интерфейс библиотеки содержит в себе следующие методы:

- Метод, позволяющий установить фишку на поле

  boolean makeMove(int x, int y, int z);

- Метод, позволяющий передвинуть фишку в доступную ячейку

  boolean makeMove(int fromX, int fromY, int fromZ, int toX, int toY, int toZ);

- Метод, позволяющий удалить фишку противника

  void removePiece(int x, int y, int z);

- Метод, возвращающий игровое поле

  Cell[][][] getField();

- Метод, позволяющий узнать, есть ли на поле новая мельница

  boolean isMill();

- Метод, позволяющий узнать, все ли фишки расставлены

  boolean isAllPiecesSet();

- Метод, возвращающий активного игрока

  Player getActivePlayer();

1.7 Выводы

В данном разделе рассмотрен процесс проектирования приложения для игры в Мельницу. Описаны концепция приложения и минимально работоспособный продукт, приведены правила игры, перечислены выделенные подпроекты, а также описаны методы, входящие в интерфейс библиотеки.

2 Реализация приложения для игры в Мельницу

2.1 Среда разработки

Операционная система: Windows 7
Среда разработки: Android Studio 2.3.1
Компилятор: javac, `JDK 1.8.0_121`
Система автоматической сборки: Gradle 3.3

2.2 Выделенные классы

В библиотеке были выделены следующие классы:

- **Cell** - содержит информацию о клетке игрового поля. Позволяет получить координаты клетки и ее статус, добавить или удалить фишку.
- **CellStatus** - перечисление статусов клетки.
- **Piece** - содержит информацию о фишке. Позволяет получить текущее расположение фишки на игровом поле, ее статус и цвет.
- **PieceStatus** - перечисление статусов фишки.
- **Board** - содержит информацию об игровом поле. Позволяет установить фишку на игровое поле, передвинуть ее на доступную клетку и удалить фишку. Содержит метод, позволяющий узнать, есть ли на поле неиспользованная мельница.
- **Player** - содержит информацию об игроке, его фишках и мельницах. Позволяет узнать, есть ли у игрока неиспользованная мельница.
- **PlayerStatus** - перечисление статусов игрока.
- **MillsAPI** - интерфейс библиотеки. Описан в предыдущем разделе.
- **Game** - класс, реализующий интерфейс библиотеки.

Проект **app** содержит в себе четыре класса:

- **MainActivity** - главная активность, содержащая меню для переключения между остальными активностями.
- **GameActivity** - активность, в которой происходит игровой процесс.
- **RegulationsActivity** - активность, содержащая правила игры.
- **EndGameActivity** - активность, в которой отображается победитель.

2.3 Примеры работы Android-приложения

Ниже приведены снимки экрана для демонстрации работы Android-приложения.

