Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему:

«ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АРКАДНОГО 2D - ШУТЕРА»

БГУИР КП 1-40 01 01 024 ПЗ

Студент: гр. 651005 Стаселович И.А.

Руководитель: Марина И.М.

Минск 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире у среднестатистического работающего человека остается очень мало времени на отдых и личное время. Каждый тщательно подходит к выбору занятия в долгожданные минуты отдыха. Кто-то посвящает себя различным хобби, кто-то занимается спортом, а кто-то играет в компьютерные игры. Благодаря вариативности игр, любителей расслабится у компьютера за игрой сегодня сотни миллионов. А что же было 30-40 лет назад, когда обычные люди только знакомились с персональными компьютерами. Тогда были популярны игровые автоматы, находящиеся в людных местах, любой подошедший мог сыграть в приглянувшуюся игру. Я бы хотел отметить такой жанр компьютерных игр как аркада. А, в частности - аркадные 2D - шутеры. Они являются одними из самых популярных видеоигр за все время их существования. Простые в освоении, создаваемые в начале 70х годов, они заставляли снова и снова игроков бросать свои монетки в автомат. Не удивительно, что период времени с конца 70х по середину 80х называют золотым веком аркадных игр!

Данная записка содержит следующие разделы курсовой работы по разработке программного средства, реализующего вышеупомянутую игру:

1) Анализ прототипов аркадных игр и формирование требований к проектируемому программному средству;
2) Разработка алгоритма на основе сформулированных в результате анализа требований для программы;
3) Разработка программного средства. Выбор структур данных для использования.
4) Обоснование технических приемов программирования;
5) Руководство пользователя. Включает в себя описание действий, которые позволят успешно использовать приложение.

1 АНАЛИЗ ПРОТОТИПОВ АРКАДНЫХ ИГР И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

1.1 История создания и прототипы

Аркада (arcade) – игровой термин или жанр, обозначающий игры с простым или примитивным игровым процессом. Аркадные игры берут свою историю от игровых автоматов, которые располагались в публичных местах и давали поиграть всем желающим. По сути, аркады были первыми электронными играми, поскольку уровень развития техники не позволял создать что-либо большее.

Истоки аркадных шутеров, а также Shoot 'em up (с англ. — «перестреляй их всех») игр можно отследить вплоть до игры Spacewar! (рис. 1.1), одной из самых первых компьютерных игр, разработанной в 1962 году и со временем распространившейся в залах игровых автоматов в первой половине 1970-х годов.

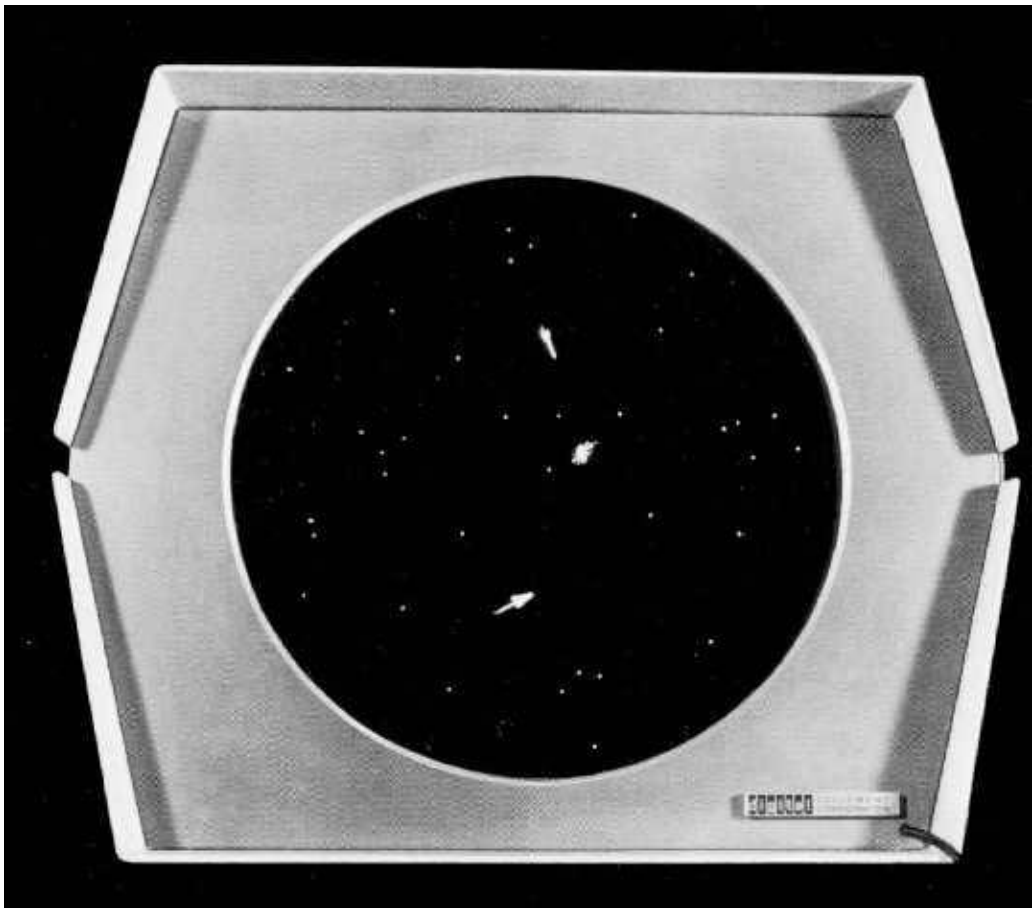


Рисунок 1.1 – “Spacewar!” 1962 от компании «Hingham Institute».

В течение 1970-х годов благодаря играм типа Space Invaders (рис 1.2) и Asteroids (рис 1.3) популярность жанра только нарастала.

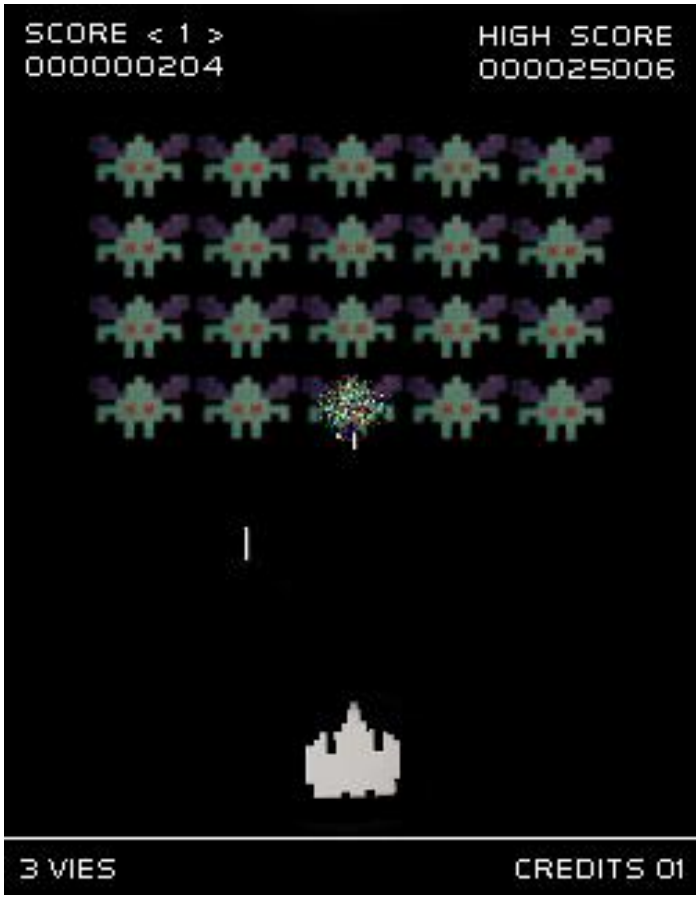


Рисунок 1.2 – “Space Invaders” 1978 от корпорации «Taito».

Рисунок 1.3 – “Asteroids” 1979 от компании «Atari».

