Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: Программирование для мобильных платформ**

Работу выполнила: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. Н. Баева

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т. А. Приходько

**Тема работы:** Библиотека AWT и SWING для построения графического интерфейса пользователя. События и их обработка.

**Ход работы:**

В лабораторной работе необходимо было разработать игру 2048.

Игровое поле для игры представляет собой квадрат 4×4, на котором в стартовой позиции две плитки уже открыты. Они могут располагаться как угодно, варианты начальной расстановки – «2» и «2», «2» и «4», «4» и «4». «Костяшки» можно перемещать в одну из четырёх сторон (если этому не мешает расположение), при этом, когда две плитки одинакового номинала сталкиваются друг с другом, то становятся одной плиткой, значение которой удваивается. За один ход плитка может складываться в разных местах игрового поля, но лишь один раз увеличивать свой номинал. Другими словами, если подряд расположены «2», «2» и «4», то за ход можно объединить только «2» и «2» в «4», а получившуюся «4» с блоком такого же значения – лишь во время следующего хода.

Особенностью 2048 является то, что движется всё игровое поле до границы, а не отдельные столбцы или строчки. После каждого такого хода на случайной пустой клетке появляется новая плитка номинала «2» (с вероятностью 90%) или «4» (с вероятностью 10%). Игра считается проигранной, если у игрока не осталось ходов (все клетки заполнены «костяшками», возможностей их объединить нет). Для победы в изначальной версии игры нужно было собрать плитку номиналом «2048» (откуда собственно и название).

В игре также предусмотрен подсчёт очков – суммы всех соединённых за игру плиток. Текущие очки за игру и лучший персональный результат отображаются в правом верхнем углу.

Были разработаны три класса ­– Tile (работа с отдельной плиткой на поле), Board (работа с игровым полем) и Game (логика игры и работа с графическим интерфейсом).

В классе Tile каждая плитка определяется целочисленным значением и цветом (public int value; и Color tileColor; соответственно). Далее следуют конструкторы пустой плитки (со значением 0) и не пустой плитки, сеттеры и геттеры значения плитки (они будут использоваться при сложении двух плиток). После следует переопределённый метод toString(), который возвращает значение плитки как строку (для графического интерфейса). Последним методом в классе был сеттер цвета плитки в зависимости от её значения, в частности, от степени двойки. Для цвета используется класс Color в пространстве RGB из библиотеки AWT. На рисунке 1 представлен фрагмент кода сеттера.

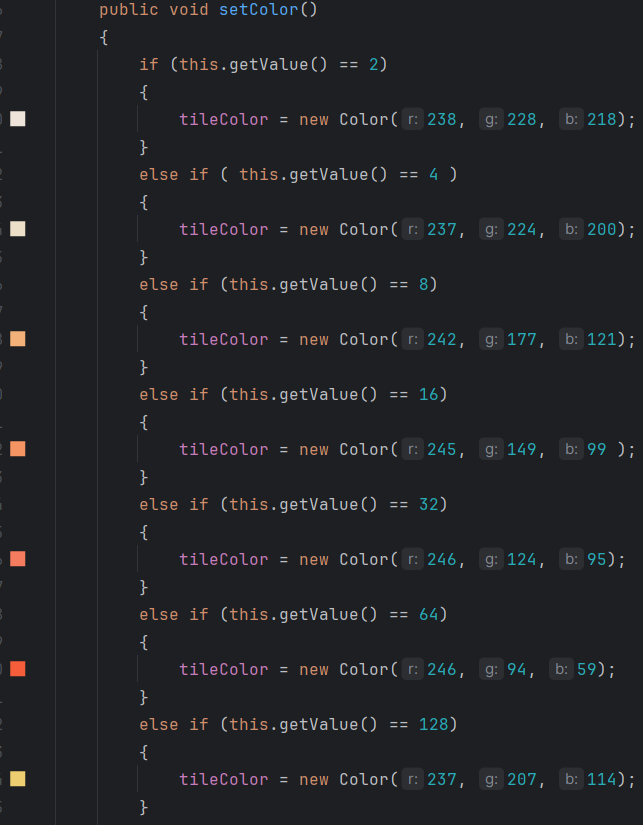


Рисунок 1 – Сеттер цвета плитки.