Рис. 1: Игровое меню

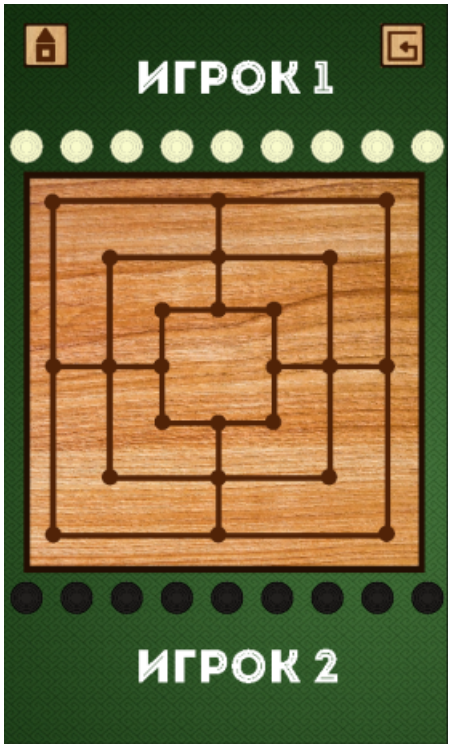

Рис. 2: Начало игры

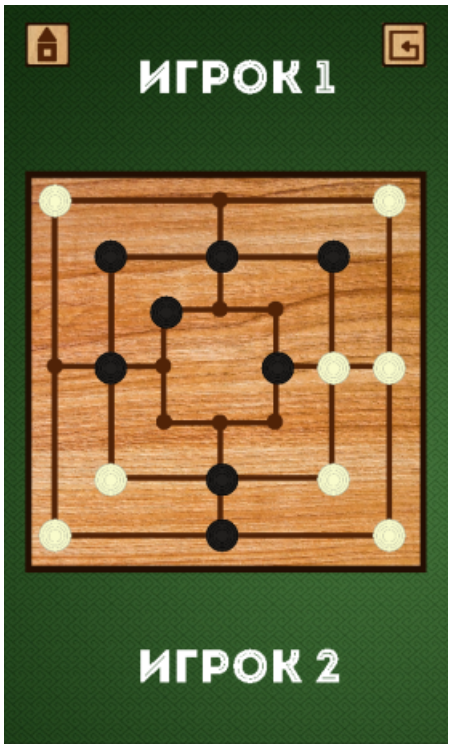

Рис. 3: Игра в процессе

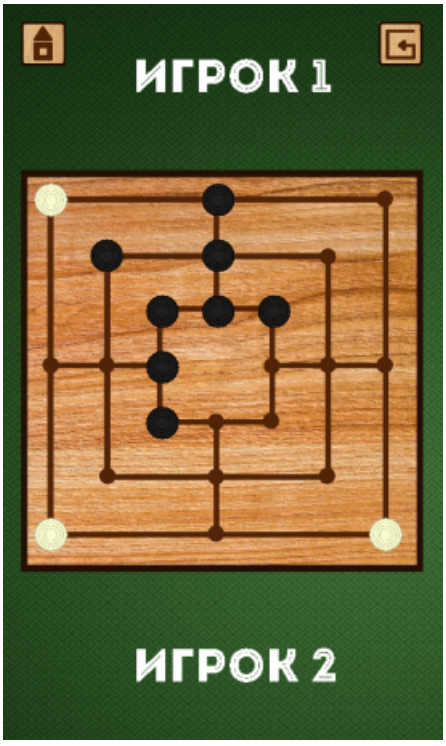

Рис. 4: Игра в процессе

Рис. 5: Конец игры

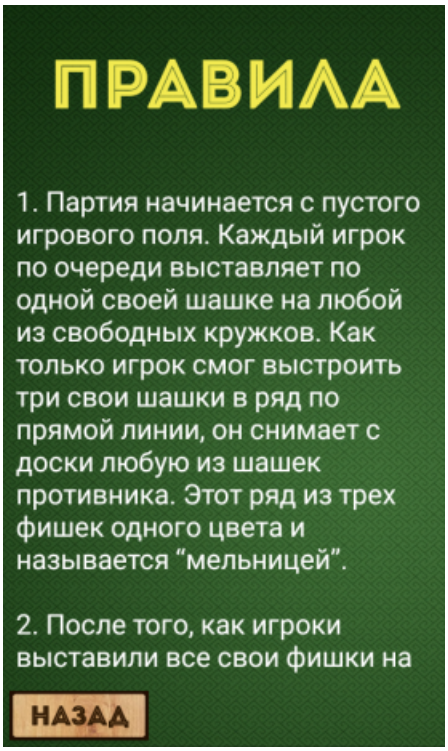

Рис. 6: Правила

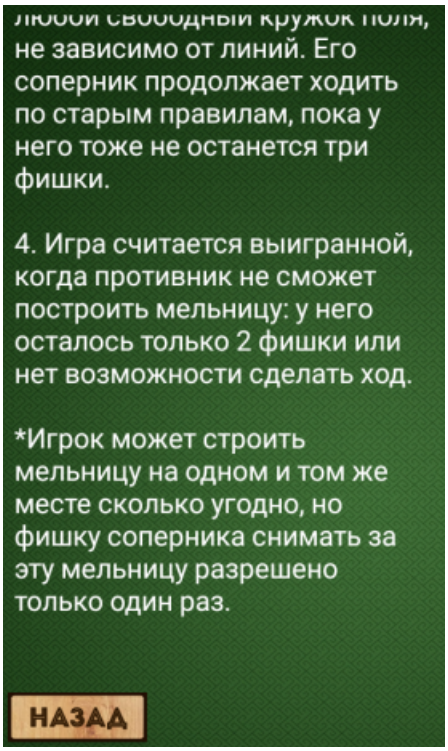

Рис. 7: Правила

2.4 Выводы

В данном разделе были описаны все классы, выделенные в процессе работы над проектом. Также были сделаны снимки экрана, демонстрирующие работу Android-приложения.

3 Процесс обеспечения качества и тестирование

3.1 Тестирование

Для проверки работы библиотеки использовались автоматические тесты, покрывающие основную функциональность ядра. Также в процессе разработки приложения проводилось ручное тестирование программы.

3.2 Выводы

В данном разделе описан процесс тестирования программы.

4 Выводы

В результате работы над курсовым проектом было реализовано приложение, предназначенное для игры двух игроков в Мельницу. В процессе создания приложения был увеличен опыт написания программ на языке Java, а также получены навыки создания Android-приложения.

5 Приложение 1

Листинг 1: MillsAPI.java

```
1 package ru.vlasova.mills.core;
2
3 public interface MillsAPI {
4     boolean makeMove(int x, int y, int z);
5     boolean makeMove( int fromX, int fromY, int fromZ, int toX, int toY, int toZ);
```

```
6       void removePiece(int x, int y, int z);
7       Cell[][][] getField();
8       boolean isMill();
9       boolean isAllPiecesSet();
10      Player getActivePlayer();
11  }
```

Листинг 2: Game.java

```
1   package ru.vlasova.mills.core;
2
3   public class Game implements MillsAPI {
4       private Board board;
5       private Player whitePlayer;
6       private Player blackPlayer;
7       private Player activePlayer;
8
9       public Game() {
10          board = new Board();
11          whitePlayer = new Player(0);
12          blackPlayer = new Player(1);
13          activePlayer = whitePlayer;
14      }
15
16      @Override
17      public boolean makeMove(int x, int y, int z) {
18          try {
19              if (activePlayer.isNewMill()) {
20                  removePiece(x, y, z);
21                  return false;
22              } else if (!board.isMill(activePlayer)) {
23                  board.setPiece(activePlayer, x, y, z);
24                  if (!board.isMill(activePlayer))
25                      changeActivePlayer();
26              }
27              return true;
28          } catch (RuntimeException e) {
29              return true;
30          }
31      }
32
33      @Override
34      public boolean makeMove(int fromX, int fromY, int fromZ, int toX, int toY, int toZ) {
35          try {
36              if (activePlayer.getStatus().equals(PlayerStatus.BASIC))
37                  board.moveOneCoord(activePlayer.getColor(), fromX, fromY, fromZ, toX, toY, toZ
    ↪ );
38              if (activePlayer.getStatus().equals(PlayerStatus.FINAL))
39                  board.moveAnyCoord(activePlayer.getColor(), fromX, fromY, fromZ, toX, toY, toZ
    ↪ );
40              if (!board.isMill(activePlayer))
41                  changeActivePlayer();
42              return true;
43          } catch (RuntimeException e){
44              return false;
45          }
46      }
47
48      @Override
49      public void removePiece(int x, int y, int z) throws RuntimeException{
50          try {
51              Piece piece = board.removePiece(activePlayer.getColor(), x, y, z);
52              changeActivePlayer();
53              activePlayer.removePiece(piece);
54
55          } catch (RuntimeException e) {
56              activePlayer.returnMill();
57              throw e;
58          }
59      }
60
61      @Override
62      public Cell[][][] getField() {
63          return board.getCells();
64      }
65
```

```
66      @Override
67      public boolean isMill() {
68          if(board.isMill(activePlayer))
69              return true;
70          else
71              return false;
72      }
73
74      private void changeActivePlayer() {
75          if(activePlayer == whitePlayer)
76              activePlayer = blackPlayer;
77          else
78              activePlayer = whitePlayer;
79      }
80
81      @Override
82      public boolean isAllPiecesSet() {
83          if(!whitePlayer.getStatus().equals(PlayerStatus.INITIAL) && !(blackPlayer.getStatus().
    ↪ equals(PlayerStatus.INITIAL)) )
84              return true;
85          else
86              return false;
87      }
88
89      @Override
90      public Player getActivePlayer() {
91          return activePlayer;
92      }
93  }
```

