Dokumentacja projektu „Java Shotgun Game”

Opis ogólny:

Aplikacja „Java Shotgun Game” jak sama nazwa wskazuje jest grą napisaną w języku Java. Jest to gra sieciowa w którą w jednym momencie może grać dowolna ilość graczy. Sama gra polega na strzelaniu do przeciwników za pomocą myszki i unikaniu przy pomocy „odrzutu” własnej broni wszelakich przeszkód oraz pocisków wystrzelonych przez naszych wrogów.

W samej aplikacji zaimplementowane zostały dodatkowe mechanizmy optymalizujące działanie programu, takie jak wyłączenie obracania się obiektów, lub też uproszczenie ich do prostych kwadratów. Gracz może włączyć wspomniane mechanizmy przed rozpoczęciem rozgrywki w razie problemów z utrzymaniem stabilnych klatek na sekundę.

Poza samym strzelaniem w grze możemy również sprawdzić jak idzie nam w porównaniu do graczy którzy przed nami próbowali swoich sił w tej grze. Możemy to zrobić przed samą grą klikając w żółty napis „Scoreboard” w menu głównym, który uruchomi dla nas nową ramkę opisaną w klasie „CScoreboardFrame” lub w trakcie gry przytrzymując na klawiaturze „s” lub „S” co otworzy nam panel opisany w klasie CScoreboardPanel.

Przytrzymując na klawiaturze „p” lub „P” mamy również dostęp do otworzenia listy graczy obecnie znajdujących się na serwerze. (Panel z listą opisany jest w klasie „CPlayerList”)

Gra posiada również wbudowany sklep w którym możemy uleczyć swoją postać, zwiększyć jej obrażenia czy też zmienić broń, jeśli ta wylosowana na samym początku nam nie odpowiada, lub zwyczajnie chcemy spróbować czegoś innego. Dostęp do sklepu znajduje się pod klawiszem „b” oraz „B”.

Krótki opis paczek oraz klas:

1. Package Config
   - Config <- Klasa zawierająca dane o konfiguracji i balansie aplikacji

2. Package Custom_Elements
   - CButton <- Klasa zawierająca instrukcje do tworzenia własnych JButton'ów
   - CImagePanel <- Klasa zawierająca instrukcje do tworzenia panelu rysującego obrazki
   - CLabel <- Klasa zawierająca instrukcje do tworzenia własnych JLabel'ów
   - CSlider <- Klasa zawierająca instrukcje do tworzenia własnych JSlider'ów
   - CTextField <- Klasa zawierająca instrukcje do tworzenia własnych JTextField'ów
   - Sound <- Klasa zajmująca się odtwarzaniem dźwięków w grze

3. Package Custom_Enums
   - ObjectType <- Enum zawierający listę typów obiektów
   - PreMadeType <- Enmu zawierający listę dostępnych kształtów
   - WeaponType <- Enum zawierający listę dostępnych typów broni

4. Package Custom_Frames
   - CPortListFrame <- Klasa wyświetlająca ramkę zawierającą wszystkie dostępne porty
   - CQuizFrame <- Klasa wyświetlająca ramkę z Quizem kiedy postać gracza zostaje zabita
   - CScoreboardFrame <- Klasa wyświetlająca ramkę z listą najlepszych graczy

5. Package Custom_Objects
   - GameObject <- Klasa odpowiadająca za obiekty takie jak gracz, przeciwnik, pułapka czy ulepszacz
   - Packet <- Klasa która jest wymieniana między serwerem a klientem
   - PurchaseOption <- Klasa odpowiedzialna za przedmioty w sklepie
   - QuestObject <- Klasa odpowiedzialna za zadania w grze
   - QuizQuestion <- Klasa odpowiedzialna za pytania do quizu
   - SavedObject <- Klasa odpowiedzialna za elementy „Scoreboard'ów"

6. Package Custom_Panels
   - CConnectionPanel <- Klasa zawierająca instrukcje do tworzenia panelu ustawień sieciowych
   - CMenuPanel <- Klasa zawierająca instrukcje do tworzenia panelu głównego menu programu
   - CPanel <- Klasa zawierająca instrukcje do tworzenia panelu wyświetlającego elementy gry na ekranie
   - CPlayerList <- Klasa zawierająca instrukcje do tworzenia panelu wyświetlającego dostępnych graczy
   - CQuestPanel <- Klasa zawierająca instrukcje do tworzenia panelu wyświetlającego aktualne zadanie
   - CScoreboardPanel <- Klasa zawierająca instrukcje do tworzenia panelu z listą najlepszych graczy
   - CSettingsPanel <- Klasa zawierająca instrukcje do tworzenia panelu ustawień ogólnych
   - CShopPanel <- Klasa zawierająca instrukcje do tworzenia panelu sklepu

7. Package Custom_Polygons
   - CPoint <- Klasa dzięki której możemy tworzyć punkty których obracanie na osi jest możliwe
   - CPolygon <- Klasa dzięki której możemy tworzyć poligony z własnych punktów

8. Package Main_Package
   - Main <- Główna klasa programu

9. Package Online_Modules
   - ClientModule <- Moduł sieciowy dla Klienta
   - HostModule <- Główna klasa serwera (hosta)
   - ServerModule <- Moduł sieciowy dla Serwera

Klasa „CONFIG”:

```java
package Config;

import java.util.UUID;

public class CONFIG {
    // GENERAL CONFIG
    public static int BACKGROUND_VOLUME = 50;
    public static int SHOOTING_VOLUME = 50;
    public static boolean ANIMATION_ROTATION = true;
    public static boolean SIMPLE_SHAPES = false;
    public static final UUID CLIENT_UUID = UUID.randomUUID();
    public static final int WINDOW_WIDTH = 1920; // WINDOW WIDTH
    public static final int WINDOW_HEIGHT = 1080; // WINDOW HEIGHT
    public static final int PORT_WINDOW_WIDTH = 500; // WINDOW WIDTH
    public static final int PORT_WINDOW_HEIGHT = 750; // WINDOW HEIGHT
    public static final int SCOREBOARD_WINDOW_WIDTH = 500; // WINDOW WIDTH
    public static final int SCOREBOARD_WINDOW_HEIGHT = 750; // WINDOW HEIGHT
    public static final int QUIZ_WINDOW_HEIGHT = 360;
    public static final int QUIZ_WINDOW_WIDTH = 640;
    public static final double RECOIL_POWER = 5; // MAX VALUE OF REVERSE VELOCITY AFTER SHOOT
    public static final double AMOUNT_OF_TICKS = 60.0; // MAX TICKS PER SECOND
    public static final double AMOUNT_OF_FRAMES = 120.0; // MAX FRAMES PER SECOND
    public static int PORT = 7777; // MULTIPLAYER PORT
    public static String SERVER_IP = "localhost";
    public static String CLIENT_NAME = "Offline"; // CLIENT DISPLAY NAME
    public static int PORT_SEARCH_MAX = 7500;
    public static int PORT_SEARCH_MIN = 7450;
    // BOOSTER SETTINGS
    public static final int BOOSTER_HP_BUFF = 25;
    public static final int BOOSTER_DAMAGE_BUFF = 5;
    // OBJECT DRAWING
    public static final int SIZE_PLAYER = 5;
    public static final int SIZE_ENEMY = 4;
    public static final int SIZE_LARGE_BULLET = 3;
    public static final int SIZE_NORMAL_BULLET = 2;
    public static final int SIZE_SMALL_BULLET= 1;
    public static final int SIZE_BOOSTER = 4;
    public static final int SIZE_TRAP = 4;
    // WEAPONS
    public static final int SHOTGUN_RELOAD = 1;
    public static final int SNIPER_RELOAD = 1;
    public static final int PISTOL_RELOAD = 1;
    public static final int RIFLE_RELOAD = 1;
    // SHOP PRICES
    // 1. SHOTGUN || 2. SNIPER || 3. PISTOL || 4.RIFLE //
    // 5. HEAL || 6. DAMAGE //
    public static final int[] SHOP_PRICES = {100,100,100,100,100,100};
    // SHOP BOOSTS
    public static final double HEAL_AMOUNT = 100;
    public static final double DAMAGE_BOOST_AMOUNT = 100;
    // Damage
    public static final double SHOTGUN_PRIME_BULLET = 1.25;
    public static final double SHOTGUN_SECONDARY_BULLET = 0.75;
    public static final double SHOTGUN_TERTIARY_BULLET = 0.25;
    public static final double SNIPER_BULLET = 3.75;
    public static final double PISTOL_BULLET = 0.75;
    public static final double RIFLE_BULLET = 0.50;

}
```

Klasa CONFIG wypełniona jest statycznymi stałymi odpowiadającymi za balans gry, oraz jej ogólne działanie.

