Оглавление

Введение

В данной пояснительной записке рассматривается описание игры в жанре аркада «Rocket game with Dora» на основе объектно-ориентированного подхода.

Объектно-ориентированное программирование (ООП) — это подход, при котором программа рассматривается как набор объектов, взаимодействующих друг с другом. ООП ускоряет написание кода и делает его читаемым.

Использование ООП существенно повышает уровень унификации разработки и пригодность для повторного использования не только ПО, но и проектов, что в конце концов ведет к сборочному созданию ПО. Системы зачастую получаются более компактными, чем их не объектно-ориентированные эквиваленты, что означает не только уменьшение объема программного кода, но и удешевление проекта за счет использования предыдущих разработок.

В отличие от других подходов к программированию, объектно-ориентированный подход требует глубокого понимания основных принципов, или концепций, на которых он базируется.

Из числа основополагающих понятий ООП обычно относят: абстракцию данных, наследование, инкапсуляцию и полиморфизм.

В качестве основного инструмента разработки применяется IntelliJ IDEA IDE 2023.3.1 Язык программирования Java.

1 Разработка компьютерной игры

1.1 Разработка технического задания

1.1.1 Полное наименование системы и её условное обозначение

Полное наименование: Разработка игрового приложения «Rocket game with Dora» для проведения досуга.

Условное обозначение системы: «Rocket game».

1.1.2 Основание для разработки

Основанием для разработки данной информационной системы является приказ по НГТУ на курсовое проектирование по дисциплине «Программирование на языке Java».

1.1.3 Назначение и цели создания системы

1.1.3.1 Назначение системы

Данный продукт предназначен для проведения досуга.

1.1.3.2 Цели создания системы

Программный продукт разрабатывается с целью:

1) Развлечь себя в свободное время.

2) Посоревноваться с друзьями.

1.1.3.3 Основные задачи разработки

1) Создать графический интерфейс.

2) Обеспечить процесс проведения игры.

1.2 Анализ технического задания

1.2.1 Общие положения

Согласно техническому заданию необходимо разработать клиентское приложение для развлечения в свободное время.

1.2.2 Требования к системе

1.2.2.1 Требования к структуре и функционированию системы

Программный продукт, разрабатываемый в рамках курсового проекта, должен удовлетворять следующему перечню функциональных требований:

1) Удобный графический интерфейс;

2) Корректно работающее приложение;

3) Экран проигрыша;

1.2.2.2 Входные данные

Входными данными при работе с программным продуктом должны быть нажатия на кнопку мыши (левая), кнопки A, D, J, L, стрелка влево, стрелка вправо, ESC.

1.2.2.3 Выходные данные

Выходными данными при работе программы является движение игроков, улучшений, препятствий, отображение локации на экран пользователя.

1.2.2.4 Требования к надёжности

Система должна сохранять работоспособность и обеспечивать восстановление своих функций при возникновении внештатных ситуаций.

1.2.2.5 Требования к эргономике и технической эстетике

Подсистемы ввода данных, а также формирования и визуализации отчетности должны обеспечивать удобный для конечного пользователя интерфейс.

Главное окно программного продукта должно позволять пользователю начать игру и выйти из игры.

1.3 Стандарты

Программный продукт разрабатывается на основании следующих государственных стандартов:

1) 2.103-68 ЕСКД. Стадии разработок.
2) 2.104-68 ЕСКД. Основные надписи.
3) 2.105-95 ЕСКД. Общие требования к текстовым документам.
   a. -96 ЕСКД. Текстовые документы.
4) 2.111- 68 ЕСКД. Общие требования к текстовым документам.
5) 2.118-73 ЕСКД. Техническое предложение.
6) 2.120-73 ЕСКД. Технический проект.
7) 2.316-68 ЕСКД. Правила нанесения на чертежах надписей, технических требований и таблиц.
8) 7.1-2003 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

1.4 Требования к техническим средствам

Для удобства работы система должна обеспечивать отображение *GUI* с расширенным набором пользовательских элементов, что соответствует

платформе *Java*, ОС не важна. Также необходима 16 версия JDK.

Минимальные технические характеристики компьютера, на котором гарантируется стабильная работа программы:

1) процессор: с частотой 2 ГГц и более;
2) Объем оперативной памяти: 4 ГБ;
3) Монитор: разрешением 1920x1080;
4) Наличие свободного дискового пространства на жестком диске – не менее 50 Мб.

2 Разработка модели предметной области

2.1 Анализ предметной области

Программируемое приложение должно позволять пользователю/ пользователям управлять ракетами, взаимодействовать с меню.

