МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Рязанский государственный радиотехнический университет имени
В. Ф. Уткина»

Кафедра систем автоматизированного проектирования вычислительных средств

КУРСОВАЯ РАБОТА

по дисциплине

«Компьютерная графика»

на тему

«Разработка графического приложения «Twinkle Dash» с использованием библиотеки для создания графических интерфейсов libGDX»

Пояснительная записка

Выполнил: студент гр. 143

Вербицкая И.С.

Проверил: к.т.н., доцент

Митрошин А.А.

Рязань 2023

Содержание

Введение

В данной курсовой работе необходимо спроектировать и реализовать графический интерфейс с использованием библиотеки libGDX.

Описание библиотеки libGDX

libGDX – кросс-платформенный фреймворк разработки Java-игр, основанный на OpenGL. Одним из ключевых преимуществ libGDX является его способность работать на различных платформах, таких как Android, iOS, HTML5 и различных операционных системах (Windows, macOS, Linux).

Библиотека предоставляет мощный графический движок, включающий в себя поддержку 2D и 3D графики, спрайтов, текстур, анимаций, шейдеров и других визуальных эффектов, предоставляет инструменты для имитации физики и обработки коллизий, что позволяет создавать реалистичное поведение объектов в игре или приложении. Кроме того, libGDX предоставляет удобные средства для управления ресурсами, такими как изображения, звуки, шрифты и т.д., что позволяет оптимизировать использование памяти и ресурсов устройства. Библиотека обеспечивает поддержку ввода с клавиатуры, сенсорных экранов и других устройств, а также предоставляет множество инструментов для обработки пользовательского ввода.

Эта библиотека предоставляет разработчикам возможность сфокусироваться на самом процессе разработки, предоставляя абстракции для работы с графикой, звуком, вводом и другими аспектами, скрывая специфичные детали каждой платформы от разработчика.

Изучение libGDX позволит приобрести навыки создания высококачественных приложений и игр для различных устройств и платформ, от мобильных устройств до настольных компьютеров. Всё вышеперечисленное определяет **актуальность** данной курсовой работы.

Постановка задачи

В рамках данной курсовой работы необходимо было придумать графическое приложение, разработка которого позволила бы освоить основные возможности библиотеки. В результате была придумана игра «Twinkle Dash». Эта игра про шарик света, который должен перемещаться по игровому пространству, избегая столкновения с острыми шипами на стенах. По мере прохождения игры скорость полета шарика и количество шипов увеличиваются. Итак, **цель** работы заключается в разработки игры «Twinkle Dash» с использованием библиотеки libGDX.

1 Создание и импорт проекта с использованием библиотеки LibGDX

Подготовка

Процесс сборки проекта на libGDX подробно приведен на официальном сайте библиотеки (https://libgdx.com/). Для создания проекта необходим libGDX Project Setup Tool, который также можно загрузить с официального сайта (https://libgdx-nightlies.s3.amazonaws.com/libgdx-runnables/gdx-setup.jar). Для работы с libGDX Project Setup Tool в системе должна быть установлена Java.

Создание проекта

Далее необходимо открыть файл gdx-setup.jar (например, с помощью Java™ Platform SE binary или OpenJDK Platform binary). В результате будет запущено приложение libGDX Project Generator (рисунок 1).

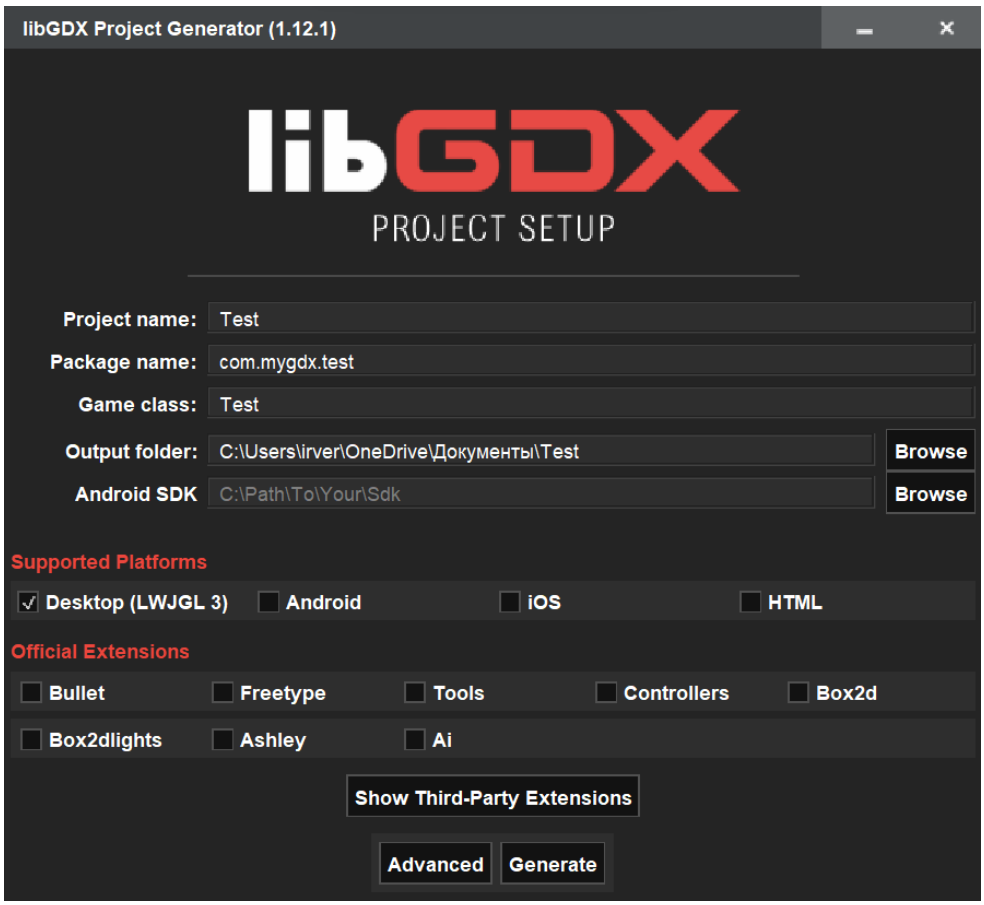

Рисунок 1 – Окно приложения libGDX Project Generator

Перед созданием проекта необходимо указать его параметры – project name (название проекта), package name (название пакетов проекта, название не должно содержать заглавных букв), game class (название класса игры, главного класса приложения), output folder (путь к папке, где будет находится или находится

проект). Параметр android SDK (пусть к android SDK) необходимо указывать только если приложение будет разрабатываться для платформы Android.

В разделе supported platforms необходимо отметить галочками те платформы, на которых планируется разработка проекта. Аналогично в разделе official extensions указываются зависимости от модулей библиотеки libGDX, которые нужны для разработки в данном проекте (например, модуль box2d для симуляции сложной физики или совместимый с ним модуль box2dlights, позволяющий управлять сложным освещением в игре).

После того, как все разделы заполнены, необходимо нажать кнопку generate, после чего в окне сообщений генератора появится соответствующее сообщение о начале сборки проекта (рисунок 2).

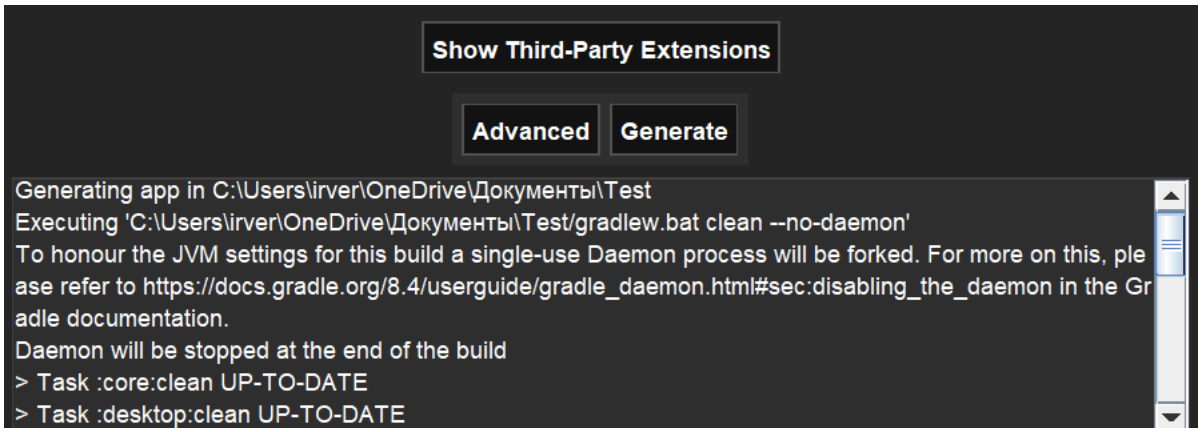

Рисунок 2 – процесс сборки проекта в libGDX Project Generator

Если проект собран успешно, также появится соответствующее сообщение (рисунок 3).

Рисунок 3 – успешная сборка проекта в libGDX Project Generator

В сообщении также указано, как импортировать проект с свою среду разработки. Рассмотрим процесс импорта более подробно на примере среды Intellj IDEA.

Импорт проекта

Для импорта созданного проекта в Intellj IDEA необходимо открыть окно Open File or Project (расположено по пути File -> Open), где нужно указать путь к своему проекту и выбрать файл build.gradle (рисунок 4).

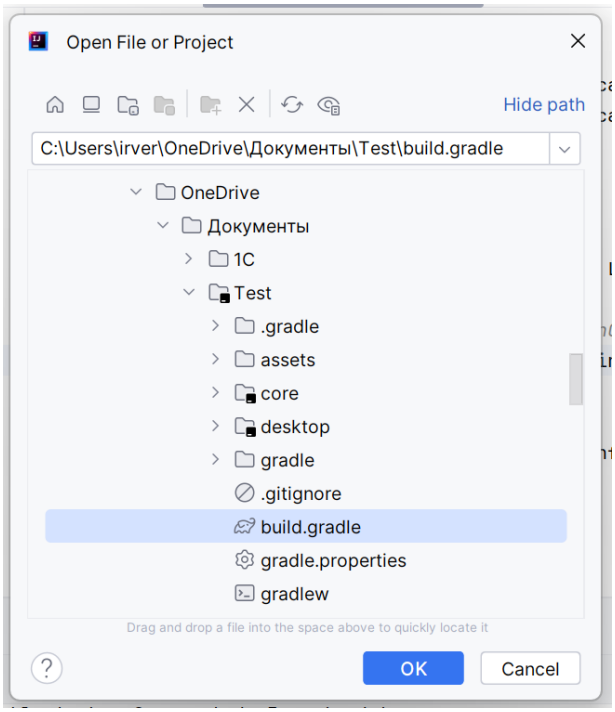

Рисунок 4 – Открытие проекта в Intellj IDEA

После нажатия кнопки ОК проект будет открыт в среде разработки и будет автоматически собран с помощью gradle. Открытие уже существующих проектов, а не новых, осуществляется аналогично.