Klasa „Main”:

```
package Main_Package;

import Custom_Elements.Sound;
import Custom_Enums.ObjectType;
import Custom_Enums.WeaponType;
import Custom_Frames.CPortListFrame;
import Custom_Frames.CScoreboardFrame;
import Custom_Objects.*;
import Custom_Panels.*;
import Config.*;
import Online_Modules.*;
import Custom_Frames.CQuizFrame;

import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class Main extends JFrame implements ActionListener, MouseListener, DocumentListener, KeyListener, ChangeListener {
    public static QuestObject activeQuestObject;
    public static GameObject playerHolder;
    public CPanel cPanel;
    public CMenuPanel cMenuPanel;
    public CConnectionPanel cConnectionPanel;
    public CSettingsPanel cSettingsPanel;
    public CQuizFrame lastChanceQuiz;
    public static boolean displayPlayerList = false;
    public static boolean displayScoreboard = false;
    public static boolean displayShop = false;
    public static int MyPlayerIndex = 0;
    public static Main GAME_INSTANCE;
    private static ClientModule CLIENT_MODULE_INSTANCE;
    public static boolean NEW_LIST_UPDATE = true; // Tells if CustomPanels.CPanel needs to update its own lists
    public static boolean ONLINE_MODE = false, GAME_RUNNING = false;
    public static List<GameObject> PlayerList = new ArrayList<>(); // List containing all clients
    public static List<GameObject> BulletList = new ArrayList<>(); // List containing all bullets
    public static List<GameObject> EnemyList = new ArrayList<>();  // List containing all enemies
    public static List<GameObject> TrapList = new ArrayList<>();  // List containing all enemies
    public static List<GameObject> BoosterList = new ArrayList<>();  // List containing all enemies
    public static Packet PacketToSend = new Packet();

    Main(){
        super("Swing Shotgun Game!");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(0,0, CONFIG.WINDOW_WIDTH, CONFIG.WINDOW_HEIGHT);
        setResizable(false);
        Image backgroundImage = null;
        try {
            backgroundImage = ImageIO.read(new File("src/Package_Images/Background.png"));
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }

        cPanel = new CPanel(backgroundImage);
        cPanel.setLayout(new GridLayout(0,3));

        activeQuestObject = new QuestObject(30,30,20,30,30,30);

        cMenuPanel = new CMenuPanel();
        cConnectionPanel = new CConnectionPanel();
        cSettingsPanel = new CSettingsPanel();
        lastChanceQuiz = new CQuizFrame();
        this.addKeyListener(this);

        cPanel.addMouseListener(this); // Add Mouse Listener

        cSettingsPanel.AnimationButton.addActionListener(this); // Add Action Listener for Animation Button
        cSettingsPanel.SimplerShapesButton.addActionListener(this); // Add Action Listener for Shapes Button
        cSettingsPanel.backgroundVolumeSlider.addChangeListener(this);
        cSettingsPanel.shootingVolumeSlider.addChangeListener(this);

        cMenuPanel.StartGameButton.addActionListener(this); // Add Action Listener for Start Game Button
        cMenuPanel.OnlineConfigButton.addActionListener(this); // Add Action Listener for Online Config Button
        cMenuPanel.SettingsButton.addActionListener(this); // Add Action Listener for Settings Button
        cMenuPanel.FullScoreboardButton.addActionListener(this); // Add Action Listener for Settings Button

        cConnectionPanel.JoinServerButton.addActionListener(this); // Add Action Listener for Join Server Button
        cConnectionPanel.StartSearchButton.addActionListener(this); // Add Action Listener for Start Search Button
        cConnectionPanel.PortInput.getDocument().addDocumentListener(this);  // Add Document Listener for Port Input

        lastChanceQuiz.submitButton.addActionListener(this);

        cPanel.add(new JLabel());
        cPanel.add(cMenuPanel);
        cPanel.add(new JLabel());

        try {
            CShopPanel.shopkeeper = ImageIO.read(new File("src/Package_Images/Shopkeeper.png"));
        }
```

```
        catch (Exception exception) {
            exception.printStackTrace();
        }

        CShopPanel.UpdateShopList();

        setContentPane(cPanel);
        setVisible(true);
    }
    public static void main(String[] args) {
        GAME_INSTANCE = new Main(); // Initialization of game window
    }
    public static void InitGame(){

        if (PlayerList.size() == 0)
            CreatePlayer();

        activeQuestObject.NewQuest();

        // Server Handling
        if (!ONLINE_MODE) {
            // Check For First Free PORT
            while (!HostModule.Available(CONFIG.PORT)){ // If Port is occupied
                CONFIG.PORT++; // Increase port
                if (CONFIG.PORT >= 65535) { // Check if port is greater or equals 65535 (the highest available port)
                    return;
                    // Return if there is no free ports ( Cancels game initialisation)
                    // Search starts from Config value which by default equals 7777
                }
            }
            ONLINE_MODE = true;
            new HostModule(); // Start Server
            CLIENT_MODULE_INSTANCE = new ClientModule(); // Start Client
        }
        GAME_RUNNING = true;

        if (CLIENT_MODULE_INSTANCE != null) {
            while (Objects.equals(CONFIG.CLIENT_NAME, "Offline"))
                continue;
        } // Waiting for getting new name

        try {
            Thread.sleep(1);
        }
        catch (Exception exception) {
            System.out.println("Sleeping Thread Error");
        }

        GAME_INSTANCE.remove(GAME_INSTANCE.cMenuPanel);
        GAME_INSTANCE.remove(GAME_INSTANCE.cConnectionPanel);
        GAME_INSTANCE.remove(GAME_INSTANCE.cSettingsPanel);
        Sound.PlayBackgroundSound();
        StartUpdating();

    }

    public static void Render(){
        GAME_INSTANCE.cPanel.repaint();
    }

    public static void CreatePlayer(){
        playerHolder = new GameObject(500, 500, 10, 100, ObjectType.PLAYER, GAME_INSTANCE.cMenuPanel.NickInput.getText(),
0,0,CONFIG.CLIENT_UUID, ObjectType.GAME, -1);
        PlayerList.add(playerHolder);
        PacketToSend.PlayerList.add(playerHolder);
        MyPlayerIndex = PlayerList.indexOf(playerHolder);
    }

    private static void StartUpdating(){
        new Thread(() -> {
            // FPS Stuff
            long lastTime = System.nanoTime();
            double nsRender = 1000000000.0 / CONFIG.AMOUNT_OF_FRAMES;
            double nsUpdate = 1000000000.0 / CONFIG.AMOUNT_OF_TICKS;
            double deltaRender = 0;
            double deltaUpdate = 0;
            int updates = 0;
            int frames = 0;
            long timer = System.currentTimeMillis();

            while(true) {
                long now = System.nanoTime();

                deltaRender += (now - lastTime) / nsRender;
                deltaUpdate += (now - lastTime) / nsUpdate;
                lastTime = now;

                if(deltaRender >= 1) {
                    deltaRender--;
                    Render();
                    frames++;
                }

                if (deltaUpdate >= 1) {
                    deltaUpdate--;
                    updates++;
                    Update();
                    Collision();
                }

                if(System.currentTimeMillis() - timer > 1000) {
                    timer += 1000;
                    GAME_INSTANCE.setTitle("PORT: " + CONFIG.PORT + " Ticks: " + updates + ", FPS: " + frames + ", " +
```

```
CONFIG.CLIENT_NAME);
                    updates = 0;
                    frames = 0;
                }
            }
        }).start();
    }
    public static void Update() {

        List<GameObject> temp = new ArrayList<>();
        temp.addAll(EnemyList);
        temp.addAll(BulletList);
        temp.addAll(PlayerList);
        temp.addAll(BoosterList);
        temp.addAll(TrapList);

        Point WindowCenter = new Point(CONFIG.WINDOW_WIDTH/2, CONFIG.WINDOW_HEIGHT/2);
        Point Temp = new Point(0,0);

        for (GameObject gameObject : temp) {

            gameObject.Move();

            if (gameObject.myType == ObjectType.ENEMY || gameObject.myType == ObjectType.PLAYER) {
                gameObject.reloadTime--;

                if (gameObject.velY < 5)
                    gameObject.velY += 0.2;

            }

            try {
                Temp.x = (int) gameObject.cordX;
                Temp.y = (int) gameObject.cordY;
                if (WindowCenter.distance(Temp) > 3000) {
                    switch (gameObject.myType) {
                        case ENEMY -> EnemyList.remove(gameObject);
                        case TRAP -> TrapList.remove(gameObject);
                        case BOOSTER -> BoosterList.remove(gameObject);
                        case BULLET_LARGE, BULLET_SMALLER, BULLET_NORMAL -> BulletList.remove(gameObject);
                    }
                }
            } catch (Exception exception) {
                exception.printStackTrace();
            }
        }

        activeQuestObject.QuestTracker();

    }

    public static void Collision(){

        try {

            List<GameObject> threatList = new ArrayList<>();
            List<GameObject> targetList = new ArrayList<>();

            threatList.addAll(BulletList);
            threatList.addAll(BoosterList);
            threatList.addAll(TrapList);

            targetList.addAll(PlayerList);
            targetList.addAll(EnemyList);

            boolean collision;

            if (threatList.size() > 0) {

                for (GameObject threat : threatList) {
                    for (GameObject target : targetList) {

                        collision = false;

                        if (threat.srcUUID.compareTo(target.myUUID) != 0)  // No Self Harm
                            if (threat.myCPolygon.GetPoly().intersects(target.myCPolygon.GetPoly().getBounds2D())) // Check
For Collision
                                collision = true;

                        if (collision) {

                            if (target.myType == ObjectType.PLAYER && threat.srcObjectType == ObjectType.PLAYER)
                                continue;

                            if (target.myType == ObjectType.ENEMY && threat.srcObjectType == ObjectType.ENEMY)
                                continue;

                            if (target.myType == ObjectType.ENEMY && threat.myType == ObjectType.TRAP)
                                continue;

                            if (target.hpCurrent - threat.damage < 0)
                                target.visible = false;

                            threat.visible = false;
                        }
                    }
                }
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    } // Collision Function
```

```
    // End of Gameplay Section

    public static void LockMenus(){
        GAME_INSTANCE.cConnectionPanel.PortInput.setBackground(Color.ORANGE);
        GAME_INSTANCE.cConnectionPanel.PortInput.setEditable(false);
        GAME_INSTANCE.cConnectionPanel.AddressInput.setBackground(Color.ORANGE);
        GAME_INSTANCE.cConnectionPanel.AddressInput.setEditable(false);
        GAME_INSTANCE.cConnectionPanel.JoinServerButton.setBackground(Color.ORANGE);
        GAME_INSTANCE.cMenuPanel.StartGameButton.setBackground(Color.GREEN);
    }

    public static void MakePurchase(int Variant) {
        // 1 Shotgun, 2 Sniper, 3 Pistol, 4 Rifle, 5 Heal, 6 Damage
        switch (Variant) {
            case 0 -> {
                if (PlayerList.get(MyPlayerIndex).CanAfford(CONFIG.SHOP_PRICES[0])) {
                    PlayerList.get(MyPlayerIndex).Charge(CONFIG.SHOP_PRICES[0]);
                    PlayerList.get(MyPlayerIndex).myWeapon = WeaponType.SHOTGUN;
                    Main.PacketToSend.PlayerList.add(PlayerList.get(MyPlayerIndex));
                }
            }
            case 1 -> {
                if (PlayerList.get(MyPlayerIndex).CanAfford(CONFIG.SHOP_PRICES[1])) {
                    PlayerList.get(MyPlayerIndex).Charge(CONFIG.SHOP_PRICES[1]);
                    PlayerList.get(MyPlayerIndex).myWeapon = WeaponType.SNIPER_RIFLE;
                    Main.PacketToSend.PlayerList.add(PlayerList.get(MyPlayerIndex));
                }
            }
            case 2 -> {
                if (PlayerList.get(MyPlayerIndex).CanAfford(CONFIG.SHOP_PRICES[2])) {
                    PlayerList.get(MyPlayerIndex).Charge(CONFIG.SHOP_PRICES[2]);
                    PlayerList.get(MyPlayerIndex).myWeapon = WeaponType.PISTOL;
                    Main.PacketToSend.PlayerList.add(PlayerList.get(MyPlayerIndex));
                }
            }
            case 3 -> {
                if (PlayerList.get(MyPlayerIndex).CanAfford(CONFIG.SHOP_PRICES[3])) {
                    PlayerList.get(MyPlayerIndex).Charge(CONFIG.SHOP_PRICES[3]);
                    PlayerList.get(MyPlayerIndex).myWeapon = WeaponType.RIFLE;
                    Main.PacketToSend.PlayerList.add(PlayerList.get(MyPlayerIndex));
                }
            }
            case 4 -> {
                if (PlayerList.get(MyPlayerIndex).CanAfford(CONFIG.SHOP_PRICES[4])) {
                    PlayerList.get(MyPlayerIndex).Charge(CONFIG.SHOP_PRICES[4]);
                    PlayerList.get(MyPlayerIndex).hpCurrent += CONFIG.HEAL_AMOUNT;
                    if (PlayerList.get(MyPlayerIndex).hpCurrent > 100)
                        PlayerList.get(MyPlayerIndex).hpCurrent = 100;
                    Main.PacketToSend.PlayerList.add(PlayerList.get(0));
                }
            }
            case 5 -> {
                if (PlayerList.get(MyPlayerIndex).CanAfford(CONFIG.SHOP_PRICES[5])) {
                    PlayerList.get(MyPlayerIndex).Charge(CONFIG.SHOP_PRICES[5]);
                    PlayerList.get(MyPlayerIndex).damage += CONFIG.DAMAGE_BOOST_AMOUNT;
                    Main.PacketToSend.PlayerList.add(PlayerList.get(0));
                }
            }
        }
    }

    private void PortCheck() {
        try {
            int tempPort = Integer.parseInt(GAME_INSTANCE.cConnectionPanel.PortInput.getText());
            if (tempPort > 65535 || tempPort <= 0) {
                GAME_INSTANCE.cConnectionPanel.JoinServerButton.setBackground(Color.RED);
            }
            else {
                CONFIG.PORT = tempPort;
                if (HostModule.Available(CONFIG.PORT)) {
                    GAME_INSTANCE.cConnectionPanel.JoinServerButton.setBackground(Color.RED);
                }
                else {
                    GAME_INSTANCE.cConnectionPanel.JoinServerButton.setBackground(Color.GREEN);
                }
            }
        }
        catch (Exception ignored) {
            System.out.println("WRONG PORT INPUTTED");
        }
    }

    @Override
    public void actionPerformed(ActionEvent e) {

        Object source = e.getSource();

        if (source == cSettingsPanel.AnimationButton) {
            CONFIG.ANIMATION_ROTATION = !CONFIG.ANIMATION_ROTATION;
            if (CONFIG.ANIMATION_ROTATION) {
                cSettingsPanel.AnimationButton.setText("Turn Off Animations");
                cSettingsPanel.AnimationButton.setBackground(Color.GREEN);
            }
            else {
                cSettingsPanel.AnimationButton.setText("Turn On Animations");
                cSettingsPanel.AnimationButton.setBackground(Color.RED);
            }
        }
        else if (source == cSettingsPanel.SimplerShapesButton) {
            CONFIG.SIMPLE_SHAPES = !CONFIG.SIMPLE_SHAPES;
            if (CONFIG.SIMPLE_SHAPES) {
                CONFIG.ANIMATION_ROTATION = false;
                cSettingsPanel.AnimationButton.setText("Turn On Animations");
```

```
                cSettingsPanel.AnimationButton.setBackground(Color.RED);

                cSettingsPanel.SimplerShapesButton.setText("Turn Off Simple Shapes");
                cSettingsPanel.SimplerShapesButton.setBackground(Color.GREEN);
            }
            else {
                CONFIG.ANIMATION_ROTATION = true;
                cSettingsPanel.AnimationButton.setText("Turn Off Animations");
                cSettingsPanel.AnimationButton.setBackground(Color.GREEN);

                cSettingsPanel.SimplerShapesButton.setText("Turn On Simple Shapes");
                cSettingsPanel.SimplerShapesButton.setBackground(Color.RED);
            }
        }
        else if (source == cMenuPanel.StartGameButton) {
            if (!GAME_RUNNING)
                InitGame();
        }
        else if (source == cMenuPanel.SettingsButton) {
            cPanel.removeAll();
            cSettingsPanel.visible = !cSettingsPanel.visible;

            if (cSettingsPanel.visible)
                cPanel.add(cSettingsPanel);
            else
                cPanel.add(new JLabel());

            cPanel.add(cMenuPanel);

            if (cConnectionPanel.visible)
                cPanel.add(cConnectionPanel);
            else
                cPanel.add(new JLabel());

            cPanel.revalidate();
            cPanel.repaint();
        }
        else if (source == cMenuPanel.OnlineConfigButton) {
            cPanel.removeAll();
            cConnectionPanel.visible = !cConnectionPanel.visible;

            if (cSettingsPanel.visible)
                cPanel.add(cSettingsPanel);
            else
                cPanel.add(new JLabel());

            cPanel.add(cMenuPanel);

            if (cConnectionPanel.visible)
                cPanel.add(cConnectionPanel);
            else
                cPanel.add(new JLabel());

            cPanel.revalidate();
            cPanel.repaint();
        }
        else if(source == cMenuPanel.FullScoreboardButton) {
            new CScoreboardFrame();
        }
        else if (source == cConnectionPanel.JoinServerButton) {
            if (!ONLINE_MODE) {
                if (!HostModule.Available(CONFIG.PORT)) {
                    CreatePlayer();
                    ONLINE_MODE = true;
                    CLIENT_MODULE_INSTANCE = new ClientModule();
                    LockMenus();
                }
                else {
                    System.out.println("THERE IS NO SERVER ON THIS PORT");
                }
            }
        }
        else if (source == cConnectionPanel.StartSearchButton) {
            try {
                new CPortListFrame(Integer.parseInt(cConnectionPanel.PortSearchStartInput.getText()),
Integer.parseInt(cConnectionPanel.PortSearchEndInput.getText()));
            }
            catch (Exception ignored) {
                System.out.println("WRONG PORT INPUTTED");
            }
        }
        else if (source == lastChanceQuiz.submitButton) {

            String string;

            if (lastChanceQuiz.jRadioButtons[0].isSelected())
                string = lastChanceQuiz.jRadioButtons[0].getText();
            else if (lastChanceQuiz.jRadioButtons[1].isSelected())
                string = lastChanceQuiz.jRadioButtons[1].getText();
            else
                string = lastChanceQuiz.jRadioButtons[2].getText();

            if (CQuizFrame.selectedQuestion.CorrectAnswer.compareTo(string) == 0) {
                CreatePlayer();
                lastChanceQuiz.setVisible(false);
                lastChanceQuiz.active = false;
                lastChanceQuiz.SelectQuestion();
            }
            else {
                dispatchEvent(new WindowEvent(GAME_INSTANCE, WindowEvent.WINDOW_CLOSING));
            }

        }
```

```
        GAME_INSTANCE.requestFocus();
    }

    @Override
    public void mousePressed(MouseEvent e) {

        if (Main.PlayerList.size() > 0) {
            Main.PlayerList.get(MyPlayerIndex).Shoot(e.getX(), e.getY());
        }

        if (Main.displayShop) {
            for (int i = 0; i < CShopPanel.fakeButtons.size(); i++) {
                if (CShopPanel.fakeButtons.get(i).intersects(new Rectangle(e.getX(), e.getY(), 10, 10))) {
                    MakePurchase(i);
                }
            }
        }
        GAME_INSTANCE.requestFocus();
    }

    @Override
    public void mouseClicked(MouseEvent e) {}

    @Override
    public void mouseReleased(MouseEvent e) {}

    @Override
    public void mouseEntered(MouseEvent e) {}

    @Override
    public void mouseExited(MouseEvent e) {}

    @Override
    public void insertUpdate(DocumentEvent e) {
        PortCheck();
    }

    @Override
    public void removeUpdate(DocumentEvent e) {
        PortCheck();
    }

    @Override
    public void changedUpdate(DocumentEvent e) {
        PortCheck();
    }

    @Override
    public void keyTyped(KeyEvent e) {

    }

    @Override
    public void keyPressed(KeyEvent e) {

        if (e.getKeyChar() == 'p' || e.getKeyChar() == 'P') {
            if (!displayPlayerList) {
                displayPlayerList = true;
            }
        }
        else if (e.getKeyChar() == 's' || e.getKeyChar() == 'S') {
            if (!displayScoreboard) {
                displayScoreboard = true;
            }
        }
        else if (e.getKeyChar() == 'b' || e.getKeyChar() == 'B') {
            displayShop = !displayShop;
        }
    }

    @Override
    public void keyReleased(KeyEvent e) {
        if (e.getKeyChar() == 'p' || e.getKeyChar() == 'P') {
            if (displayPlayerList) {
                displayPlayerList = false;
            }
        }
        else if (e.getKeyChar() == 's' || e.getKeyChar() == 'S') {
            if (displayScoreboard) {
                displayScoreboard = false;
            }
        }
    }

    @Override
    public void stateChanged(ChangeEvent e) {
        if (e.getSource() == cSettingsPanel.backgroundVolumeSlider) {
            CONFIG.BACKGROUND_VOLUME = 100 - cSettingsPanel.backgroundVolumeSlider.getValue();
            if (CONFIG.BACKGROUND_VOLUME > 80)
                CONFIG.BACKGROUND_VOLUME = 80;
        }
        else if (e.getSource() == cSettingsPanel.shootingVolumeSlider) {
            CONFIG.SHOOTING_VOLUME = 100 - cSettingsPanel.shootingVolumeSlider.getValue();
            if (CONFIG.SHOOTING_VOLUME > 80)
                CONFIG.SHOOTING_VOLUME = 80;
        }
    }
}
```