Нефункциональные требования к программному средству:

1) Эффективность: программа должна иметь минимальные требования к аппаратному обеспечению. Не должно требоваться дополнительных периферийных средств (сканер, дополнительные дисководы и т.д.);
2) Доступность: программа должна быть доступна, иметь удобный пользовательский интерфейс;

2.2 Разработка структуры классов

В соответствие с анализом предметной области можно выделить основные классы информационной системы и их атрибуты:

1) Класс Main – стартовый класс, запускающий окно с приложением:
   - Метод, создающий окно приложения;
2) Класс Window – класс, создающий окно приложения и запускающий плоскость для отрисовки элементов GUI:
   - Методы получения разрешения экрана;
3) Класс GameArea – класс, включающий в себя основную логику игры. Взаимодействие объектов, игроков, кнопок, окон, между собой:
   - Методы отрисовки окон (меню, поражения, ошибки, игровой

области);

- Методы обработки и генерации чисел
- Методы, регулирующие состояние/логику игры
- Методы, регулирующие количество объектов на экране

4) Класс Player – класс, хранящий информацию об игроке, обработку событий, связанных с игроком:

- Методы обновления данных о игроке: позиции, состояния.
- Метод обработки коллизии игрока;
- Метод отрисовки игрока

5) Класс Object – класс, хранящий информацию об объектах, летящих на игрока. Включает в себя обработку всех событий, связанных с объектами:

- Методы обновления данных о объекте: позиции, состояния.
- Метод отрисовки объекта

6) Класс Button – класс, хранящий информацию о кнопках. Включает в себя обработку всех событий, связанных с кнопками:

- Методы обновления состояния кнопки.
- Метод отрисовки объекта

7) Класс MyKeyListener – класс, обрабатывающий нажатие на клавиши:

- Методы, регулирующие события, которые происходят по нажатию/отпусканию клавиш

8) Класс MyMouseListener – класс, обрабатывающий передвижение мыши, а как же нажатие на клавиши мыши:

- Методы обновления состояния мыши.

3 Разработка структуры приложения

3.1 Разработка архитектуры

Игровое приложение «Rocket game» не очень сложно в исполнении,

поэтому для работы создано всего 8 классов в которых описаны функции игровых и неигровых объектов. Класс Main запускает игровое приложение с помощью класса Window, который создает окно приложения. В этом же классе объявляется класс GameArea.

Класс GameArea запускает таймер перерисовки окон, инициализирует объекты класса Object, а также пытается загрузить картинки. При неудачной загрузке, пользователю выведется окно оповещения об ошибке (В окне оповещения об ошибке, есть кнопка выхода из программы), иначе отрисуется окно меню.

В меню можно нажимать на кнопки, созданные с помощью класса Button, который в свою очередь с помощью класса MyMouseListener определяет нажатие на клавишу. В окне меню можно выбрать игру на одного, на двоих и на троих, кнопками «1 Player», «2 Players», «3 Players» соответственно. Кнопками с цифрами можно указать сложность, а кнопкой «exit» выйти из игры.

При нажатии на «1 Player» метод spawn, создаст одного игрока с помощью класса Player. Запустится игровой цикл. GameArea отрисовывает игровой фон, текст, вызывает отрисовку игроков и объектов. Также данный класс регулирует количество объектов на игровом поле и изменяет счет, используя параметр сложности difficult. Сложность можно регулировать в любое время, даже в игре. Достаточно выйти в меню нажав на клавиатуре [ESC]. В меню выбрать сложность любой из доступных кнопок с цифрами, после, снова нажать на клавиатуре [ESC]. Игра продолжится и будет длится, пока хотя бы 1 из игроков остается в живых. После потери всех ракет, идет отрисовка окна поражения. В этом окне можно увидеть текущий/лучший счет за игру, а также выйти в меню.

3.2 Проектирование пользовательского интерфейса и взаимодействие с ним

В данном проекте для успешной работы реализован удобный пользовательский интерфейс (рис. 1, 2, 3, 4).