Листинг 3: Board.java

```
1   package ru.vlasova.mills.core;
2
3
4   public class Board {
5       private Cell[][][] cells;
6
7       public Board() {
8           cells = new Cell[3][3][3];
9           for (int i=0; i<3; i++) {
10              for (int j = 0; j < 3; j++)
11                  for (int k = 0; k < 3; k++)
12                      cells[i][j][k] = new Cell(i, j, k);
13          }
14          setNotAvailableCells();
15      }
16
17      private void setNotAvailableCells() {
18          for (int i=0; i<3; i++)
19              cells[1][1][i].setStatus(CellStatus.NOTAVAILABLE);
20      }
21
22      public Cell[][][] getCells() {
23          return cells;
24      }
25
26      public void setPiece(Player player, int x, int y, int z) throws RuntimeException{
27          if (cells[x][y][z].getStatus().equals(CellStatus.OCCUPIED))
28              throw new RuntimeException("Cell is occupied");
29          if (cells[x][y][z].getStatus().equals(CellStatus.NOTAVAILABLE))
30              throw new RuntimeException("The cell is not available");
31          Piece piece = new Piece(player.getColor(), x, y, z);
32          player.setPiece(piece);
33          cells[x][y][z].setPiece(piece);
34      }
35
36      public void moveOneCoord(int color, int fromX, int fromY, int fromZ, int toX, int toY, int
    ↪  toZ) throws RuntimeException{
37          if (!isMoveOneCoordPossible(fromX, fromY, fromZ, toX, toY, toZ))
38              throw new RuntimeException("the move is impossible");
39          if (cells[toX][toY][toZ].getStatus().equals(CellStatus.OCCUPIED))
40              throw new RuntimeException("the cell is occupied");
41          if (cells[fromX][fromY][fromZ].getPiece().getColor() != color)
42              throw new RuntimeException("you can't move opponent's piece");
43          cells[toX][toY][toZ].setPiece(cells[fromX][fromY][fromZ].removePiece());
```

```
44      }
45
46      private boolean isMoveOneCoordPossible(int fromX, int fromY, int fromZ, int toX, int toY,
        ↪ int toZ) {
47          if (Math.abs(fromX - toX) > 1 || Math.abs(fromY - toY) > 1 || Math.abs(fromZ - toZ) >
        ↪ 1)
48              return false;
49          if (fromX != toX && fromY != toY)
50              return false;
51          if (fromX != toX && fromZ != toZ)
52              return false;
53          if (fromY != toY && fromZ != toZ)
54              return false;
55          if ((fromX == 0 && toX == 0 && (fromY == 2 || fromY == 0)) && fromZ != toZ)
56              return false;
57          if ((fromX == 2 && toX ==2 && (fromY == 2 || fromY == 0)) && fromZ != toZ)
58              return false;
59          return true;
60      }
61
62      public void moveAnyCoord(int color, int fromX, int fromY, int fromZ, int toX, int toY, int
        ↪  toZ) throws RuntimeException{
63          if (cells[fromX][fromY][fromZ].getPiece().getColor() != color)
64              throw new RuntimeException("you␣can't␣move␣opponent's␣piece");
65          if (cells[toX][toY][toZ].getStatus().equals(CellStatus.EMPTY))
66              cells[toX][toY][toZ].setPiece(cells[fromX][fromY][fromZ].removePiece());
67          else
68              throw new RuntimeException(("the␣cell␣is␣occupied"));
69      }
70
71      public boolean isMill(Player player) {
72          if (isMillByX(player, 0) || isMillByX(player, 2) || isMillByY(player, 0)
73                  || isMillByY(player, 2) || isMillByZHorizontal(player) || isMillByZVertical(
        ↪ player))
74              return true;
75          else
76              return false;
77      }
78
79      private boolean isMillByY(Player player, int x) {
80          int count = 0;
81          Cell[] cellsOfMill = new Cell[3];
82          for (int i=0; i<3; i++) {
83              for (int j=0; j<3; j++) {
84                  if (cells[x][j][i].getStatus().equals(CellStatus.OCCUPIED) && cells[x][j][i].
        ↪ getPiece().getColor() == player.getColor()) {
85                      cellsOfMill[count] = cells[x][j][i];
86                      count++;
87                  }
88              }
89              if (checkMill(count, player, cellsOfMill))
90                  return true;
91              count = 0;
92          }
93          return false;
94      }
95
96      private boolean isMillByX(Player player, int y) {
97          int count = 0;
98          Cell[] cellsOfMill = new Cell[3];
99          for (int i=0; i<3; i++) {
100             for (int j=0; j<3; j++) {
101                 if (cells[j][y][i].getStatus().equals(CellStatus.OCCUPIED) && cells[j][y][i].
        ↪ getPiece().getColor() == player.getColor()) {
102                     cellsOfMill[count] = cells[j][y][i];
103                     count++;
104                 }
105             }
106             if (checkMill(count, player, cellsOfMill))
107                 return true;
108             count = 0;
109         }
110         return false;
111     }
112
113     private boolean isMillByZHorizontal(Player player) {
```

```
114         int count = 0;
115         Cell[] cellsOfMill = new Cell[3];
116         for (int i=0; i<2; i++) {
117             for (int j=0; j<3; j++) {
118                 if (cells[i*2][1][j].getStatus().equals(CellStatus.OCCUPIED) && cells[i*2][1][
    ↪ j].getPiece().getColor() == player.getColor()) {
119                     cellsOfMill[count] = cells[i*2][1][j];
120                     count++;
121                 }
122             }
123             if (checkMill(count, player, cellsOfMill))
124                 return true;
125             count = 0;
126         }
127         return false;
128     }
129 
130     private boolean isMillByZVertical(Player player) {
131         int count = 0;
132         Cell[] cellsOfMill = new Cell[3];
133         for (int i=0; i<2; i++) {
134             for (int j=0; j<3; j++) {
135                 if (cells[1][i*2][j].getStatus().equals(CellStatus.OCCUPIED) && cells[1][i*2][
    ↪ j].getPiece().getColor() == player.getColor()) {
136                     cellsOfMill[count] = cells[1][i*2][j];
137                     count++;
138                 }
139             }
140             if (checkMill(count, player, cellsOfMill))
141                 return true;
142             count = 0;
143         }
144         return false;
145     }
146 
147     private boolean checkMill(int count, Player player, Cell[] cellsOfMill) {
148         if (count == 3) {
149             for (Cell[] cells : player.getMills()) {
150                 if (cells[0] == cellsOfMill[0] && cells[1] == cellsOfMill[1] && cells[2] ==
    ↪ cellsOfMill[2]) {
151                     count = 0;
152                     break;
153                 }
154             }
155         }
156         if (count == 3) {
157             player.addMill(cellsOfMill);
158             return true;
159         }
160         else
161             return false;
162     }
163 
164 
165     public Piece removePiece(int color, int x, int y, int z) throws RuntimeException {
166         if (cells[x][y][z].getPiece().getColor() == color)
167             throw new RuntimeException("you can remove only opponent's piece");
168         if (cells[x][y][z].getStatus().equals(CellStatus.EMPTY))
169             throw new RuntimeException("the cell is empty");
170         if (cells[x][y][z].getStatus().equals(CellStatus.NOTAVAILABLE))
171             throw new RuntimeException("the cell is not available");
172         return cells[x][y][z].removePiece();
173     }
174 }
```