Najważniejsze elementy klasy „Main”

Funkcja „StartUpdating”:

```
private static void StartUpdating(){
    new Thread(() -> {
        // FPS Stuff
        long lastTime = System.nanoTime();
        double nsRender = 1000000000.0 / CONFIG.AMOUNT_OF_FRAMES;
        double nsUpdate = 1000000000.0 / CONFIG.AMOUNT_OF_TICKS;
        double deltaRender = 0;
        double deltaUpdate = 0;
        int updates = 0;
        int frames = 0;
        long timer = System.currentTimeMillis();

        while(true) {
            long now = System.nanoTime();

            deltaRender += (now - lastTime) / nsRender;
            deltaUpdate += (now - lastTime) / nsUpdate;
            lastTime = now;

            if(deltaRender >= 1) {
                deltaRender--;
                Render();
                frames++;
            }

            if (deltaUpdate >= 1) {
                deltaUpdate--;
                updates++;
                Update();
                Collision();
            }

            if(System.currentTimeMillis() - timer > 1000) {
                timer += 1000;
                GAME_INSTANCE.setTitle("PORT: " + CONFIG.PORT + " Ticks: " + updates + ", FPS: " + frames + ", " +
CONFIG.CLIENT_NAME);
                updates = 0;
                frames = 0;
            }
        }
    }).start();
}
```

Funkcja „StartUpdating” jest najważniejszą funkcją w całym programie. Jest ona tak zwanym game-loop'em, odpowiada za wywoływanie aktualizacji gry oraz rysowanie obiektów na ekranie.

Funkcja „Update”:

```
public static void Update() {

    List<GameObject> temp = new ArrayList<>();
    temp.addAll(EnemyList);
    temp.addAll(BulletList);
    temp.addAll(PlayerList);
    temp.addAll(BoosterList);
    temp.addAll(TrapList);

    Point WindowCenter = new Point(CONFIG.WINDOW_WIDTH/2, CONFIG.WINDOW_HEIGHT/2);
    Point Temp = new Point(0,0);

    for (GameObject gameObject : temp) {

        gameObject.Move();

        if (gameObject.myType == ObjectType.ENEMY || gameObject.myType == ObjectType.PLAYER) {
            gameObject.reloadTime--;

            if (gameObject.velY < 5)
                gameObject.velY += 0.2;

        }

        try {
            Temp.x = (int) gameObject.cordX;
            Temp.y = (int) gameObject.cordY;
            if (WindowCenter.distance(Temp) > 3000) {
                switch (gameObject.myType) {
                    case ENEMY -> EnemyList.remove(gameObject);
```

```
                    case TRAP -> TrapList.remove(gameObject);
                    case BOOSTER -> BoosterList.remove(gameObject);
                    case BULLET_LARGE, BULLET_SMALLER, BULLET_NORMAL -> BulletList.remove(gameObject);
                }
            }
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }

    activeQuestObject.QuestTracker();

}
```

Funkcja „Update" domyślnie wywoływana jest przez wątek tworzony w funkcji „StartUpdating" 60 razy na sekundę. Odpowiada ona za aktualizację pozycji, czasu przeładowywania oraz prędkości poruszania wszystkich obiektów w programie.

Funkcja „Collision":

```
public static void Collision(){

    try {

        List<GameObject> threatList = new ArrayList<>();
        List<GameObject> targetList = new ArrayList<>();

        threatList.addAll(BulletList);
        threatList.addAll(BoosterList);
        threatList.addAll(TrapList);

        targetList.addAll(PlayerList);
        targetList.addAll(EnemyList);

        boolean collision;

        if (threatList.size() > 0) {

            for (GameObject threat : threatList) {
                for (GameObject target : targetList) {

                    collision = false;

                    if (threat.srcUUID.compareTo(target.myUUID) != 0)  // No Self Harm
                        if (threat.myCPolygon.GetPoly().intersects(target.myCPolygon.GetPoly().getBounds2D())) // Check For
Collision
                            collision = true;

                    if (collision) {

                        if (target.myType == ObjectType.PLAYER && threat.srcObjectType == ObjectType.PLAYER)
                            continue;

                        if (target.myType == ObjectType.ENEMY && threat.srcObjectType == ObjectType.ENEMY)
                            continue;

                        if (target.myType == ObjectType.ENEMY && threat.myType == ObjectType.TRAP)
                            continue;

                        if (target.hpCurrent - threat.damage < 0)
                            target.visible = false;

                        threat.visible = false;
                    }
                }
            }
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
} // Collision Function
```

Jak sama nazwa wskazuje funkcja odpowiada za sprawdzanie kolizji między obiektami. Jest to bardzo prosta wersja takowej funkcji, ponieważ w przeciwieństwie do serwerowej, klient'owa wersja tej funkcji jedynie ukrywa trafiony obiekt jeśli został on trafiony, ja poziom zdrowia spadł poniżej 0.

