Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №6

«Реализация простейшей игры»

Выполнил:

Студент 4 курса

Группы АС-56

Волк Н.А.

Проверил:

Козинский А.А.

Брест 2022

Лабораторная работа №6

**Цель работы:** Разработка приложения, представляющего собой реализацию простейшей игры.

**Задачи лабораторной работы:**

Реализация простейшей игры по собственному сценарию.

(Возможно использование стороннего кода, но необходимо разобраться в нем, воспроизвести, отладить и запустить на исполнение). Сдается КОД с пояснениями.

Найдите среди известных игр ту, сценарий и КОД которой вам удастся найти и воспроизвести (Змейка, Быки и Коровы, морской бой, минер, пасьянс, тетрис, стрелялки, гонки, Agar, спиннер, тенис, сквош,…). ПОВТОРЫ не принимаются. Повтором считается точное воспроизведение одного и того же сценария без собственноручного внесения изменений.

Воспроизведите игру в мобильном приложении. На проверку сдать исходный код и перечень собственных дополнений.

Реализованная игра: Тетрис

Перечень собственных дополнений: добавление возможности переключения режимов игры(hard mode и easy mode), возможности ставить игру на паузу а также постепенное повышение сложности игры за каждые 10 очков счета.

**Код программы:**

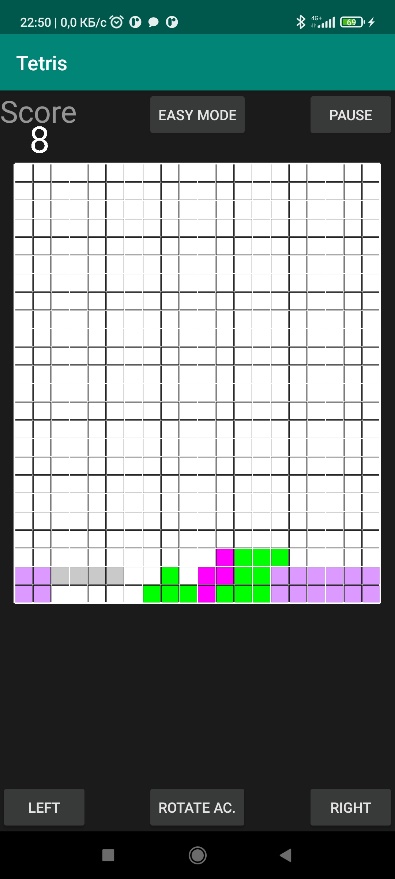
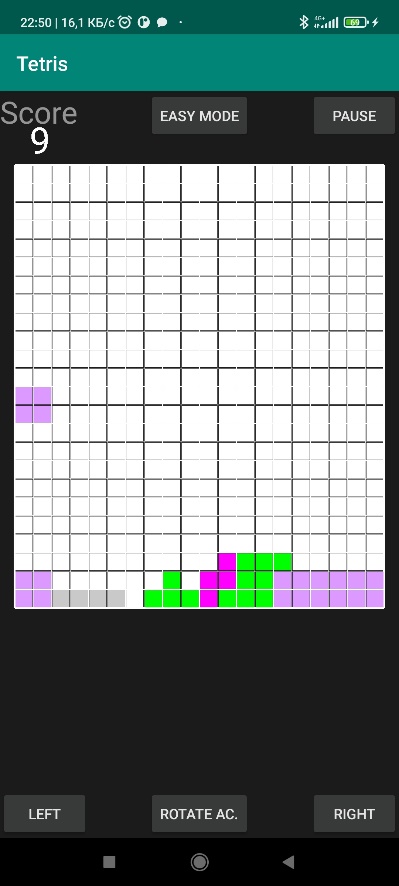
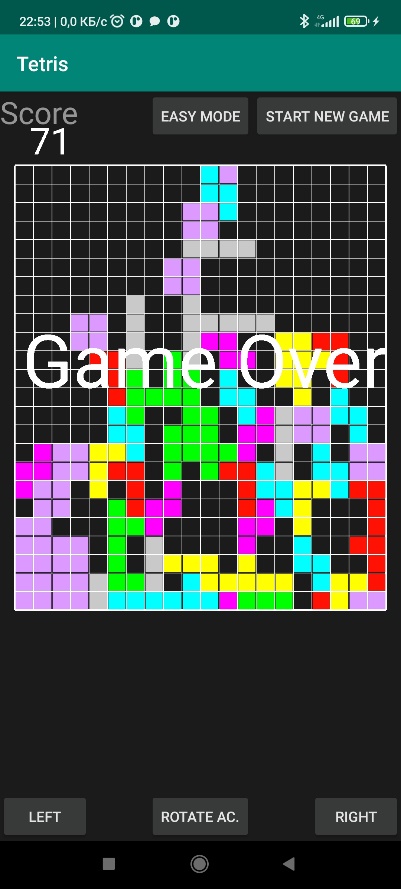
Game.java