2 Разработка графического приложения

2.1 Описание классов

Для реализации игры была разработана система классов, рассмотрим их подробнее.

2.1.1 Пакет ru.rsreu.verbickaya.twinkledash

ru.rsreu.verbickaya.twinkledash – корневой пакет проекта, он содержит единственный класс TwinkleDash. Класс TwinkleDash наследует класс Game и является главным классом игры, обеспечивая единую точку входа в приложение.

2.1.2 Пакет ru.rsreu.verbickaya.twinkledash.screens

Этот пакет содержит классы, наследующие класс ScreenAdapteer. Каждый класс отвечает за свое окно игры: AuthorScreen – за окно с информацией об авторе, GameplayScreen – за окно самого игрового процесса, GreetingScreen – за окно заставки игры, MenuScreen – за окно главного меню, NameScreen – за окно ввода имени, RecordsScreen – за окно со списком рекордов.

2.1.3 Пакет ru.rsreu.verbickaya.twinkledash.utils

Этот пакет предназначен для хранения классов, не относящихся напрямую к игровым сущностям. Содержит классы-утилиты Assets, RecordsProcessor и Utils. Класс Assets реализует логику доступа к игровым файлам, RecordsProcessor – логику работы с рекордами игры, Utils – прочие утилиты проекта.

2.1.4 Пакет ru.rsreu.verbickaya.twinkledash.actors

Содержит классы, представляющие собой игровые сущности и наследующие класс Actor. Класс Twinkle реализует шарик, который, по сути, является главным элементом игровой сцены, он описывает логику управления шариком и логику его поведения. Класс Wall – боковые стенки игрового поля, от которых отталкивается шарик (Twinkle), класс Ground – «потолок» и «пол» поля, к которым шарик не может прикасаться. Класс Spike реализует шип на игровой сцене, при соприкосновении с шипом шарик «умирает» и раунд игры заканчивается. Класс SpikesController отвечает за управление группой шипов и реализует логику их появления и исчезновения на сцене.

2.2 Описание интерфейса игры

Как уже было сказано ранее, в игре есть несколько экранов, которые могут показываться пользователю. При запуске игры пользователь видит заставку (рисунок 5), после которой открывается окно ввода имени (рисунок 6). Поле ввода имени ограничено по длине и допустимым символам ввода (можно вводить только цифры и латинские буквы). Кнопка OK доступна только если имя введено. Кнопка Cancel отменяет вход в игру.

Рисунок 5 - Заставка

Рисунок 6 – Окно ввода имени

После успешного ввода имени пользователь попадает в главное меню (рисунок 7), где ему доступен переход к окну рекордов (рисунок 8) кнопкой Records и окну информации об авторе (рисунок 9) кнопкой About author. Из обоих окон можно вернуться в меню с помощью кнопки Menu. Кроме того, на окне рекордов доступен сброс рекордов игры с помощью кнопки Clear records. Все три окна, как и окно игры, отображают кнопку Exit, которая отвечает за выход из приложения.

Рисунок 7 – Главное меню

Рисунок 8 – Окно рекордов

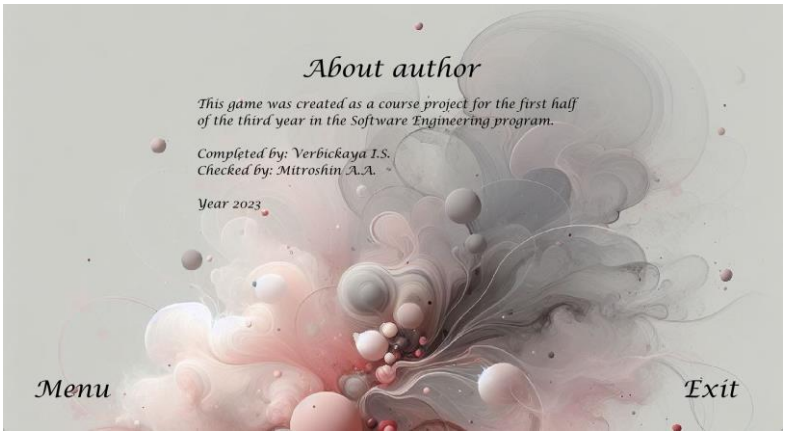

Рисунок 9 – Окно информации об авторе

При нажатии на кнопку Play запускается непосредственно игра (рисунок 10). Управление игрой, как и всем приложением в целом, осуществляется нажатием левой кнопкой мыши по необходимым элементам. Изначально на игровом окне отображается «плавающий» вверх-вниз шарик. При нажатии на экран он начинает свой полет и летит вправо. Столкнувшись со стенкой, он разворачивается влево и так аналогично до конца раунда. На шарик действует гравитация – его тянет вниз, поэтому чтобы лететь ему необходимо регулярно «прыгать» вверх, прыжки также осуществляются кликом мышки.

Рисунок 10 – Начало игры

Каждое столкновение со стенкой и последующий за ним разворот засчитывается в очки за раунд, которые отображены в правом верхнем углу экрана (current score). Изначально на боковых стенках нет шипов, но после первого же столкновения они начинают появляться (рисунок 11).

Рисунок 11 – Игровой процесс

По мере набора очков увеличивается сложность игры – увеличивается количество шипов на стенах и скорость полета шарика. Кроме того, меняется и цветовое оформление сцены (рисунок 12, рисунок 13)

Рисунок 12 – Вариант цветового оформления сцены

Рисунок 13 – Вариант цветового оформления сцены

Если шарик попадает не на боковую стенку, а на шип, раунд заканчивается и шарик «умирает», падая вниз. Для возобновления игры требуется еще один клик по экрану, после чего игровая сцена возвращается к начальному состоянию. Если количество очков, которое набил игрок, входит в десятку рекордов, он получает соответствующее сообщение при окончании раунда (рисунок 14) и его результат записывается в таблицу рекордов.

Рисунок 14 – Окончание игры при рекордном количестве очков

Для удобства игрового процесса на игровом экране в левом верхнем углу отображается лучшее количество очков (best score), которое получил игрок за сеанс игры (начиная с момента запуска) (рисунок 15).

Рисунок 15 – Отображение лучшего количества очков на сцене

Смысл игры в целом сводится к получению наибольшего количества очков, а также в расслаблении, которое чувствует игрок при использовании данного графического приложения. Удовольствие от игры достигается за счет лаконичного и приятного графического и звукового оформления.

Заключение

В ходе данной курсовой работы была проведена разработка и реализация графического приложения «Twinkle Dash», основанного на библиотеке libGDX. Работа была направлена на овладение основными возможностями данной библиотеки с целью создания игрового контента.

Итак, данная работа позволила не только успешно реализовать графическое приложение, но и обрести практические навыки, необходимые для разработки кросс-платформенных игр и приложений с использованием библиотеки libGDX. Цель работы была достигнута.

Список использованных источников

1. https://libgdx.com/wiki/start/project-generation
2. https://habr.com/ru/articles/143405/
3. https://gamedev.ru/code/forum/?id=176333
4. https://habr.com/ru/articles/143479/
5. https://www.alexkorablev.ru/2016/03/14/libgdx-fonts/
6. https://libgdx.com/wiki/graphics/2d/fonts/bitmap-fonts
7. https://libgdx.com/wiki/tools/hiero
8. https://suvitruf.ru/2012/10/31/2464/
9. https://suvitruf.ru/2013/01/25/2959/

Приложение А. Исходный код проекта

Класс DesktopLauncher

```
package ru.rsreu.verbickaya.twinkledash;
import com.badlogic.gdx.backends.lwjgl3.Lwjgl3Application;
import com.badlogic.gdx.backends.lwjgl3.Lwjgl3ApplicationConfiguration;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
public class DesktopLauncher {
    public static void main (String[] arg) {
       Lwjgl3ApplicationConfiguration config = new Lwjgl3ApplicationConfiguration();
       config.setForegroundFPS(60);
      config.setFullscreenMode(Lwjgl3ApplicationConfiguration.getDisplayMode());
       config.setResizable(false);
       config.setTitle("Twinkle Dash");
       new Lwjgl3Application(new TwinkleDash(), config);
    }
}
```

Класс TwinkleDash

```
package ru.rsreu.verbickaya.twinkledash;
import com.badlogic.gdx.*;
import ru.rsreu.verbickaya.twinkledash.screens.GreetingScreen;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
public class TwinkleDash extends Game {
    public static final int WIDTH = 1920;
    public static final int HEIGHT = 1080;
    public static final int CENTER_X = WIDTH/2;
    public static final int CENTER_Y = HEIGHT/2;
    public static final int TWINKLE_RADIUS = 30;
    public static final int GROUND_HEIGHT = 20;
    public static final int GROUND_WIDTH = WIDTH;
    public static final int WALL_HEIGHT = HEIGHT;
    public static final int WALL_WIDTH = 300;
    public static final int SPIKE_WIDTH = 88;
    public static final int SPIKE_HEIGHT = SPIKE_WIDTH;
    public static final int LEVEL_STEP = 4;
    public static final int LEVEL_COUNT = 8;
    private int bestScore = 0;
    private String userName;
    @Override
    public void create () {
       Assets.load();
       setScreen(new GreetingScreen(this));
    }
    @Override
    public void pause() {
       super.pause();
    }
    @Override
    public void dispose() {
       super.dispose();
       Assets.dispose();
    }
    public void setBestScore(int score) {
       this.bestScore = score;
    }
    public int getBestScore() {
       return this.bestScore;
    }
    public String getUserName() {
       return userName;
    }
    public void setUserName(String userName) {
       this.userName = userName;
    }
}
```