Funkcja nie usuwa żadnych obiektów a jedynie ukrywa je by w jak najlepszy sposób uniknąć szansy na desynchronizację z serwerem. W razie błędu serwer prześle pakiet zawierający poprawne dane i obiekt ponownie stanie się widzialny. Jeśli jednak pakiet popierający decyzję Klienta obiekt zostanie usunięty.

Funkcja „InitGame”:

```
public static void InitGame(){

    if (PlayerList.size() == 0)
        CreatePlayer();

    activeQuestObject.NewQuest();

    // Server Handling
    if (!ONLINE_MODE) {
        // Check For First Free PORT
        while (!HostModule.Available(CONFIG.PORT)){ // If Port is occupied
            CONFIG.PORT++; // Increase port
            if (CONFIG.PORT >= 65535) { // Check if port is greater or equals 65535 (the highest available port)
                return;
                // Return if there is no free ports ( Cancels game initialisation)
                // Search starts from Config value which by default equals 7777
            }
        }
        ONLINE_MODE = true;
        new HostModule(); // Start Server
        CLIENT_MODULE_INSTANCE = new ClientModule(); // Start Client
    }
    GAME_RUNNING = true;

    if (CLIENT_MODULE_INSTANCE != null) {
        while (Objects.equals(CONFIG.CLIENT_NAME, "Offline"))
            continue;
    } // Waiting for getting new name

    try {
        Thread.sleep(1);
    }
    catch (Exception exception) {
        System.out.println("Sleeping Thread Error");
    }

    GAME_INSTANCE.remove(GAME_INSTANCE.cMenuPanel);
    GAME_INSTANCE.remove(GAME_INSTANCE.cConnectionPanel);
    GAME_INSTANCE.remove(GAME_INSTANCE.cSettingsPanel);
    Sound.PlayBackgroundSound();
    StartUpdating();

}
```

Funkcja odpowiada za inicjalizację gry po wciśnięciu przycisku „Play”.

Funkcja początkowo sprawdza czy gracz został już utworzony i czy aplikacja jest w trybie online. Jest tak, ponieważ w programie można włączyć sam serwer bez uruchamiania samej gry, lub dołączenia do już istniejącej rozgrywki.

W przypadku pierwszym kiedy próbujemy utworzyć serwer jeśli jakiś zajmuje dany port możemy się spotkać z błędem uniemożliwiającym dalszą pracę programu.

W drugim przypadku gdyby program nie sprawdzał czy obiekt gracza został już utworzony, klient wysłałby dwa obiekty co skutkowałoby pojawieniem się

dodatkowego, zbędnego gracza na mapie który nie byłby kontrolowanego przez nikogo.

Klasa „HostModule”

```
package Online_Modules;

import Config.CONFIG;
import Custom_Enums.ObjectType;
import Custom_Objects.GameObject;
import Custom_Objects.Packet;
import Custom_Objects.SavedObject;

import javax.swing.*;
import java.awt.*;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.*;
import java.util.List;

public class HostModule extends JFrame {
    public static List<ServerModule> threadList;
    private int clientCount = 0;
    private static int packSize = 1;
    private static final UUID myUUID = UUID.randomUUID();
    public static Packet GlobalPackage;
    public static Random randomizer = new Random();
    public HostModule() {

        super("Sample");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(0,0,400,900);
        setVisible(true);
        setResizable(false);
        GlobalPackage = new Packet();

        threadList = new ArrayList<>();

        ServerSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(CONFIG.PORT);
        } catch (Exception exception) {
            exception.printStackTrace();
        }

        ServerSocket finalServerSocket = serverSocket;

        StartUpdating();

        new Thread(() -> {
            while (true) {
                Socket socket;
                try {

                    System.out.println("Serwer czeka na klienta...");
                    socket = finalServerSocket.accept();
                    System.out.println("Serwer widzi nowego klienta");
                    clientCount++;

                    ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
                    ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());

                    ServerModule serverModule = new ServerModule(socket, ois, oos, clientCount);
                    threadList.add(serverModule);

                    serverModule.packetToSend.BulletList.addAll(GlobalPackage.BulletList);
                    serverModule.packetToSend.PlayerList.addAll(GlobalPackage.PlayerList);
                    serverModule.packetToSend.EnemyList.addAll(GlobalPackage.EnemyList);
                    serverModule.packetToSend.TrapList.addAll(GlobalPackage.TrapList);
                    serverModule.packetToSend.BoosterList.addAll(GlobalPackage.BoosterList);

                    setTitle("Current Client Count: " + clientCount);

                    Spawning();

                } catch (Exception exception) {
                    exception.printStackTrace();
                    System.out.println("HM_001");
                }

            }
        }).start();

    }
    // Gameplay Section
    private static void StartUpdating(){
        new Thread(() -> {
            // FPS Stuff
            long lastTime = System.nanoTime();
            double nsUpdate = 1000000000.0 / CONFIG.AMOUNT_OF_TICKS;
```

```
            double deltaUpdate = 0;

            long timer = System.currentTimeMillis();

            while(true) {
                long now = System.nanoTime();

                deltaUpdate += (now - lastTime) / nsUpdate;
                lastTime = now;

                if (deltaUpdate >= 1) {
                    deltaUpdate--;
                    Update(); // Call Update Function to update positions
                    Collision(); // Call Collision Function to check for collisions
                }

                if(System.currentTimeMillis() - timer > 1000) {
                    timer += 1000;
                }
            }
        }).start();
    } // Start Updating Thread
    public static void Update() {

        List<GameObject> temp = new ArrayList<>();
        temp.addAll(GlobalPackage.EnemyList);
        temp.addAll(GlobalPackage.BulletList);
        temp.addAll(GlobalPackage.PlayerList);
        temp.addAll(GlobalPackage.BoosterList);
        temp.addAll(GlobalPackage.TrapList);

        Point WindowCenter = new Point(CONFIG.WINDOW_WIDTH/2, CONFIG.WINDOW_HEIGHT/2);
        Point Temp = new Point(0,0);

        for (GameObject gameObject : temp) {

            gameObject.Move();

            if (gameObject.myType == ObjectType.ENEMY || gameObject.myType == ObjectType.PLAYER) {
                gameObject.reloadTime--;

                if (gameObject.velY < 5) {
                    gameObject.velY += 0.2;
                }
            }

            try {
                Temp.x = (int) gameObject.cordX;
                Temp.y = (int) gameObject.cordY;
                if (WindowCenter.distance(Temp) > 3000) {
                    switch (gameObject.myType) {
                        case ENEMY -> GlobalPackage.EnemyList.remove(gameObject);
                        case TRAP -> GlobalPackage.TrapList.remove(gameObject);
                        case BOOSTER -> GlobalPackage.BoosterList.remove(gameObject);
                        case BULLET_LARGE, BULLET_SMALLER, BULLET_NORMAL -> GlobalPackage.BulletList.remove(gameObject);
                    }
                }
            }
            catch (Exception exception) {
                exception.printStackTrace();
            }
        }
    } // Optimisation Function
    public static void Collision(){

        try {

            List<GameObject> threatList = new ArrayList<>();
            List<GameObject> targetList = new ArrayList<>();

            threatList.addAll(GlobalPackage.BulletList);
            threatList.addAll(GlobalPackage.BoosterList);
            threatList.addAll(GlobalPackage.TrapList);

            targetList.addAll(GlobalPackage.PlayerList);
            targetList.addAll(GlobalPackage.EnemyList);

            boolean collision;

            if (threatList.size() > 0) {

                for (GameObject threat : threatList) {
                    for (GameObject target : targetList) {

                        collision = false;

                        if (threat.srcUUID.compareTo(target.myUUID) != 0)  // No Self Harm
                            if (threat.myCPolygon.GetPoly().intersects(target.myCPolygon.GetPoly().getBounds2D())) // Check
For Collision
                                collision = true;

                        if (collision) {

                            if (target.myType == ObjectType.PLAYER && threat.srcObjectType == ObjectType.PLAYER)
                                continue;

                            if (target.myType == ObjectType.ENEMY && threat.srcObjectType == ObjectType.ENEMY)
                                continue;

                            if (target.myType == ObjectType.ENEMY && threat.myType == ObjectType.TRAP)
                                continue;

                            threat.hpCurrent = -1;
```

```
                            target.hpCurrent -= threat.damage;
                            target.statDamageTaken += threat.damage;



                            if (threat.myType == ObjectType.BOOSTER) {
                                target.damage += CONFIG.BOOSTER_DAMAGE_BUFF;
                                target.hpCurrent += CONFIG.BOOSTER_HP_BUFF;

                                if (target.hpCurrent > 100)
                                    target.hpCurrent = 100;
                            }

                            boolean additionalPlayerPackage = false;

                            if (threat.srcObjectType == ObjectType.PLAYER) {
                                try {
                                    GlobalPackage.PlayerList.get(threat.SrcIndexInGlobalList).statDamageDealt +=
threat.damage;

                                    additionalPlayerPackage = true;
                                }
                                catch (Exception ignored){
                                    // Threat source might not be in game anymore
                                }
                            }

                            if (target.hpCurrent <= 0) {
                                switch (target.myType) {
                                    case ENEMY -> {
                                        GlobalPackage.EnemyList.remove(target);
                                        if (threat.srcObjectType == ObjectType.PLAYER) {
                                            if (GlobalPackage.PlayerList.size() > 0) {
                                                try {

GlobalPackage.PlayerList.get(threat.SrcIndexInGlobalList).statKillCount++;
                                                    additionalPlayerPackage = true;
                                                }
                                                catch (Exception ignored) {
                                                    // Threat source might not be in game anymore
                                                }
                                            }
                                        }
                                    }
                                    case PLAYER -> {
                                        GlobalPackage.PlayerList.remove(target);
                                        SavedObject.Save(target.objectName, target.MyScore);
                                    }
                                }
                            }

                            for (ServerModule serverModule : threadList) {

                                if (additionalPlayerPackage)

serverModule.packetToSend.PlayerList.add(GlobalPackage.PlayerList.get(threat.SrcIndexInGlobalList));

                                switch (threat.myType) {
                                    case BOOSTER -> {
                                        serverModule.packetToSend.BoosterList.add(threat);
                                        target.statBoosterCount++;
                                    }
                                    case TRAP -> {
                                        serverModule.packetToSend.TrapList.add(threat);
                                        target.statTrapCount++;
                                    }
                                    default -> serverModule.packetToSend.BulletList.add(threat);
                                }

                                switch (target.myType) {
                                    case ENEMY -> serverModule.packetToSend.EnemyList.add(target);
                                    case PLAYER -> serverModule.packetToSend.PlayerList.add(target);
                                }

                            }

                            switch (threat.myType) {
                                case BOOSTER -> GlobalPackage.BoosterList.remove(threat);
                                case TRAP -> GlobalPackage.TrapList.remove(threat);
                                default -> GlobalPackage.BulletList.remove(threat);
                            }
                            if (GlobalPackage.EnemyList.size() == 0)
                                EnemySpawning();
                        }
                    }
                }
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    } // Collision Function
    // End of Gameplay Section

    public static void Spawning(){
        new Thread(() -> {
            while (true) {

                Random random = new Random();

                switch (random.nextInt(3)) {
                    case 0 -> EnemySpawning();
                    case 1 -> BoosterSpawning();
                    case 2 -> TrapSpawning();
```

```
                }

                if (random.nextInt(50) == 1) {
                    packSize++;
                }

                try {
                    Thread.sleep(5000);
                }
                catch (Exception e){
                    e.printStackTrace();
                }

            }
        }).start();
    }
    public static void EnemySpawning(){

        Random random = new Random();

        for (int i = 0; i < packSize; i++) {

            GameObject tempEnemy = new GameObject(random.nextInt(CONFIG.WINDOW_WIDTH - 100) + 50,
random.nextInt(CONFIG.WINDOW_HEIGHT - 100) + 50, 10, 1, ObjectType.ENEMY, "Enemy",0, 0, myUUID, ObjectType.GAME,-1);

            for (ServerModule serverModule : threadList) {
                serverModule.packetToSend.EnemyList.add(tempEnemy);
            }

            GlobalPackage.EnemyList.add(tempEnemy);

            new Thread(() -> {
                while (GlobalPackage.EnemyList.contains(tempEnemy)) {
                    if (GlobalPackage.PlayerList.size() > 0) {
                        int i1 = GlobalPackage.EnemyList.indexOf(tempEnemy);
                        //System.out.println(GlobalPackage.EnemyList.get(i).StringPosition());
                        GlobalPackage.EnemyList.get(i1).Shoot(
                                GlobalPackage.PlayerList.get(randomizer.nextInt(GlobalPackage.PlayerList.size())).cordX,
                                GlobalPackage.PlayerList.get(randomizer.nextInt(GlobalPackage.PlayerList.size())).cordY);
                    }
                    try {
                        Thread.sleep(2000);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }).start();
        }
    }
    public static void BoosterSpawning(){
        Random random = new Random();

        for (int i = 0; i < packSize; i++) {

            GameObject tempBooster = new GameObject(random.nextInt(CONFIG.WINDOW_WIDTH),
random.nextInt(CONFIG.WINDOW_HEIGHT), 0, 100, ObjectType.BOOSTER, "",0, 1, myUUID,ObjectType.GAME, -1);
            tempBooster.objectName = "BOOSTER";

            for (ServerModule serverModule : threadList) {
                serverModule.packetToSend.BoosterList.add(tempBooster);
            }

            GlobalPackage.BoosterList.add(tempBooster);

        }
    }
    public static void TrapSpawning(){

        Random random = new Random();

        for (int i = 0; i < packSize; i++) {

            GameObject tempTrap = new GameObject(random.nextInt(CONFIG.WINDOW_WIDTH), random.nextInt(CONFIG.WINDOW_HEIGHT),
50, 100, ObjectType.TRAP,"", 0, 0, myUUID,ObjectType.GAME, -1);
            tempTrap.objectName = "TRAP";

            for (ServerModule serverModule : threadList) {
                serverModule.packetToSend.TrapList.add(tempTrap);
            }

            GlobalPackage.TrapList.add(tempTrap);

        }

    }
    public static boolean Available(int PORT) {

        ServerSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(PORT);
            serverSocket.close();
            return true;
        }
        catch (Exception e) {
            if (serverSocket != null) {
                try {
                    serverSocket.close();
                }
                catch (Exception exception) {
                    exception.printStackTrace();
                    System.out.println("HM_004");
                }
```

```
            }
        }

        return false;

    } // Check if port is Available

}
```