package com.tetris.tetris;  
  
import android.content.Intent;  
import android.graphics.Color;  
import android.os.Bundle;  
import android.os.Handler;  
import android.os.Looper;  
import android.support.v7.app.AppCompatActivity;  
import android.view.View;  
import android.widget.Button;  
import android.widget.FrameLayout;  
import android.widget.RelativeLayout;  
import android.widget.TextView;  
  
public class Game extends AppCompatActivity implements View.OnClickListener {  
  
 DrawView drawView;  
 GameState gameState;  
 RelativeLayout gameButtons;  
 Button left;  
 Button right;  
 Button rotateAc;  
 FrameLayout game;  
 Button pause;  
 TextView score;  
 Button difficultyToggle;  
 Handler handler;  
 Runnable loop;  
 int delayFactor;  
 int delay;  
 int delayLowerLimit;  
  
 @Override  
 public void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 gameState = new GameState(24, 20, TetraminoType.*getRandomTetramino*());  
  
 drawView = new DrawView(this, gameState);  
 drawView.setBackgroundColor(Color.*WHITE*);  
  
 game = new FrameLayout(this);  
 gameButtons = new RelativeLayout(this);  
  
 delay = 500;  
 delayLowerLimit = 200;  
 delayFactor = 2;  
  
 left = new Button(this);  
 left.setText(R.string.*left*);  
 left.setId(R.id.*left*);  
  
 right = new Button(this);  
 right.setText(R.string.*right*);  
 right.setId(R.id.*right*);  
  
 rotateAc = new Button(this);  
 rotateAc.setText(R.string.*rotate\_ac*);  
 rotateAc.setId(R.id.*rotate\_ac*);  
  
 pause = new Button(this);  
 pause.setText(R.string.*pause*);  
 pause.setId(R.id.*pause*);  
  
 score = new TextView(this);  
 score.setText(R.string.*score*);  
 score.setId(R.id.*score*);  
 score.setTextSize(30);  
  
 difficultyToggle = new Button(this);  
 difficultyToggle.setText(R.string.*easy*);  
 difficultyToggle.setId(R.id.*difficulty*);  
  
 RelativeLayout.LayoutParams rl = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.*WRAP\_CONTENT*, RelativeLayout.LayoutParams.*WRAP\_CONTENT*);  
 RelativeLayout.LayoutParams leftButton = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.*WRAP\_CONTENT*, RelativeLayout.LayoutParams.*WRAP\_CONTENT*);  
 RelativeLayout.LayoutParams rightButton = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.*WRAP\_CONTENT*, RelativeLayout.LayoutParams.*WRAP\_CONTENT*);  
 RelativeLayout.LayoutParams downButton = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.*WRAP\_CONTENT*, RelativeLayout.LayoutParams.*WRAP\_CONTENT*);  
 RelativeLayout.LayoutParams pausebutton = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.*WRAP\_CONTENT*, RelativeLayout.LayoutParams.*WRAP\_CONTENT*);  
 RelativeLayout.LayoutParams scoretext = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.*WRAP\_CONTENT*, RelativeLayout.LayoutParams.*WRAP\_CONTENT*);  
 RelativeLayout.LayoutParams speedbutton = new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.*WRAP\_CONTENT*, RelativeLayout.LayoutParams.*WRAP\_CONTENT*);  
  
 gameButtons.setLayoutParams(rl);  
 gameButtons.addView(left);  
 gameButtons.addView(right);  
 gameButtons.addView(rotateAc);  
 gameButtons.addView(pause);  
 gameButtons.addView(score);  
 gameButtons.addView(difficultyToggle);  
  
 leftButton.addRule(RelativeLayout.*ALIGN\_PARENT\_LEFT*, RelativeLayout.*TRUE*);  
 leftButton.addRule(RelativeLayout.*ALIGN\_PARENT\_BOTTOM*, RelativeLayout.*TRUE*);  
  