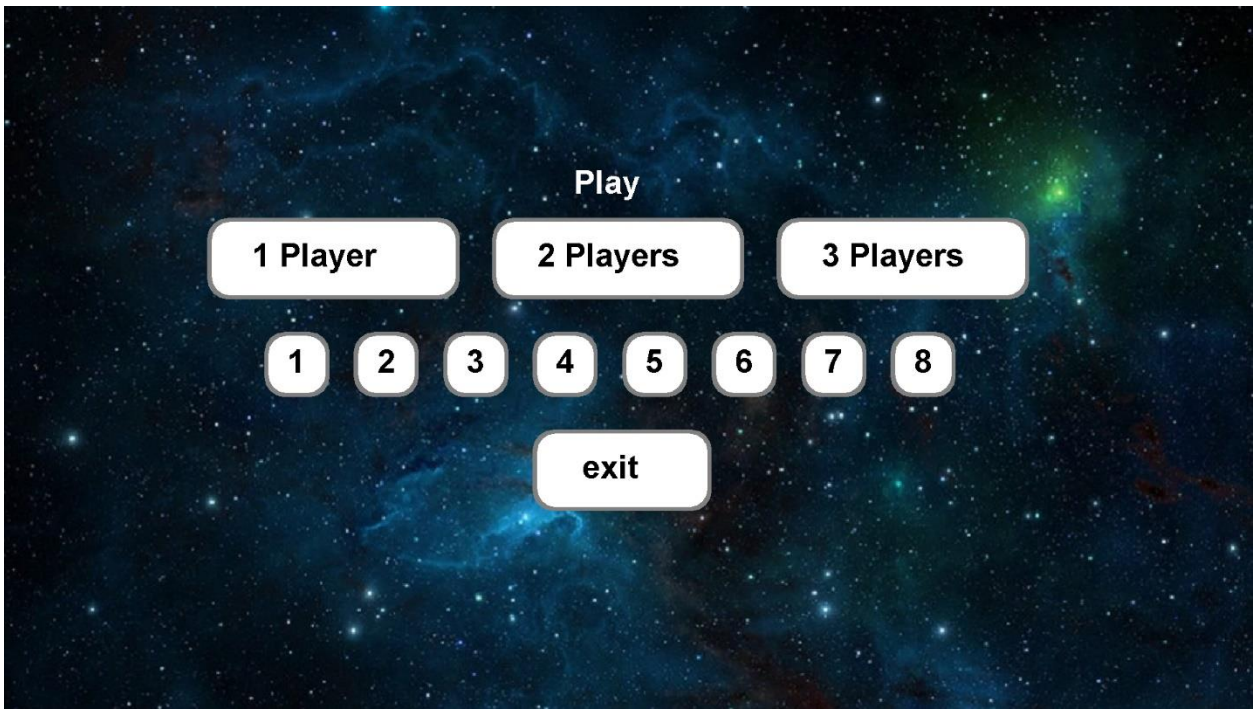

Рис. 1 – Меню приложения.