Класс Board работает с игровым полем. Сперва инициализируются массив элементов из вышеопределённого класса Tile, затем целочисленные переменные размерности сетки, границы и игрового счёта. Далее следует конструктор игрового поля размером 4х4. В нём с помощью цикла for заполняется массив board типа Tile. После следуют геттеры игрового счёта и самого большого значения плитки (аналогично цикл, проходящий по всей доске и поиск плитки с максимальным значением). Как и в предыдущем классе, переопределяется метод toString(), который возвращает игровую доску в виде строки для графического интерфейса. Фрагмент соответствующих методов представлены на рисунке 2.

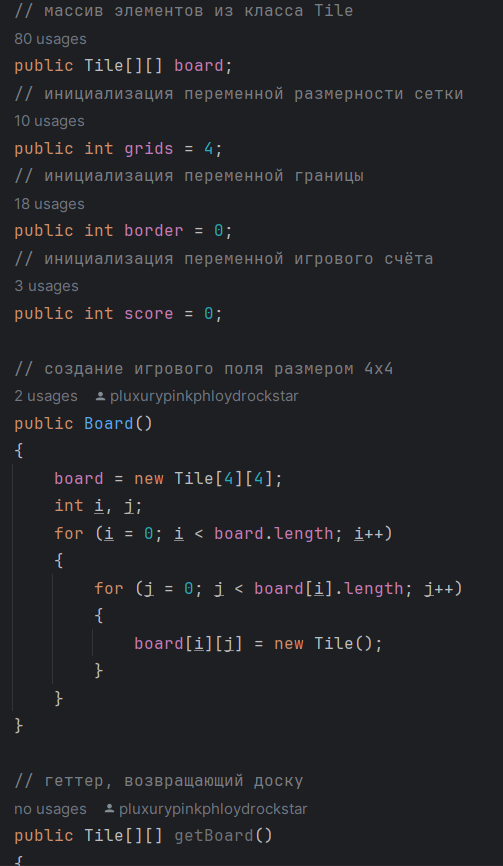


Рисунок 2 – Инициализация переменных, конструкторов и геттеров/сеттеров в классе Board

Следующие методы отвечают за складывание плиток и проверяют случаи окончания игры. Метод spawn() генерирует случайное число 2 или 4 на пустой плитке игрового поля. Он использует цикл while для поиска пустой плитки, а затем присваивает ей значение 2 с вероятностью 80% или 4 с вероятностью 20%. Этот метод вызывается каждый раз, когда игрок делает ход. Метод blackOut() проверяет, полностью ли заполнено игровое поле. Он подсчитывает количество плиток, у которых значение больше 0, и возвращает true, если их 16, то есть все плитки заняты. Этот метод может использоваться для определения, есть ли еще возможные ходы для игрока. Метод gameOver() проверяет, окончена ли игра. Он возвращает true, если игровое поле заполнено и ни одна из плиток не может объединиться с соседней плиткой по горизонтали или вертикали. Этот метод использует вложенные циклы for и условные операторы if для проверки всех плиток на равенство соседних значений.

Плитки в игре можно сдвигать вверх, вниз, влево или вправо с помощью клавиш-стрелок или клавиш WASD. Метод verticalMove принимает три параметра: row, col и direction. Он сравнивает плитку в указанной строке и столбце (compare) с плиткой на границе (initial). Граница — это верхняя или нижняя строка доски, в зависимости от направления движения. Если initial имеет значение 0 или равно compare, то они объединяются в одну плитку, а их значения складываются. Если нет, то граница сдвигается на одну строку вверх или вниз и вызывается метод verticalMove снова. Метод up вызывает метод verticalMove с параметром direction равным “up” для каждой плитки на доске. Он начинает с верхней строки и двигается вниз по каждому столбцу. Он объединяет плитки с одинаковыми значениями и сдвигает их вверх. Метод down вызывает метод verticalMove с параметром direction равным “down” для каждой плитки на доске. Он начинает с нижней строки и двигается вверх по каждому столбцу. Он объединяет плитки с одинаковыми значениями и сдвигает их вниз.