 rightButton.addRule(RelativeLayout.*ALIGN\_PARENT\_RIGHT*, RelativeLayout.*TRUE*);  
 rightButton.addRule(RelativeLayout.*ALIGN\_PARENT\_BOTTOM*, RelativeLayout.*TRUE*);  
  
 downButton.addRule(RelativeLayout.*CENTER\_HORIZONTAL*, RelativeLayout.*TRUE*);  
 downButton.addRule(RelativeLayout.*ALIGN\_PARENT\_BOTTOM*, RelativeLayout.*TRUE*);  
  
 pausebutton.addRule(RelativeLayout.*ALIGN\_PARENT\_RIGHT*, RelativeLayout.*TRUE*);  
 pausebutton.addRule(RelativeLayout.*ALIGN\_PARENT\_TOP*, RelativeLayout.*TRUE*);  
  
 scoretext.addRule(RelativeLayout.*ALIGN\_PARENT\_LEFT*, RelativeLayout.*TRUE*);  
 scoretext.addRule(RelativeLayout.*ALIGN\_PARENT\_TOP*, RelativeLayout.*TRUE*);  
  
 speedbutton.addRule(RelativeLayout.*CENTER\_HORIZONTAL*, RelativeLayout.*TRUE*);  
 speedbutton.addRule(RelativeLayout.*ALIGN\_PARENT\_TOP*, RelativeLayout.*TRUE*);  
  
 left.setLayoutParams(leftButton);  
 right.setLayoutParams(rightButton);  
 rotateAc.setLayoutParams(downButton);  
 pause.setLayoutParams(pausebutton);  
 score.setLayoutParams(scoretext);  
 difficultyToggle.setLayoutParams(speedbutton);  
  
 game.addView(drawView);  
 game.addView(gameButtons);  
 setContentView(game);  
  
 View leftButtonListener = findViewById(R.id.*left*);  
 leftButtonListener.setOnClickListener(this);  
  
 View rightButtonListener = findViewById(R.id.*right*);  
 rightButtonListener.setOnClickListener(this);  
  
 View rotateACButtonListener = findViewById(R.id.*rotate\_ac*);  
 rotateACButtonListener.setOnClickListener(this);  
  
 View pauseButtonListener = findViewById(R.id.*pause*);  
 pauseButtonListener.setOnClickListener(this);  
  
 View speedButtonListener = findViewById(R.id.*difficulty*);  
 speedButtonListener.setOnClickListener(this);  
  
 handler = new Handler(Looper.*getMainLooper*());  
 loop = new Runnable() {  
 public void run() {  
 if (gameState.status) {  
 if (!gameState.pause) {  
 boolean success = gameState.moveFallingTetraminoDown();  
 if (!success) {  
 gameState.paintTetramino(gameState.falling);  
 gameState.lineRemove();  
  
 gameState.pushNewTetramino(TetraminoType.*getRandomTetramino*());  
  
 if (gameState.score % 10 == 9 && delay >= delayLowerLimit) {  
 delay = delay / delayFactor + 1;  
 }  
 gameState.incrementScore();  
 }  
 drawView.invalidate();  
 handler.postDelayed(this, delay);  
 } else {  
 handler.postDelayed(this, delay);  
 }  
 } else {  
 pause.setText(R.string.*start\_new\_game*);  
 }  
 }  
  
 };  
 loop.run();  
 }  
  
 @Override  
 public void onClick(View action) {  
 if (action == left) {  
 gameState.moveFallingTetraminoLeft();  
  
 } else if (action == right) {  
 gameState.moveFallingTetraminoRight();  
  
 } else if (action == rotateAc) {  
 gameState.rotateFallingTetraminoAntiClock();  
  
 } else if (action == pause) {  
 if (gameState.status) {  
 if (gameState.pause) {  
 gameState.pause = false;  
 pause.setText(R.string.*pause*);  
  
 } else {  
 pause.setText(R.string.*play*);  
 gameState.pause = true;  
  
 }  
 } else {  
 pause.setText(R.string.*start\_new\_game*);  
 Intent intent = new Intent(Game.this, MainActivity.class);  
 startActivity(intent);  
  
 }  
 } else if (action == difficultyToggle) {  
 if (!gameState.difficultMode) {  
 delay = delay / delayFactor;  
 gameState.difficultMode = true;  
 difficultyToggle.setText(R.string.*hard*);  
  
 } else {  
 delay = delay \* delayFactor;  
 difficultyToggle.setText(R.string.*easy*);  
 gameState.difficultMode = false;  
  