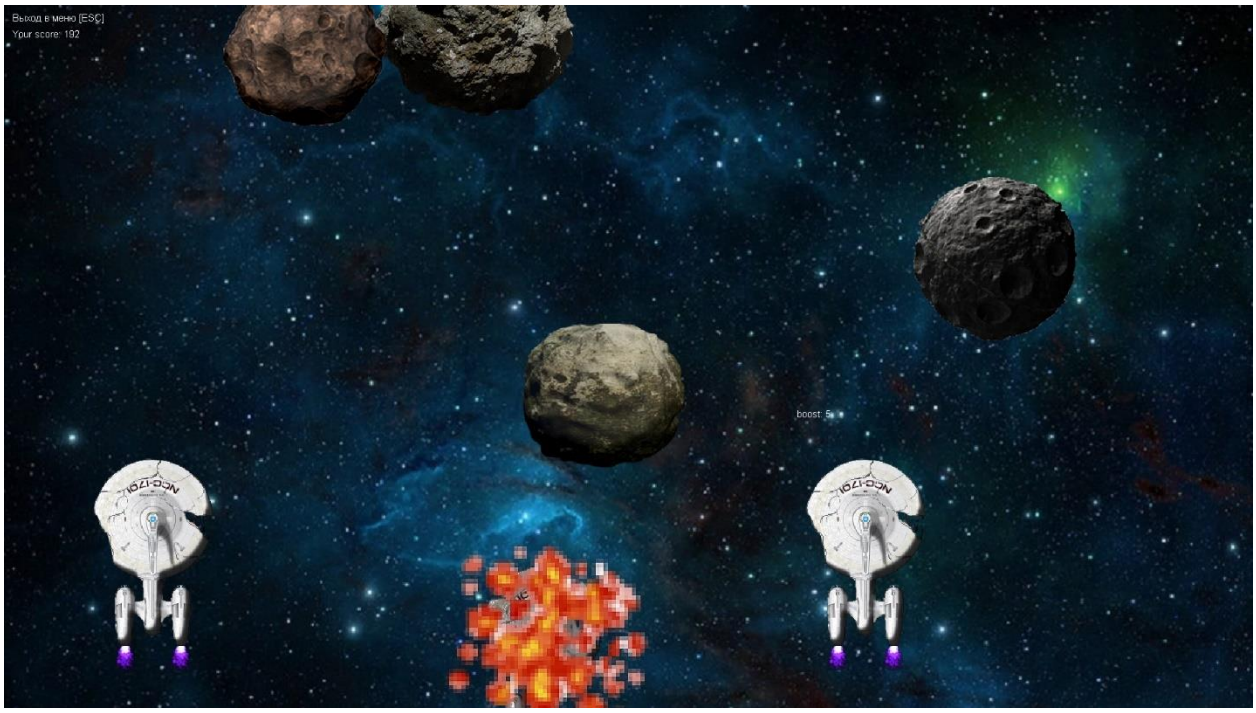

Рис. 2 – Геймплей.

Рис. 3 - Интерфейс окна оповещения об ошибке.

Рис. 4 – Интерфейс окна проигрыша.

Основной сценарий работы пользователя с приложением.

1) Пользователь запускает приложение и видит стартовый экран с кнопками, нажимая на которые пользователь активирует события: играть одному, вдвоем, втроем, установить сложность или выйти из игры.
2) Пользователь приступает к управлению персонажем один или вместе с товарищами уворачиваясь от препятствий.
3) После проигрыша пользователь возвращается в меню.

Заключение

В данной курсовой работе было разработано игровое приложение «Rocket game», используя объектно-ориентированный подход. Этот метод позволяет упростить сложные объекты, составляя их из более маленьких и простых. В проекте удалось выполнить все выше поставленные задачи:

1) Создать графический интерфейс.
2) Обеспечить корректную реакцию приложения на действия игрока.
3) Обеспечить закрытие приложения

Тем не менее, данное приложение не является конечным результатом, так как существует ещё множество различных путей для развития и улучшения работы программы.

В перспективе можно рассмотреть следующий перечень улучшений:

- Добавить поддержку разных разрешений.
- Добавить больше эффектов и картинок препятствий.
- Добавить сохранение лучшего счета в отдельный файл и считывание его при запуске.
- Добавить музыку.

Список литературы

1. Кэти Сьерра, Берт Бейтс «Изучаем Java», 2-е издание - Эксмо, 2012 – 720 с.
2. https://www.youtube.com/c/KaarinGaming

3. https://blog.skillfactory.ru/glossary/oop-obektno-orientirovannoe-programmirovanie/

Приложения

Приложение 1: Код класса Main

```
public class Main {
    /**
     *
     * @author Terrick
     */
    public static void main(String[] args) {
        Window window = new Window();
    }
}
```

Приложение 2: Код класса Window.

```
import javax.swing.*;
/**
 *
 * @author Terrick
 */
public class Window extends JFrame {

    public Window()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("game_rocket_with_DORA");
        setExtendedState(JFrame.MAXIMIZED_BOTH);
        setUndecorated(true);
        add(new GameArea());
        setVisible(true);
    }

    //автоматическое получение ширины экрана
    public static int getScreenWidth() {
        return
java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment().getMaximumWindowBounds().
width;
    }
    //автоматическое получение высоты экрана
    public static int getScreenHeight() {
        return
java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment().getMaximumWindowBounds().
height;
    }

}
```