Класс Game, отвечающий за реализацию графического интерфейса, расширяет класс JPanel и реализует интерфейс KeyListener. Он предназначен для создания и управления игрой. Метод Game() — это конструктор класса, который инициализирует некоторые поля и объекты, необходимые для игры. Он создает новый объект класса Board, который представляет игровое поле, состоящее из 4x4 плиток. Он также создает новый объект класса JFrame, который является окном, в котором отображается игра. Он называет окно “2048” и присваивает строковое представление игрового поля переменной gameBoard. Метод setUpGUI() — это статический метод, который настраивает графический интерфейс для игры. Он добавляет слушателя событий к объекту frame, чтобы он мог реагировать на нажатие клавиш. Он также добавляет объект newGame в контейнер contentPane объекта frame, чтобы он мог отрисовывать игровое поле на панели. Он устанавливает размер окна 600x400 пикселей, делает его видимым и запрещает изменять его размер. Метод keyPressed(KeyEvent e) — это переопределенный метод интерфейса KeyListener, который вызывается, когда пользователь нажимает клавишу на клавиатуре. Он проверяет, какая клавиша была нажата, и выполняет соответствующее действие. Если нажата клавиша W или стрелка вверх, он вызывает метод game.up(), который сдвигает все плитки вверх и объединяет их, если они одинаковые. Затем он вызывает метод game.spawn(), который добавляет новую плитку со значением 2 или 4 на случайное свободное место. Затем он обновляет переменную gameBoard и перерисовывает окно. Аналогично, если нажата клавиша S или стрелка вниз, он вызывает метод game.down(), который сдвигает все плитки вниз. Если нажата клавиша A или стрелка влево, он вызывает метод game.left(), который сдвигает все плитки влево. Если нажата клавиша D или стрелка вправо, он вызывает метод game.right(), который сдвигает все плитки вправо.

Переопределённый метод run() является точкой входа в приложение на Java Swing. Он вызывает метод invokeLater() класса SwingUtilities и передает ему новый объект Runnable, который содержит логику для создания графического интерфейса пользователя (GUI) с помощью setUpGUI(). Метод invokeLater() предназначен для запуска асинхронной операции в потоке обработки событий (event dispatch thread, EDT). Это означает, что код внутри метода run() будет выполнен позже, когда EDT будет готов к обработке новых задач. Это необходимо для обеспечения безопасности потоков при работе с компонентами Swing, так как они не являются потокобезопасными и должны быть изменены только из EDT. Соответствующий метод представлен на рисунке 3.

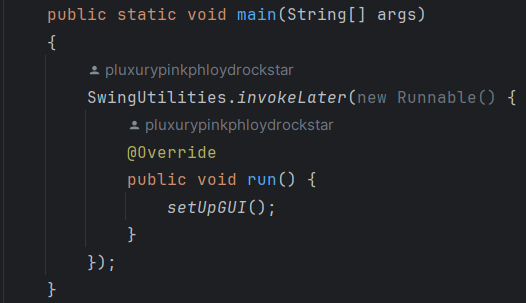


Рисунок 3 – Точка входа в приложение.

Диаграмма классов получившейся игры представлена на рисунке 4.