1.2 Характерные особенности жанра

- *Игра на одном экране*

В классических аркадах весь игровой процесс сосредоточен на одном экране. Прежде всего это обусловлено исторически, что произошло из-за технических ограничений, но в то же время это значительно влияло на геймдизайн. Так, игроки в любой момент времени могли видеть весь игровой мир и принимать решения, исходя из полной информации о его состоянии.

- *Бесконечная игра*

Потенциально игроки могут играть в аркаду бесконечное время, и соответственно, не могут выиграть. Это влияло на то, что игроки делали вызов сами себе — насколько долго они смогут продержаться. Относительно геймдизайна в аркадах игрок никогда не выигрывал, и каждая игра заканчивалась поражением.

- *Множество жизней*

Обычно, классическая аркада предлагает игроку несколько попыток (жизней). Такой подход позволяет новичкам получить большую возможность изучить игровые механики до того, как игра заканчивается.

- *Игровой счёт*

Практически все классические аркады включают в себя игровой счёт, когда игрок получает очки за выполнение различных целей или задач, например, поражение противника снарядом. Здесь имеется другой важный концепт аркад, унаследованный от игр пинбола: очки позволяют игроку понять, насколько хорошо он играл, и несмотря на то, что выиграть невозможно.

- *Быстрое обучение, простой игровой процесс*

Для классических аркад характерно то, что игрокам легко научиться геймплею. Вместе с тем, если игрок погибает в аркаде, то это практически всегда происходит по его вине. В таких играх нет «специальных комбинаций клавиш», которые игрок должен выучить из документации для того, чтобы сделать что-то особенное. В то же время, простой игровой процесс не подразумевает что он «плохой» или «ограниченный», — он может быть «элегантным» и «отполированным».

- *Нет сюжета/истории*

Классические аркады практически всегда избегали попыток рассказать какую-либо историю, и данная тенденция продолжается для современных аркад. Играм жанра всегда требовалось, чтобы игроки быстро поняли, что происходит — это научная фантастика, война, спорт или что-то ещё. Геймдизайнеры классических аркадных игр не чувствовали, что им нужно наполнять свои миры чем-то и отдельно объяснять игрокам почему они должны стрелять в те или иные цели различной формы.

1.3 Формирование требований к проектируемому программному средству

Подробно изучив некоторое количество программных средств, реализующих Shoot 'em up игры, я пришел к некоторым выводам, что из себя должна представлять данная программа в моем понимании:

1. Использование основных принципов оригинальных аркадных игр, то есть:
   - бесконечный игровой процесс;
   - игра на одном экране;
   - отсутствие сюжета;
   - простое управление;
   - множество жизней;
   - игровой счет;
2. Простой интуитивно понятный и приятный интерфейс;
3. Возможность сохранения счета игры определенного игрока с его именем;
4. Просмотр таблицы всех сохраненных счетов игроков.

2 РАЗРАБОТКА АЛГОРИТМА

2.1 Анализ требований к программному средству и разработка функциональных требований

В результате анализа требований к программному средству были составлены следующие функциональные требования:

1. Наличие главного меню, из которого игрок может начать игровой процесс, либо открыть таблицу сохраненных счетов;
2. Наличие персонажа, которым управляет игрок, в данном случае - звездолет;
3. Реализация движения звездолета (по горизонтали);
4. Реализация полета астероидов (по вертикали) в сторону звездолета;
5. Реализация генерирования случайных начальных координат и скорости движения астероидов;
6. Реализация генерирования случайного количества астероидов от 1 до 5 за единицу времени;
7. Реализация выстрелов звездолета;
8. Реализация столкновений снарядов и астероидов;
9. Реализация разрушения астероидов после попадания в них снарядов;
10. Реализация начисления определенного количества очков за каждый разрушенный астероид;
11. Наличие индикатора жизней звездолета;
12. Наличие индикатора очков игрока;
13. Реализация столкновений звездолета и астероидов;
14. Реализация потери жизней звездолета после столкновения с астероидом;
15. Реализация проигрыша (конца игры) после потери определенного количества здоровья;
16. Возможность сохранения набранных очков в таблицу счетов игроков;
17. Возможность введения игроком своего уникального имени;
18. Реализация записи уникального имени и счета игрока в файл для дальнейшего вывода в таблицу;
19. Возможность возврата в главное меню игры после поражения;
20. Реализация обнуления счета и очищения игрового поля после поражения для успешного запуска новой игры.

2.2 Разработка алгоритма программного средства

Укрупненная схема программы представлена на рисунке 2.1.

Рисунок 2.1 – Укрупнённая схема программы

2.3 Разработка алгоритма игрового процесса

При запуске приложения открывается главное меню, при нажатии на кнопку «start» пользователь попадает в игру. Происходит переход приложения из состояния MENU в состояние IN_GAME. Соответственно, происходит очищение компонентов главного меню и создание звездолета, индикатора жизней и счета игрока. С помощью внутреннего двигателя, который производит обновление данных игры и их отрисовку на экран 60 раз в 1 секунду, происходит создание астероидов над игровым полем и реализация их падения вниз, также реализуется движение звездолета вправо и влево, исходя из нажатий игрока на клавиши «влево» и «вправо». В добавок, реализуются выстрелы снарядов из звездолета, при нажатии на «пробел», их движение вверх, удаление подстреленных астероидов и попавших в них снарядов, а также начисление игроку 100 очков счета за каждый разрушенных астероид и отрисовка нового счета игрока на экран. При столкновении звездолета и астероида, происходит отнятие 1/3 здоровья у персонажа, а также отрисовывается новый укороченный индикатор здоровья. После того, как у звездолета не останется здоровья произойдет конец игры и переход приложения из состояния IN_GAME в состояние DEATH_MENU.

2.4 Разработка алгоритма сохранения результатов игроков

После проигрыша, игроку представляется возможность ввести свое имя в новом окне, которое открывается после нажатия на кнопку «save», и записать свои очки вместе с этим именем в файл счетов всех игроков. Находясь в главном меню, пользователь может открыть таблицу очков нажав на кнопку «scores». После этого произойдет открытие нового окна с таблицей, на которую будут выведены все сохраненные данные игроков.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка используемых данных

Одной из основных задач данного программного средства является сохранение и беспрепятственный доступ к результатам игроков, было решено использовать текстовый файл «scores.txt» для выполнения данной функции.

Для создания формы с необходимым функционалом используется библиотека Swing и компонент JFrame.

Для реализации состояний приложения используется перечисляемый тип GameState, структура которого представлена на рисунке 3.1

```
public enum GameState {
    MENU,
    IN_GAME,
    DEATH_MENU
}
```

Рисунок 3.1 – Структура перечисляемого типа GameState

Для создания дружелюбного и приятного пользовательского интерфейса было решено использовать различного вида спрайты кнопок, звездолета, астероидов, снарядов, фона и логотипа игры. Для реализации классов спрайтов используется конструкция interface, представленная на рисунке 3.2

```
public interface Sprite {
    void render(Graphics2D graphics2D);

    int getX();

    int getY();

    int getWidth();

    int getHeight();

    void erase();

}
```

Рисунок 3.2 – Структура интерфейса Sprite

Также, мною было решено добавить фоновую музыку в приложение. Для этого я использовал библиотеку JLayer.

Хранение всех используемых сторонних ресурсов, то есть, спрайтов, музыки, файла с данными игроков, происходит в папке res данного проекта.

3.2 Разработка схемы работы приложения

При запуске приложения пользователь видит только главное меню. В главном меню игроку доступно три кнопки: «start», «scores» и «exit». При нажатии на кнопку «exit» произойдет выход из приложения. При нажатии на кнопку «scores» произойдет открытие дополнительного диалогового окна с таблицей всех сохранённых счетов игроков, которые загружаются в приложение из файла. При нажатии на кнопку «start» произойдет изменение интерфейса и начнется игровой процесс. Пользователю будут доступны некоторые возможности и функции. Игрок сможет управлять движением звездолета и стрелять, используя клавиатуру. При нажатии на клавишу «esc» произойдет выход в главное меню. После поражения пользователь будет находится в так называемом меню «смерти», в которое будет выведен конечный счет игрока. Также в этом меню игроку доступны две кнопки: «save» и «menu». После нажатия на кнопку «save» произойдет открытие дополнительного диалогового окна с компонентом для ввода имени игрока и кнопкой «ок». После ввода имени и нажатия на кнопку «ок» произойдет сохранение данных игрока (его счет за прошлую игру и введенное им имя) в файл, и диалоговое окно пропадет. После нажатия на кнопку «menu» пользователь попадет в главное меню и при повторном нажатии на кнопку «scores» будет выводиться новая таблица счетов игроков с новыми обновленными данными, а при нажатии на кнопку «start» будет запускаться новая игра.