Приложение 3: Код класса GameArea.

```
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
/**
 *
 * @author Terrick
 */
public class GameArea extends JPanel {

    private static final byte[] COUNT_MASS_IMG = {10, 2};
    private static final int DELAY_FRAME    = 25;
    private static final int DFT_COUNT_DFCT  = 8;
    private static final int DFT_COUNT_BTN   = 4 + DFT_COUNT_DFCT;
    public  enum STATES{MENU, PLAY1P, PLAY2P, PLAY3P, PlAYAREA, ERRORLOAD,
GAMEOVER}

    private static STATES           state      = STATES.MENU;
    private static int              difficult  = 4;
    private static int              score      = 0;
    private static int      localBetterScore    = 0;
    private static ArrayList<Player> players    = new ArrayList<>();
    private static Object[]         gameObject = new Object[COUNT_MASS_IMG[0]];
    private static Image[]          images     = new Image[COUNT_MASS_IMG[1]];
    private static Button[]         buttons    = new Button[DFT_COUNT_BTN];
    private static Font             font       = new Font( Font.SANS_SERIF , Font.PLAIN|
Font.BOLD , 40);
    private static MyMouseListener   mouse      = new MyMouseListener();
    private        Timer            timerRepaint;

    static ArrayList<Player> getPlayers()              { return players;    }
    static void              setState(STATES localState) { state = localState;}
    static void              setDifficult(int dfct)      { difficult = dfct;  }
    static STATES            getState()                  { return state;      }

    public GameArea() {
        setFocusable(true);
        requestFocus();
        addKeyListener(new MyKeyListener());
        addMouseListener(mouse);

        timerRepaint = new Timer(DELAY_FRAME, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                repaint();
```

```
            }
        });
        timerRepaint.start();
        for (int count = 0; count < COUNT_MASS_IMG.length; count++) {
            for (int i = 0; i < COUNT_MASS_IMG[count]; i++){
                try{
                    if (count == 0) {
                        gameObject[i] = new Object(ImageIO.read(new
    File("images\\gameArea\\gameObject\\" + i + ".png")), i);
                    }
                    if (count == 1) {
                        images[i] = ImageIO.read(new File("images\\gameArea\\1" + i + ".png"));
                    }
                }
                catch (IOException exp){
                    state = STATES.ERRORLOAD;
                    break;
                }
            }
        }

        buttons[0] = new Button(650, 520, 220, 100, 60, "exit", mouse);
        buttons[1] = new Button(250, 260, 310, 100, 55, "1 Player", mouse);
        buttons[2] = new Button(600, 260, 310, 100, 55, "2 Players", mouse);
        buttons[3] = new Button(950, 260, 310, 100, 55, "3 Players", mouse);

        for (int i = DFT_COUNT_BTN - DFT_COUNT_DFCT; i < DFT_COUNT_BTN; i++) {
            Integer label = i - (DFT_COUNT_BTN - DFT_COUNT_DFCT) + 1;
            buttons[i] = new Button(210 + label * 110, 400, 80, 80, 28, label.toString(), mouse);
        }
    }
    //метод регулирующий вызов других методов.
    public void paintComponent(Graphics gr)
    {
        switch (state) {
            case PLAY1P    -> spawn(1);
            case PLAY2P    -> spawn(2);
            case PLAY3P    -> spawn(3);
            case PlAYAREA  -> playMethod(gr);
            case MENU      -> menuMethod(gr);
            case GAMEOVER  -> gameOverMethod(gr);
            default        -> errorLoadMethod(gr);
        }
    }

    //метод регулирующий вызов методов отрисовки окон по нажатию кнопки ESC.
    static void switchMenu(){
        switch (state) {
            case PlAYAREA  -> state = STATES.MENU;
            case GAMEOVER  -> state = STATES.MENU;
            case MENU      -> {
                if (players.size() > 0) state = STATES.PlAYAREA;
```

```
            }
        }
    }

    //метод регулирующий количество игроков на игровом поле.
    private void spawn(int count) {
        players.clear();
        for (byte i = 0; i < count; i ++) {
            players.add(new Player(300 + 300 * i));
        }
        state =  STATES.PlAYAREA;if (FinalScore() > localBetterScore) {
            localBetterScore = FinalScore();
        }
        Player.setCountActPlayer((byte)count);
        score = 0;
        for (Object object: gameObject) {
            object.clear();
        }
    }
    //метод отрисовки игрового цикла
    private void playMethod(Graphics gr) {
        score += difficult/3 + 2;
        gr.setColor(Color.WHITE);
        gr.drawImage(images[0], 0, 0, null);

        for (Player player : players) {
            player.draw(gr);
            for (Object object: gameObject) {
                if (object.getActive()) {
                    if (player.touchObject(object.getType(), object.getX(), object.getY(),
object.getWidth(), object.getHeight())) {
                        object.destruction();
                    }
                }
            }
        }

        if (Object.getCountObjectActive() < difficult/2 + 3) {
            gameObject[random(0, COUNT_MASS_IMG[0] - 1)].start();
        }
        for ( Object object: gameObject) {
            object.draw(gr);
            object.down();
        }

        gr.drawString("Выход в меню [ESC]", 10, 20);
        gr.drawString("Your score: " + FinalScore(), 10, 40);
    }

    //метод отрисовки меню
    private void menuMethod(Graphics gr) {
        gr.drawImage(images[0], 0, 0, null);
```

```
        gr.setFont(font);
        gr.setColor(Color.WHITE);
        gr.drawString("Play", 700, 230);
        for (Button button: buttons){
            button.draw(gr);
            button.touchButton();
        }
    }

    //метод отрисовки окна поражения
    private void gameOverMethod(Graphics gr){
        gr.drawImage(images[0], 0, 0, null);
        gr.drawImage(images[1], 550, 400, null);
        gr.setFont(font);
        gr.setColor(Color.WHITE);
        if (FinalScore() > localBetterScore) {
            localBetterScore = FinalScore();
        }
        gr.drawString("текущий счет: " + FinalScore(), 580, 560);
        gr.drawString(" лучший счет: " + localBetterScore, 580, 610);
        gr.drawString("Выход в меню [ESC]", 580, 840);
    }

    //метод отрисовки окна, предупреждающего об ошибке
    private void errorLoadMethod(Graphics gr){
        gr.setFont(font);
        gr.drawString("Возникли какие либо ошибки", 440, 450);
        buttons[0].draw(gr);
        buttons[0].touchButton();
    }

    //метод генерирующий случайное лисло в заданном диапозоне
    private int random(int min, int max){
        return (int)(Math.random() * (max - min + 1) + min);
    }
    //метод обрабатывающий игровой счет
    private int FinalScore() { return score/20;}
}
```