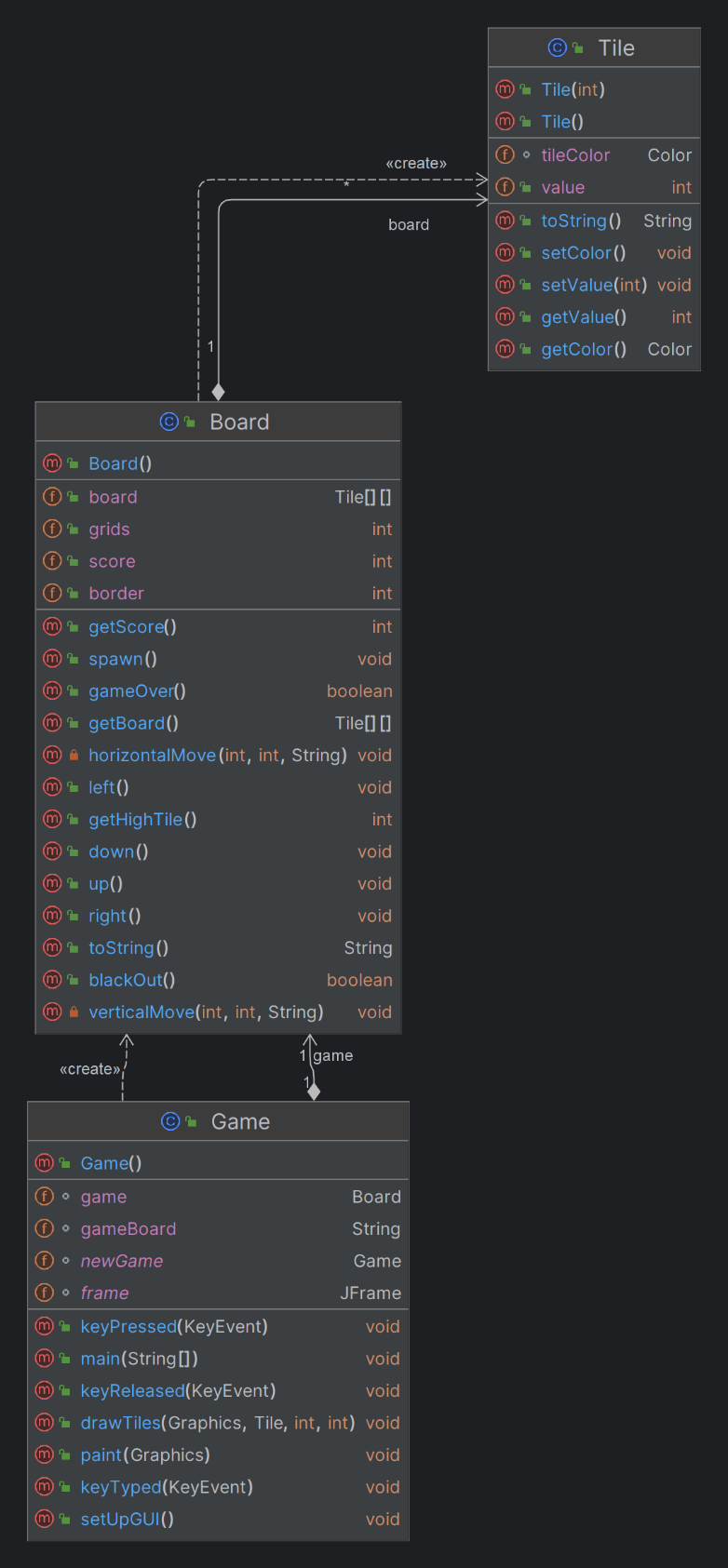


Рисунок 4 – Диаграмма классов.

**Листинг программ**

Файл Tile.java

package Game\_2048;  
  
import java.awt.Color;  
  
// Класс Tile(Плитка) работает с отдельными плитками.  
  
public class Tile {  
  
 // целочисленное значение плитки  
 public int value;  
 // цвет плитки  
 Color tileColor;  
  
 // создание пустой плитки со значением 0  
 public Tile()  
 {  
 value = 0;  
 }  
  
 // создание не пустой плитки со значением  
 public Tile(int number)  
 {  
 value = number;  
 }  
  
 // сеттер значения плитки (используется при сложении двух плиток вместе)  
 public void setValue(int value)  
 {  
 this.value = value;  
 }  
  
 // геттер значения плитки  
 public int getValue()  
 {  
 return value;  
 }  
  
 // возвращение значения плитки как строки (для графического интерфейса)  
 @Override  
 public String toString()  
 {  
 return "" + value;  
 }  
  
 // сеттер цвета плитки (в зависимости от ее значения)  
 public void setColor()  
 {  
 if (this.getValue() == 2)  
 {  
 tileColor = new Color(238, 228, 218);  
 }  
 else if ( this.getValue() == 4 )  
 {  
 tileColor = new Color(237, 224, 200);  
 }  
 else if (this.getValue() == 8)  
 {  
 tileColor = new Color(242, 177, 121);  
 }  
 else if (this.getValue() == 16)  
 {  
 tileColor = new Color(245, 149, 99 );  
 }  
 else if (this.getValue() == 32)  
 {  
 tileColor = new Color(246, 124, 95);  
 }  
 else if (this.getValue() == 64)  
 {  
 tileColor = new Color(246, 94, 59);  
 }  
 else if (this.getValue() == 128)  
 {  
 tileColor = new Color(237, 207, 114);  
 }  
 else if (this.getValue() == 256)  
 {  
 tileColor = new Color(237, 204, 97);  
 }  
 else if (this.getValue() == 512)  
 {  
 tileColor = new Color(237, 200, 80);  
 }  
 else if (this.getValue() == 1024 )  
 {  
 tileColor = new Color(237, 197, 63);  
 }  
 else  
 {  
 tileColor = new Color(237, 194, 46);  
 }  
 }  
  