Najważniejsze elementy klasy „HostModule”:

Konstruktor:

```
public HostModule() {

    super("Sample");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(0,0,400,900);
    setVisible(true);
    setResizable(false);
    GlobalPackage = new Packet();

    threadList = new ArrayList<>();

    ServerSocket serverSocket = null;

    try {
        serverSocket = new ServerSocket(CONFIG.PORT);
    } catch (Exception exception) {
        exception.printStackTrace();
    }

    ServerSocket finalServerSocket = serverSocket;

    StartUpdating();

    new Thread(() -> {
        while (true) {
            Socket socket;
            try {

                System.out.println("Serwer czeka na klienta...");
                socket = finalServerSocket.accept();
                System.out.println("Serwer widzi nowego klienta");
                clientCount++;

                ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
                ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());

                ServerModule serverModule = new ServerModule(socket, ois, oos, clientCount);
                threadList.add(serverModule);

                serverModule.packetToSend.BulletList.addAll(GlobalPackage.BulletList);
                serverModule.packetToSend.PlayerList.addAll(GlobalPackage.PlayerList);
                serverModule.packetToSend.EnemyList.addAll(GlobalPackage.EnemyList);
                serverModule.packetToSend.TrapList.addAll(GlobalPackage.TrapList);
                serverModule.packetToSend.BoosterList.addAll(GlobalPackage.BoosterList);

                setTitle("Current Client Count: " + clientCount);

                Spawning();

            } catch (Exception exception) {
                exception.printStackTrace();
                System.out.println("HM_001");
            }

        }
    }).start();

}
```

W konstruktorze wywoływany jest wątek odpowiadający za nasłuchiwanie na nadciągających klientów.

Kiedy klient zostaje znaleziony tworzony jest obiekt klasy „ServerModule” do którego dopisane zostają globalne listy pakietów (Listy przechowują wszystkie żyjące w danej chwili obiekty), tak by nowo przybyły klient widział to samo co reszta klientów.

Utworzony obiekt „serverModule” zostaje dodany do listy, by w przyszłości z łatwością można było przekazywać pakiety które dotrą do serwera w przyszłości.

Funkcje „StartUpdating” oraz „Update”:

```
private static void StartUpdating(){
    new Thread(() -> {
        // FPS Stuff
        long lastTime = System.nanoTime();
        double nsUpdate = 1000000000.0 / CONFIG.AMOUNT_OF_TICKS;
        double deltaUpdate = 0;

        long timer = System.currentTimeMillis();

        while(true) {
            long now = System.nanoTime();

            deltaUpdate += (now - lastTime) / nsUpdate;
            lastTime = now;

            if (deltaUpdate >= 1) {
                deltaUpdate--;
                Update(); // Call Update Function to update positions
                Collision(); // Call Collision Function to check for collisions
            }

            if(System.currentTimeMillis() - timer > 1000) {
                timer += 1000;
            }
        }
    }).start();
} // Start Updating Thread
public static void Update() {

    List<GameObject> temp = new ArrayList<>();
    temp.addAll(GlobalPackage.EnemyList);
    temp.addAll(GlobalPackage.BulletList);
    temp.addAll(GlobalPackage.PlayerList);
    temp.addAll(GlobalPackage.BoosterList);
    temp.addAll(GlobalPackage.TrapList);

    Point WindowCenter = new Point(CONFIG.WINDOW_WIDTH/2, CONFIG.WINDOW_HEIGHT/2);
    Point Temp = new Point(0,0);

    for (GameObject gameObject : temp) {

        gameObject.Move();

        if (gameObject.myType == ObjectType.ENEMY || gameObject.myType == ObjectType.PLAYER) {
            gameObject.reloadTime--;

            if (gameObject.velY < 5) {
                gameObject.velY += 0.2;
            }
        }

        try {
            Temp.x = (int) gameObject.cordX;
            Temp.y = (int) gameObject.cordY;
            if (WindowCenter.distance(Temp) > 3000) {
                switch (gameObject.myType) {
                    case ENEMY -> GlobalPackage.EnemyList.remove(gameObject);
                    case TRAP -> GlobalPackage.TrapList.remove(gameObject);
                    case BOOSTER -> GlobalPackage.BoosterList.remove(gameObject);
                    case BULLET_LARGE, BULLET_SMALLER, BULLET_NORMAL -> GlobalPackage.BulletList.remove(gameObject);
                }
            }
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }
    }
} // Optimisation Function
```

Obydwie funkcje działają niemal identycznie jak te z klasy „Main” z tą różnicą, że w przypadku klasy „HostModule” funkcje w żaden sposób nie rysują niczego na ekranie.

Serwer z założenia nie posiada żadnego interfejsu, dlatego też nie potrzebuje żadnych aktualizacji wyglądu

Funkcja „Collision”:

```
public static void Collision(){

    try {

        List<GameObject> threatList = new ArrayList<>();
        List<GameObject> targetList = new ArrayList<>();

        threatList.addAll(GlobalPackage.BulletList);
        threatList.addAll(GlobalPackage.BoosterList);
        threatList.addAll(GlobalPackage.TrapList);

        targetList.addAll(GlobalPackage.PlayerList);
        targetList.addAll(GlobalPackage.EnemyList);

        boolean collision;

        if (threatList.size() > 0) {

            for (GameObject threat : threatList) {
                for (GameObject target : targetList) {

                    collision = false;

                    if (threat.srcUUID.compareTo(target.myUUID) != 0)  // No Self Harm
                        if (threat.myCPolygon.GetPoly().intersects(target.myCPolygon.GetPoly().getBounds2D())) // Check For
Collision
                            collision = true;

                    if (collision) {

                        if (target.myType == ObjectType.PLAYER && threat.srcObjectType == ObjectType.PLAYER)
                            continue;

                        if (target.myType == ObjectType.ENEMY && threat.srcObjectType == ObjectType.ENEMY)
                            continue;

                        if (target.myType == ObjectType.ENEMY && threat.myType == ObjectType.TRAP)
                            continue;

                        threat.hpCurrent = -1;
                        target.hpCurrent -= threat.damage;
                        target.statDamageTaken += threat.damage;



                        if (threat.myType == ObjectType.BOOSTER) {
                            target.damage += CONFIG.BOOSTER_DAMAGE_BUFF;
                            target.hpCurrent += CONFIG.BOOSTER_HP_BUFF;

                            if (target.hpCurrent > 100)
                                target.hpCurrent = 100;
                        }

                        boolean additionalPlayerPackage = false;

                        if (threat.srcObjectType == ObjectType.PLAYER) {
                            try {
                                GlobalPackage.PlayerList.get(threat.SrcIndexInGlobalList).statDamageDealt += threat.damage;
                                additionalPlayerPackage = true;
                            }
                            catch (Exception ignored){
                                // Threat source might not be in game anymore
                            }
                        }

                        if (target.hpCurrent <= 0) {
                            switch (target.myType) {
                                case ENEMY -> {
                                    GlobalPackage.EnemyList.remove(target);
                                    if (threat.srcObjectType == ObjectType.PLAYER) {
                                        if (GlobalPackage.PlayerList.size() > 0) {
                                            try {
                                                GlobalPackage.PlayerList.get(threat.SrcIndexInGlobalList).statKillCount++;
                                                additionalPlayerPackage = true;
                                            }
                                            catch (Exception ignored) {
                                                // Threat source might not be in game anymore
                                            }
                                        }
                                    }
                                }
                                case PLAYER -> {
                                    GlobalPackage.PlayerList.remove(target);
                                    SavedObject.Save(target.objectName, target.MyScore);
                                }
                            }
                        }

                        for (ServerModule serverModule : threadList) {

                            if (additionalPlayerPackage)

serverModule.packetToSend.PlayerList.add(GlobalPackage.PlayerList.get(threat.SrcIndexInGlobalList));

                            switch (threat.myType) {
                                case BOOSTER -> {
                                    serverModule.packetToSend.BoosterList.add(threat);
                                    target.statBoosterCount++;
                                }
```

```
                            case TRAP -> {
                                serverModule.packetToSend.TrapList.add(threat);
                                target.statTrapCount++;
                            }
                            default -> serverModule.packetToSend.BulletList.add(threat);
                        }

                        switch (target.myType) {
                            case ENEMY -> serverModule.packetToSend.EnemyList.add(target);
                            case PLAYER -> serverModule.packetToSend.PlayerList.add(target);
                        }

                    }

                    switch (threat.myType) {
                        case BOOSTER -> GlobalPackage.BoosterList.remove(threat);
                        case TRAP -> GlobalPackage.TrapList.remove(threat);
                        default -> GlobalPackage.BulletList.remove(threat);
                    }
                    if (GlobalPackage.EnemyList.size() == 0)
                        EnemySpawning();
                }
            }
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
} // Collision Function
// End of Gameplay Section
```

W przypadku klasy „HostModule" funkcja „Collision" jest znacznie bardziej rozbudowana. Dokładnie sprawdzane są parametry obiektów i w razie wystąpienia kolizji obiekty usuwane są z list serwerowych a następnie do klientów przesyłane są informacje potwierdzające lub informujące o śmierci danego obiektu.