Класс GreetingScreen

```
package ru.rsreu.verbickaya.twinkledash.screens;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.ScreenAdapter;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.Image;
import com.badlogic.gdx.utils.Align;
import com.badlogic.gdx.utils.Timer;
import com.badlogic.gdx.utils.viewport.StretchViewport;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
public class GreetingScreen extends ScreenAdapter {
    private TwinkleDash game;
    private Stage stage;
    private Image greeting;
    public GreetingScreen(TwinkleDash game) {
        Assets.playGameMusic();
        this.game = game;
        stage = new Stage(new StretchViewport(TwinkleDash.WIDTH, TwinkleDash.HEIGHT));
        greeting = new Image(Assets.logo);
        greeting.setPosition(TwinkleDash.CENTER_X,TwinkleDash.CENTER_Y, Align.center);
        stage.addActor(greeting);
    }
    @Override
    public void dispose() {
        stage.dispose();
    }
    @Override
    public void show() {
        Timer.schedule(new Timer.Task() {
            @Override
            public void run() {
                game.setScreen(new NameScreen(game));
            }
        }, 2.5f);
    }
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(247/255f, 240/255f, 234/255f, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
        stage.act(delta);
        stage.draw();
    }
}
```

Класс NameScreen

```
package ru.rsreu.verbickaya.twinkledash.screens;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.ScreenAdapter;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.Label;
import com.badlogic.gdx.scenes.scene2d.ui.TextButton;
import com.badlogic.gdx.scenes.scene2d.ui.TextField;
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;
import com.badlogic.gdx.utils.Align;
import com.badlogic.gdx.utils.viewport.StretchViewport;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
public class NameScreen extends ScreenAdapter {
    private TwinkleDash game;
    private Stage stage;
    private Label title;
    private TextButton cancelButton;
    private TextButton acceptButton;
```

```
    private TextField nameField;
    public NameScreen(TwinkleDash game) {
        this.game = game;
        stage = new Stage(new StretchViewport(TwinkleDash.WIDTH, TwinkleDash.HEIGHT));
        initTitle();
        initButtons();
        initTextField();
        stage.setKeyboardFocus(nameField);
        Gdx.input.setInputProcessor(stage);
    }
    @Override
    public void dispose() {
        stage.dispose();
    }
    private void initButtons() {
        TextButton.TextButtonStyle buttonStyle1 = new TextButton.TextButtonStyle();
        buttonStyle1.font = Assets.font;
        buttonStyle1.fontColor = Color.BLACK;
        cancelButton = new TextButton("Cancel", buttonStyle1);
        TextButton.TextButtonStyle buttonStyle2 = new TextButton.TextButtonStyle();
        buttonStyle2.font = Assets.font;
        buttonStyle2.fontColor = Color.GRAY;
        acceptButton = new TextButton("OK", buttonStyle2);
        cancelButton.setPosition(TwinkleDash.CENTER_X + 500, TwinkleDash.CENTER_Y - 200,
Align.center);
        acceptButton.setPosition(TwinkleDash.CENTER_X - 500, TwinkleDash.CENTER_Y - 200,
Align.center);
        initButtonListeners();
        stage.addActor(cancelButton);
        stage.addActor(acceptButton);
    }
    private void initButtonListeners() {
        acceptButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                if (!(nameField.getText().equals(null) ||
nameField.getText().equals(""))) {
                    game.setUserName(nameField.getText());
                    game.setScreen(new MenuScreen(game));
                    Assets.stopGameMusic();
                    Assets.playButtonSound();
                }
            }
        });
        cancelButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                Gdx.app.exit();
            }
        });
    }
    private void initTitle() {
        title = new Label("Please enter your name before the playing", new
Label.LabelStyle(Assets.font, Color.BLACK));
        title.setPosition(TwinkleDash.CENTER_X, TwinkleDash.CENTER_Y + 200,
Align.center);
        stage.addActor(title);
    }
    private void initTextField() {
        TextureRegion whitePixel = Assets.white_pixel;
        TextureRegionDrawable fieldBackground = new TextureRegionDrawable(whitePixel);
        TextField.TextFieldStyle textFieldStyle = new TextField.TextFieldStyle();
        textFieldStyle.font = Assets.big_font;
        textFieldStyle.fontColor = Color.BLACK;
        textFieldStyle.background = fieldBackground;
        nameField = new TextField("", textFieldStyle);
        nameField.setAlignment(Align.center);
        nameField.setSize(TwinkleDash.WIDTH*0.6f, TwinkleDash.HEIGHT*0.15f);
```

```
        nameField.setMaxLength(10);
        nameField.setPosition(TwinkleDash.CENTER_X, TwinkleDash.CENTER_Y,Align.center);
        initTextFieldFilter();
        stage.addActor(nameField);
    }
    private void initTextFieldFilter() {
        final char[] allowedCharacters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890".toCharArray();
        nameField.setTextFieldFilter((textField, c) -> {
            for (char allowedChar : allowedCharacters) {
                if (c == allowedChar) {
                    return true;
                }
            }
            return false;
        });
    }
    @Override
    public void render(float delta) {
        Gdx.graphics.getGL20().glClearColor(247/255f, 240/255f, 234/255f, 1);
        Gdx.graphics.getGL20().glClear(GL20.GL_COLOR_BUFFER_BIT |
GL20.GL_DEPTH_BUFFER_BIT);
        updateAcceptButton();
        stage.act();
        stage.draw();

    }
     private void updateAcceptButton() {
         TextButton.TextButtonStyle buttonStyle = acceptButton.getStyle();
         if (nameField.getText().equals(null) || nameField.getText().equals(""))
             buttonStyle.fontColor = Color.GRAY;
         else buttonStyle.fontColor = Color.BLACK;
         acceptButton.setStyle(buttonStyle);
     }
}
```

Класс MenuScreen

```
package ru.rsreu.verbickaya.twinkledash.screens;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.ScreenAdapter;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.Image;
import com.badlogic.gdx.scenes.scene2d.ui.Label;
import com.badlogic.gdx.scenes.scene2d.ui.TextButton;
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
import com.badlogic.gdx.utils.Align;
import com.badlogic.gdx.utils.viewport.StretchViewport;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
public class MenuScreen extends ScreenAdapter {
    private TwinkleDash game;
    private Stage stage;
    private Label title;
    private TextButton playButton;
    private TextButton recordsButton;
    private TextButton exitButton;
    private TextButton authorButton;
    private Image background;
    public MenuScreen(TwinkleDash game) {
        Assets.playMenuMusic();
        this.game = game;
        stage = new Stage(new StretchViewport(TwinkleDash.WIDTH, TwinkleDash.HEIGHT));
        initBackground();
        initTitle();
        initButtons();
        Gdx.input.setInputProcessor(stage);
    }
```

```
    @Override
    public void dispose() {
        stage.dispose();
    }
    private void initBackground() {
        background = new Image(Assets.main_bckgrnd);
        background.setWidth(TwinkleDash.WIDTH);
        background.setHeight(TwinkleDash.WIDTH);
        background.setPosition(0,TwinkleDash.HEIGHT-TwinkleDash.WIDTH + 110);
        stage.addActor(background);
    }
    private void initTitle() {
        title = new Label("Twinkle Dash", new Label.LabelStyle(Assets.big_font,
Color.BLACK));
        title.setPosition(TwinkleDash.CENTER_X, TwinkleDash.HEIGHT * 0.85f,
Align.center);
        stage.addActor(title);
    }
    private void initButtons() {
        TextButton.TextButtonStyle buttonStyle = new TextButton.TextButtonStyle();
        buttonStyle.font = Assets.font;
        buttonStyle.fontColor = Color.BLACK;
        playButton = new TextButton("Play", buttonStyle);
        recordsButton = new TextButton("Records", buttonStyle);
        exitButton = new TextButton("Exit", buttonStyle);
        authorButton = new TextButton("About author", buttonStyle);
        playButton.setPosition(TwinkleDash.CENTER_X, TwinkleDash.HEIGHT * 0.7f,
Align.center);
        recordsButton.setPosition(TwinkleDash.CENTER_X, TwinkleDash.HEIGHT * 0.6f,
Align.center);
        exitButton.setPosition(TwinkleDash.WIDTH * 0.91f, 108.0f, Align.center);
        authorButton.setPosition(TwinkleDash.WIDTH * 0.15f, TwinkleDash.HEIGHT * 0.1f,
Align.center);
        initButtonListeners();
        stage.addActor(playButton);
        stage.addActor(recordsButton);
        stage.addActor(exitButton);
        stage.addActor(authorButton);
    }
    private void initButtonListeners() {
        playButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                Assets.stopMenuMusic();
                game.setScreen(new GameplayScreen(game));
                Assets.playButtonSound();
            }
        });
        recordsButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                game.setScreen(new RecordsScreen(game));
                Assets.playButtonSound();
            }
        });
        exitButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                Gdx.app.exit();
            }
        });
        authorButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                game.setScreen(new AuthorScreen(game));
                Assets.playButtonSound();
```

```
            }
        });
    }
    @Override
    public void render(float delta) {
        Gdx.graphics.getGL20().glClearColor(1, 1, 1, 1);
        Gdx.graphics.getGL20().glClear(GL20.GL_COLOR_BUFFER_BIT |
GL20.GL_DEPTH_BUFFER_BIT);
        stage.act();
        stage.draw();
    }
}
```