 // геттер цвета плитки  
 public Color getColor()  
 {  
 this.setColor();  
 return tileColor;  
 }  
  
}

Файл Board.java

package Game\_2048;  
  
// Класс Board (Игровое поле) работает с игровым полем.  
  
public class Board {  
  
 // массив элементов из класса Tile  
 public Tile[][] board;  
 // инициализация переменной размерности сетки  
 public int grids = 4;  
 // инициализация переменной границы  
 public int border = 0;  
 // инициализация переменной игрового счёта  
 public int score = 0;  
  
 // создание игрового поля размером 4x4  
 public Board()  
 {  
 board = new Tile[4][4];  
 int i, j;  
 for (i = 0; i < board.length; i++)  
 {  
 for (j = 0; j < board[i].length; j++)  
 {  
 board[i][j] = new Tile();  
 }  
 }  
 }  
  
 // геттер, возвращающий доску  
 public Tile[][] getBoard()  
 {  
 return board;  
 }  
  
 // геттер, возвращающий игровой счёт  
 public int getScore()  
 {  
 return score;  
 }  
  
 // поиск самой большой по значению плитки  
 public int getHighTile()  
 {  
 int high = board[0][0].getValue();  
 int i, j;  
 for (i = 0; i < board.length; i++)  
 {  
 for (j = 0; j < board[i].length; j++)  
 {  
 if (board[i][j].getValue() > high)  
 {  
 high = board[i][j].getValue();  
 }  
 }  
 }  
 return high;  
 }  
  
 // возвращение игровой доски в виде строки (для графического интерфейса)  
 @Override  
 public String toString()  
 {  
 String s = "";  
 int i, j;  
 for (i = 0; i < board.length; i++)  
 {  
 for (j = 0; j < board[i].length; j++)  
 {  
 s += board[i][j].toString() + " ";  
 }  
 s += "\n";  
 }  
 return s;  
 }  
  
 // генерация 2 или 4 на пустом месте каждый раз, когда делается ход  
 public void spawn()  
 {  
 boolean empty = true;  
 while (empty)  
 {  
 int row = (int)(Math.random() \* 4);  
 int col = (int)(Math.random() \* 4);  
 double x = Math.random();  
 if (board[row][col].getValue() == 0)  
 {  
 if (x < 0.2)  
 {  
 board[row][col] = new Tile(4);  
 empty = false;  
 }  
 else  
 {  
 board[row][col] = new Tile(2);  
 empty = false;  
 }  
 }  
 }  
 }  
  
 // проверка, полностью ли заполнено игровое поле (когда все плитки заняты, но не выигрышной комбинацией)  
 public boolean blackOut()  
 {  
 int count = 0;  
 int i, j;  
 for (i = 0; i < board.length; i++)  
 {  
 for (j = 0; j < board[i].length; j++)  
 {  
 if (board[i][j].getValue() > 0)  
 {  
 count++;  
 }  
 }  
 }  
 if (count == 16)  
 {  
 return true;  
 }  
 return false;  
 }  
  