4 ОБОСНОВАНИЕ ТЕХНИЧЕСКИХ ПРИЕМОВ ПРОГРАММИРОВАНИЯ

Программа включает в себя 16 пакетов (рис 4.1):

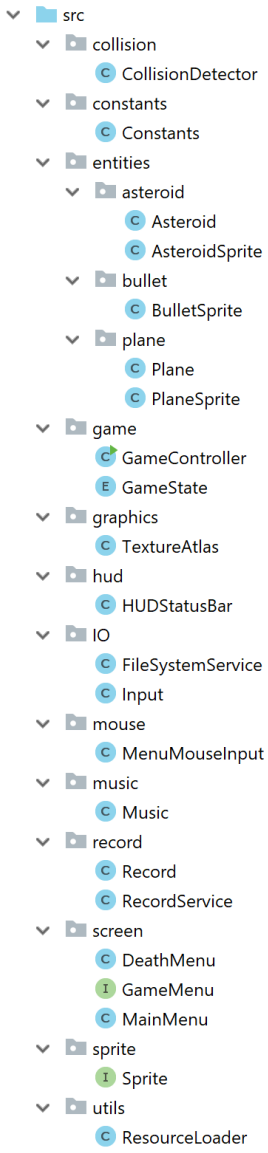


Рисунок 4.1 – Структура проекта

– collision package

В данном пакете находится класс CollisionDetector, в котором реализован метод checkCollision для проверки столкновений звездолета и снарядов с астероидами.

– constants package

В данном пакете находится класс Constants, который содержит в себе необходимые константные значения для всей программы.

– entities.asteroid package

В данном пакете находятся классы Asteroid и AsteroidSprite. Класс Asteroid расширяет класс AsteroidSprite, а AsteroidSprite, в свою очередь, реализует интерфейс Sprite из пакета sprites package. Их основная задача – реализация астероидов, их координат, размеров, скорости, отрисовки, удаления, а также передача их урона и очков за их разрушение.

– entities.bullet package

В данном пакете находится класс BulletSprite, который реализует снаряды, их координаты, размеры, скорость, удаление и отрисовку.

– entities.plane package

В данном пакете находятся классы Plane и PlaneSprite. Класс Plane расширяет класс PlaneSprite, а PlaneSprite, в свою очередь, реализует интерфейс Sprite из пакета sprites package. Их основная задача – реализация звездолета, его начальных координат, движения, размеров, скорости, отрисовки, получения урона, выстрелов, а также передача текущего здоровья звездолета.

– game package

В данном пакете находятся класс GameController и структура перечисляемого типа GameState. Класс GameController – основной класс программы, в котором находится запускаемый main метод проекта. Реализует интерфейс Runnable для реализации подпроцесса(потока) обновления экрана программы.

– graphics package

В данном пакете находится класс TextureAtlas, который отвечает за загрузку спрайтов и вырезание из них текстур по определённым координатам.

– hud package

В данном пакете находится класс HUDStatusBar, который отвечает изменение и отрисовку очков игрока и здоровья звездолета.

– IO package

В данном пакете находятся классы FileSystemService и Input. Класс FileSystemService отвечает за запись рекордов в текстовый файл. Класс Input расширяет класс JComponent и считывает нажатия всех клавиш на клавиатуре в приложении.

– mouse package

В данном пакете находится класс MenuMouseInput, который отвечает за нажатия кнопки во всех состояниях приложения.

– music package

В данном пакете находится класс Music, который реализует интерфейс Runnable для создания подпроцесса(потока) воспроизведения аудиофайла для фоновой музыки.

– record package

В данном пакете находятся классы Record и RecordService, которые реализуют считывание рекордов из файла и занесение их в определенном виде в массив строк.

– screen package

В данном пакете находятся классы DeathMenu, MainMenu и интерфейс GameMenu. Данные два класса реализуют интерфейс GameMenu, который описывает их основные методы. DeathMenu отвечает за отрисовку меню «смерти», за нажатия на кнопки и за создание диалогового окна для ввода имени игрока. MainMenu отвечает за отрисовку главного меню, за нажатия на кнопки и за создание диалогового окна для вывод таблицы счетов.

– sprite package

В данном пакете находится интерфейс Sprite, описывающий все методы, которые должны иметь классы, реализующий данный интерфейс.

– utils package

В данном пакете находится класс ResourceLoader, который реализует загрузку ресурсов в программу из папки «res».

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для того чтобы начать работу с приложением обязательно наличие всех перечисленных в 5 пункте пакетов с классами, используемых библиотек, а также папки «res» со всеми нужными ресурсами. После запуска приложения пользователь оказывается в главном меню (рис 5.1), и начинает играть фоновая музыка.