Приложение 4: Код класса Player.

```
import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import javax.swing.Timer;
/**
 *
 * @author Terrick
```

```
 */
public class Player {
    private static final byte DFT_LIFE        = 4;
    private static final byte DFT_FIRE_FRAME  = 6;
    private static final byte DFT_DSTR_FRAME  = 8;
    private static final byte DFT_SPEED       = 12;
    private static final byte DFT_BOOST       = 2;
    private static final byte DFT_COUNT_EFT   = 9;
    private static final int  DFT_TIME_SPEED  = 10;
    private static final int  DFT_TIME_SHIELD = 5;

    private static  Image[]  lifeImages      = new Image[DFT_LIFE];
    private static  Image[]  fireImages      = new Image[DFT_FIRE_FRAME];
    private static  Image[]  effectImages    = new Image[DFT_COUNT_EFT];
    private static  byte     countActPlayer;

    private byte   life                 = DFT_LIFE-1;
    private int    x                    = Window.getScreenWidth()/2;
    private int    y                    = Window.getScreenHeight() - Window.getScreenHeight()/3;
    private int    speed                = DFT_SPEED;
    private byte   numberFireFrame      = 0;
    private byte   numberDstrFrame      = 0;
    private byte   frameHelper          = 0;
    private boolean active              = true;
    private int    height, width, timeSpeed, timeShield = 0;
    private Timer  timerUpdate;
    private boolean btnLeft, btnRight = false;

    static void setCountActPlayer(byte count) { countActPlayer = count;  }
    public void setBtnLeft(boolean btnLeft)   { this.btnLeft = btnLeft;  }
    public void setBtnRight(boolean btnRight) { this.btnRight = btnRight; }

    Player(int x){
        this.x = x;
        spawn();
    }
    //метод создания игрока
    private void spawn(){
        timerUpdate = new Timer(1000, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (timeShield > 0) timeShield--;
                if (timeSpeed > 0) timeSpeed--;
                else speed = DFT_SPEED;
            }
        });
        timerUpdate.start();
        for (int i = 0; i < DFT_LIFE; i++){
            try { lifeImages[i] = ImageIO.read(new File("images\\gameArea\\player\\life\\" + i +
".png")); }
            catch (IOException exp){
                GameArea.setState(GameArea.STATES.ERRORLOAD);
                break;
```

```
            }
        }
        for (int i = 0; i < DFT_FIRE_FRAME; i++){
            try { fireImages[i] = ImageIO.read(new File("images\\gameArea\\player\\fireFrame\\" + i
+ ".png")); }
            catch (IOException exp){
                GameArea.setState(GameArea.STATES.ERRORLOAD);
                break;
            }
        }
        for (int i = 0; i < DFT_COUNT_EFT; i++){
            try {effectImages[i] = ImageIO.read(new File("images\\gameArea\\player\\effects\\" + i +
".png")); }
            catch (IOException exp){
                GameArea.setState(GameArea.STATES.ERRORLOAD);
                break;
            }
        }
        height = lifeImages[0].getHeight(null);
        width = lifeImages[0].getWidth(null);
    }

    // метод, регулирующий передвижение игрока
    public void move(){
        if (active) {
            if (btnLeft && x > 0){
                x -= speed;
            }
            if (btnRight && x + width < Window.getScreenWidth()) {
                x += speed;
            }
        }
    }

    //метод, отрисовывающий игрока
    public void draw(Graphics gr){
        if (active) {
            move();
            gr.drawImage(lifeImages[life], x, y, null);
            if (numberFireFrame == DFT_FIRE_FRAME) numberFireFrame = 0;
            gr.drawImage(fireImages[numberFireFrame], x + 25, y + height, null);
            numberFireFrame++;
            if (timeShield > 0) {
                gr.drawImage(effectImages[8], x - 30, y - 30, null);
                gr.drawString("shild: " + timeShield, x - 10, y - 40);
            }
            if (timeSpeed > 0) gr.drawString("boost: " + timeSpeed, x - 10, y - 50);
        }
        else drawDestruction(gr);
    }

    //метод, отрисовывающий анимацию разрущения корабля
```

```
    public void drawDestruction(Graphics gr){
        gr.drawImage(lifeImages[life], x, y, null);
        if (numberDstrFrame < DFT_DSTR_FRAME) {
            gr.drawImage(effectImages[numberDstrFrame], x - 70, y - 50, null);

            if (frameHelper == 5) {
                numberDstrFrame++;
                frameHelper = 0;
            }
            frameHelper++;
        }
        if (y < Window.getScreenWidth()) y += Object.getSpeed();
        else if (countActPlayer == 0) {
            GameArea.setState(GameArea.STATES.GAMEOVER);
        }
        timerUpdate.stop();
    }

    //метод, проверяющий активацию коллизии игрока с обьектами.
    // При активации коллизии запускает соответствующее действие
    public Boolean touchObject(Object.TYPE object, int ObjectX, int ObjectY, int ObjectW, int
ObjectH ){
        if (active) {
            if ((x <= ObjectX + ObjectW && x + width >= ObjectX && y + 10 <= ObjectY +
ObjectH*3/4 && y + 10 + height >= ObjectY + ObjectH/4)
                    || (x <=ObjectX + ObjectW*3/4 && x + width >= ObjectX + ObjectW/4 && y + 10
< ObjectY + ObjectH && y + 10 + height >= ObjectY)) {
                switch (object) {
                    case METEORITE -> {
                        if (life > 0) {
                            if (timeShield == 0) life--;
                        }
                    }
                    case HEALTH -> {
                        if (life < DFT_LIFE - 1) life++;
                    }
                    case BOOST -> {
                        speed *= DFT_BOOST;
                        timeSpeed = DFT_TIME_SPEED;
                    }
                    case SHIELD -> {
                        timeShield = DFT_TIME_SHIELD;
                    }
                }

                if (life == 0) {
                    active = false;
                    countActPlayer --;
                }
                return true;
            }
        }
```

```
        return false;
    }
}
```