Klasa „ServerModule":

```
package Online_Modules;

import Custom_Objects.GameObject;
import Custom_Objects.Packet;
import Main_Package.Main;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class ServerModule {
    private boolean clientIdSent = false;
    final int clientID;
    final String clientName;
    final Socket socket;
    final ObjectInputStream objectInputStream;
    final ObjectOutputStream objectOutputStream;
    public Packet packetToSend;
    ServerModule(Socket socket, ObjectInputStream objectInputStream, ObjectOutputStream
objectOutputStream, int clientID) {
        this.socket = socket;
        this.clientID = clientID;
        this.clientName = "Client_0" + clientID;
        this.objectInputStream = objectInputStream;
        this.objectOutputStream = objectOutputStream;

        packetToSend = new Packet();

        new Thread(new Runnable() {
            @Override
            public void run() {
                Packet tempPacket = new Packet();
                Packet AckPacket;

                Packet toBeRemoved = new Packet();

                boolean Running = true;
```

```
                while (Running) {
                    try { // Wysyłanie
                        if (!clientIdSent) {
                            objectOutputStream.writeObject(clientID);
                            clientIdSent = true;
                        }
                        else {
                            objectOutputStream.reset();
                            objectOutputStream.flush();

                            tempPacket.PlayerList.addAll(packetToSend.PlayerList);
                            tempPacket.EnemyList.addAll(packetToSend.EnemyList);
                            tempPacket.BulletList.addAll(packetToSend.BulletList);
                            tempPacket.TrapList.addAll(packetToSend.TrapList);
                            tempPacket.BoosterList.addAll(packetToSend.BoosterList);

                            packetToSend.PlayerList.clear();
                            packetToSend.EnemyList.clear();
                            packetToSend.BulletList.clear();
                            packetToSend.TrapList.clear();
                            packetToSend.BoosterList.clear();

                            objectOutputStream.writeUnshared(tempPacket);
                            tempPacket.PlayerList.clear();
                            tempPacket.EnemyList.clear();
                            tempPacket.BulletList.clear();
                            tempPacket.TrapList.clear();
                            tempPacket.BoosterList.clear();
                        }
                    }
                    catch (Exception exception) {
                        exception.printStackTrace();
                    }

                    try { // Odbieranie
                        AckPacket = (Packet) objectInputStream.readObject();
                        for (ServerModule serverThread : HostModule.threadList) {
                            if (serverThread.clientID != clientID) {

serverThread.packetToSend.PlayerList.addAll(AckPacket.PlayerList);

serverThread.packetToSend.EnemyList.addAll(AckPacket.EnemyList);

serverThread.packetToSend.BulletList.addAll(AckPacket.BulletList);
                            }
                        }

                        if (AckPacket.BulletList.size() > 0) {
                            for (GameObject AckBullet : AckPacket.BulletList) {
                                if (AckBullet != null) {
                                    boolean missing = true;
                                    for (GameObject LocalBullet :
HostModule.GlobalPackage.BulletList) {
                                        if (LocalBullet.myUUID.compareTo(AckBullet.myUUID) ==
0) {
                                            if (AckBullet.hpCurrent > 0) {
                                                LocalBullet.ReplaceMe(AckBullet);
                                            }
                                            else {

//HostModule.GlobalPackage.BulletList.remove(LocalBullet);
                                                toBeRemoved.BulletList.add(LocalBullet);
                                            }
                                            missing = false;
                                        }
                                    }
                                    if (missing) {
                                        HostModule.GlobalPackage.BulletList.add(AckBullet);
                                    }
                                }
                            }
                        } // Handle Bullet List

                        if (AckPacket.EnemyList.size() > 0) {
                            for (GameObject AckEnemy : AckPacket.EnemyList) {
                                if (AckEnemy != null) {
```

```
                                    boolean missing = true;
                                    for (GameObject LocalEnemy :
HostModule.GlobalPackage.EnemyList) {
                                        if (LocalEnemy.myUUID.compareTo(AckEnemy.myUUID) == 0)
{
                                            if (AckEnemy.hpCurrent > 0) {
                                                LocalEnemy.ReplaceMe(AckEnemy);
                                            }
                                            else {

//HostModule.GlobalPackage.BulletList.remove(LocalBullet);
                                                toBeRemoved.EnemyList.add(LocalEnemy);
                                            }
                                            missing = false;
                                        }
                                    }
                                    if (missing) {
                                        HostModule.GlobalPackage.EnemyList.add(AckEnemy);
                                    }
                                }
                            }
                        } // Handle Enemy List

                        if (AckPacket.PlayerList.size() > 0) {
                            for (GameObject AckPlayer : AckPacket.PlayerList) {
                                if (AckPlayer != null) {
                                    boolean missing = true;
                                    for (GameObject LocalPlayer :
HostModule.GlobalPackage.PlayerList) {
                                        if (LocalPlayer.myUUID.compareTo(AckPlayer.myUUID) ==
0) {
                                            if (AckPlayer.hpCurrent > 0) {
                                                LocalPlayer.ReplaceMe(AckPlayer);
                                            } else {

//HostModule.GlobalPackage.PlayerList.remove(LocalPlayer);
                                                toBeRemoved.PlayerList.add(LocalPlayer);
                                            }
                                            missing = false;
                                        }
                                    }
                                    if (missing) {
                                        int x = HostModule.GlobalPackage.PlayerList.size();
                                        AckPlayer.MyIndexInGlobalList =  x;
                                        HostModule.GlobalPackage.PlayerList.add(AckPlayer);

packetToSend.PlayerList.add(HostModule.GlobalPackage.PlayerList.get(x));
                                    }
                                }
                            }
                        }

                        if (AckPacket.TrapList.size() > 0) {
                            for (GameObject AckTrap : AckPacket.TrapList) {
                                if (AckTrap != null) {
                                    boolean missing = true;
                                    for (GameObject LocalTrap :
HostModule.GlobalPackage.TrapList) {
                                        if (LocalTrap.myUUID.compareTo(AckTrap.myUUID) == 0) {
                                            if (AckTrap.hpCurrent > 0) {
                                                LocalTrap.ReplaceMe(AckTrap);
                                            }
                                            else {

//HostModule.GlobalPackage.BulletList.remove(LocalBullet);
                                                toBeRemoved.TrapList.add(LocalTrap);
                                            }
                                            missing = false;
                                        }
                                    }
                                    if (missing) {
                                        HostModule.GlobalPackage.TrapList.add(AckTrap);
                                    }
                                }
                            }
                        } // Handle Trap List
```

```
                        if (AckPacket.BoosterList.size() > 0) {
                            for (GameObject AckBooster : AckPacket.BoosterList) {
                                if (AckBooster != null) {
                                    boolean missing = true;
                                    for (GameObject LocalBooster :
HostModule.GlobalPackage.BoosterList) {
                                        if (LocalBooster.myUUID.compareTo(AckBooster.myUUID)
== 0) {
                                            if (AckBooster.hpCurrent > 0) {
                                                LocalBooster.ReplaceMe(AckBooster);
                                            }
                                            else {

//HostModule.GlobalPackage.BulletList.remove(LocalBullet);
                                                toBeRemoved.BoosterList.add(LocalBooster);
                                            }
                                            missing = false;
                                        }
                                    }
                                    if (missing) {
                                        HostModule.GlobalPackage.BoosterList.add(AckBooster);
                                    }
                                }
                            }
                        } // Handle Trap List

                        AckPacket.ClearPacket();

                        HostModule.GlobalPackage.BulletList.removeAll(toBeRemoved.BulletList);
                        HostModule.GlobalPackage.EnemyList.removeAll(toBeRemoved.EnemyList);
                        HostModule.GlobalPackage.PlayerList.removeAll(toBeRemoved.PlayerList);

HostModule.GlobalPackage.BoosterList.removeAll(toBeRemoved.BoosterList);
                        HostModule.GlobalPackage.TrapList.removeAll(toBeRemoved.TrapList);

                        toBeRemoved.BulletList.clear();
                        toBeRemoved.EnemyList.clear();
                        toBeRemoved.PlayerList.clear();
                        toBeRemoved.BoosterList.clear();
                        toBeRemoved.TrapList.clear();

                    }
                    catch (Exception exception) {
                        exception.printStackTrace();
                        Running = false;
                    }
                }
            }
        }).start();

    }

}
```

Klasa odpowiada za przechwytywanie, wysyłanie oraz przetwarzanie pakietów.