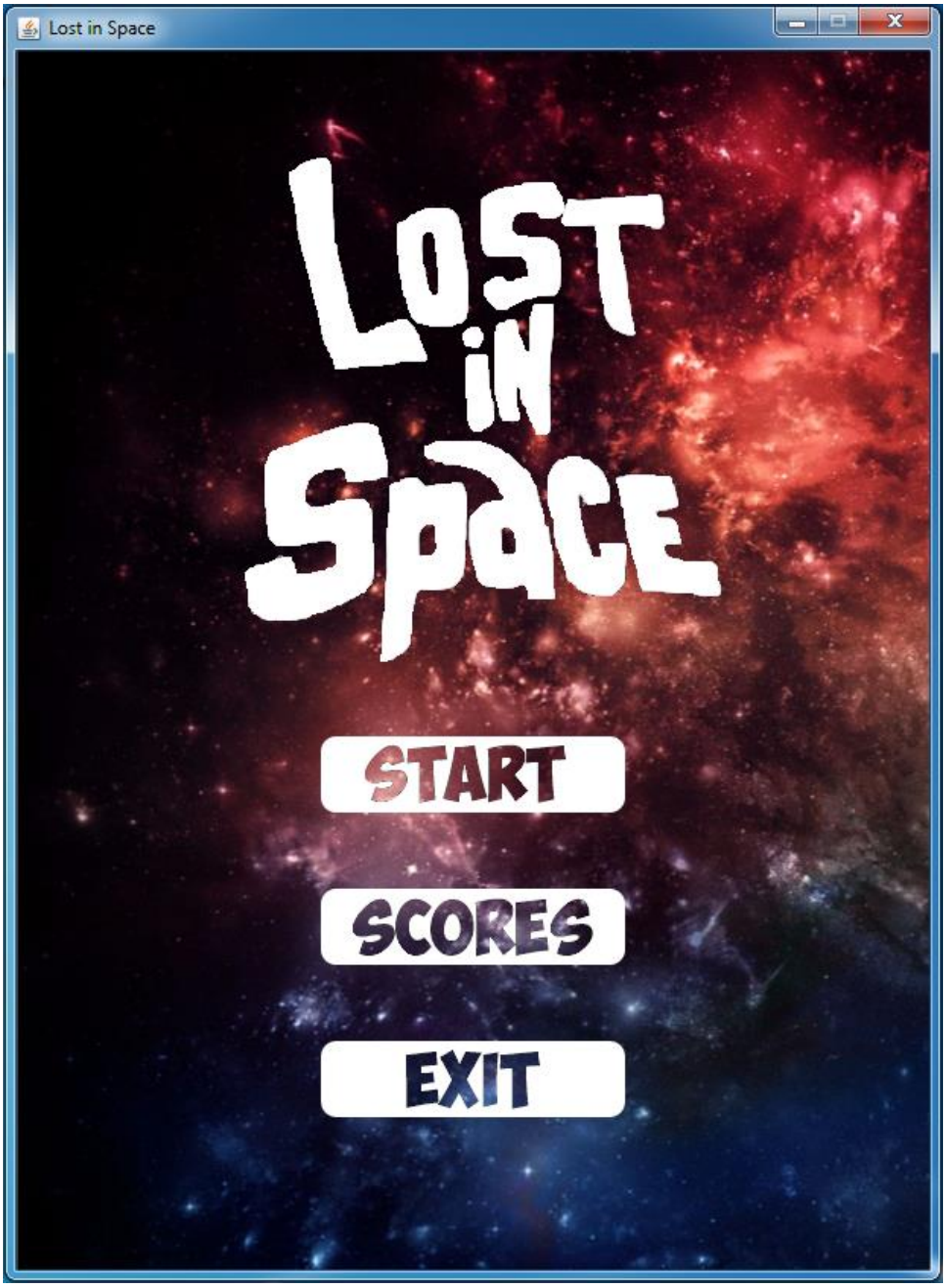


Рисунок 5.1 – Главное меню

В главном меню пользователю доступны разные функции.

При нажатии на кнопку «exit» произойдет закрытие приложения.

При нажатии на кнопку «scores» появится дополнительное окно с таблицей всех сохраненных счетов игроков (рис. 5.2).

Рисунок 5.2 – Таблица счетов

После нажатия на кнопку «start» начинается игровой процесс (рис 5.3).

Рисунок 5.3 – Игровое поле

После поражения пользователь будет видеть меню «смерти», на котором будет выведен его счет за прошлую игру (рис 5.4).

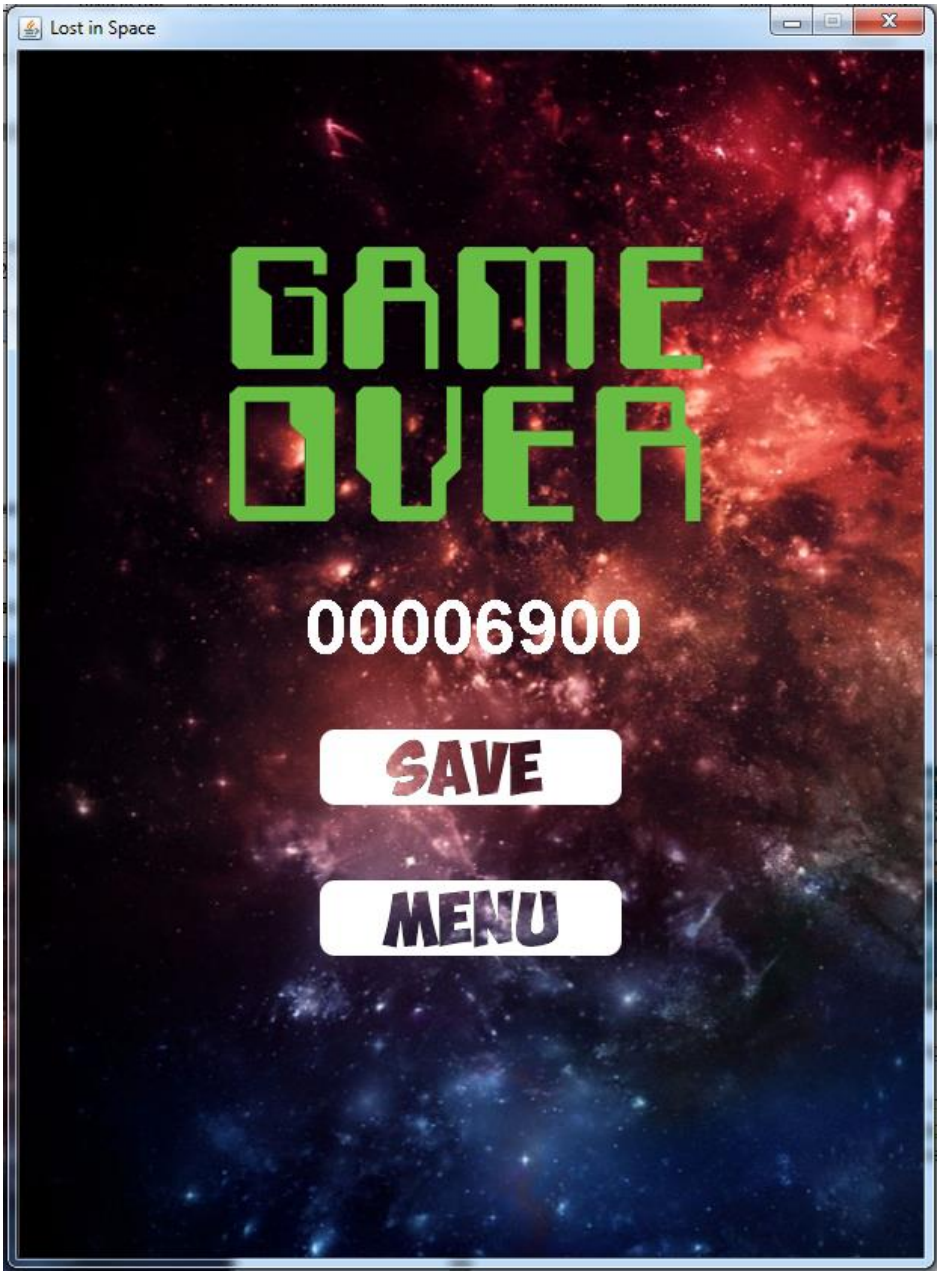


Рисунок 5.4 – Меню «смерти»

При нажатии на кнопку «save» появится дополнительное окно для ввода имени игрока и сохранения текущего счета в таблицу (рис 5.5).

Рисунок 5.5 – Окно ввода имени игрока

После нажатия на кнопку «menu» пользователь попадает на главное меню (рис 5.1) и при нажатии на кнопку «scores» будет выведена обновленная таблица счетов с внесенными данными прошлой сохраненной игры (рис 5.6).

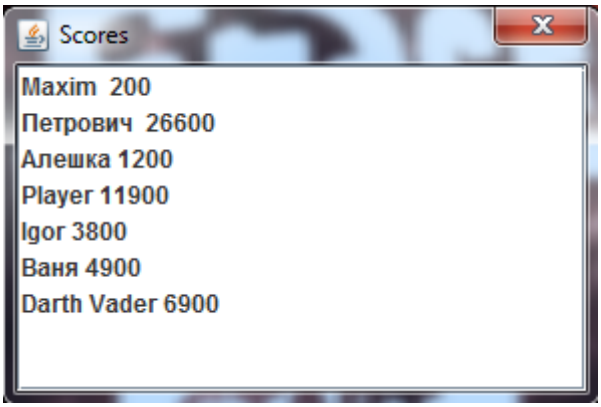


Рисунок 5.6 – Обновленная таблица счетов

ЗАКЛЮЧЕНИЕ

В результате работы над курсовым проектом было создано приложение, реализующее аркадный 2D - шутер со всем необходимым для него функционалом. Разработка приложения включала в себя решение множества задач, в ходе чего были изучены основы создания игры. Далее были изучены некоторые возможности создания приложений в IntelliJ IDEA на Java и формирование конкретных функциональных требований к программе на основе возможностей языка. Затем были разработаны структуры данных, разработана примерная архитектура приложения. Далее были детализированы все классы и их методы. Выполнив все вышеперечисленные этапы и собрав их воедино было полученное исправно работающее приложение.