Приложение 5: Код класса Object.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
/**
 *
 * @author Terrick
 */
class Object {
    public enum TYPE{METEORITE, BOOST, HEALTH, SHIELD}
    private static int     speed             = 6;
    private        int     reloading         = 0;
    private        Boolean active            = false;
    private static int     countObjectActive   = 0;
    private        int     x,y, width, height;
    private        Timer   timerUpdate;
    private        Image   img;
    private        TYPE    type;

    static void    setSpeed(int speed)         { Object.speed = speed;     }
    static int     getCountObjectActive()      { return countObjectActive; }
    static int     getSpeed()                  { return Object.speed;      }
    public boolean getActive()                 { return active;            }
    public TYPE    getType()                   { return type;              }
    public int     getX()                      { return x;                 }
    public int     getY()                      { return y;                 }
    public int     getHeight()                 { return height;            }
    public int     getWidth()                  { return width;             }

    Object(Image img, int number) {
        timerUpdate = new Timer(1000, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (reloading > 0) reloading--;
            }
        });
        timerUpdate.start();
        this.img = img;
        this.height = img.getHeight(null);
        this.width = img.getWidth(null);
        this.x =  random(0,  Window.getScreenWidth() - width );

        switch (number){
            case 7  -> this.type = TYPE.HEALTH;
            case 8  -> this.type = TYPE.SHIELD;
            case 9  -> this.type = TYPE.BOOST;
            default -> this.type = TYPE.METEORITE;
```

```
    }
}

//метод активации объекта
void start() {
    if (!active) {
        if (reloading == 0)
        {
            y = -height;
            x = random(0,  Window.getScreenWidth() - width );
            countObjectActive ++;
            active = true;
        }
    }
}

//метод, передвигающий объект
void down(){
    if (active) {
        y += speed;
        if (y > Window.getScreenHeight() + 50) {
            countObjectActive --;
            active = false;
        }
    }
}
//метод отрисовки
void draw(Graphics gr){
    if (active) {
        gr.drawImage(img, x, y, null);
    }
}
//метод генерации случайного числа в заданном диапозоне
private int random(int min, int max){
    return (int)(Math.random() * (max - min + 1) + min);
}

//метод уничтожения объекта
public void destruction(){
    switch (type){
        case HEALTH  -> reloading = random(5, 15);
        case BOOST   -> reloading = random(5, 15);
        case SHIELD  -> reloading = random(5, 15);
    }
    countObjectActive --;
    active = false;
}

//метод, для деактивации обьектов.
// Используется при повторном запуске игры.
public void clear(){
    countObjectActive = 0;
```

```
        active = false;
    }
}
```

Приложение 6: Код класса Button.