Листинг 4: Player.java

```
1 package ru.vlasova.mills.core;
2 
3 import java.util.ArrayList;
4 
5 public class Player {
6     private PlayerStatus status;
7     private int color;
8     private ArrayList<Piece> pieces;
9     private ArrayList<Cell[]> mills;
```

```
10      private int countOfUsedMills = 0;
11      private int countOfDestroyed = 0;
12
13      public Player(int color) {
14          status = PlayerStatus.INITIAL;
15          this.color = color;
16          pieces = new ArrayList<>();
17          mills = new ArrayList<>();
18      }
19
20      public void addMill(Cell[] cells) {
21          mills.add(cells);
22      }
23
24      public void setStatus(PlayerStatus status) {
25          this.status = status;
26      }
27
28      public void setPiece(Piece piece) throws RuntimeException{
29          if(!status.equals(PlayerStatus.INITIAL))
30              throw new RuntimeException("All pieces are placed");
31          pieces.add(piece);
32          if(pieces.size()==9)
33              status = PlayerStatus.BASIC;
34      }
35
36      public int getColor() {
37          return color;
38      }
39
40      public PlayerStatus getStatus() {
41          return status;
42      }
43
44      public void removePiece(Piece piece) {
45          for(Piece p : pieces)
46              if(p.getX() == piece.getX() && p.getY() == piece.getY() &&
47                      piece.getZ() == p.getZ() && p.getColor() == piece.getColor() && p.
    ↪ getStatus() == PieceStatus.NEW) {
48                  p.setStatus(PieceStatus.DESTROYED);
49                  countOfDestroyed++;
50              }
51          if (countOfDestroyed == 6)
52              status = PlayerStatus.FINAL;
53          if (countOfDestroyed == 7)
54              status = PlayerStatus.LOSER;
55      }
56
57      public void returnMill() {
58          countOfUsedMills--;
59      }
60
61      public ArrayList<Cell[]> getMills() {
62          return mills;
63      }
64
65      public boolean isNewMill() {
66          if (mills.size()>countOfUsedMills) {
67              countOfUsedMills++;
68              return true;
69          }
70          else
71              return false;
72      }
73  }
```

Листинг 5: PlayerStatus.java

```
1  package ru.vlasova.mills.core;
2
3  public enum PlayerStatus {
4      INITIAL, BASIC, FINAL, LOSER;
5  }
```

Листинг 6: Piece.java

```
package ru.vlasova.mills.core;

public class Piece {
    int color;
    PieceStatus status;
    int x;
    int y;
    int z;

    public Piece(int color, int x, int y, int z) {
        this.color = color;
        this.status = PieceStatus.NEW;
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public void setStatus(PieceStatus status) {
        this.status = status;
    }

    public PieceStatus getStatus() {
        return status;
    }

    public int getColor() {
        return color;
    }

    public void setCoord(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getZ() {
        return z;
    }
}
```

Листинг 7: PieceStatus.java

```
package ru.vlasova.mills.core;

public enum PieceStatus {
    NEW, INGAME, DESTROYED
}
```

Листинг 8: Cell.java

```
package ru.vlasova.mills.core;

public enum CellStatus {
    NOTAVAILABLE, EMPTY, OCCUPIED
}
```