 // проверка окончена ли игра (когда игровое поле заполнено и ни одна из плиток не может объединиться)  
 public boolean gameOver()  
 {  
 int count = 0;  
 int i, j;  
 for (i = 0; i < board.length; i++)  
 {  
 for (j = 0; j < board[i].length; j++)  
 {  
 if (board[i][j].getValue() > 0)  
 {  
 if (i == 0 && j == 0)  
 {  
 if (board[i][j].getValue() != board[i + 1][j].getValue() && board[i][j].getValue() != board[i][j + 1].getValue())  
 {  
 count++;  
 }  
 }  
 else if (i == 0 && j == 3)  
 {  
 if (board[i][j].getValue() != board[i + 1][j].getValue() && board[i][j].getValue() != board[i][j - 1].getValue())  
 {  
 count++;  
 }  
 }  
 else if (i == 3 && j == 3)  
 {  
 if (board[i][j].getValue() != board[i - 1][j].getValue() && board[i][j].getValue() != board[i][j - 1].getValue())  
 {  
 count++;  
 }  
 }  
 else if (i == 3 && j == 0)  
 {  
 if ( board[i][j].getValue() != board[i - 1][j].getValue() && board[i][j].getValue() != board[i][j + 1].getValue())  
 {  
 count++;  
 }  
 }  
 else if (i == 0 && (j == 1 || j == 2))  
 {  
 if ( board[i][j].getValue() != board[i + 1][j].getValue() && board[i][j].getValue() != board[i][j + 1].getValue() && board[i][j].getValue() != board[i][j - 1].getValue())  
 {  
 count++;  
 }  
 }  
 else if (i == 3 && (j == 1 || j == 2))  
 {  
 if ( board[i][j].getValue() != board[i - 1][j].getValue() && board[i][j].getValue() != board[i][j + 1].getValue() && board[i][j].getValue() != board[i][j - 1].getValue())  
 {  
 count++;  
 }  
 }  
 else if (j == 0 && (i == 1 || i == 2))  
 {  
 if ( board[i][j].getValue() != board[i][j + 1].getValue() && board[i][j].getValue() != board[i - 1][j].getValue() && board[i][j].getValue() != board[i + 1][j].getValue() )  
 {  
 count++;  
 }  
 }  
 else if (j == 3 && (i == 1 || i == 2))  
 {  
 if ( board[i][j].getValue() != board[i][j - 1].getValue() && board[i][j].getValue() != board[i - 1][j].getValue() && board[i][j].getValue() != board[i + 1][j].getValue())  
 {  
 count++;  
 }  
 }  
 else  
 {  
 if (board[i][j].getValue() != board[i][j - 1].getValue() && board[i][j].getValue() != board[i][j + 1].getValue() && board[i][j].getValue() != board[i - 1][j].getValue() && board[i][j].getValue() != board[i + 1][j].getValue())  
 {  
 count++;  
 }  
 }  
 }  
 }  
 }  
 if (count == 16 )  
 {  
 return true;  
 }  
 return false;  
 }  
  
 // сравнение значения двух плиток по вертикали и определение, совпадают ли они или нет  
 // если значение 0 (пустая плитка) — их значения суммируются  
 // рекурсия для прохождения всего столбца  
 private void verticalMove(int row, int col, String direction)  
 {  
 Tile initial = board[border][col];  
 Tile compare = board[row][col];  
 if (initial.getValue() == 0 || initial.getValue() == compare.getValue())  
 {  
 if (row > border || (direction.equals("down") && (row < border)))  
 {  
 int addScore = initial.getValue() + compare.getValue();  
 if (initial.getValue() != 0 )  
 {  
 score += addScore;  
 }  
 initial.setValue(addScore);  
 compare.setValue(0);  
 }  
 }  
 else  
 {  
 if (direction.equals("down"))  
 {  
 border--;  
 }  
 else  
 {  
 border++;  
 }  
 verticalMove(row, col, direction);  
 }  
 }  
  
 // нажатие клавиши «w» или стрелки вверх  
 // вызов метода verticalMove с параметром «вверх» для каждой плитки  
 public void up()  
 {  
 int i, j;  
 for (i = 0; i < grids; i++)  
 {  
 border = 0;  
 for (j = 0; j < grids; j++)  
 {  
 if (board[j][i].getValue() != 0)  
 {  
 if (border <= j)  
 {  
 verticalMove(j, i, "up");  
 }  
 }  
 }  
 }  
 }  
  
 // нажатие клавиши «s» или стрелки вниз  
 // вызов метода verticalMove с параметром «вниз» для каждой плитки  
 public void down()  
 {  
 int i, j;  
 for (i = 0; i < grids; i++)  
 {  
 border = (grids - 1);  
 for (j = grids - 1; j >= 0; j--)  
 {  
 if (board[j][i].getValue() != 0)  
 {  
 if (border >= j)  
 {  
 verticalMove(j, i, "down");  
 }  
 }  
 }  
 }  
 }  
  