Klasa „ClientModule”:

```
package Online_Modules;

import Custom_Objects.GameObject;
import Config.CONFIG;
import Custom_Objects.Packet;
import Custom_Panels.CPlayerList;
import Custom_Panels.CScoreboardPanel;
import Main_Package.Main;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class ClientModule {
    public int ClientID = -1;
    public String ClientName;
```

```
public ClientModule(){
    // Basic JFrame Thingy

    Socket socket;
    ObjectInputStream objectInputStream = null;
    ObjectOutputStream objectOutputStream = null;

    try {
        socket = new Socket("localhost",  CONFIG.PORT);
        objectOutputStream = new ObjectOutputStream(socket.getOutputStream());
        objectInputStream = new ObjectInputStream(socket.getInputStream());
    }
    catch (Exception e){
        System.out.println("Error during creating connection");
    }

    ObjectInputStream finalObjectInputStream = objectInputStream;
    ObjectOutputStream finalObjectOutputStream = objectOutputStream;

    new Thread(() -> {
        Packet tempPacket = new Packet();
        Packet AckPacket;

        Packet toBeRemoved = new Packet();

        while (true) {
            try {
                if (ClientID == -1) {
                    int tmp = (int) finalObjectInputStream.readObject();
                    ClientName = "Client_0" + tmp;
                    ClientID = tmp;
                    CONFIG.CLIENT_NAME = ClientName;
                }
                else {

                    try {
                        AckPacket = (Packet) finalObjectInputStream.readObject();
                        if (AckPacket.BulletList.size() > 0) {
                            for (GameObject AckBullet : AckPacket.BulletList) {
                                if (AckBullet != null) {
                                    boolean missing = true;
                                    for (GameObject LocalBullet : Main.BulletList) {
                                        if (LocalBullet.myUUID.compareTo(AckBullet.myUUID) == 0) {
                                            if (AckBullet.hpCurrent > 0) {
                                                LocalBullet.ReplaceMe(AckBullet);
                                            }
                                            else {
                                                toBeRemoved.BulletList.add(LocalBullet);
                                               // Main.BulletList.remove(LocalBullet);
                                            }
                                            missing = false;
                                        }
                                    }
                                    if (missing) {
                                        Main.BulletList.add(AckBullet);
                                    }
                                }
                            }
                        } // Handle Bullet List

                        if (AckPacket.EnemyList.size() > 0) {
                            for (GameObject AckEnemy : AckPacket.EnemyList) {
                                if (AckEnemy != null) {
                                    boolean missing = true;
                                    for (GameObject LocalEnemy : Main.EnemyList) {
                                        if (LocalEnemy.myUUID.compareTo(AckEnemy.myUUID) == 0) {
                                            if (AckEnemy.hpCurrent > 0) {
                                                LocalEnemy.ReplaceMe(AckEnemy);
                                            }
                                            else {
                                                toBeRemoved.EnemyList.add(LocalEnemy);
                                                // Main.EnemyList.remove(LocalEnemy);
                                            }
                                            missing = false;

                                        }
                                    }
                                    if (missing) {
                                        Main.EnemyList.add(AckEnemy);
                                    }
                                }
                            }
                        } // Handle Enemy List

                        if (AckPacket.PlayerList.size() > 0) {
                            for (GameObject AckPlayer : AckPacket.PlayerList) {
                                if (AckPlayer != null) {
                                    boolean missing = true;
                                    for (GameObject LocalPlayer : Main.PlayerList) {
                                        if (LocalPlayer.myUUID.compareTo(AckPlayer.myUUID) == 0) {
                                            if (AckPlayer.hpCurrent > 0) {
                                                LocalPlayer.ReplaceMe(AckPlayer);
                                            }
                                            else {
                                               // Main.PlayerList.remove(LocalPlayer);
                                                toBeRemoved.PlayerList.add(LocalPlayer);
                                            }
                                            missing = false;
                                        }
                                    }
                                    if (missing) {
                                        Main.PlayerList.add(AckPlayer);
                                    }
```

```
                                }
                            }
                        } // Handle Player List

                        if (AckPacket.TrapList.size() > 0) {
                            for (GameObject AckTrap : AckPacket.TrapList) {
                                if (AckTrap != null) {
                                    boolean missing = true;
                                    for (GameObject localTrap : Main.TrapList) {
                                        if (localTrap.myUUID.compareTo(AckTrap.myUUID) == 0) {
                                            if (AckTrap.hpCurrent > 0) {
                                                localTrap.ReplaceMe(AckTrap);
                                            }
                                            else {
                                               // Main.PlayerList.remove(LocalPlayer);
                                                toBeRemoved.TrapList.add(localTrap);
                                            }
                                            missing = false;
                                        }
                                    }
                                    if (missing) {
                                        Main.TrapList.add(AckTrap);
                                    }
                                }
                            }
                        } // Handle Trap List

                        if (AckPacket.BoosterList.size() > 0) {
                            for (GameObject AckBooster : AckPacket.BoosterList) {
                                if (AckBooster != null) {
                                    boolean missing = true;
                                    for (GameObject LocalBooster : Main.BoosterList) {
                                        if (LocalBooster.myUUID.compareTo(AckBooster.myUUID) == 0) {
                                            if (AckBooster.hpCurrent > 0) {
                                                LocalBooster.ReplaceMe(AckBooster);
                                            }
                                            else {
                                               // Main.PlayerList.remove(LocalPlayer);
                                                toBeRemoved.BoosterList.add(LocalBooster);
                                            }
                                            missing = false;
                                        }
                                    }
                                    if (missing) {
                                        Main.BoosterList.add(AckBooster);
                                    }
                                }
                            }
                        } // Handle Player List

                        if (AckPacket.PlayerList.size() > 0 || AckPacket.EnemyList.size() > 0 ||
AckPacket.BulletList.size() > 0)
                            Main.NEW_LIST_UPDATE = true;

                        if (AckPacket.PlayerList.size() > 0 || AckPacket.EnemyList.size() > 0) {
                            CScoreboardPanel.UpdateScoreboardList();
                            CPlayerList.UpdatePlayerList();
                        }

                        AckPacket.ClearPacket();

                        Main.BulletList.removeAll(toBeRemoved.BulletList);
                        Main.EnemyList.removeAll(toBeRemoved.EnemyList);
                        Main.PlayerList.removeAll(toBeRemoved.PlayerList);
                        Main.BoosterList.removeAll(toBeRemoved.BoosterList);
                        Main.TrapList.removeAll(toBeRemoved.TrapList);

                        toBeRemoved.BulletList.clear();
                        toBeRemoved.EnemyList.clear();
                        toBeRemoved.PlayerList.clear();
                        toBeRemoved.BoosterList.clear();
                        toBeRemoved.TrapList.clear();

                        try {
                            Main.PlayerList.get(Main.MyPlayerIndex);
                        }
                        catch (Exception exception){
                            if (!Main.GAME_INSTANCE.lastChanceQuiz.active) {
                                Main.GAME_INSTANCE.lastChanceQuiz.active = true;
                                Main.GAME_INSTANCE.lastChanceQuiz.setVisible(true);
                            }
                        }

                    }
                    catch (Exception ignored) {
                        return;
                    }

                }

            } catch (Exception e) {
                System.out.println("CM_001");
                break;
            }

            try {
                tempPacket.PlayerList.addAll(Main.PacketToSend.PlayerList);
                tempPacket.BulletList.addAll(Main.PacketToSend.BulletList);

                Main.PacketToSend.ClearPacket();

                finalObjectOutputStream.reset();
                finalObjectOutputStream.flush();
```

```
                    finalObjectOutputStream.writeUnshared(tempPacket);

                    tempPacket.ClearPacket();

                } catch (Exception e2) {
                    e2.printStackTrace();
                }

            }
        }).start();
    }

}
```

Ta klasa jest odpowiednikiem klasy „ServerModule" dla Klienta. Odpowiada za przechwytywanie, przesyłanie oraz przetwarzanie pakietów.

Klasa „Packet"

```
package Custom_Objects;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class Packet implements Serializable {
    public List<GameObject> PlayerList = new ArrayList<>();
    public List<GameObject> EnemyList = new ArrayList<>();
    public List<GameObject> BulletList = new ArrayList<>();
    public List<GameObject> BoosterList = new ArrayList<>();
    public List<GameObject> TrapList = new ArrayList<>();
    public void ClearPacket() {
        this.PlayerList.clear();
        this.EnemyList.clear();
        this.BulletList.clear();
        this.TrapList.clear();
        this.BoosterList.clear();
    }
}
```

Obiekty tej klasy są przesyłane między serwerem a klientem.

Klasa „GameObject"

```
package Custom_Objects;

import Config.CONFIG;
import Custom_Elements.Sound;
import Custom_Enums.*;
import Custom_Polygons.CPolygon;
import Main_Package.Main;
import Online_Modules.*;

import java.awt.*;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.UUID;

public class GameObject implements Serializable {
    public final UUID srcUUID;
    public final UUID myUUID = UUID.randomUUID();
    public final ObjectType srcObjectType;
    public CPolygon myCPolygon;
    public Color MyColor;
    public int MyScore, MyIndexInGlobalList, SrcIndexInGlobalList;
    public int statKillCount, statTrapCount, statBoosterCount, statDamageTaken, statDamageDealt, statShootCount;
    public double cordX,cordY, velY,velX;
    public double damage, hpCurrent, reloadTime;
    public ObjectType myType;
    public int rotation = 5;
    public int wallet = 1000000000;
    public String objectName;
    public WeaponType myWeapon;
    public boolean visible = true;

    public GameObject(double cordX, double cordY, double damage, double hpCurrent, ObjectType myType, String objectName,
double velX, double velY, UUID srcUUID, ObjectType srcObjectType, int SrcIndexInGlobalList){

        this.srcUUID = srcUUID;
        this.srcObjectType = srcObjectType;
        this.SrcIndexInGlobalList = SrcIndexInGlobalList;

        this.cordX = cordX;
        this.cordY = cordY;
```

```
        this.velX = velX;
        this.velY = velY;

        this.damage = damage;
        this.hpCurrent = hpCurrent;

        this.myType = myType;
        this.objectName = objectName;

        myCPolygon = new CPolygon();

        switch (myType) {
            case PLAYER -> { myCPolygon.InitPreMade(PreMadeType.PENTAGON,CONFIG.SIZE_PLAYER); MyColor = Color.GREEN; }
            case ENEMY -> { myCPolygon.InitPreMade(PreMadeType.SQUARE,CONFIG.SIZE_ENEMY); MyColor = Color.ORANGE; }
            case BULLET_LARGE -> { myCPolygon.InitPreMade(PreMadeType.TRIANGLE,CONFIG.SIZE_LARGE_BULLET); MyColor =
Color.RED; }
            case BULLET_NORMAL -> { myCPolygon.InitPreMade(PreMadeType.TRIANGLE,CONFIG.SIZE_NORMAL_BULLET); MyColor =
Color.RED; }
            case BULLET_SMALLER -> { myCPolygon.InitPreMade(PreMadeType.TRIANGLE,CONFIG.SIZE_SMALL_BULLET); MyColor =
Color.RED; }
            case TRAP -> { myCPolygon.InitPreMade(PreMadeType.SQUARE,CONFIG.SIZE_TRAP); MyColor = Color.PINK; }
            case BOOSTER -> { myCPolygon.InitPreMade(PreMadeType.SQUARE,CONFIG.SIZE_BOOSTER); MyColor = Color.WHITE; }
        }

        switch (new Random().nextInt(4)) {
            case 0 -> myWeapon = WeaponType.SHOTGUN;
            case 1 -> myWeapon = WeaponType.SNIPER_RIFLE;
            case 2 -> myWeapon = WeaponType.PISTOL;
            case 3 -> myWeapon = WeaponType.RIFLE;
        }

        if (myType == ObjectType.BULLET_LARGE || myType == ObjectType.BULLET_NORMAL || myType == ObjectType.BULLET_SMALLER)
            myWeapon = null;

    }

    public void Move(){

        cordY += velY;
        cordX += velX;

        myCPolygon.UpdateOffsets(cordX, cordY);

        if (CONFIG.ANIMATION_ROTATION)
            myCPolygon.Rotate(true, rotation);

        if (myType == ObjectType.PLAYER || myType == ObjectType.ENEMY) {
            if (cordX > (CONFIG.WINDOW_WIDTH + 50)) {
                cordX = 0;
            } else if (cordX < -50) {
                cordX = CONFIG.WINDOW_WIDTH;
            }
            if (cordY > CONFIG.WINDOW_HEIGHT) {
                cordY = -50;

            }
        }

    }
    public void ReplaceMe(GameObject gameObject){
        this.visible = gameObject.visible;
        this.myCPolygon = gameObject.myCPolygon;
        this.MyColor = gameObject.MyColor;
        this.MyScore = gameObject.MyScore;
        this.MyIndexInGlobalList = gameObject.MyIndexInGlobalList;
        this.SrcIndexInGlobalList = gameObject.SrcIndexInGlobalList;

        this.statKillCount = gameObject.statKillCount;
        this.statTrapCount = gameObject.statTrapCount;
        this.statBoosterCount = gameObject.statBoosterCount;
        this.statDamageDealt = gameObject.statDamageDealt;
        this.statDamageTaken = gameObject.statDamageTaken;
        this.statShootCount = gameObject.statShootCount;

        this.cordX = gameObject.cordX;
        this.cordY = gameObject.cordY;
        this.velY = gameObject.velY;
        this.velX = gameObject.velX;

        this.damage = gameObject.damage;
        this.hpCurrent = gameObject.hpCurrent;
        this.reloadTime = gameObject.reloadTime;
        this.myType = gameObject.myType;
        this.rotation = gameObject.rotation;
        this.wallet = gameObject.wallet;
        this.objectName = gameObject.objectName;
        this.myWeapon = gameObject.myWeapon;
    }
    public void Shoot(double targetCordX, double targetCordY){

        double tempSourceX = cordX, tempSourceY = cordY;

        double distX = Math.abs(tempSourceX - targetCordX);
        double distY = Math.abs(tempSourceY - targetCordY);

        double proportions = CONFIG.RECOIL_POWER / (distX + distY);

        double speedX = distX * proportions;
        double speedY = distY * proportions;

        if (tempSourceX > targetCordX && speedX > 0)
            speedX *= -1;
```

```
        if (tempSourceX < targetCordX && speedX < 0)
            speedX *= -1;
        if (tempSourceY > targetCordY && speedY > 0)
            speedY *= -1;
        if (tempSourceY < targetCordY && speedY < 0)
            speedY *= -1;

        if (reloadTime <= 0) {
            List<GameObject> Bullets = new ArrayList<>();
            statShootCount++;
            if (myType == ObjectType.PLAYER)
                Sound.PlayOofSound();

            if (myWeapon == WeaponType.SHOTGUN) {
                Bullets.add(new GameObject(cordX, cordY, damage * CONFIG.SHOTGUN_PRIME_BULLET, 100.0,
ObjectType.BULLET_LARGE, "Bullet",speedX * 1.3, speedY, myUUID, myType, MyIndexInGlobalList));
                Bullets.add(new GameObject(cordX, cordY * 0.9, damage * CONFIG.SHOTGUN_SECONDARY_BULLET, 100.0,
ObjectType.BULLET_NORMAL, "Bullet",speedX * 1.2, speedY, myUUID, myType, MyIndexInGlobalList));
                Bullets.add(new GameObject(cordX, cordY * 1.1, damage * CONFIG.SHOTGUN_SECONDARY_BULLET, 100.0,
ObjectType.BULLET_NORMAL, "Bullet",speedX * 1.2, speedY, myUUID, myType, MyIndexInGlobalList));
                Bullets.add(new GameObject(cordX, cordY * 0.8, damage * CONFIG.SHOTGUN_TERTIARY_BULLET, 100.0,
ObjectType.BULLET_SMALLER, "Bullet",speedX * 1.1, speedY, myUUID, myType, MyIndexInGlobalList));
                Bullets.add(new GameObject(cordX, cordY * 1.2, damage * CONFIG.SHOTGUN_TERTIARY_BULLET, 100.0,
ObjectType.BULLET_SMALLER, "Bullet",speedX * 1.1, speedY, myUUID, myType, MyIndexInGlobalList));
                velX = (-1.3 * speedX);
                velY = (-1.3 * speedY);
                reloadTime = CONFIG.SHOTGUN_RELOAD;
            }
            else if (myWeapon == WeaponType.PISTOL) {
                Bullets.add(new GameObject(cordX, cordY, damage * CONFIG.PISTOL_BULLET, 100.0, ObjectType.BULLET_NORMAL,
"Bullet",speedX, speedY, myUUID, myType, MyIndexInGlobalList));
                velX = (-1 * speedX);
                velY = (-1 * speedY);
                reloadTime = CONFIG.PISTOL_RELOAD;
            }
            else if (myWeapon == WeaponType.RIFLE) {
                Bullets.add(new GameObject(cordX, cordY, damage * CONFIG.RIFLE_BULLET, 100.0, ObjectType.BULLET_SMALLER,
"Bullet",speedX * 1.1, speedY, myUUID, myType, MyIndexInGlobalList));
                Bullets.add(new GameObject(cordX, cordY, damage * CONFIG.RIFLE_BULLET, 100.0, ObjectType.BULLET_SMALLER,
"Bullet",speedX * 1.3, speedY, myUUID, myType, MyIndexInGlobalList));
                Bullets.add(new GameObject(cordX, cordY, damage * CONFIG.RIFLE_BULLET, 100.0, ObjectType.BULLET_SMALLER,
"Bullet",speedX * 1.5, speedY, myUUID, myType, MyIndexInGlobalList));
                velX = (-1.1 * speedX);
                velY = (-1.1 * speedY);
                reloadTime = CONFIG.RIFLE_RELOAD;
            }
            else if (myWeapon == WeaponType.SNIPER_RIFLE) {
                Bullets.add(new GameObject(cordX, cordY, damage * CONFIG.SNIPER_BULLET, 100.0, ObjectType.BULLET_LARGE,
"Bullet",speedX * 3, speedY, myUUID,myType, MyIndexInGlobalList));
                velX = (-1.3 * speedX);
                velY = (-1.3 * speedY);
                reloadTime = CONFIG.SNIPER_RELOAD;
            }

            if (myType == ObjectType.PLAYER) {
                Main.PacketToSend.PlayerList.add(this);
                Main.BulletList.addAll(Bullets);
                Main.PacketToSend.BulletList.addAll(Bullets);
            } else if (myType == ObjectType.ENEMY) {
                HostModule.GlobalPackage.BulletList.addAll(Bullets);
                for (ServerModule serverModule : HostModule.threadList) {
                    serverModule.packetToSend.BulletList.addAll(Bullets);
                    serverModule.packetToSend.EnemyList.add(this);
                }
            }

        }
    }
    public boolean CanAfford(int price) {
        return wallet - price > 0;
    }
    public void Charge(int price) {
        if (CanAfford(price))
            wallet -= price;
    }
}
```