Класс AuthorScreen

```
package ru.rsreu.verbickaya.twinkledash.screens;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.ScreenAdapter;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.Image;
import com.badlogic.gdx.scenes.scene2d.ui.Label;
import com.badlogic.gdx.scenes.scene2d.ui.TextButton;
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
import com.badlogic.gdx.utils.Align;
import com.badlogic.gdx.utils.viewport.StretchViewport;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
import ru.rsreu.verbickaya.twinkledash.utils.Utils;
public class AuthorScreen extends ScreenAdapter {
    private TwinkleDash game;
    private Stage stage;
    private Label title;
    private Label text;
    private TextButton exitButton;
    private TextButton menuButton;
    private Image background;
    public AuthorScreen(TwinkleDash game) {
        this.game = game;
        stage = new Stage(new StretchViewport(TwinkleDash.WIDTH, TwinkleDash.HEIGHT));
        initBackground();
        initText();
        initTitle();
        initButtons();
        Gdx.input.setInputProcessor(stage);
    }
    @Override
    public void dispose() {
        stage.dispose();
    }
    private void initButtons() {
        TextButton.TextButtonStyle buttonStyle = new TextButton.TextButtonStyle();
        buttonStyle.font = Assets.font;
        buttonStyle.fontColor = Color.BLACK;
        exitButton = new TextButton("Exit", buttonStyle);
        menuButton = new TextButton("Menu", buttonStyle);
        exitButton.setPosition(TwinkleDash.WIDTH * 0.91f, 108.0f, Align.center);
        menuButton.setPosition(TwinkleDash.WIDTH * 0.09f, TwinkleDash.HEIGHT * 0.1f,
Align.center);
        initButtonListeners();
        stage.addActor(exitButton);
        stage.addActor(menuButton);
    }
    private void initButtonListeners() {
        menuButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                game.setScreen(new MenuScreen(game));
```

```
                Assets.playButtonSound();
            }
        });
        exitButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                Gdx.app.exit();
            }
        });
    }
    private void initTitle() {
        title = new Label("About author", new Label.LabelStyle(Assets.font,
Color.BLACK));
        title.setPosition(TwinkleDash.CENTER_X, TwinkleDash.HEIGHT * 0.85f,
Align.center);
        stage.addActor(title);
    }
    private void initText() {
        text = new Label(Utils.getAuthorText(), new Label.LabelStyle(Assets.little_font,
Color.BLACK));
        text.setWrap(true);
        text.setWidth(TwinkleDash.WIDTH * 0.5f);
        text.setPosition(TwinkleDash.CENTER_X, TwinkleDash.HEIGHT * 0.65f, Align.center);
        stage.addActor(text);
    }
    private void initBackground() {
        background = new Image(Assets.author_bckgrnd);
        background.setWidth(TwinkleDash.WIDTH);
        background.setHeight(TwinkleDash.WIDTH);
        background.setPosition(0,TwinkleDash.HEIGHT-TwinkleDash.WIDTH + 120);
        stage.addActor(background);
    }
    @Override
    public void render(float delta) {
        Gdx.graphics.getGL20().glClearColor(1, 1, 1, 1);
        Gdx.graphics.getGL20().glClear(GL20.GL_COLOR_BUFFER_BIT |
GL20.GL_DEPTH_BUFFER_BIT);
        stage.act();
        stage.draw();
    }
}
```

Класс RecordsScreen

```
package ru.rsreu.verbickaya.twinkledash.screens;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.ScreenAdapter;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.Image;
import com.badlogic.gdx.scenes.scene2d.ui.Label;
import com.badlogic.gdx.scenes.scene2d.ui.TextButton;
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
import com.badlogic.gdx.utils.Align;
import com.badlogic.gdx.utils.viewport.StretchViewport;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
import ru.rsreu.verbickaya.twinkledash.utils.RecordsProcessor;
public class RecordsScreen  extends ScreenAdapter {
    private TwinkleDash game;
    private Stage stage;
    private Label title;
    private Label text;
    private TextButton exitButton;
    private TextButton menuButton;
    private TextButton clearButton;
    private Image background;
    public RecordsScreen(TwinkleDash game) {
```

```
        this.game = game;
        stage = new Stage(new StretchViewport(TwinkleDash.WIDTH, TwinkleDash.HEIGHT));
        initBackground();
        initText();
        initTitle();
        initButtons();
        Gdx.input.setInputProcessor(stage);
    }
    @Override
    public void dispose() {
        stage.dispose();
    }
    private void initButtons() {
        TextButton.TextButtonStyle buttonStyle = new TextButton.TextButtonStyle();
        buttonStyle.font = Assets.font;
        buttonStyle.fontColor = Color.BLACK;
        exitButton = new TextButton("Exit", buttonStyle);
        menuButton = new TextButton("Menu", buttonStyle);
        clearButton = new TextButton("Clear records",buttonStyle);
        exitButton.setPosition(TwinkleDash.WIDTH * 0.91f, TwinkleDash.HEIGHT * 0.1f,
Align.center);
        menuButton.setPosition(TwinkleDash.WIDTH * 0.09f, TwinkleDash.HEIGHT * 0.1f,
Align.center);
        clearButton.setPosition(TwinkleDash.WIDTH * 0.5f, TwinkleDash.HEIGHT * 0.1f,
Align.center);
        initButtonListeners();
        stage.addActor(exitButton);
        stage.addActor(menuButton);
        stage.addActor(clearButton);
    }
    private void initButtonListeners() {
        menuButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                game.setScreen(new MenuScreen(game));
                Assets.playButtonSound();
            }
        });
        clearButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                RecordsProcessor.clearRecords();
                text.setText(RecordsProcessor.getRecordsText());
                Assets.playButtonSound();
            }
        });
        exitButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                Gdx.app.exit();
            }
        });
    }
    private void initTitle() {
        title = new Label("Records", new Label.LabelStyle(Assets.font, Color.BLACK));
        title.setPosition(TwinkleDash.CENTER_X, TwinkleDash.HEIGHT * 0.85f,
Align.center);
        stage.addActor(title);
    }
    private void initText() {
        text = new Label(RecordsProcessor.getRecordsText(), new
Label.LabelStyle(Assets.little_font, Color.BLACK));
        text.setWrap(true);
        text.setAlignment(Align.center);
        text.setWidth(TwinkleDash.WIDTH * 0.5f);
        text.setHeight(TwinkleDash.HEIGHT);
        text.setPosition(TwinkleDash.CENTER_X, TwinkleDash.CENTER_Y, Align.center);
```

```
        stage.addActor(text);
    }
    private void initBackground() {
        background = new Image(Assets.records_bckgrnd);
        background.setWidth(TwinkleDash.WIDTH);
        background.setHeight(TwinkleDash.WIDTH);
        background.setPosition(0, TwinkleDash.HEIGHT - TwinkleDash.WIDTH + 170);
        stage.addActor(background);
    }
    @Override
    public void render(float delta) {
        Gdx.graphics.getGL20().glClearColor(1, 1, 1, 1);
        Gdx.graphics.getGL20().glClear(GL20.GL_COLOR_BUFFER_BIT |
GL20.GL_DEPTH_BUFFER_BIT);
        stage.act();
        stage.draw();
    }
}
```