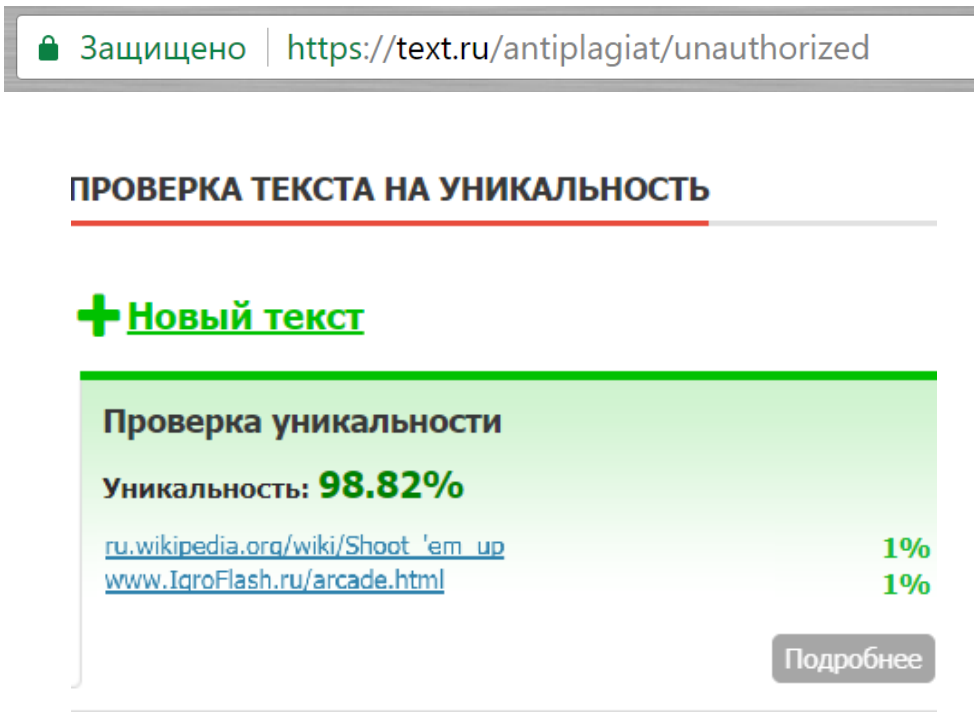
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Аркадные игры, описание игрового жанра [электронный ресурс]. – Электронные данные. – Режим доступа: http://www.igroflash.ru/arcade.html;

[2] Жанр Shoot ‘em up [электронный ресурс] – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Shoot_%27em_up;

[3] ГОСТ 19.701–90 (ИСО 5807–85) [Текст]. – Единая система программной документации: Сб. ГОСТов. - М.: Стандартинформ, 2005 с.

ПРИЛОЖЕНИЕ А
проверка на антиплагиат



ПРИЛОЖЕНИЕ Б
исходный код программы

CollisionDetector class

```
package collision;

import sprite.Sprite;

public class CollisionDetector {


    public static boolean checkCollision(Sprite target1, Sprite target2) {


        return target2.getY() + target2.getHeight() >= target1.getY()
                && target2.getY() <= target1.getY()
                && target1.getX() + target1.getWidth() >= target2.getX()
                && target1.getX() <= target2.getX() + target2.getWidth();

    }

}
```

Constants class

```
package constants;

public class Constants {

    // SCREEN
    public static final int SCREEN_WIDTH = 600;
    public static final int SCREEN_HEIGHT = 800;
    //! SCREEN

    // VEL
    public static final int PLANE_VELOCITY_X = 8;
    public static final int BULLET_VELOCITY_Y = 12;
    public static final int ASTEROID_VELOCITY = 6 ;
    //!VEL

    // textures
    public static final String BULLET_TEXTURE_NAME = "bullet.png";
    // !textures

    // timing
    public static final int SHOOT_TIME_INTERVAL = 200;
    //! timing


    // gen time
    public static final int ASTEROID_SPAWN_INTERVAL = 1000;
    // !gen time


    // score
    public static final int ASTEROID_SCORE = 100;
    // !score

    // damage
    public static final int ASTEROID_DAMAGE = 33;
    // !damage

}
```

Asteroid class

```
package entities.asteroid;

import constants.Constants;
import game.GameController;

public class Asteroid extends AsteroidSprite {
```

```
    private int score = Constants.ASTEROID_SCORE;
    private int damage = Constants.ASTEROID_DAMAGE;

    public Asteroid(String textureFileName, GameController context) {
        super(textureFileName, context);
    }

    public int getScore() {
        return score;
    }

    public void setScore(int score) {
        this.score = score;
    }

    public int getDamage() {
        return damage;
    }

    public void setDamage(int damage) {
        this.damage = damage;
    }

    @Override
    public String toString() {
        return "Asteroid{" + "score=" + score +
                ", damage=" + damage +
                '}';
    }
}
```

AsteroidSprite class

```
package entities.asteroid;

import constants.Constants;
import game.GameController;
import graphics.TextureAtlas;
import sprite.Sprite;

import java.awt.*;

public class AsteroidSprite implements Sprite {


    private TextureAtlas texture;

    private GameController context;

    private final int WIDTH = 80;
    private final int HEIGHT = 64;

    // default position of plane
    private int x = Constants.SCREEN_WIDTH / 2 - (WIDTH / 2);
    private int y = 0;

    //velocity
    private int velY = Constants.ASTEROID_VELOCITY;


    public AsteroidSprite(String textureFileName, GameController context) {
        texture = new TextureAtlas(textureFileName);
        this.context = context;
    }

    // 60 times in sec
    @Override
    public void render(Graphics2D graphics2D) {
        y += velY;
        if (texture != null) {
            graphics2D.drawImage(texture.cut(0, 0, WIDTH, HEIGHT), (int) x, (int) y, null);
        }
    }

    public int getX() {
        return x;
    }
```

```
    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void setYVelocity(int YVelocity) {
        this.velY = YVelocity;
    }

    @Override
    public int getHeight() {
        return HEIGHT;
    }

    public int getWidth() {
        return WIDTH;
    }

    @Override
    public void erase() {
        this.texture = null;
    }
}
```

BulletSprite class

```
package entities.bullet;

import constants.Constants;
import graphics.TextureAtlas;
import sprite.Sprite;

import java.awt.*;

public class BulletSprite implements Sprite {

    private TextureAtlas texture;

    private final int WIDTH = 9;
    private final int HEIGHT = 18;

    // default position of plane
    private int x;
    private int y;

    //velocity
    private int velY = Constants.BULLET_VELOCITY_Y;


    public BulletSprite(String textureFileName) {
        texture = new TextureAtlas(textureFileName);
    }


    @Override
    public void render(Graphics2D graphics2D) {
        y -= velY;
        if (texture != null) {
            graphics2D.drawImage(texture.cut(0, 0, 9, 18), (int) x, (int) y, null);
        }

    }



    public int getX() {
        return x;
    }

    public void setX(int x) {
```

```
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getHeight() {
        return HEIGHT;
    }

    public int getWidth() {
        return WIDTH;
    }

    @Override
    public void erase() {
        this.texture = null;
    }
}
```

Plane class

```
package entities.plane;

import constants.Constants;
import entities.bullet.BulletSprite;
import game.GameController;
import game.GameController;
import hud.HUDStatusBar;

import java.awt.*;

public class Plane extends PlaneSprite {



    private int health = 99;

    public Plane(String textureFileName, GameController context) {
        super(textureFileName, context);
    }

    public void render(Graphics2D graphics2D) {
        super.render(graphics2D);
    }


    // shoot bullet
    public void shoot() {
        BulletSprite bulletSprite = new BulletSprite(Constants.BULLET_TEXTURE_NAME);
        bulletSprite.setX(x + (WIDTH / 2) - (bulletSprite.getWidth() / 2));
        bulletSprite.setY(y - bulletSprite.getHeight());

        context.addBulletToContext(bulletSprite);
    }

    public void takeDamage(int damage) {
        health -= damage;
    }

    public int getHealth() {
        return health;
    }
}
```