Листинг 9: CellStatus.java

```
package ru.vlasova.mills.core;

public enum CellStatus {
    NOTAVAILABLE, EMPTY, OCCUPIED
}
```

Листинг 10: MainActivity.java

```
package ru.vlasova.mills.android;

import android.content.Intent;
import android.graphics.Typeface;
import android.os.Bundle;
import android.os.SystemClock;
import android.provider.Settings;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.
    ↪ LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.activity_main);

        Typeface millTypeface = Typeface.createFromAsset(getAssets(), "fonts/MillFont.ttf");
        Typeface buttonTypeface = Typeface.createFromAsset(getAssets(), "fonts/ButtonFont.otf"
    ↪ );

        final TextView tv = (TextView) findViewById(R.id.mill);
        tv.setTypeface(millTypeface);

        Button start = (Button) findViewById(R.id.buttonStart);
        start.setTypeface(buttonTypeface);
        View.OnClickListener startListener = new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intentWinner = new Intent(MainActivity.this, EndGameActivity.class);
                startActivity(intentWinner);
                Intent intentGame = new Intent(MainActivity.this, GameActivity.class);
                startActivity(intentGame);
                finish();
            }
        };
        start.setOnClickListener(startListener);

        Button regulations = (Button) findViewById(R.id.buttonRegulations);
        regulations.setTypeface(buttonTypeface);
        View.OnClickListener regulationsListener = new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, RegulationsActivity.class);
                startActivity(intent);
                finish();
            }
        };
        regulations.setOnClickListener(regulationsListener);

        Button exit = (Button) findViewById(R.id.buttonExit);
        exit.setTypeface(buttonTypeface);
        View.OnClickListener exitListener = new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
                System.runFinalization();
                System.exit(0);
            }
        };
        exit.setOnClickListener(exitListener);
    }

}
```

Листинг 11: GameActivity.java

```
package ru.vlasova.mills.android;