Класс GameplayScreen

```
package ru.rsreu.verbickaya.twinkledash.screens;
import com.badlogic.gdx.*;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.scenes.scene2d.ui.Image;
import com.badlogic.gdx.scenes.scene2d.ui.Label;
import com.badlogic.gdx.scenes.scene2d.ui.TextButton;
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;
import com.badlogic.gdx.utils.Align;
import com.badlogic.gdx.utils.viewport.StretchViewport;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.actors.controllers.SpikesController;
import ru.rsreu.verbickaya.twinkledash.actors.entities.Ground;
import ru.rsreu.verbickaya.twinkledash.actors.entities.Spike;
import ru.rsreu.verbickaya.twinkledash.actors.entities.Twinkle;
import ru.rsreu.verbickaya.twinkledash.actors.entities.Wall;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
import ru.rsreu.verbickaya.twinkledash.utils.RecordsProcessor;
import ru.rsreu.verbickaya.twinkledash.utils.Utils;
public class GameplayScreen extends ScreenAdapter {
    protected OrthographicCamera camera;
    protected TwinkleDash game;
    private Stage gameplayStage;
    private Stage stage;
    private State screenState = State.PREGAME;
    private enum State {PREGAME, PLAYING, DYING, DEAD}
    private boolean justTouched;
    private boolean allowRestart = false;
    private int score = 0;
    private int level = 0;
    private TextButton exitButton;
    private TextButton menuButton;
    private Label current_score_label;
    private Label current_score;
    private Label best_score_label;
    private Label best_score;
    private Label record_score_label;
    private Ground bottom_ground;
    private Ground upper_ground;
    private Wall right_wall;
    private Wall left_wall;
    private Image background;
    private Image whitePixel;
    private Twinkle twinkle;
```

```
    private SpikesController spikes;
    public GameplayScreen(TwinkleDash game) {
        this.game = game;
        Assets.playGameMusic();
        camera = new OrthographicCamera(TwinkleDash.WIDTH, TwinkleDash.HEIGHT);
        gameplayStage = new Stage(new StretchViewport(TwinkleDash.WIDTH,
TwinkleDash.HEIGHT, camera));
        stage = new Stage(new StretchViewport(TwinkleDash.WIDTH, TwinkleDash.HEIGHT));
        initWhitePixel();
        initBackground();
        initTwinkle();
        initBorders();
        initSpikes();
        initButtons();
        initLabels();
        initInputProcessor();
        gameplayStage.addActor(background);
        gameplayStage.addActor(twinkle);
        gameplayStage.addActor(bottom_ground);
        gameplayStage.addActor(upper_ground);
        gameplayStage.addActor(left_wall);
        gameplayStage.addActor(right_wall);
        for (Spike spike : spikes.getConstantSpikes())
            gameplayStage.addActor(spike);
        stage.addActor(exitButton);
        stage.addActor(menuButton);
        stage.addActor(current_score);
        stage.addActor(best_score);
        stage.addActor(best_score_label);
        stage.addActor(current_score_label);
    }
    private void initLabels() {
        Label.LabelStyle labelStyle = new Label.LabelStyle(Assets.little_font,
Assets.getTextColor(0));
        best_score_label = new Label("Best\nscore",labelStyle);
        best_score_label.setAlignment(Align.center);
        best_score_label.setPosition(TwinkleDash.WIDTH * 0.08f, TwinkleDash.HEIGHT *
0.8f, Align.center);
        current_score_label = new Label("Current\nscore",labelStyle);
        current_score_label.setAlignment(Align.center);
        current_score_label.setPosition(TwinkleDash.WIDTH * 0.92f, TwinkleDash.HEIGHT *
0.8f, Align.center);
        Label.LabelStyle messageLabelStyle = new Label.LabelStyle(Assets.font,
Color.BLACK);
        record_score_label = new Label(game.getUserName() + ", your record is in top
10!\nCheck the records table.",messageLabelStyle);
        record_score_label.setAlignment(Align.center);
record_score_label.setPosition(TwinkleDash.CENTER_X,TwinkleDash.CENTER_Y,Align.center);
        Label.LabelStyle numLabelStyle = new Label.LabelStyle(Assets.big_font,
Assets.getTextColor(0));
        best_score = new Label(Integer.toString(game.getBestScore()), numLabelStyle);
        best_score.setPosition(TwinkleDash.WIDTH * 0.08f, TwinkleDash.HEIGHT * 0.7f,
Align.center);
        best_score.setAlignment(Align.center);
        current_score = new Label(Integer.toString(score), numLabelStyle);
        current_score.setAlignment(Align.center);
        current_score.setPosition(TwinkleDash.WIDTH * 0.92f, TwinkleDash.HEIGHT * 0.7f,
Align.center);
    }
    private void initWhitePixel() {
        whitePixel = new Image(Assets.white_pixel);
        whitePixel.setWidth(TwinkleDash.WIDTH);
        whitePixel.setHeight(TwinkleDash.HEIGHT);
    }
    private void initTwinkle() {
        twinkle = new Twinkle();
        twinkle.setPosition(TwinkleDash.WIDTH * .25f, TwinkleDash.HEIGHT / 2,
Align.center);
        twinkle.addAction(Utils.getFloatyAction());
        twinkle.setState(Twinkle.State.PREGAME);
```

```
    }
    private void initBackground() {
        background = new Image(Assets.getBackgroundTexture(0));
        int square_side;
        if (TwinkleDash.WIDTH - TwinkleDash.WALL_WIDTH * 2 >
                TwinkleDash.HEIGHT - TwinkleDash.GROUND_HEIGHT * 2)
            square_side = TwinkleDash.WIDTH - TwinkleDash.WALL_WIDTH * 2;
        else square_side = TwinkleDash.HEIGHT - TwinkleDash.GROUND_HEIGHT * 2;
        background.setWidth(square_side);
        background.setHeight(square_side);
        background.setPosition(TwinkleDash.CENTER_X - square_side / 2,
TwinkleDash.CENTER_Y - square_side / 2);
    }
    private void initBorders() {
        bottom_ground = new Ground();
        bottom_ground.setPosition(0, 0);
        upper_ground = new Ground();
        upper_ground.setPosition(0, TwinkleDash.HEIGHT - TwinkleDash.GROUND_HEIGHT);
        left_wall = new Wall();
        left_wall.setPosition(0, 0);
        right_wall = new Wall();
        right_wall.setPosition(TwinkleDash.WIDTH - TwinkleDash.WALL_WIDTH, 0);
    }
    private void initSpikes() {
        spikes = new SpikesController();
    }
    private void initButtons() {
        TextButton.TextButtonStyle buttonStyle = new TextButton.TextButtonStyle();
        buttonStyle.font = Assets.font;
        buttonStyle.fontColor = Assets.getTextColor(0);
        exitButton = new TextButton("Exit", buttonStyle);
        menuButton = new TextButton("Menu", buttonStyle);
        exitButton.setPosition(TwinkleDash.WIDTH * 0.92f, TwinkleDash.HEIGHT * 0.1f,
Align.center);
        menuButton.setPosition(TwinkleDash.WIDTH * 0.08f, TwinkleDash.HEIGHT * 0.1f,
Align.center);
        initButtonListeners();
    }
    private void initButtonListeners() {
        menuButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                Assets.stopGameMusic();
                game.setScreen(new MenuScreen(game));
                Assets.playButtonSound();
            }
        });
        exitButton.addListener(new ClickListener() {
            @Override
            public void clicked(com.badlogic.gdx.scenes.scene2d.InputEvent event, float
x, float y) {
                Gdx.app.exit();
            }
        });
    }
    @Override
    public void render(float delta) {
        Gdx.graphics.getGL20().glClearColor(1, 1, 1, 1f);
        Gdx.graphics.getGL20().glClear(GL20.GL_COLOR_BUFFER_BIT |
GL20.GL_DEPTH_BUFFER_BIT);
        switch (screenState) {
            case PREGAME:
                updateAndDrawStages();
                break;
            case PLAYING:
                renderPlaying();
                break;
            case DYING:
            case DEAD:
```

```
            renderDeadOrDying();
            break;
    }
}
private void renderDeadOrDying() {
    if (twinkle.getState() == Twinkle.State.DEAD) {
        screenState = State.DEAD;
        allowRestart = true;
    }
    updateAndDrawStages();
}
private void showWhiteScreen() {
    gameplayStage.addActor(whitePixel);
    whitePixel.addAction(Actions.fadeOut(5));
}
private void showRecordsMessage() {
    gameplayStage.addActor(record_score_label);
    record_score_label.addAction(Actions.fadeOut(5));
}
private void renderPlaying() {
    if (justTouched) {
        twinkle.jump();
        justTouched = false;
    }
    gameplayStage.act();
    stage.act();
    controlGameplay();
    gameplayStage.draw();
    stage.draw();
}
private void updateAndDrawStages() {
    gameplayStage.act();
    gameplayStage.draw();
    stage.act();
    stage.draw();
}
@Override
public void dispose() {
    gameplayStage.dispose();
    stage.dispose();
}
private void controlGameplay() {
    for (Spike spike : spikes.getConstantSpikes()) {
        if (Utils.isCollision(spike.getBounds(), twinkle.getBounds())) {
            stopTheWorld();
        }
    }
    for (Spike spike : spikes.getDynamicSpikes()) {
        if (Utils.isCollision(spike.getBounds(), twinkle.getBounds())) {
            stopTheWorld();
        }
    }
    if (twinkle.isBehindWalls() && twinkle.getState().equals(Twinkle.State.ALIVE)) {
        score++;
        twinkle.reflect();
        for (Spike spike : spikes.getDynamicSpikes()) spike.remove();
        spikes.changeSide(score);
        for (Spike spike : spikes.getDynamicSpikes()) gameplayStage.addActor(spike);
        if (getLevel(score) != level) {
            level = getLevel(score);
            updateLevelColors(level % TwinkleDash.LEVEL_COUNT);
        }
        updateCurrentScoreLabel();
        if (score > game.getBestScore()) {
            game.setBestScore(score);
            updateBestScoreLabel();
        }
    }
}
private void updateBestScoreLabel() {
```

```
        best_score.setText(game.getBestScore());
    }
    private void updateCurrentScoreLabel() {
        current_score.setText(score);
    }
    private int getLevel(int score) {
        return score / TwinkleDash.LEVEL_STEP;
    }
    private void stopTheWorld() {
        twinkle.setToDying();
        showWhiteScreen();
        screenState = State.DYING;
        if (RecordsProcessor.isInTop(score) && score != 0 && score ==
game.getBestScore()) {
            showRecordsMessage();
            RecordsProcessor.addRecord(game.getUserName(), game.getBestScore());
        }
    }
    private void initInputProcessor() {
        InputMultiplexer multiplexer = new InputMultiplexer();
        multiplexer.addProcessor(stage);
        multiplexer.addProcessor(gameplayStage);
        multiplexer.addProcessor(new InputAdapter() {
            @Override
            public boolean touchDown(int screenX, int screenY, int pointer, int button) {
                switch (screenState) {
                    case DYING:
                        justTouched = true;
                        break;
                    case DEAD:
                        if (allowRestart) {
                            Assets.playResetSound();
                            game.setScreen(new GameplayScreen(game));
                        }
                        justTouched = true;
                        break;
                    case PLAYING:
                        justTouched = true;
                        break;
                    case PREGAME:
                        justTouched = true;
                        screenState = State.PLAYING;
                        twinkle.setState(Twinkle.State.ALIVE);
                        twinkle.clearActions();
                        //gameplayStage.addActor(twinkle);
                        break;
                }
                return true;
            }
        });
        Gdx.input.setInputProcessor(multiplexer);
    }
    private void updateLevelColors(int color_level) {
        //фон
        TextureRegion newBackgroundRegion;
        newBackgroundRegion = new
TextureRegion(Assets.getBackgroundTexture(color_level));
        background.setDrawable(new TextureRegionDrawable(newBackgroundRegion));
        // сам шарик
        twinkle.changeColor(color_level);
        // стены и пики
        right_wall.changeColor(color_level);
        left_wall.changeColor(color_level);
        bottom_ground.changeColor(color_level);
        upper_ground.changeColor(color_level);
        spikes.changeColor(color_level);
        // шрифт на кнопках
        TextButton.TextButtonStyle buttonStyle = exitButton.getStyle();
        buttonStyle.fontColor = Assets.getTextColor(color_level);
        exitButton.setStyle(buttonStyle);
```

```
        menuButton.setStyle(buttonStyle);
        // label-ы
        Label.LabelStyle numLabelStyle = new
Label.LabelStyle(Assets.big_font,Assets.getTextColor(color_level));
        best_score.setStyle(numLabelStyle);
        current_score.setStyle(numLabelStyle);
        Label.LabelStyle labelStyle = new
Label.LabelStyle(Assets.little_font,Assets.getTextColor(color_level));
        best_score_label.setStyle(labelStyle);
        current_score_label.setStyle(labelStyle);
    }
}
```