 }  
 }  
 }  
}

GameState.java

package com.tetris.tetris;  
  
import android.util.SparseArray;  
  
class GameState {  
  
 boolean status;  
 int score;  
 boolean pause;  
 BasicBlock[][] board;  
 Tetramino falling;  
 boolean difficultMode;  
 private int rows;  
 private int columns;  
 private Integer ctr;  
 private SparseArray<Tetramino> tetraminos;  
  
 GameState(int rows, int columns, TetraminoType fallingTetraminoType) {  
  
 this.rows = rows;  
 this.columns = columns;  
 this.pause = false;  
 ctr = 0;  
 score = 0;  
 this.status = true;  
 difficultMode = false;  
  
 board = new BasicBlock[rows][columns];  
 for (int row = 0; row < rows; row++) {  
 for (int column = 0; column < columns; column++) {  
 board[row][column] = new BasicBlock(row, column);  
 }  
 }  
  
 tetraminos = new SparseArray<>();  
  
 falling = new Tetramino(fallingTetraminoType, this.ctr);  
  
 tetraminos.put(this.ctr, falling);  
 }  
  
 private BasicBlock getCoordinateBlock(Coordinate coordinate) {  
  
 return this.board[coordinate.y][coordinate.x];  
 }  
  
 private boolean isConflicting(Coordinate coordinate) {  
  
 if (coordinate.x < 0 || coordinate.x >= this.columns || coordinate.y < 0 || coordinate.y >= this.rows)  
 return true;  
  
 return this.getCoordinateBlock(coordinate).state == BasicBlockState.*ON\_TETRAMINO*;  
  
 }  
  
 private boolean canTetraminoDisplace(Tetramino tetramino, Coordinate displacement) {  
  
 for (BasicBlock block : tetramino.blocks) {  
 if (block.state == BasicBlockState.*ON\_TETRAMINO*) {  
 Coordinate shifted = Coordinate.*add*(block.coordinate, displacement);  
 if (isConflicting(shifted)) {  
 return false;  
 }  
 }  
 }  
 return true;  
 }  
  
 boolean moveFallingTetraminoDown() {  
  
 if (canTetraminoDisplace(falling, new Coordinate(1, 0))) {  
 falling.moveDown();  
 return true;  
 } else {  
 return false;  
 }  
  
 }  
  
 boolean moveFallingTetraminoLeft() {  
  
 if (canTetraminoDisplace(falling, new Coordinate(0, -1))) {  
 falling.moveLeft();  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 boolean moveFallingTetraminoRight() {  
  
 if (canTetraminoDisplace(falling, new Coordinate(0, 1))) {  
 falling.moveRight();  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 boolean rotateFallingTetraminoAntiClock() {  
 if (falling.type == TetraminoType.*SQUARE\_SHAPED*) {  
 return true;  
 } else {  
 for (BasicBlock block : falling.blocks) {  
 if (block.state == BasicBlockState.*ON\_EMPTY*)  
 continue;  
  
 BasicBlock referenceBlock = falling.blocks[0];  
 Coordinate baseCoordinate = Coordinate.*sub*(block.coordinate, referenceBlock.coordinate);  
 if (isConflicting(Coordinate.*add*(Coordinate.*rotateAntiClock*(baseCoordinate), referenceBlock.coordinate))) {  
 return false;  
 }  
 }  
 falling.performClockWiseRotation();  
 return true;  
 }  
 }  
  
 void paintTetramino(Tetramino tetramino) {  
 for (BasicBlock block : tetramino.blocks) {  
 if (block.state == BasicBlockState.*ON\_EMPTY*)  
 continue;  
 this.getCoordinateBlock(block.coordinate).set(block);  
 }  
 }  
  
 void pushNewTetramino(TetraminoType tetraminoType) {  
 this.ctr++;  
  
 falling = new Tetramino(tetraminoType, this.ctr);  
 this.tetraminos.put(this.ctr, falling);  
 for (BasicBlock block : falling.blocks) {  
 if (this.getCoordinateBlock(block.coordinate).state == BasicBlockState.*ON\_TETRAMINO*)  
 this.status = false;  
 }  
 }  
  
 void incrementScore() {  
  
 this.score++;  
 }  
  
 void lineRemove() {  
 boolean removeLines;  
 do {  
 removeLines = false;  
 for (int row = this.rows - 1; row >= 0; row--) {  
 boolean rowIsALine = true;  
 for (int column = 0; column < this.columns; column++) {  
 if (this.board[row][column].state != BasicBlockState.*ON\_TETRAMINO*) {  
 rowIsALine = false;  
 break;  
 }  
 }  
 if (!rowIsALine) {  
 continue;  
 }  
  
 for (int column = 0; column < this.columns; column++) {  
 Tetramino tetramino = this.tetraminos.get((this.board[row][column].tetraId));  
  