```

```
3  import android.app.Activity;
4  import android.app.AlertDialog;
5  import android.content.Context;
6  import android.content.DialogInterface;
7  import android.content.Intent;
8  import android.graphics.BitmapFactory;
9  import android.graphics.Canvas;
10 import android.graphics.Color;
11 import android.graphics.Matrix;
12 import android.graphics.Point;
13 import android.graphics.Typeface;
14 import android.os.Bundle;
15 import android.view.Display;
16 import android.view.MotionEvent;
17 import android.view.View;
18 import android.graphics.Bitmap;
19 import android.graphics.Paint;
20 import android.view.WindowManager;
21 import android.widget.LinearLayout;
22
23 import ru.vlasova.mills.core.Cell;
24 import ru.vlasova.mills.core.CellStatus;
25 import ru.vlasova.mills.core.Game;
26 import ru.vlasova.mills.core.PieceStatus;
27 import ru.vlasova.mills.core.PlayerStatus;
28
29 public class GameActivity extends Activity {
30
31     protected void onCreate(Bundle savedInstanceState) {
32         super.onCreate(savedInstanceState);
33         getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.
    ↪ LayoutParams.FLAG_FULLSCREEN);
34         setContentView(new GameView(this));
35     }
36
37     public void create() {}
38
39
40     class GameView extends View {
41         private Bitmap fieldBitmap;
42         private Bitmap background;
43         private Figure[] whiteFigures;
44         private Figure[] blackFigures;
45         private Paint paint;
46         private Point point;
47         private Game game;
48         private int cellSize;
49         boolean afterRemove;
50         int xForMove = -1, yForMove = -1, zForMove = -1;
51         private Point fieldCoords;
52         private int figureSize;
53         LinearLayout linearLayout;
54
55         GameView(Context context) {
56             super(context);
57             Display display = getWindowManager().getDefaultDisplay();
58             point = new Point();
59             display.getSize(point);
60             int x = (int) (point.x * 0.9);
61             fieldBitmap = Bitmap.createScaledBitmap(BitmapFactory.decodeResource(getResources
    ↪ (), R.drawable.field), x, x, false);
62             background = Bitmap.createScaledBitmap(BitmapFactory.decodeResource(getResources()
    ↪ , R.drawable.background), point.x, point.y, false);
63             paint = new Paint(Paint.ANTI_ALIAS_FLAG);
64             fieldCoords = new Point();
65             cellSize = fieldBitmap.getHeight()/14;
66             fieldCoords.x = (point.x - fieldBitmap.getWidth())/2;
67             fieldCoords.y = (point.y - fieldBitmap.getWidth())/2;
68             game = new Game();
69
70             whiteFigures = new Figure[9];
71             blackFigures = new Figure[9];
72             int size = fieldBitmap.getHeight()/8;
73             for (int i=0; i<9; i++) {
74                 whiteFigures[i] = new Figure(0, i*size, fieldCoords.y-size, size);
75                 blackFigures[i] = new Figure(1, i*size, fieldCoords.y+fieldBitmap.getWidth(),
```

```
    ↪ size);
            }

        }

        protected void onDraw(Canvas canvas) {
            canvas.drawBitmap(background, 0, 0, null);
            canvas.drawBitmap(fieldBitmap, fieldCoords.x, fieldCoords.y, paint);
            drawField(canvas);

            for (Figure figure : whiteFigures) {
                if (figure.getStatus().equals(PieceStatus.NEW))
                    figure.draw(canvas);
            }
            for (Figure figure : blackFigures) {
                if (figure.getStatus().equals(PieceStatus.NEW))
                    figure.draw(canvas);
            }
            drawDetails(canvas);
        }

        public boolean onTouchEvent(MotionEvent event) {
            double eventX = event.getX();
            double eventY = event.getY();
            int z = calculateZ(eventX, eventY);
            Point coordXY = calculateXY(eventX, eventY, z);
            int x = coordXY.x;
            int y = coordXY.y;
            try {
                if (!game.isAllPiecesSet() && x != -1 && y != -1 && z != -1) {
                    makeInitiatingMove(x, y, z);
                    invalidate();
                }
                else if (x != -1 && y != -1 && z != -1 && !game.getActivePlayer().isNewMill())
    ↪ {
                    movePiece(x, y, z);
                    checkWinner();
                }
                else if (x != -1 && y != -1 && z != -1) {
                    game.removePiece(x, y, z);
                    checkWinner();
                    invalidate();

                }
            } catch (RuntimeException e) {
                invalidate();
                return false;
            }
            if (y == -1)
                checkButtons(eventX, eventY);
            return false;
        }

        public void drawField(Canvas fieldCanvas) {
            int color;
            Figure figure;
            Cell[][][] cells = game.getField();
            for(int i=0; i<3; i++)
                for(int j=0; j<3; j++)
                    for(int k=0; k<3; k++)
                        if(cells[i][j][k].getStatus().equals(CellStatus.OCCUPIED)) {
                            color = cells[i][j][k].getPiece().getColor();
                            figure = findFigure(i, j, k, color);
                            if (figure != null && !isAllInitialized()) {

                                figure.draw(fieldCanvas);
                            }
                            if (figure == null && !isAllInitialized())
                                initializeFigure(fieldCanvas, i, j, k, color);
                            else if (isAllInitialized() && figure != null){
                                figure.draw(fieldCanvas);
                            }
                        }
        }

        private void initializeFigure(Canvas canvas, int x, int y, int z, int color) {
```

```
150             if (color == 0) {
151                 for (Figure figure : whiteFigures) {
152                     if (figure.getStatus().equals(PieceStatus.NEW)) {
153                         initializeFigure(canvas, x, y, z, figure);
154                         break;
155                     }
156                 }
157             }
158             else {
159                 for (Figure figure : blackFigures) {
160                     if (figure.getStatus().equals(PieceStatus.NEW)) {
161                         initializeFigure(canvas, x, y, z, figure);
162                         break;
163                     }
164                 }
165             }
166         }
167 
168         private void initializeFigure(Canvas canvas, int x, int y, int z, Figure figure) {
169             figure.setCoords(x, y, z);
170             int drawX = calculateCoordAfterMove(x, z)+fieldCoords.x-cellSize/2-10;
171             int drawY = calculateCoordAfterMove(y, z)+fieldCoords.y-cellSize/2-10;
172             figure.setDrawCoords(drawX, drawY);
173             figure.draw(canvas);
174             figure.setStatus(PieceStatus.INGAME);
175         }
176 
177         private boolean isAllInitialized() {
178             for (Figure figure : whiteFigures)
179                 if (figure.getStatus().equals(PieceStatus.NEW))
180                     return false;
181             for (Figure figure : blackFigures)
182                 if (figure.getStatus().equals(PieceStatus.NEW))
183                     return false;
184             return true;
185         }
186 
187         private Figure findFigure(int x, int y, int z, int color) {
188             if (color == 0) {
189                 for (Figure figure : whiteFigures) {
190                     if (figure.checkCoords(x, y, z))
191                         return figure;
192                 }
193             }
194             else {
195                 for (Figure figure : blackFigures) {
196                     if (figure.checkCoords(x, y, z))
197                         return figure;
198                 }
199             }
200             return null;
201         }
202 
203         /** private void drawFigure(Canvas canvas, int x, int y, int color) {
204             if (color == 0) {
205                 for (Figure figure : whiteFigures) {
206                     if (figure.getStatus().equals(PieceStatus.NEW)) {
207                         if (drawFigure(figure, canvas, x, y))
208                             break;
209                     }
210                 }
211             }
212             else {
213                 for (Figure figure : blackFigures) {
214                     if (figure.getStatus().equals(PieceStatus.NEW)) {
215                         if (drawFigure(figure, canvas, x, y))
216                             break;
217                     }
218                 }
219             }
220         }
221 
222         private boolean drawFigure(Figure figure, Canvas canvas, int x, int y) {
223             if (figure.getStatus().equals(PieceStatus.NEW)) {
224                 figure.setCoords(x, y);
225                 figure.setStatus(PieceStatus.INGAME);
```

```
226                 figure.draw(canvas);
227                 return true;
228             }
229             return false;
230         }*/
231 
232         private int calculateCoordAfterMove(int coord, int z) {
233             int trueCoord;
234             if(coord==0)
235                 trueCoord = cellSize*(2*z+1)-figureSize;
236             else if(coord==1)
237                 trueCoord = cellSize*7-figureSize;
238             else
239                 trueCoord = cellSize*(13-2*z)-figureSize;
240             return trueCoord;
241         }
242 
243         private int calculateCoordBeforeMove(double coord, int z) {
244             int trueCoord;
245             if(coord>=(cellSize/2) && coord<=cellSize*(2*z+2))
246                 trueCoord = 0;
247             else if(coord>=6*cellSize && coord<=8*cellSize)
248                 trueCoord = 1;
249             else if(coord>=(cellSize*(13-2*z)-cellSize/2) && coord<=14*cellSize-cellSize/3)
250                 trueCoord = 2;
251             else trueCoord = -1;
252             return trueCoord;
253 
254         }
255 
256         private Point calculateXY(double x, double y, int z) {
257             double trueX = x - (point.x - fieldBitmap.getWidth())/2;
258             double trueY = y - (point.y - fieldBitmap.getHeight())/2;
259             return new Point(calculateCoordBeforeMove(trueX, z), calculateCoordBeforeMove(
    ↪ trueY, z));
260         }
261 
262         private int calculateZ(double x, double y) {
263             double trueX = x - (point.x - fieldBitmap.getWidth())/2;
264             double trueY = y - (point.y - fieldBitmap.getHeight())/2;
265             int z;
266             if(trueX<=2*cellSize || trueX>=13*cellSize)
267                 z = 0;
268             else if((trueX>2*cellSize && trueX<4*cellSize && trueY>=2*cellSize && trueY<=11*
    ↪ cellSize)
269                     || (trueX>10*cellSize && trueX<13*cellSize && trueY>=2*cellSize && trueY
    ↪ <=11*cellSize))
270                 z = 1;
271             else if((trueX>=4*cellSize && trueX<=6*cellSize && trueY>=5*cellSize && trueY<=9*
    ↪ cellSize)
272                     || (trueX>=8*cellSize && trueX<=10*cellSize && trueY>=5*cellSize && trueY
    ↪ <=9*cellSize))
273                 z = 2;
274             else if(trueX>6*cellSize && trueX<8*cellSize) {
275                 if(trueY<=1.5*cellSize || trueY>=12.5*cellSize)
276                     z = 0;
277                 else if((trueY>=2*cellSize && trueY<=4*cellSize && trueX>=2*cellSize && trueX
    ↪ <=11*cellSize)
278                         || (trueY>=10*cellSize && trueY <=12*cellSize && trueY>=2*cellSize &&
    ↪ trueX<=11*cellSize))
279                     z = 1;
280                 else if((trueY>=4*cellSize && trueY<=6*cellSize && trueX>=5*cellSize && trueX
    ↪ <=9*cellSize)
281                         || (trueY>=8*cellSize && trueY<=10*cellSize && trueX>=5*cellSize &&
    ↪ trueX<=9*cellSize))
282                     z = 2;
283                 else z = -1;
284             }
285             else
286                 z = -1;
287             return z;
288         }
289 
290         public void movePiece(int x, int y, int z) {
291             Figure figure;
292             if (game.isMill()) {
```

```
293                 game.removePiece(x, y, z);
294             }
295             else if (xForMove == -1) {
296                 figure = findFigure(x, y, z, game.getActivePlayer().getColor());
297                 if (figure != null) {
298                     xForMove = x;
299                     yForMove = y;
300                     zForMove = z;
301                 }
302             }
303             else {
304                 figure = findFigure(xForMove, yForMove, zForMove, game.getActivePlayer().
    ↪ getColor());
305                 if (game.makeMove(xForMove, yForMove, zForMove, x, y, z)) {
306                     figure.setCoords(x, y, z);
307                     figure.setDrawCoords(calculateCoordAfterMove(x, z) + fieldCoords.x -
    ↪ cellSize / 2 - 10,
308                             calculateCoordAfterMove(y, z) + fieldCoords.y - cellSize / 2 - 10)
    ↪ ;
309                     invalidate();
310                 }
311                 xForMove = -1;
312                 yForMove = -1;
313                 zForMove = -1;
314             }
315         }
316 
317         /**public void deleteFigure(int x, int y, int z,int color) {
318             x = calculateCoordAfterMove(x, z);
319             y = calculateCoordAfterMove(y, z);
320             if (color == 0) {
321                 for (Figure figure : whiteFigures) {
322                     if (figure.checkCoords(x, y))
323                         figure.delete();
324                     break;
325                 }
326             }
327             else {
328                 for (Figure figure : blackFigures) {
329                     if (figure.checkCoords(x, y))
330                         figure.delete();
331                     break;
332                 }
333             }
334         }*/
335 
336         private void makeInitiatingMove(int x, int y, int z) throws RuntimeException{
337             if (game.makeMove(x, y, z)) {
338                 invalidate();
339             }
340             else {
341                 Figure figure = findFigure(x, y, z, game.getActivePlayer().getColor());
342                 figure.destroy();
343                 invalidate();
344             }
345         }
346 
347         private void drawDetails(Canvas canvas) {
348             Paint fontPaint = new Paint();
349             Typeface fontType = Typeface.createFromAsset(getAssets(), "fonts/ButtonFont.otf");
350             fontPaint.setTypeface(fontType);
351             fontPaint.setColor(Color.WHITE);
352             fontPaint.setTextSize(50);
353             float x = point.x/2-point.x/5;
354             canvas.drawText("ИГРОК 1", x, point.y/8, fontPaint);
355             canvas.drawText("ИГРОК 2", x, point.y-point.y/12, fontPaint);
356 
357             int size = point.x/10;
358             Bitmap replay = Bitmap.createScaledBitmap(BitmapFactory.decodeResource(
    ↪ getResources(), R.drawable.replay), size, size, false);
359             canvas.drawBitmap(replay, point.x-2*size+size/2, size/2, null);
360             Bitmap home = Bitmap.createScaledBitmap(BitmapFactory.decodeResource(getResources
    ↪ (), R.drawable.home), size, size, false);
361             canvas.drawBitmap(home, size/2, size/2, null);
362         }
363 
```

```
364         private void checkButtons(double x, double y) {
365             int size = point.x/10;
366             if (x > size/2 && x < size/2+size && y > size/2 && y < size/2+size) {
367                 Intent intent = new Intent(GameActivity.this, MainActivity.class);
368                 startActivity(intent);
369                 finish();
370             }
371             int coordX = point.x-2*size+size/2;
372             if (x > coordX & x < coordX+size && y > size/2 && y < size/2+size) {
373                 Intent intent = new Intent(GameActivity.this, GameActivity.class);
374                 startActivity(intent);
375                 finish();
376             }
377         }
378 
379         private void checkWinner() {
380             if (game.getActivePlayer().getStatus().equals(PlayerStatus.LOSER)) {
381                 finish();
382             }
383         }
384     }
385 
386     class Figure {
387         private int x;
388         private int y;
389         private int z;
390         private int drawX;
391         private int drawY;
392         private int color;
393         private int size;
394         private Bitmap bitmap;
395         private PieceStatus status;
396 
397         Figure(int color, int drawX, int drawY, int size) {
398             this.color = color;
399             this.drawX = drawX;
400             this.drawY = drawY;
401             this.x = -1;
402             this.y = -1;
403             this.z = -1;
404             this.size = size;
405             this.status = PieceStatus.NEW;
406             if (color == 0)
407                 this.bitmap = Bitmap.createScaledBitmap(BitmapFactory.decodeResource(
    ↪ getResources(), R.drawable.white), size, size, false);
408             else
409                 this.bitmap = Bitmap.createScaledBitmap(BitmapFactory.decodeResource(
    ↪ getResources(), R.drawable.black), size, size, false);
410         }
411 
412         public void setCoords(int x, int y, int z) {
413             this.x = x;
414             this.y = y;
415             this.z = z;
416         }
417 
418         public void setDrawCoords(int x, int y) {
419             this.drawX = x;
420             this.drawY = y;
421         }
422 
423         public Bitmap getBitmap() {
424             return bitmap;
425         }
426 
427         public boolean checkCoords(int x, int y, int z) {
428             return this.x == x && this.y == y && this.z == z;
429         }
430 
431         public void draw(Canvas canvas){
432             canvas.drawBitmap(bitmap, drawX, drawY, null);
433         }
434 
435         public PieceStatus getStatus() {
436             return status;
437         }
```

```
438
439         public void setStatus(PieceStatus status) {
440             this.status = status;
441         }
442
443         public void moveBitmap(Canvas canvas, int x, int y) {
444             Matrix matrix = new Matrix();
445             matrix.preTranslate(x-drawX, y-drawY);
446             canvas.drawBitmap(bitmap, matrix, null);
447         }
448
449         public void destroy() {
450             status = PieceStatus.DESTROYED;
451             x = -1;
452             y = -1;
453             z = -1;
454         }
455
456     }
457 }
```