Класс Twinkle

```
package ru.rsreu.verbickaya.twinkledash.actors.entities;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.Circle;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.utils.Align;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import static java.lang.Math.abs;
public class Twinkle extends Actor {
    private static final int RADIUS = TwinkleDash.TWINKLE_RADIUS;
    private static final float GRAVITY = 900f;
    private static final float JUMP = 450f; // скорость прыжка
    private static float VELOCITY = 450f; // скорость движения влево/вправо
    private static float MAX_VELOCITY = 1000f; // предельная скорость
    private static float VELOCITY_STEP = 5f; // шаг увеличения скорости
    private float speed_y = JUMP;
    private float speed_x = VELOCITY;
    private Vector2 speed; // скорость
    private Vector2 speedUp; // ускорение
    private TextureRegion region;
    private Circle bounds; // границы (окружность) для коллизий
    private State state;
    public enum State { PREGAME, ALIVE, DYING, DEAD }
    public Twinkle() {
        region = new TextureRegion(Assets.getTwinkleTextureRegion(0));
        setWidth(RADIUS);
        setHeight(RADIUS);
        state = State.ALIVE;
        speed = new Vector2(VELOCITY, 0);
        speedUp = new Vector2(0, -GRAVITY);
        bounds = new Circle(getX() + getWidth() / 2, getY() + getHeight() / 2, RADIUS);
        setOrigin(Align.center);
    }
    public void changeColor(int level) {
        region = new TextureRegion(Assets.getTwinkleTextureRegion(level));
    }
    public void jump() {
        speed.y = speed_y;
        Assets.playJumpSound();
    }
    public void reflect() {
        speed_x *= -1;
        if (abs(speed_x) + VELOCITY_STEP <= MAX_VELOCITY) {
            if (speed_x < 0) speed_x -= VELOCITY_STEP; else speed_x += VELOCITY_STEP;
            speed_y = abs(speed_x);
            speedUp = new Vector2(0, -abs(speed_x)*2);
        }
        speed = new Vector2(speed_x, 0);
        Assets.playLevelSound();
    }
    @Override
```

```
    public void act(float delta) {
        super.act(delta);
        switch (state){
            case PREGAME:
                break;
            case ALIVE:
                actAlive(delta);
                break;
            case DEAD:
            case DYING:
                actDying(delta);
                break;
        }
        updateBounds();
    }
    private void actDying(float delta) {
        speedUp.y = -GRAVITY;
        applyAccel(delta);
        updatePosition(delta);
        controlBorders();
    }
    private void updateBounds() {
        bounds.setX(getX() + getWidth() / 2);
        bounds.setY(getY() + getHeight() / 2);
       // bounds.x = getX();
        //bounds.y = getY();
    }
    private void actAlive(float delta) {
        applyAccel(delta);
        updatePosition(delta);
        controlBorders();
    }
    private void controlBorders() {
        if (isBelowGround()) {
            setY(TwinkleDash.GROUND_HEIGHT);
            setState(State.DEAD);
        }
        if (isAboveCeiling()) {
            setY(TwinkleDash.HEIGHT - TwinkleDash.GROUND_HEIGHT - getHeight());
        }
        if (isBehindLeftWall()) {
            setX(TwinkleDash.WALL_WIDTH);
        }
        if (isBehindRightWall()) {
            setX(TwinkleDash.WIDTH - TwinkleDash.WALL_WIDTH - getWidth());
        }
    }
    public boolean isAboveCeiling() {
        return (getY(Align.top) >= TwinkleDash.HEIGHT-TwinkleDash.GROUND_HEIGHT);
    }
    public boolean isBelowGround() {
        return (getY(Align.bottom) <= TwinkleDash.GROUND_HEIGHT);
    }
    public boolean isBehindWalls() { return (getX(Align.left) <= TwinkleDash.WALL_WIDTH
|| getX(Align.right) >= TwinkleDash.WIDTH-TwinkleDash.WALL_WIDTH); }
    public boolean isBehindLeftWall() {
        return getX(Align.left) <= TwinkleDash.WALL_WIDTH;
    }
    public boolean isBehindRightWall() {
        return getX(Align.right) >= TwinkleDash.WIDTH - TwinkleDash.WALL_WIDTH;
    }
    private void updatePosition(float delta) {
        setX(getX() + speed.x * delta);
        setY(getY() + speed.y * delta);
    }
    private void applyAccel(float delta) {
        speed.add(speedUp.x * delta, speedUp.y * delta);
    }
    @Override
    public void draw(Batch batch, float parentAlpha) {
```

```
        batch.setColor(Color.WHITE);
        switch (state){
            case ALIVE:
            case PREGAME:
                drawAlive(batch);
                break;
            case DEAD:
            case DYING:
                drawDead(batch);
                break;
        }
    }
    private void drawAlive(Batch batch) {
        batch.draw(region, getX(), getY(), getOriginX(), getOriginY(), getWidth(),
getHeight(), getScaleX(),
                getScaleY(), getRotation());
    }
    private void drawDead(Batch batch) {
        batch.draw(region, getX(), getY(), getOriginX(), getOriginY(), getWidth(),
getHeight(), getScaleX(),
                getScaleY(), getRotation());
        // Тут может измениться освещение (интенсивность) ((если будет))
    }
    public void setToDying() {
        Assets.playHitSound();
        addAction(Actions.delay(.25f,Actions.run(new Runnable() {
            @Override
            public void run() {
                Assets.playDieSound();
            }
        })));
        state = State.DYING;
        speed.y = 0;
    }
    public Circle getBounds() {
        return bounds;
    }
    public void setBounds(Circle bounds) {
        this.bounds = bounds;
    }
    public State getState() {
        return state;
    }
    public void setState(State state) {
        this.state = state;
    }
}
```

Класс Wall

```
package ru.rsreu.verbickaya.twinkledash.actors.entities;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.scenes.scene2d.Actor;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
public class Wall  extends Actor {
    private static final float WIDTH = TwinkleDash.WALL_WIDTH;
    private static final float HEIGHT = TwinkleDash.WALL_HEIGHT;
    private TextureRegion region;
    public Wall() {
        region = new TextureRegion(Assets.getWallTextureRegion(0));
        setWidth(WIDTH);
        setHeight(HEIGHT);
    }
    public void changeColor(int level) {
        region = new TextureRegion(Assets.getWallTextureRegion(level));
    }
    @Override
```

```
    public void act(float delta) {
        super.act(delta);
    }
    @Override
    public void draw(Batch batch, float parentAlpha) {
        batch.setColor(Color.WHITE);
        batch.draw(region, getX(), getY(), getOriginX(), getOriginY(), getWidth(),
getHeight(), getScaleX(),
                getScaleY(), getRotation());
    }
    public TextureRegion getRegion() {
        return region;
    }
}
```

Класс Ground

```
package ru.rsreu.verbickaya.twinkledash.actors.entities;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.scenes.scene2d.Actor;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
public class Ground extends Actor {
    private static final float WIDTH = TwinkleDash.GROUND_WIDTH;
    private static final float HEIGHT = TwinkleDash.GROUND_HEIGHT;
    private TextureRegion region;
    public Ground() {
        region = new TextureRegion(Assets.getGroundTextureRegion(0));
        setWidth(WIDTH);
        setHeight(HEIGHT);
    }
    public void changeColor(int level) {
        region = new TextureRegion(Assets.getGroundTextureRegion(level));
    }
    @Override
    public void act(float delta) {
        super.act(delta);
    }
    @Override
    public void draw(Batch batch, float parentAlpha) {
        batch.setColor(Color.WHITE);
        batch.draw(region, getX(), getY(), getOriginX(), getOriginY(), getWidth(),
getHeight(), getScaleX(),
                getScaleY(), getRotation());
    }
    public TextureRegion getRegion() {
        return region;
    }
}
```

Класс Spike

```
package ru.rsreu.verbickaya.twinkledash.actors.entities;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.utils.Align;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.utils.Assets;
import com.badlogic.gdx.math.Polygon;
import static ru.rsreu.verbickaya.twinkledash.utils.Utils.getTrianglePeaks;
public class Spike extends Actor {
    private static final int WIDTH = TwinkleDash.SPIKE_WIDTH;
    private static final int HEIGHT = TwinkleDash.SPIKE_HEIGHT;
    private SpikeDirection direction;
    private TextureRegion region;
    private Polygon bounds; // границы для отслеживания коллизий
    public enum SpikeDirection {up, down, left, right};
```

```
    public Spike(SpikeDirection direction, int shift_x, int shift_y) {
        this.direction = direction;
        region = new TextureRegion(Assets.getSpikeTextureRegion(0, direction));
        float[] peaks = getTrianglePeaks(WIDTH, HEIGHT, shift_x, shift_y, direction);
        bounds = new Polygon(peaks);
        setWidth(WIDTH);
        setHeight(HEIGHT);
        setOrigin(Align.center);
    }
    public Spike(SpikeDirection direction, int shift_x, int shift_y, int level) {
        this.direction = direction;
        region = new TextureRegion(Assets.getSpikeTextureRegion(level, direction));
        float[] peaks = getTrianglePeaks(WIDTH, HEIGHT, shift_x, shift_y, direction);
        bounds = new Polygon(peaks);
        setWidth(WIDTH);
        setHeight(HEIGHT);
        setOrigin(Align.center);
    }
    public void changeColor(int level) {
        region = new TextureRegion(Assets.getSpikeTextureRegion(level, direction));
    }
    @Override
    public void act(float delta) {
        super.act(delta);
    }
    @Override
    public void draw(Batch batch, float parentAlpha) {
        batch.setColor(Color.WHITE);
        batch.draw(region, getX(), getY(), getOriginX(), getOriginY(), getWidth(),
getHeight(), getScaleX(),
                getScaleY(), getRotation());
    }
    public Polygon getBounds() {
        return bounds;
    }
    public void setBounds(Polygon bounds) {
        this.bounds = bounds;
    }
}
```