PlaneSprite class

```
package entities.plane;

import constants.Constants;
import game.GameController;
```

```java
import graphics.TextureAtlas;
import sprite.Sprite;

import java.awt.*;

public class PlaneSprite implements Sprite {


    protected TextureAtlas texture;
    protected Graphics2D graphics2D;

    protected GameController context;


    protected final int WIDTH = 70;
    protected final int HEIGHT = 78;

    // default position of plane
    protected int x = Constants.SCREEN_WIDTH / 2 - (WIDTH / 2);
    protected int y = Constants.SCREEN_HEIGHT - HEIGHT;

    //velocity
    protected int velX = Constants.PLANE_VELOCITY_X;


    public PlaneSprite(String textureFileName, GameController context) {
        texture = new TextureAtlas(textureFileName);
        this.context = context;
    }

    // 60 times in sec
    @Override
    public void render(Graphics2D graphics2D) {
        if (texture != null) {
            graphics2D.drawImage(texture.cut(0, 0, WIDTH, HEIGHT), x, y, null);
        }
    }


    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void left() {

        // left wall collision detection
        if (!(x <= 0)) {
            x -= velX;
        }
    }

    public void right() {

        // right wall collision detection
        if (!(x >= (Constants.SCREEN_WIDTH - WIDTH))) {
            x += velX;
        }
    }



    @Override
    public int getWidth() {
        return WIDTH;
    }
```

```
    @Override
    public int getHeight() {
        return HEIGHT;
    }

    @Override
    public void erase() {
        this.texture = null;
    }
}
```

GameController class

```
package game;

import IO.Input;
import collision.CollisionDetector;
import constants.Constants;
import entities.asteroid.Asteroid;
import entities.bullet.BulletSprite;
import entities.plane.Plane;
import graphics.TextureAtlas;
import hud.HUDStatusBar;
import mouse.MenuMouseInput;
import music.Music;
import screen.DeathMenu;
import screen.MainMenu;
import sprite.Sprite;

import java.awt.Canvas;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.event.KeyEvent;
import java.awt.image.BufferStrategy;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;

import javax.swing.JFrame;
import javax.swing.JPanel;

public class GameController implements Runnable {


    private JFrame frame;
    private Canvas canvas;
    private BufferStrategy bufferStrategy;

    public static final String TITLE = "Lost in Space";


    public static final String ATLAS_FILE_NAME = "plane.png";


    private ArrayList<Sprite> spriteList = new ArrayList<Sprite>();
    private ArrayList<Sprite> asteroidList = new ArrayList<Sprite>();
    private ArrayList<Sprite> bulletList = new ArrayList<Sprite>();


    private HUDStatusBar hudStatusBar;

    private Input input;

    private Plane plane;

    private DeathMenu deathMenu;
    private MainMenu mainMenu;
    private boolean initialRendering = true;


    private long lastShootTime;
    private long lastAsteroidSpawnTime;

    private GameState gameState = GameState.MENU;
```

```
private TextureAtlas bgImage;


private long desiredFPS = 60;
private long desiredDeltaLoop = (1000 * 1000 * 1000) / desiredFPS;

private boolean running = true;


private GameController() {


    input = new Input();
    bgImage = new TextureAtlas("space.jpg");

    plane = new Plane(ATLAS_FILE_NAME, this);
    spriteList.add(plane);

    frame = new JFrame(TITLE);
    frame.add(input);

    deathMenu = new DeathMenu(frame, this);
    mainMenu = new MainMenu(this, frame);

    JPanel panel = (JPanel) frame.getContentPane();
    panel.setPreferredSize(new Dimension(Constants.SCREEN_WIDTH, Constants.SCREEN_HEIGHT));
    panel.setLayout(null);

    canvas = new Canvas();
    canvas.setBounds(0, 0, Constants.SCREEN_WIDTH,  Constants.SCREEN_HEIGHT);
    canvas.setIgnoreRepaint(true);

    // attach mouse listener to
    MenuMouseInput menuMouseInput= new MenuMouseInput(this);
    menuMouseInput.setDeathMenu(deathMenu);
    menuMouseInput.setMainMenu(mainMenu);
    canvas.addMouseListener(menuMouseInput);

    panel.add(canvas);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setResizable(false);
    frame.setVisible(true);


    canvas.createBufferStrategy(3);
    bufferStrategy = canvas.getBufferStrategy();

    canvas.requestFocus();
}


public void run() {

    long beginLoopTime;
    long endLoopTime;
    long deltaLoop;
    System.out.println(running);


    while (running) {


        beginLoopTime = System.nanoTime();

        render();

        update();

        endLoopTime = System.nanoTime();
        deltaLoop = endLoopTime - beginLoopTime;

        if (deltaLoop > desiredDeltaLoop) {
            //Do nothing. We are already late.
        } else {
            try {
```

```
                Thread.sleep((desiredDeltaLoop - deltaLoop) / (1000 * 1000));
            } catch (InterruptedException e) {
                //Do nothing
            }
        }
    }
}

private void render() {

    // first render
    if (initialRendering) {
        plane = new Plane(ATLAS_FILE_NAME, this);
        System.out.println("initial rendering");
        spriteList.add(plane);
        initialRendering = false;

        hudStatusBar = new HUDStatusBar(plane.getHealth(), this);
    }

    Graphics2D graphics = (Graphics2D) bufferStrategy.getDrawGraphics();
    graphics.clearRect(0, 0, Constants.SCREEN_WIDTH, Constants.SCREEN_HEIGHT);

    graphics.drawImage(bgImage.cut(0, 0, Constants.SCREEN_WIDTH, Constants.SCREEN_HEIGHT), 0, 0, null);


    if (gameState == GameState.IN_GAME ) {
        //System.out.println("here");


        // create asteroids
        if (System.currentTimeMillis() - lastAsteroidSpawnTime > Constants.ASTEROID_SPAWN_INTERVAL) {
            Random random = new Random();
            int numberOfAsteroids = random.nextInt(4) + 1;

            for (int i = 0; i < numberOfAsteroids; i++) {
                Asteroid asteroidSprite = new Asteroid("asteroid.png", this);
                int xPosition = random.nextInt(Constants.SCREEN_WIDTH - asteroidSprite.getWidth());
                int yVelocity = random.nextInt(Constants.ASTEROID_VELOCITY) + 2;
                asteroidSprite.setX(xPosition);
                asteroidSprite.setY(-100);
                asteroidSprite.setYVelocity(yVelocity);

                spriteList.add(asteroidSprite);
                asteroidList.add(asteroidSprite);
            }

            lastAsteroidSpawnTime = System.currentTimeMillis();
        }


        Iterator<Sprite> iterator = spriteList.iterator();
        // render all sprites
        while (iterator.hasNext()) {
            Sprite sprite = iterator.next();
            sprite.render(graphics);

            // sprite far off the screen -> delete all references
            if (sprite.getY() < -100) {
                iterator.remove();
            }

        }

        hudStatusBar.render(graphics);
        plane.render(graphics);

    } else if (gameState == GameState.DEATH_MENU) {
        deathMenu.render(graphics, hudStatusBar.getScore());
    } else if (gameState == GameState.MENU) {
        mainMenu.render(graphics);
    }


    graphics.dispose();
    bufferStrategy.show();
}
```

```
protected void update() {


    if (gameState == GameState.IN_GAME && plane.getHealth() > 0) {
        if (input.getKey(KeyEvent.VK_LEFT)) {
            plane.left();
        }

        if (input.getKey(KeyEvent.VK_RIGHT)) {
            plane.right();
        }

        if (input.getKey(KeyEvent.VK_ESCAPE)) {
            gameState = GameState.MENU;
        }

        if (input.getKey(KeyEvent.VK_SPACE)) {

            if (System.currentTimeMillis() - lastShootTime > Constants.SHOOT_TIME_INTERVAL) {
                plane.shoot();
                lastShootTime = System.currentTimeMillis();
            }
        }

        // bullet|asteroid collision detection
        for (Sprite bullet : bulletList) {
            for (Sprite asteroid : asteroidList) {

                boolean collided = CollisionDetector.checkCollision(bullet, asteroid);
                if (collided) {


                    ((BulletSprite) bullet).setY(Constants.SCREEN_HEIGHT - 1000);
                    ((BulletSprite) bullet).setX(Constants.SCREEN_WIDTH + 2000);
                    ((Asteroid) asteroid).setY(Constants.SCREEN_HEIGHT - 1000);
                    ((Asteroid) asteroid).setX(Constants.SCREEN_WIDTH + 2000);

                    hudStatusBar.setScore(((Asteroid) asteroid).getScore());

                    bullet.erase();
                    asteroid.erase();
                }
            }

        }

        // plane|steroid collision detection
        for (Sprite asteroid : asteroidList) {
            boolean collided = CollisionDetector.checkCollision(plane, asteroid);

            if (collided) {
                asteroid.erase();
                ((Asteroid) asteroid).setY(Constants.SCREEN_HEIGHT - 1000);
                ((Asteroid) asteroid).setX(Constants.SCREEN_WIDTH + 2000);
                plane.takeDamage(((Asteroid) asteroid).getDamage());
                hudStatusBar.setHealth(plane.getHealth());
            }
        }

        if (plane.getHealth() <= 0) {
            gameState = GameState.DEATH_MENU;
        }
    }

}


// clear all sprites
public void clearGameArea() {
    Iterator<Sprite> iterator = spriteList.iterator();
    hudStatusBar.refresh();

    while (iterator.hasNext()) {
        Sprite sprite = iterator.next();
        sprite.erase();
```

```
            // sprite far off the screen -> delete all references
            if (sprite.getY() < -100) {
                iterator.remove();
            }

        }

        spriteList.clear();
        bulletList.clear();
        asteroidList.clear();
        initialRendering = true;
    }

    // create bullet
    public void addBulletToContext(Sprite sprite) {
        this.spriteList.add(sprite);
        this.bulletList.add(sprite);
    }


    public GameState getGameState() {
        return gameState;
    }

    public void setGameState(GameState gameState) {
        this.gameState = gameState;
    }

    public static void main(String[] args) throws FileNotFoundException {

        Music m = new Music();
        new Thread(m).start();

        GameController ex = new GameController();
        new Thread(ex).start();

    }
}
```