 // аналогичное сравнение двух плиток по горизонтали и определение совпадают они или нет  
 private void horizontalMove(int row, int col, String direction)  
 {  
 Tile initial = board[row][border];  
 Tile compare = board[row][col];  
 if (initial.getValue() == 0 || initial.getValue() == compare.getValue())  
 {  
 if (col > border || (direction.equals("right") && (col < border)))  
 {  
 int addScore = initial.getValue() + compare.getValue();  
 if (initial.getValue() != 0 )  
 {  
 score += addScore;  
 }  
 initial.setValue(addScore);  
 compare.setValue(0);  
 }  
 }  
 else  
 {  
 if (direction.equals("right"))  
 {  
 border--;  
 }  
 else  
 {  
 border++;  
 }  
 horizontalMove(row, col, direction);  
 }  
 }  
  
 // нажатие клавиши «a» или стрелки влево  
 // вызов метода HorizontalMove с параметром «влево» для каждой плитки  
 public void left()  
 {  
 int i, j;  
 for (i = 0; i < grids; i++)  
 {  
 border = 0;  
 for (j = 0; j < grids; j++)  
 {  
 if (board[i][j].getValue() != 0)  
 {  
 if (border <= j)  
 {  
 horizontalMove(i, j,"left");  
 }  
 }  
 }  
 }  
 }  
  
 // нажатие клавиши «d» или стрелки вправо  
 // вызов метода HorizontalMove с параметром «вправо» для каждой плитки  
 public void right()  
 {  
 int i, j;  
 for (i = 0; i < grids; i++)  
 {  
 border = (grids - 1);  
 for (j = (grids - 1);j >= 0; j--)  
 {  
 if (board[i][j].getValue() != 0)  
 {  
 if (border >= j)  
 {  
 horizontalMove(i, j, "right");  
 }  
 }  
 }  
 }  
 }  
  
}

Файл Game.java

package Game\_2048;  
  
import javax.swing.\*;  
  
import java.awt.Color;  
import java.awt.Graphics;  
import java.awt.Graphics2D;  
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
  
  
public class Game extends JPanel implements KeyListener {  
 // создание нового объекта класса Board, который представляет игровое поле  
 Board game = new Board();  
 // создание нового объекта класса Game, который используется для настройки графического интерфейса  
 static Game *newGame* = new Game();  
 // создание нового объекта класса JFrame с заголовком  
 static JFrame *frame* = new JFrame("2048");  
 // строковое представление игрового поля  
 String gameBoard = game.toString();  
  
 // настройка графического интерфейса  
 public static void setUpGUI()  
 {  
 // добавление слушателя событий к объекту  
 *frame*.addKeyListener(*newGame*);  
 // добавление объекта newGame в контейнер contentPane объекта frame  
 *frame*.getContentPane().add(*newGame*);  
 // установка размером окна игры  
 *frame*.setSize(600, 400);  
 // установка окна видимым  
 *frame*.setVisible(true);  
 // нельзя изменять размер окна  
 *frame*.setResizable(false);  
 }  
  
 // используется keyPressed по нажатию по кнопке клавиатуры  
 // проверка нажатия соответствующих клавиши Wasd или клавиши со стрелками  
 // обновление JFrame при каждом нажатии  
 @Override  
 public void keyPressed(KeyEvent e)  
 {  
 if (e.getKeyChar() == 'w' || e.getKeyCode() == KeyEvent.*VK\_UP*)  
 {  
 game.up();  
 game.spawn();  
 gameBoard = game.toString();  
 *frame*.repaint();  
 }  
 else if (e.getKeyChar() == 's' || e.getKeyCode() == KeyEvent.*VK\_DOWN*)  
 {  
 game.down();  
 game.spawn();  
 gameBoard = game.toString();  
 *frame*.repaint();  
 }  
 else if (e.getKeyChar() == 'a' || e.getKeyCode() == KeyEvent.*VK\_LEFT*)  
 {  
 game.left();  
 game.spawn();  
 gameBoard = game.toString();  
 *frame*.repaint();  
 }  
 else if (e.getKeyChar() == 'd' || e.getKeyCode() == KeyEvent.*VK\_RIGHT*)  
 {  
 game.right();  
 game.spawn();  
 gameBoard = game.toString();  
 *frame*.repaint();  
 }  
 else if (e.getKeyCode() == KeyEvent.*VK\_ENTER*)  
 {  
 game = new Board();  
 game.spawn();  
 game.spawn();  
 *frame*.repaint();  
 }  
 }  
  