Класс SpikesController

```
package ru.rsreu.verbickaya.twinkledash.actors.controllers;
import com.badlogic.gdx.utils.Array;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.actors.entities.Spike;
import ru.rsreu.verbickaya.twinkledash.utils.Utils;
import java.util.List;
public class SpikesController {
    private Array<Spike> constant_spikes;
    private Array<Spike> dynamic_spikes;
    private boolean left_side = true;
    private int level = 0;
    public SpikesController() {
        this.constant_spikes = new Array<>();
        this.dynamic_spikes = new Array<>();
        setBottomSpikes();
        setUpperSpikes();
    }
    public void changeColor(int level) {
        this.level = level;
        for (Spike spike: constant_spikes) spike.changeColor(level);
        for (Spike spike: dynamic_spikes) spike.changeColor(level);
    }
    public void changeSide(int score) {
        int s = TwinkleDash.HEIGHT - 2 * TwinkleDash.GROUND_HEIGHT - 2 *
TwinkleDash.SPIKE_HEIGHT;
        int k = s / TwinkleDash.SPIKE_WIDTH;
        int count = score/TwinkleDash.LEVEL_STEP + 2;
        if (count > k - 1) count = k - 1;
```

```
        dynamic_spikes.clear();
        if (left_side) setLeftSpikes(k, count); else setRightSpikes(k, count);
        left_side = !left_side;
    }
    private void setRightSpikes(int general_count, int positions_count) {
        int x = TwinkleDash.WIDTH - TwinkleDash.WALL_WIDTH - TwinkleDash.SPIKE_HEIGHT;
        int y;
        List<Integer> positions =
Utils.getRandomPositions(general_count,positions_count);
        for (int i: positions) {
            y = TwinkleDash.GROUND_HEIGHT + TwinkleDash.SPIKE_HEIGHT + i *
TwinkleDash.SPIKE_WIDTH;
            Spike spike = new Spike(Spike.SpikeDirection.left, x, y, level);
            spike.setPosition(x, y);
            dynamic_spikes.add(spike);
        }
    }
    private void setLeftSpikes(int general_count, int positions_count) {
        int x = TwinkleDash.WALL_WIDTH;
        int y;
        List<Integer> positions =
Utils.getRandomPositions(general_count,positions_count);
        for (int i: positions) {
            y = TwinkleDash.GROUND_HEIGHT + TwinkleDash.SPIKE_HEIGHT + i *
TwinkleDash.SPIKE_WIDTH;
            Spike spike = new Spike(Spike.SpikeDirection.right, x, y, level);
            spike.setPosition(x, y);
            dynamic_spikes.add(spike);
        }
    }
    private void setBottomSpikes() {
        int s = TwinkleDash.WIDTH - TwinkleDash.WALL_WIDTH * 2;
        int k = s / TwinkleDash.SPIKE_WIDTH;
        for (int i = 0; i < k; i++) {
            int x = TwinkleDash.WALL_WIDTH + TwinkleDash.SPIKE_WIDTH * i;
            int y = TwinkleDash.GROUND_HEIGHT;
            Spike spike = new Spike(Spike.SpikeDirection.up, x, y, level);
            spike.setPosition(x, y);
            constant_spikes.add(spike);
        }
    }
    private void setUpperSpikes() {
        int s = TwinkleDash.WIDTH - TwinkleDash.WALL_WIDTH * 2;
        int k = s / TwinkleDash.SPIKE_WIDTH;
        for (int i = 0; i < k; i++) {
            int x = TwinkleDash.WALL_WIDTH + TwinkleDash.SPIKE_WIDTH * i;
            int y = TwinkleDash.HEIGHT-TwinkleDash.GROUND_HEIGHT-
TwinkleDash.SPIKE_HEIGHT;
            Spike spike = new Spike(Spike.SpikeDirection.down, x, y, level);
            spike.setPosition(x,y);
            constant_spikes.add(spike);
        }
    }
    public Array<Spike> getConstantSpikes() {
        return constant_spikes;
    }
    public Array<Spike> getDynamicSpikes() {
        return dynamic_spikes;
    }
}
```

Класс Assets

```
package ru.rsreu.verbickaya.twinkledash.utils;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.Pixmap;
import com.badlogic.gdx.graphics.Texture;
```

```
import com.badlogic.gdx.graphics.g2d.*;
import com.badlogic.gdx.utils.Array;
import ru.rsreu.verbickaya.twinkledash.TwinkleDash;
import ru.rsreu.verbickaya.twinkledash.actors.entities.Spike;
import static ru.rsreu.verbickaya.twinkledash.utils.Utils.getTrianglePeaks;
public class Assets {
    private static final int RADIUS = TwinkleDash.TWINKLE_RADIUS;
    private static final int GROUND_WIDTH = TwinkleDash.GROUND_WIDTH;
    private static final int GROUND_HEIGHT = TwinkleDash.GROUND_HEIGHT;
    private static final int WALL_WIDTH = TwinkleDash.WALL_WIDTH;
    private static final int WALL_HEIGHT = TwinkleDash.WALL_HEIGHT;
    private static final int SPIKE_HEIGHT = TwinkleDash.SPIKE_HEIGHT;
    private static final int SPIKE_WIDTH = TwinkleDash.SPIKE_WIDTH;
    private static SpriteBatch batch;
    private static Sound hit_sound;
    private static Sound jump_sound;
    private static Sound die_sound;
    private static Sound newLevel_sound;
    private static Sound buttonClick_sound;
    private static Sound reset_sound;
    private static Music menu_music;
    private static Music game_music;
    public static TextureRegion white_pixel;
    public static Texture main_bckgrnd;
    public static Texture records_bckgrnd;
    public static Texture author_bckgrnd;
    public static Texture logo;
    private static Array<Texture> game_level_backgrounds;
    private static Array<Color> game_level_twinkleColors;
    private static Array<Color> game_level_bordersAndSpikesColors;
    private static Array<Color> game_level_textColors;
    public static BitmapFont big_font;
    public static BitmapFont font;
    public static BitmapFont little_font;
    public static final String RECORD_PATH = "text/records.txt";
    public static final String AUTHOR_PATH = "text/author.txt";
    public static void playButtonSound() { buttonClick_sound.play(); }
    public static void playHitSound(){
        hit_sound.play();
    }
    public static void playDieSound(){
        die_sound.play();
    }
    public static void playJumpSound() {
        jump_sound.play();
    }
    public static void playLevelSound() {
        newLevel_sound.play();
    }
    public static void playResetSound() { reset_sound.play(); }
    public static void playMenuMusic() { menu_music.setLooping(true); menu_music.play();
}
    public static void stopMenuMusic() { menu_music.stop(); }
    public static void playGameMusic() { game_music.setLooping(true); game_music.play();
}
    public static void stopGameMusic() { game_music.stop(); }
    public static TextureRegion getTwinkleTextureRegion(int level) {
        int i;
        if (level < game_level_twinkleColors.size) i = level;
        else i = game_level_twinkleColors.size - 1;
        return createCircleTexture(game_level_twinkleColors.get(i));
    }
    public static TextureRegion getGroundTextureRegion(int level) {
        int i;
        if (level < game_level_bordersAndSpikesColors.size) i = level;
        else i = game_level_bordersAndSpikesColors.size - 1;
        return createRectangleTexture(game_level_bordersAndSpikesColors.get(i),
GROUND_WIDTH, GROUND_HEIGHT);
    }
    public static TextureRegion getWallTextureRegion(int level) {
```

```
        int i;
        if (level < game_level_bordersAndSpikesColors.size) i = level;
        else i = game_level_bordersAndSpikesColors.size - 1;
        return createRectangleTexture(game_level_bordersAndSpikesColors.get(i),
WALL_WIDTH, WALL_HEIGHT);
    }
    public static TextureRegion getSpikeTextureRegion(int level, Spike.SpikeDirection
direction) {
        int i;
        if (level < game_level_bordersAndSpikesColors.size) i = level;
        else i = game_level_bordersAndSpikesColors.size - 1;
        return createTriangleTexture(game_level_bordersAndSpikesColors.get(i),
SPIKE_WIDTH, SPIKE_HEIGHT, direction);
    }
    public static Texture getBackgroundTexture(int level) {
        int i;
        if (level < game_level_backgrounds.size) i = level;
        else i = game_level_backgrounds.size - 1;
        return game_level_backgrounds.get(i);
    }
    public static Color getTextColor(int level) {
        int i;
        if (level < game_level_textColors.size) i = level;
        else i = game_level_textColors.size - 1;
        return game_level_textColors.get(i);
    }
    public static void load() {
        batch = new SpriteBatch();
        initFonts();
        initSoundsAndMusic();
        initBackgrounds();
    }
    private static void initFonts() {
        big_font = new BitmapFont(Gdx.files.internal("fonts/big_font.fnt"));
        font = new BitmapFont(Gdx.files.internal("fonts/font.fnt"));
        little_font = new BitmapFont(Gdx.files.internal("fonts/little_font.fnt"));
    }
    private static void initSoundsAndMusic() {
        die_sound = Gdx.audio.newSound(Gdx.files.internal("sounds/die.mp3"));
        hit_sound = Gdx.audio.newSound(Gdx.files.internal("sounds/hit.wav"));
        jump_sound = Gdx.audio.newSound(Gdx.files.internal("sounds/jump.wav"));
        newLevel_sound = Gdx.audio.newSound(Gdx.files.internal("sounds/level.wav"));
        buttonClick_sound = Gdx.audio.newSound(Gdx.files.internal("sounds/button.wav"));
        reset_sound = Gdx.audio.newSound(Gdx.files.internal("sounds/reset.mp3"));
        game_music = Gdx.audio.newMusic(Gdx.files.internal("music/game.mp3"));
        menu_music = Gdx.audio.newMusic(Gdx.files.internal("music/menu.mp3"));
    }
    private static void initBackgrounds() {
        main_bckgrnd = new Texture(Gdx.files.internal("backgrounds/menu.jpg"));
        author_bckgrnd = new Texture(Gdx.files.internal("backgrounds/author.jpg"));
        records_bckgrnd = new Texture(Gdx.files.internal("backgrounds/records.jpg"));
        logo = new Texture(Gdx.files.internal("backgrounds/logo.jpg"));
        initLevelColors();
    }
    private static void initLevelColors() {
        white_pixel = createRectangleTexture(Color.WHITE, WALL_HEIGHT, GROUND_WIDTH);
        game_level_backgrounds = new Array<Texture>();
        game_level_twinkleColors = new Array<Color>();
        game_level_bordersAndSpikesColors = new Array<Color>();
        game_level_textColors = new Array<Color>();
        String path = "backgrounds/game_levels/";
        // уровень 0
        game_level_backgrounds.add(new Texture(Gdx.files.internal(path +
"light/white_gray.jpg")));
        //game_level_twinkleColors.add(new Color(230/255f,182/255f,170/255f,1)); // ОЧЕНЬ
КРАСИВЫЙ ЦВЕТ
        game_level_twinkleColors.add(new Color(64/255f,69/255f,75/255f,1));
        game_level_bordersAndSpikesColors.add(new Color(104/255f,109/255f,115/255f,1));
        game_level_textColors.add(new Color(230/255f,230/255f,232/255f,1));
        // уровень 1
```

```
        game_level_backgrounds.add(new Texture(Gdx.files.internal(path +
"light/blue_pink.jpg")));
        game_level_twinkleColors.add(new Color(83/255f,118/255f,130/255f,1));
        game_level_bordersAndSpikesColors.add(new Color(110/255f,156/255f,172/255f,1));
        game_level_textColors.add(new Color(218/255f,234/255f,237/255f,1));
        // уровень 2
        game_level_backgrounds.add(new Texture(Gdx.files.internal(path +
"dark/black_white.jpg")));
        game_level_twinkleColors.add(new Color(1,1,1,1));
        game_level_bordersAndSpikesColors.add(new Color(94/255f,105/255f,
0.45882353f,1));
        game_level_textColors.add(new Color(1,1,1,1));
        // уровень 3
        game_level_backgrounds.add(new Texture(Gdx.files.internal(path +
"light/yellow_pink.jpg")));
        game_level_twinkleColors.add(new Color(117/255f,112/255f,94/255f,1));
        game_level_bordersAndSpikesColors.add(new Color(255/255f,105/255f,86/255f,1));
        game_level_textColors.add(new Color(255/255f,240/255f,212/255f,1));
        // уровень 4
        game_level_backgrounds.add(new Texture(Gdx.files.internal(path +
"dark/blue_pink.jpg")));
        game_level_twinkleColors.add(new Color(245/255f,161/255f,150/255f,1));
        game_level_bordersAndSpikesColors.add(new Color(250/255f,133/255f,139/255f,1));
        game_level_textColors.add(new Color(14/255f,42/255f,54/255f,1));
        // уровень 5
        game_level_backgrounds.add(new Texture(Gdx.files.internal(path +
"light/pink_blue.jpg")));
        game_level_twinkleColors.add(new Color(14/255f,25/255f,33/255f,1));
        game_level_bordersAndSpikesColors.add(new Color(35/255f,62/255f,82/255f,1));
        game_level_textColors.add(new Color(243/255f,166/255f,158/255f,1));
        // уровень 6
        game_level_backgrounds.add(new Texture(Gdx.files.internal(path +
"dark/green_yellow.jpg")));
        game_level_twinkleColors.add(new Color(255/255f,242/255f,170/255f,1));
        game_level_bordersAndSpikesColors.add(new Color(90/255f,125/255f,93/255f,1));
        game_level_textColors.add(new Color(15/255f,23/255f,20/255f,1));
        // уровень 7
        game_level_backgrounds.add(new Texture(Gdx.files.internal(path +
"dark/red_blue.jpg")));
        game_level_twinkleColors.add(new Color(95/255f,235/255f,222/255f,1));
        game_level_bordersAndSpikesColors.add(new Color(247/255f,31/255f,34/255f,1));
        game_level_textColors.add(new Color(72/255f,1/255f,31/255f,1));
    }
    private static TextureRegion createCircleTexture(Color color) {
        Pixmap pixmap = new Pixmap(RADIUS * 2, RADIUS * 2, Pixmap.Format.RGBA8888);
        pixmap.setColor(color);
        pixmap.fillCircle(RADIUS, RADIUS, RADIUS);
        Texture texture = new Texture(pixmap);
        TextureRegion textureRegion = new TextureRegion(texture);
        pixmap.dispose();
        return textureRegion;
    }
    private static TextureRegion createRectangleTexture(Color color, int width, int
height) {
        Pixmap pixmap = new Pixmap(width, height, Pixmap.Format.RGBA8888);
        pixmap.setColor(color);
        pixmap.fillRectangle(0, 0, width, height);
        Texture texture = new Texture(pixmap);
        TextureRegion textureRegion = new TextureRegion(texture);
        pixmap.dispose();
        return textureRegion;
    }
    private static TextureRegion createTriangleTexture(Color color, int width, int
height, Spike.SpikeDirection direction) {
        Pixmap pixmap = new Pixmap(width, height, Pixmap.Format.RGBA8888);
        pixmap.setColor(color);
        float[] peaks = getTrianglePeaks(width, height, direction);
        pixmap.fillTriangle((int) peaks[0],(int) peaks[1],(int) peaks[2],
                (int) peaks[3],(int) peaks[4],(int) peaks[5]);
        Texture texture = new Texture(pixmap);
```

```
        TextureRegion textureRegion = new TextureRegion(texture);
        pixmap.dispose();
        return textureRegion;
    }
    public static void dispose(){
        if (batch != null) {
            batch.dispose();
        }
        hit_sound.dispose();
        jump_sound.dispose();
        die_sound.dispose();
        newLevel_sound.dispose();
        buttonClick_sound.dispose();
        reset_sound.dispose();
        menu_music.dispose();
        game_music.dispose();
        main_bckgrnd.dispose();
        author_bckgrnd.dispose();
        records_bckgrnd.dispose();
        logo.dispose();
        for (Texture texture: game_level_backgrounds) {
            texture.dispose();
        }
        big_font.dispose();
        font.dispose();
        little_font.dispose();
    }
}
```