GameState enum

```
package game;

public enum GameState {
    MENU,
    IN_GAME,
    DEATH_MENU
}
```

TextureAtlas class

```
package graphics;

import utils.ResourceLoader;

import java.awt.image.BufferedImage;

public class TextureAtlas {

    BufferedImage image;

    public TextureAtlas(String imageName){
        image = ResourceLoader.loadImage(imageName);
    }

    public BufferedImage cut(int x, int y, int w, int h){
        return image.getSubimage(x, y, w, h);
    }

}
```

HUDStatusBar class

```
package hud;

import constants.Constants;
```

```
import game.GameController;
import sprite.Sprite;

import java.awt.*;

public class HUDStatusBar implements Sprite{

    private int targetHealth;

    private GameController context;

    private String scorePattern = "%s";
    private int score;

    private final int HEIGHT = 32;

    // default position of hud
    private int x = 10;
    private int y = 10;

    public HUDStatusBar(Integer health, GameController context) {
        this.context = context;
        this.targetHealth = health;
    }


    @Override
    public void render(Graphics2D graphics2D) {
        // render health bar
        graphics2D.setColor(Color.RED);
        graphics2D.fillRect(x, y, (int)(targetHealth * 2.4), HEIGHT);

        // render score
        String strScore = String.valueOf(score);

        StringBuilder base = new StringBuilder("");

        for (int i= 0; i < 8 - strScore.length(); i++) {
            base.append("0");
        }

        base.append(strScore);

        graphics2D.setColor(Color.WHITE);
        Font font = new Font("arial", Font.BOLD, 40);
        graphics2D.setFont(font);
        graphics2D.drawString(String.format(scorePattern, base), (x + Constants.SCREEN_WIDTH - 200), y + 30);
    }


    public int getScore() {
        return score;
    }

    public void setScore(int score) {

            this.score += score;

    }

    @Override
    public int getX() {
        return 0;
    }

    @Override
    public int getY() {
        return 0;
    }

    @Override
    public int getWidth() {
        return 0;
    }

    @Override
    public int getHeight() {
```

```
        return 0;
    }

    @Override
    public void erase() {
    }

    public void setHealth(int health) {
        this.targetHealth = health;
    }

    public void refresh() {
        this.score = 0;
    }
}
```

FileSystemService class

```
package IO;

import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;

public class FileSystemService {

    public FileSystemService() {
    }



    public void writeToFile(String filename, String text) {

        try(FileWriter writer = new FileWriter(filename, true)) {

            writer.write(text + "\n");
        } catch (IOException e) {
            e.printStackTrace();
        }


    }

    public List<String> readLines(String filename) {

        try {
            return Files.readAllLines(Paths.get(filename));
        } catch (IOException e) {
            e.printStackTrace();
        }

        return null;
    }
}
```

Input class

```
package IO;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.KeyListener;
import java.util.Arrays;

public class Input extends JComponent {

    private boolean[] map;

    public Input() {

        map = new boolean[256];
```

```
        for (int i = 0; i < map.length; i++) {

            final int KEY_CODE = i;

            getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(i, 0, false), i * 2);
            getActionMap().put(i * 2, new AbstractAction() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    map[KEY_CODE] = true;
                }
            });

            getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(i, 0, true), i * 2 + 1);
            getActionMap().put(i * 2 + 1, new AbstractAction() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    map[KEY_CODE] = false;
                }
            });

        }

    }

    public boolean[] getMap() {
        return Arrays.copyOf(map, map.length);
    }

    public boolean getKey(int keyCode) {
        return map[keyCode];
    }

}
```

MenuMouseInput class

```
package mouse;

import constants.Constants;
import game.GameController;
import game.GameState;
import screen.DeathMenu;
import screen.GameMenu;
import screen.MainMenu;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class MenuMouseInput implements MouseListener {

    private final GameController context;
    private GameMenu deathMenu;
    private GameMenu mainMenu;


    public MenuMouseInput(GameController gameController) {
        context = gameController;
    }


    @Override
    public void mouseClicked(MouseEvent e) {

    }

    @Override
    public void mousePressed(MouseEvent e) {
        int x = e.getX();
        int y = e.getY();


        if (context.getGameState() == GameState.MENU) {
            System.out.println("Main Menu action");
            performMenuAction(x, y, mainMenu);
        } else if (context.getGameState() == GameState.DEATH_MENU) {
            System.out.println("Death Menu action");
            performMenuAction(x, y, deathMenu);
```

```
        }

    }

    private void performMenuAction(int x, int y, GameMenu menu) {
        if (x > Constants.SCREEN_WIDTH / 2  - 100 && x < (Constants.SCREEN_WIDTH / 2  - 100 ) + 200) {
            if (y > 450 && y < 450 + 50) {
                System.out.println("first button click");
                menu.firstButtonClick();
            } else if (y > 550 && y < 600)  {
                System.out.println("second button click");
                menu.secondButtonClick();
            } else if (y > 650 && y < 700) {
                System.out.println("third button click");
                ((MainMenu)menu).thirdButtonClick();;
            }
        }



    }

    @Override
    public void mouseReleased(MouseEvent e) {

    }

    @Override
    public void mouseEntered(MouseEvent e) {

    }

    @Override
    public void mouseExited(MouseEvent e) {

    }

    public GameMenu getDeathMenu() {
        return deathMenu;
    }

    public void setDeathMenu(GameMenu deathMenu) {
        this.deathMenu = deathMenu;
    }

    public GameMenu getMainMenu() {
        return mainMenu;
    }

    public void setMainMenu(GameMenu mainMenu) {
        this.mainMenu = mainMenu;
    }
}
```

Music class

```
package music;

import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.Player;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Music implements Runnable {

    public Music() {

    }

    @Override
    public void run() {
        while (true) {
            try {

                FileInputStream f = new FileInputStream("res/John Williams - Main Title and Escape.mp3");
```

```
            try {
                Player player = new Player(f);
                player.play();
            } catch (JavaLayerException e) {
                e.printStackTrace();
            } finally {
                try {
                    f.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

  }
}
```