 @Override  
 public void keyReleased(KeyEvent e) {  
 // *TODO Auto-generated method* }  
  
 @Override  
 public void keyTyped(KeyEvent e) {  
 // *TODO Auto-generated method* }  
  
 // отрисовка отдельной плитки  
 public void drawTiles(Graphics g, Tile tile, int x, int y)  
 {  
 int tileValue = tile.getValue();  
 int length = String.*valueOf*(tileValue).length();  
 // создание объекта Graphics2D из объекта Graphics  
 Graphics2D g2 = (Graphics2D)g;  
 // установка цвета рисования - серого  
 g2.setColor(Color.*lightGray*);  
 // заполнение прямоугольника с закругленными углами размером 50x50 пикселей  
 g2.fillRoundRect(x, y, 50, 50, 5, 5);  
 // установка цвета рисования для заголовка цифры - чёрный  
 g2.setColor(Color.*black*);  
 // проверка значения плитки  
 if (tileValue > 0)  
 {  
 // установка цвета рисования в зависимости от значения плитки  
 g2.setColor(tile.getColor());  
 // аналогичное заполнение  
 g2.fillRoundRect(x, y, 50, 50, 5, 5);  
 // установка цвета рисования для заголовка цифры - серого  
 g2.setColor(Color.*black* );  
 // вывод значения плитки в центр прямоугольника  
 g.drawString("" + tileValue, x + 25 - 3 \* length, y + 25);  
 }  
 }  
  
 // отрисовка графического интерфейса  
 @Override  
 public void paint(Graphics g)  
 {  
 // отрисовка фона окна  
 super.paint(g);  
 // создание объекта Graphics2D из объекта Graphics  
 Graphics2D g2 = (Graphics2D)g;  
 // вывод соответствующих заголовков  
 g2.drawString("2048", 250, 20);  
 g2.drawString("Счёт: " + game.getScore(),200 - 4 \* String.*valueOf*(game.getScore()).length(), 40);  
 g2.drawString("Наибольшее значение: " + game.getHighTile(), 280 - 4 \* String.*valueOf*(game.getHighTile()).length(),40);  
 // начало игры  
 g2.drawString("Нажмите 'Enter' чтобы начать", 210, 315);  
 g2.drawString("Используйте клавиши 'wasd' или клавиши со стрелками для игры", 125, 340);  
 // установка цвета рисования для окна - серый  
 g2.setColor(Color.*gray*);  
 // заполнение прямоугольника цветом  
 g2.fillRect(140, 50, 250, 250);  
 int i, j;  
 // отрисовка плиток  
 for (i = 0; i < 4; i++)  
 {  
 for (j = 0; j < 4; j++)  
 {  
 drawTiles(g, game.board[i][j], j \* 60 + 150, i \* 60 + 60);  
 }  
 }  
 // случай, когда игровое поле переполнено  
 if (game.blackOut())  
 {  
 g2.drawString("Нажмите 'Enter' чтобы начать заново", 200, 360);  
 g2.setColor(Color.*gray*);  
 g2.fillRect(140, 50, 250, 250);  
 for (i = 0; i < 4; i++)  
 {  
 for (j = 0; j < 4; j++)  
 {  
 drawTiles(g, game.board[i][j], j \* 60 + 150, i \* 60 + 60);  
 }  
 }  
 }  
 // случай, когда игра окончена  
 if (game.gameOver())  
 {  
 g2.setColor(Color.*gray*);  
 g2.fillRect(140, 50, 250, 250);  
 for (i = 0; i < 4; i++)  
 {  
 for (j = 0; j < 4; j++)  
 {  
 g2.setColor(Color.*RED*);  
 g2.fillRoundRect(j \* 60 + 150, i \* 60 + 60, 50, 50, 5, 5);  
 g2.setColor(Color.*black* );  
 g.drawString("ИГРА", j \* 60 + 160, i \* 60 + 75);  
 g.drawString("ОКОНЧЕНА", j \* 60 + 160, i \* 60 + 95);  
 }  
 }  
 }  
 }  
  
 public static void main(String[] args)  
 {  
 SwingUtilities.*invokeLater*(new Runnable() {  
 @Override  
 public void run() {  
 *setUpGUI*();  
 }  
 });  
 }  
}