Класс RecordsProcessor

```
package ru.rsreu.verbickaya.twinkledash.utils;
import java.io.*;
import java.util.*;
public class RecordsProcessor {
    private static final String PATH = Assets.RECORD_PATH;
    public static String getRecordsText() {
        StringBuilder text = new StringBuilder("Top 10 records:" + "\n" + "\n");
        List<String> records = getTopRecords();
        int i = 1;
        for (String record: records) {
            text.append(i);
            text.append(". " );
            text.append(record);
            text.append("\n");
            i++;
        }
        return text.toString();
    }
    private static List<String> getTopRecords() {
        Map<String, Integer> recordsMap = new HashMap<>();
        try {
            File file = new File(PATH);
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] parts = line.split("\\s+");
                if (parts.length == 2) {
                    String name = parts[0];
                    int score = Integer.parseInt(parts[1]);
                    recordsMap.put(name, score);
                }
            }
            scanner.close();
        } catch (FileNotFoundException e) { }
        List<Map.Entry<String, Integer>> sortedRecords = new
ArrayList<>(recordsMap.entrySet());
sortedRecords.sort(Map.Entry.comparingByValue(Comparator.reverseOrder()));
        List<String> topRecords = new ArrayList<>();
        int count = 0;
```

```
        for (Map.Entry<String, Integer> entry : sortedRecords) {
            topRecords.add(entry.getKey() + " " + entry.getValue());
            count++;
            if (count >= 10) {
                break;
            }
        }
        return topRecords;
    }
    public static void addRecord(String name, int record) {
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(PATH, true));
            writer.write(name + " " + record + "\n");
            writer.close();
        } catch (IOException e) {}
    }
    public static boolean isInTop(int record) {
        List<Integer> topRecords = new ArrayList<>();
        try {
            File file = new File(PATH);
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] parts = line.split("\\s+");
                if (parts.length == 2) {
                    int score = Integer.parseInt(parts[1]);
                    topRecords.add(score);
                }
            }
            scanner.close();
        } catch (FileNotFoundException e) {}
        topRecords.sort(Comparator.reverseOrder());
        return topRecords.size() < 10 || record > topRecords.get(9);
    }
    public static void clearRecords() {
        try {
            FileWriter fileWriter = new FileWriter(PATH, false);
            fileWriter.close();
        } catch (IOException e) { };
    }
}
```

Класс Utils

```
package ru.rsreu.verbickaya.twinkledash.utils;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.files.FileHandle;
import com.badlogic.gdx.math.*;
import com.badlogic.gdx.scenes.scene2d.Action;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.scenes.scene2d.actions.MoveByAction;
import com.badlogic.gdx.scenes.scene2d.actions.SequenceAction;
import ru.rsreu.verbickaya.twinkledash.actors.entities.Spike;
import java.util.*;
public class Utils {
    // анимация "плавания" вверх-вниз
    public static Action getFloatyAction() {
        MoveByAction a1 = Actions.moveBy(0, 20f, 1f, Interpolation.sine);
        MoveByAction a2 = Actions.moveBy(0, -20f, 1f, Interpolation.sine);
        SequenceAction sa = Actions.sequence(a1, a2);
        return Actions.forever(sa);
    }
    // определение координат вершин треугольника-spike по его ширине, высоте и по
направлению
    public static float[] getTrianglePeaks(int w, int h, Spike.SpikeDirection direction)
{
        float x1, x2, x3, y1, y2, y3;
        x1 = x2 = x3 = y1 = y2 = y3 = 0;
        switch (direction) {
            case down:
```

```
                x2 = w;
                x3 = w / 2;
                y3 = h;
                break;
            case up:
                y1 = h;
                x2 = w;
                y2 = h;
                x3 = w / 2;
                break;
            case left:
                x1 = w;
                y1 = h;
                x2 = w;
                y3 = h / 2;
                break;
            case right:
                y1 = h;
                x3 = w;
                y3 = h / 2;
                break;
        }
        float[] peaks = new float[]{x1, y1, x2, y2, x3, y3};
        return peaks;
    }
    // определение координат вершин треугольника-spike по его ширине, высоте
    // + сдвиг по осям, и по направлению
    public static float[] getTrianglePeaks(int w, int h, int x, int y,
Spike.SpikeDirection direction) {
        float x1, x2, x3, y1, y2, y3;
        x1 = x2 = x3 = y1 = y2 = y3 = 0;
        switch (direction) {
            case down:
                x2 = w;
                x3 = w / 2;
                y3 = h;
                break;
            case up:
                y1 = h;
                x2 = w;
                y2 = h;
                x3 = w / 2;
                break;
            case left:
                x1 = w;
                y1 = h;
                x2 = w;
                y3 = h / 2;
                break;
            case right:
                y1 = h;
                x3 = w;
                y3 = h / 2;
                break;
        }
        float[] peaks = new float[]{x1 + x, y1 + y, x2 + x, y2 + y, x3 + x, y3 + y};
        return peaks;
    }
    // проверка пересечения полигона с окружностью
    public static boolean isCollision(Polygon p, Circle c) {
        float[] vertices = p.getTransformedVertices();
        Vector2 center = new Vector2(c.x, c.y);
        float squareRadius = c.radius * c.radius;
        for (int i = 0; i < vertices.length; i += 2) {
            if (i == 0) {
                if (Intersector.intersectSegmentCircle(new Vector2(
                        vertices[vertices.length - 2],
                        vertices[vertices.length - 1]), new Vector2(
                        vertices[i], vertices[i + 1]), center, squareRadius))
                    return true;
```

```
            } else {
                if (Intersector.intersectSegmentCircle(new Vector2(
                        vertices[i - 2], vertices[i - 1]), new Vector2(
                        vertices[i], vertices[i + 1]), center, squareRadius))
                    return true;
            }
        }
        return false;
    }
    // в пределах до general_count определяет count случайных позиций spikes
    public static List<Integer> getRandomPositions(int general_count, int count) {
        List<Integer> positions = new ArrayList<Integer>();
        Random rand = new Random();
        while (positions.size() < count) {
            int position = rand.nextInt(general_count);
            if (!positions.contains(position)) {
                positions.add(position);
            }
        }
        return positions;
    }
    //чтение текста "Об авторе" из файла
    public static String getAuthorText() {
        FileHandle fileHandle = Gdx.files.internal(Assets.AUTHOR_PATH);
        return fileHandle.readString();
    }
}
```