 BasicBlock blockToClear = this.board[row][column];  
 blockToClear.setEmptyBlock(blockToClear.coordinate);  
  
 if (tetramino == null) {  
 continue;  
 }  
  
 for (BasicBlock block : tetramino.blocks) {  
 if (block.state == BasicBlockState.*ON\_EMPTY*) {  
 continue;  
 }  
  
 if (block.coordinate.y == row && block.coordinate.x == column) {  
 block.state = BasicBlockState.*ON\_EMPTY*;  
  
 this.ctr++;  
 Tetramino upperTetramino = tetramino.copy(this.ctr);  
 this.tetraminos.put(this.ctr, upperTetramino);  
 for (BasicBlock upperBlock : upperTetramino.blocks) {  
 if (upperBlock.coordinate.y >= block.coordinate.y) {  
 upperBlock.state = BasicBlockState.*ON\_EMPTY*;  
 } else {  
 this.getCoordinateBlock(upperBlock.coordinate).tetraId = upperBlock.tetraId;  
 }  
 }  
  
 this.ctr++;  
 Tetramino lowerTetramino = tetramino.copy(this.ctr);  
 this.tetraminos.put(this.ctr, lowerTetramino);  
 for (BasicBlock lowerBlock : lowerTetramino.blocks) {  
 if (lowerBlock.coordinate.y <= block.coordinate.y) {  
 lowerBlock.state = BasicBlockState.*ON\_EMPTY*;  
 } else {  
 this.getCoordinateBlock(lowerBlock.coordinate).tetraId = lowerBlock.tetraId;  
 }  
 }  
  
 this.tetraminos.remove(block.tetraId);  
 break;  
 }  
  
 }  
 }  
 this.adjustTheMatrix();  
 this.incrementScore();  
 removeLines = true;  
 break;  
 }  
 } while (removeLines);  
 }  
  
 private void adjustTheMatrix() {  
 for (int row = this.rows - 1; row >= 0; row--) {  
 for (int column = 0; column < this.columns; column++) {  
 Tetramino T = (this.tetraminos).get((this.board[row][column].tetraId));  
  
 if (T != null)  
 this.shiftTillBottom(T);  
 }  
 }  
 }  
  
 private void shiftTillBottom(Tetramino tetramino) {  
 boolean shiftTillBottom;  
 do {  
 boolean shouldShiftDown = true;  
 shiftTillBottom = false;  
  
 for (BasicBlock block : tetramino.blocks) {  
 if (block.state == BasicBlockState.*ON\_EMPTY*)  
 continue;  
  
 Coordinate newCoordinate = Coordinate.*add*(block.coordinate, new Coordinate(1, 0));  
  
 if (isTetraPresent(newCoordinate, tetramino))  
 continue;  
  
 if (isConflicting(newCoordinate))  
 shouldShiftDown = false;  
 }  
  
 if (shouldShiftDown) {  
 for (BasicBlock block : tetramino.blocks) {  
 if (block.state == BasicBlockState.*ON\_EMPTY*)  
 continue;  
this.getCoordinateBlock(block.coordinate).setEmptyBlock(block.coordinate);  
  
 block.coordinate.y++;  
 }  
  
 for (BasicBlock block : tetramino.blocks) {  
 if (block.state == BasicBlockState.*ON\_EMPTY*)  
 continue;  
  
 this.getCoordinateBlock(block.coordinate).set(block);  
  
 }  
 shiftTillBottom = true;  
 }  
 } while (shiftTillBottom);  
 }  
  
 private boolean isTetraPresent(Coordinate coordinate, Tetramino tetramino) {  
 for (BasicBlock block : tetramino.blocks) {  
 if (block.state == BasicBlockState.*ON\_EMPTY*)  
 continue;  
  
 if (Coordinate.*isEqual*(block.coordinate, coordinate))  
 return true;  
  
 }  
  
 return false;  
 }  
}

Пример выполнения программы:

***Вывод:*** разработал приложение, предоставляющее возможность использования жестов.