Record class

```
package record;

public class Record {

   private String name;
   private int score;

   public Record(String name, int score) {
      this.name = name;
      this.score = score;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public int getScore() {
      return score;
   }

   public void setScore(int score) {
      this.score = score;
   }

   @Override
   public String toString() {
      return name + " " + score;
   }
}
```

RecordService class

```
package record;

import IO.FileSystemService;
import utils.ResourceLoader;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class RecordService {


   private FileSystemService fileSystemService;

   public RecordService() {
      this.fileSystemService = new FileSystemService();
   }
```

```
    public ArrayList<Record> findAllRecords() {
        ArrayList<Record> records = new ArrayList<>();

        String RECORDS_FILENAME = "scores.txt";
        List<String> strings = fileSystemService.readLines(ResourceLoader.resolve(RECORDS_FILENAME));
        for (String s : strings) {
            Record record = new Record(s.split(":")[0], Integer.valueOf(s.split(":")[1]));
            records.add(record);
        }


        System.out.println(records);
        return records;

    }
}
```

DeathMenu class

```
package screen;

import IO.FileSystemService;
import constants.Constants;
import game.GameController;
import game.GameState;
import graphics.TextureAtlas;
import hud.HUDStatusBar;
import utils.ResourceLoader;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class DeathMenu implements GameMenu {


    private final Font font;

    private final JFrame parentFrame;
    private final JPanel modalContent;
    private final JDialog dialog;

    private int score;
    private String scorePattern = "%s";
    private final JTextField textField;
    private TextureAtlas gameOverImage;
    private TextureAtlas saveImage;
    private TextureAtlas menuImage;

    private FileSystemService fileSystemService;
    private GameController context;

    public DeathMenu(JFrame parentFrame, GameController context) {

        fileSystemService = new FileSystemService();
        this.context = context;

        gameOverImage = new TextureAtlas("gameOver.png");
        saveImage = new TextureAtlas("save.png");
        menuImage = new TextureAtlas("menu.png");

        this.parentFrame = parentFrame;
        font = new Font("arial", Font.BOLD, 50);
        modalContent = new JPanel(new GridBagLayout());

        dialog = new JDialog(parentFrame, "Enter your name", true);
        dialog.getContentPane().add(modalContent);

        textField = new JTextField("Player", 20);
        textField.setVisible(true);

        JButton jButton = new JButton("Ok");

        jButton.addActionListener(new SaveResultActionListener());
```

```
        jButton.setVisible(true);

        modalContent.add(textField);
        modalContent.add(jButton);

    }


    public void render(Graphics2D graphics, int score) {
        this.score = score;

        // render score
        String strScore = String.valueOf(score);

        StringBuilder base = new StringBuilder("");

        for (int i= 0; i < 8 - strScore.length(); i++) {
            base.append("0");
        }
        base.append(strScore);

        graphics.setFont(this.font);
        graphics.setColor(Color.WHITE);

        graphics.drawImage(gameOverImage.cut(0, 0, 314, 182), 140, 130, null);
        graphics.drawImage(saveImage.cut(0, 0, 200, 50), Constants.SCREEN_WIDTH / 2  - 100, 450, null);
        graphics.drawImage(menuImage.cut(0, 0, 200, 50), Constants.SCREEN_WIDTH / 2  - 100, 550, null);

        graphics.drawString(String.format(scorePattern, base), 190, 400);

        dialog.setBounds((int) parentFrame.getLocationOnScreen().getX() + (Constants.SCREEN_WIDTH / 2) - 150,
                (int) parentFrame.getLocationOnScreen().getY() + Constants.SCREEN_HEIGHT / 2 - 100, 300, 200);

    }



    public class SaveResultActionListener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            // todo validate name

            //ResourceLoader.resolve("records.txt");

            String text = String.format("%s:%s", textField.getText(), score);



            System.out.println(text);
            fileSystemService.writeToFile(ResourceLoader.resolve("scores.txt"), text.trim());
            dialog.setVisible(false);
        }

    }

    @Override
    public void firstButtonClick() {
        dialog.setVisible(true);
    }

    @Override
    public void secondButtonClick() {
        context.setGameState(GameState.MENU);
    }
}
```

GameMenu interface

```
package screen;

// menu with two buttons
public interface GameMenu {

    void firstButtonClick();
```

```
    void secondButtonClick();
}
```

MainMenu class

```
package screen;

import constants.Constants;
import game.GameController;
import game.GameState;
import graphics.TextureAtlas;
import record.Record;
import record.RecordService;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

public class MainMenu implements GameMenu{


    private final DefaultListModel model;

    private final JDialog dialog;
    private TextureAtlas logoImage;
    private TextureAtlas startImage;
    private TextureAtlas scoresImage;
    private TextureAtlas exitImage;
    private final RecordService recordService;
    private final JFrame parentFrame;

    private GameController context;

    public MainMenu(GameController context, JFrame parentFrame) {
        recordService = new RecordService();

        logoImage = new TextureAtlas("logo.png");
        startImage = new TextureAtlas("start.png");
        scoresImage = new TextureAtlas("scores.png");
        exitImage = new TextureAtlas("exit.png");

        this.context = context;
        this.parentFrame = parentFrame;
        dialog = new JDialog(parentFrame, "Scores", true);


        // default empty list
        model = new DefaultListModel<String>();
        JList<String> list = new JList(model);
        list.setVisible(true);

        // create list view
        JScrollPane modalContent = new JScrollPane(list);
        dialog.getContentPane().add(modalContent);

    }

    public void render(Graphics2D graphics) {
        graphics.setColor(Color.WHITE);

        graphics.drawImage(logoImage.cut(0, 0, 313, 331), 150, 70, null);
        graphics.drawImage(startImage.cut(0, 0, 200, 50), Constants.SCREEN_WIDTH / 2  - 100, 450, null);
        graphics.drawImage(scoresImage.cut(0, 0, 200, 50), Constants.SCREEN_WIDTH / 2  - 100, 550, null);
        graphics.drawImage(exitImage.cut(0, 0, 200, 50), Constants.SCREEN_WIDTH / 2  - 100, 650, null);

        dialog.setBounds((int) parentFrame.getLocationOnScreen().getX() + (Constants.SCREEN_WIDTH / 2) - 150,
                (int) parentFrame.getLocationOnScreen().getY() + Constants.SCREEN_HEIGHT / 2 - 100, 300, 200);

    }

    @Override
    public void firstButtonClick() {
        context.clearGameArea();
        context.setGameState(GameState.IN_GAME);
    }
```

```
    @Override
    public void secondButtonClick() {

        // find all records
        model.clear();
        ArrayList<Record> records = recordService.findAllRecords();
//        DefaultListModel<String> model = new DefaultListModel<String>();
        for (Record record : records) {
            System.out.println(record);
            model.addElement(record.toString());
        }

        dialog.setVisible(true);

    }

    public void thirdButtonClick() {
        System.exit(0);
    }
}
```

Sprite interface

```
package sprite;

import java.awt.*;

public interface Sprite {
    void render(Graphics2D graphics2D);

    int getX();

    int getY();

    int getWidth();

    int getHeight();

    void erase();

}
```

ResourceLoader class

```
package utils;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class ResourceLoader {

    private static final String PATH = "res/";

    public static BufferedImage loadImage(String fileName) {

        BufferedImage image = null;

        try {

            image = ImageIO.read(new File(PATH + fileName));

        } catch (IOException e) {
            e.printStackTrace();
        }

        return image;

    }

    public static String resolve(String resourceName) {
        return PATH + resourceName;
    }

}
```

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 1–40 01 01 024 ПЗ	Пояснительная записка	23 с.
	Графические документы	
ГУИР 651005 024 ПД	Схема программы	Формат А1

					БГУИР КП 1-40 01 01 024 Д1			
Изм.	Л.	№ докум.	Подп.	Дата	ПС реализующее аркадный 2D - шутер Ведомость курсового проекта		Лист	Листов
Разраб.		Стаселович И.А.		16.12.17		Т	44	44
Пров.		Марина И.М.		20.12.17		Кафедра ПОИТ Гр. 651005		