Листинг 12: RegulationsActivity.java

```
1  package ru.vlasova.mills.android;
2
3  import android.app.Activity;
4  import android.content.Intent;
5  import android.content.res.AssetManager;
6  import android.graphics.Typeface;
7  import android.os.Bundle;
8  import android.support.v7.app.AppCompatActivity;
9  import android.view.View;
10 import android.view.WindowManager;
11 import android.widget.Button;
12 import android.widget.ScrollView;
13 import android.widget.TextView;
14
15 import java.io.BufferedReader;
16 import java.io.File;
17 import java.io.FileInputStream;
18 import java.io.IOException;
19 import java.io.InputStreamReader;
20
21 import java.io.InputStream;
22 import java.io.Reader;
23 import java.net.URL;
24 import java.nio.Buffer;
25
26 public class RegulationsActivity extends Activity {
27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.
    ↪ LayoutParams.FLAG_FULLSCREEN);
31         setContentView(R.layout.activity_regulations);
32
33         TextView tv = (TextView) findViewById(R.id.regulations);
34         Typeface buttonTypeface = Typeface.createFromAsset(getAssets(), "fonts/ButtonFont.otf"
    ↪ );
35         tv.setTypeface(buttonTypeface);
36
37         TextView text = (TextView) findViewById(R.id.text);
38         try {
39             text.setText(readFile());
40         } catch (IOException e) {}
41
42         Button button = (Button) findViewById(R.id.back);
43         button.setTypeface(buttonTypeface);
44         View.OnClickListener exitListener = new View.OnClickListener() {
45             @Override
46             public void onClick(View v) {
47                 Intent intent = new Intent(RegulationsActivity.this, MainActivity.class);
48                 startActivity(intent);
49                 finish();
50             }
51         };
```

```
52          button.setOnClickListener(exitListener);
53      }
54
55      private String readFile() throws IOException {
56          AssetManager manager = this.getAssets();
57          InputStreamReader isr = new InputStreamReader(manager.open("files/regulations.txt"), "
    ↪ windows-1251");
58          BufferedReader br = new BufferedReader(isr);
59          String str = "";
60          String result = "";
61          while ((str = br.readLine()) != null) {
62              result += str;
63              result += "\n";
64          }
65          return result;
66      }
67  }
```