```
import java.awt.*;
/**
 *
 * @author Terrick
 */
public class Button {
    private int    DFT_PADDING = 5;
    private int    DFT_ROUND   = 50;
    private int    x, y, width, height, Left_padding;
    private static MyMouseListener mouse;
    private String label;

    Button(int x, int y, int width, int height, int Left_padding, String label, MyMouseListener
mouse) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.label = label;
        this.Left_padding = Left_padding;
        this.mouse = mouse;
    }

    // метод отрисовки
    public void draw(Graphics gr) {
        gr.setColor(Color.gray);
        gr.fillRoundRect(x, y, width, height, DFT_ROUND, DFT_ROUND);
        gr.setColor(Color.WHITE);
        gr.fillRoundRect(x + DFT_PADDING, y + DFT_PADDING, width - DFT_PADDING * 2,
height - DFT_PADDING * 2, DFT_ROUND - DFT_PADDING, DFT_ROUND -
DFT_PADDING);
        gr.setColor(Color.BLACK);
        gr.drawString(label, x + Left_padding, y + height / 2 + 10);
    }

    //метод, определяющий нажатие на кнопку.
    //регулирует действия по нажатию на кнопку
    public void touchButton() {
        if (mouse.getKey() == 1) {
            if (mouse.getMouseX() > x && mouse.getMouseX() < x + width &&
mouse.getMouseY() > y && mouse.getMouseY() < y + height) {

                switch (label) {
                    case "1 Player"  -> GameArea.setState(GameArea.STATES.PLAY1P);
                    case "2 Players" -> GameArea.setState(GameArea.STATES.PLAY2P);
```

```
                case "3 Players" -> GameArea.setState(GameArea.STATES.PLAY3P);
                case "exit" -> System.exit(0);
                default -> {
                    GameArea.setDifficult(Integer.parseInt(label));
                    Object.setSpeed(3 + Integer.parseInt(label)/3);
                }

            }
        }
    }
}
```

Приложение 7: Код класса MyMouseListener.

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
/**
 *
 * @author Terrick
 */
public class MyMouseListener implements MouseListener {
    private static int    key_;
    private static Point  location;
    private static double mouseX = 0;
    private static double mouseY = 0;

    public int getKey()    { return  key_;        }
    public int getMouseX() { return (int) mouseX; }
    public int getMouseY() { return (int) mouseY; }


    @Override
    public void mouseClicked(MouseEvent e) {
    }

    @Override
    public void mousePressed(MouseEvent e) {
        key_ = e.getButton();
        location = MouseInfo.getPointerInfo().getLocation();
        mouseX = location.getX();
        mouseY = location.getY();

    }

    @Override
    public void mouseReleased(MouseEvent e) {
        key_ = 0;
```

```
    }

    @Override
    public void mouseEntered(MouseEvent e) {
    }

    @Override
    public void mouseExited(MouseEvent e) {

    }
}
```

Приложение 8: Код класса MyKeyListener.

```
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;
/**
 *
 * @author Terrick
 */
public class MyKeyListener implements KeyListener {
    private ArrayList<Player> players = GameArea.getPlayers();
    private int[] KEYCODE = {KeyEvent.VK_A, KeyEvent.VK_D, KeyEvent.VK_J,
KeyEvent.VK_L, KeyEvent.VK_LEFT, KeyEvent.VK_RIGHT};
    private int key_;

    public void keyPressed(KeyEvent e) {
        int c = 0;
        for (Player player : players) {
            if (e.getKeyCode() == KEYCODE[c]) {
                player.setBtnLeft(true);
            }
            if (e.getKeyCode() == KEYCODE[c+1]) {
                player.setBtnRight(true);
            }
            c += 2;
        }
    }
    public void keyReleased(KeyEvent e) {
        key_ = e.getKeyCode();

        if (key_ == KeyEvent.VK_ESCAPE) {
            GameArea.switchMenu();
        }

        int c = 0;
        for (Player player : players) {
            if (e.getKeyCode() == KEYCODE[c]) {
                player.setBtnLeft(false);
            }
```

```
            if (e.getKeyCode() == KEYCODE[c+1]) {
                player.setBtnRight(false);
            }
            c += 2;
        }
    }
    public void keyTyped(KeyEvent e) {}
}
```