Листинг 13: EndGameActivity.java

```
1   package ru.vlasova.mills.android;
2
3   import android.app.Activity;
4   import android.content.Intent;
5   import android.graphics.Typeface;
6   import android.os.Bundle;
7   import android.view.View;
8   import android.view.WindowManager;
9   import android.widget.Button;
10  import android.widget.TextView;
11
12  public class EndGameActivity extends Activity {
13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.
    ↪ LayoutParams.FLAG_FULLSCREEN);
17          setContentView(R.layout.activity_endgame);
18
19          Typeface buttonTypeface = Typeface.createFromAsset(getAssets(), "fonts/ButtonFont.otf"
    ↪ );
20          TextView tv = (TextView) findViewById(R.id.winner);
21          tv.setTypeface(buttonTypeface);
22
23          Button replay = (Button) findViewById(R.id.buttonReplay);
24          replay.setTypeface(buttonTypeface);
25          View.OnClickListener replayListener = new View.OnClickListener() {
26              @Override
27              public void onClick(View v) {
28                  Intent intent = new Intent(EndGameActivity.this, GameActivity.class);
29                  startActivity(intent);
30                  finish();
31              }
32          };
33          replay.setOnClickListener(replayListener);
34
35          Button menu = (Button) findViewById(R.id.buttonMenu);
36          menu.setTypeface(buttonTypeface);
37          View.OnClickListener menuListener = new View.OnClickListener() {
38              @Override
39              public void onClick(View v) {
40                  Intent intent = new Intent(EndGameActivity.this, MainActivity.class);
41                  startActivity(intent);
42                  finish();
43              }
44          };
45          menu.setOnClickListener(menuListener);
46      }
47  }
```