Jedna z ważniejszych klas w programie. Odpowiada za wszystkich graczy, przeciwników, ulepszaczy oraz pułapek występujące w grze.

Klasa „QuestObject”

```
package Custom_Objects;

import Main_Package.Main;

import java.util.Random;

public class QuestObject {
```

```
    public int targetStatKillCount, targetStatTrapCount, targetStatBoosterCount, targetStatDamageTaken,
targetStatDamageDealt, targetStatShootCount;
    public int rewardCash = 100, rewardHeal = 100;

    public QuestObject(int targetStatKillCount, int targetStatTrapCount, int targetStatBoosterCount, int
targetStatDamageTaken, int targetStatDamageDealt, int targetStatShootCount) {
        this.targetStatKillCount = targetStatKillCount;
        this.targetStatTrapCount = targetStatTrapCount;
        this.targetStatBoosterCount = targetStatBoosterCount;
        this.targetStatDamageTaken = targetStatDamageTaken;
        this.targetStatDamageDealt = targetStatDamageDealt;
        this.targetStatShootCount = targetStatShootCount;
    }

    public void NewQuest(){

        this.targetStatKillCount = 0;
        this.targetStatTrapCount = 0;
        this.targetStatBoosterCount = 0;
        this.targetStatDamageTaken = 0;
        this.targetStatDamageDealt = 0;
        this.targetStatShootCount = 0;

        Random random = new Random();

        for (int i = 0; i < (random.nextInt(5) + 1); i++) {
            boolean failure = true;
            do {
                switch (random.nextInt(6)) {
                    case 0 -> {
                        if (targetStatKillCount == 0) {
                            targetStatKillCount = (int) (3 + (Main.PlayerList.get(Main.MyPlayerIndex).statKillCount) * 1.2);
                            failure = false;
                        }
                    }
                    case 1 -> {
                        if (targetStatTrapCount == 0) {
                            targetStatTrapCount = (int) (3 + (Main.PlayerList.get(Main.MyPlayerIndex).statTrapCount) * 1.2);
                            failure = false;
                        }
                    }
                    case 2 -> {
                        if (targetStatBoosterCount == 0) {
                            targetStatBoosterCount = (int) (3 + (Main.PlayerList.get(Main.MyPlayerIndex).statBoosterCount) *
1.2);
                            failure = false;
                        }
                    }
                    case 3 -> {
                        if (targetStatDamageTaken == 0) {
                            targetStatDamageTaken = (int) (3 + (Main.PlayerList.get(Main.MyPlayerIndex).statDamageTaken) *
1.2);
                            failure = false;
                        }
                    }
                    case 4 -> {
                        if (targetStatDamageDealt == 0) {
                            targetStatDamageDealt = (int) (3 + (Main.PlayerList.get(Main.MyPlayerIndex).statDamageDealt) *
1.2);
                            failure = false;
                        }
                    }
                    case 5 -> {
                        if (targetStatShootCount == 0) {
                            targetStatShootCount = (int) (3 + (Main.PlayerList.get(Main.MyPlayerIndex).statShootCount) *
1.2);
                            failure = false;
                        }
                    }
                }
            }
            while (failure);
        }
    }

    public void QuestTracker() {
        if (Main.PlayerList.contains(Main.playerHolder)) {
            boolean Success = true;

            if (Main.PlayerList.get(Main.MyPlayerIndex).statKillCount < targetStatKillCount)
                Success = false;
            else if (Main.PlayerList.get(Main.MyPlayerIndex).statTrapCount < targetStatTrapCount)
                Success = false;
            else if (Main.PlayerList.get(Main.MyPlayerIndex).statBoosterCount < targetStatBoosterCount)
                Success = false;
            else if (Main.PlayerList.get(Main.MyPlayerIndex).statDamageTaken < targetStatDamageTaken)
                Success = false;
            else if (Main.PlayerList.get(Main.MyPlayerIndex).statDamageDealt < targetStatDamageDealt)
                Success = false;
            else if (Main.PlayerList.get(Main.MyPlayerIndex).statShootCount < targetStatShootCount)
                Success = false;

            if (Success) {
                Main.PlayerList.get(Main.MyPlayerIndex).hpCurrent += rewardHeal;
                if (Main.PlayerList.get(Main.MyPlayerIndex).hpCurrent > 100)
                    Main.PlayerList.get(Main.MyPlayerIndex).hpCurrent = 100;
                Main.PlayerList.get(Main.MyPlayerIndex).wallet += rewardCash;
                Main.PacketToSend.PlayerList.add(Main.PlayerList.get(Main.MyPlayerIndex));
                Main.activeQuestObject.NewQuest();
            }

        }
    }
```

```
}
```

Klasa odpowiedzialna za całą mechanikę „Questów" czyli w przypadku tego programu proceduralnie generowanych celów / wyzwań stawianych przed graczem za które ten może otrzymać określone w klasie „CONFIG" nagrody.

Klasa „SavedObject":

```
package Custom_Objects;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

public class SavedObject implements Comparable<SavedObject> {
    public String Name;
    public int Score;
    public SavedObject(String Name, int Score) {
        this.Name = Name;
        this.Score = Score;
    }
    @Override
    public int compareTo(SavedObject o) {
        return o.Score - Score;
    }

    public String toScoreboardString(int index){
        return (index + 1) + ". " + Name + " Score: " + Score;
    }

    public static void Save(String nick, int Score) {
        try {
            File file = new File("Scoreboard.txt");
            FileWriter fileWriter = new FileWriter(file,true);
            String output = nick + "-" + Score + "\n";
            fileWriter.write(output);
            fileWriter.close();
        }
        catch (Exception ignored) {

        }
    }

    public static List<SavedObject> Read() {
        List<SavedObject> list = new ArrayList<>();
        String[] temp;
        try {
            File file = new File("Scoreboard.txt");
            Scanner scanner = new Scanner(new FileInputStream(file.getAbsolutePath()), StandardCharsets.UTF_8);

            while(scanner.hasNextLine()){
                temp = scanner.nextLine().split("-");
                list.add(new SavedObject(temp[0], Integer.parseInt(temp[1])));
            }
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }
        Collections.sort(list);
        return list;
    }

}
```

Klasa używana jest do tworzenia i sortowania rankingiem graczy.

Klasy „CPoint" oraz „CPolygon:

```
package Custom_Polygons;

import java.awt.*;
import java.io.Serializable;

public class CPoint implements Serializable {
    public double cordX, cordY;
    public double scale = 0.5;
```

```
    public CPoint(double cordX, double cordY){
        this.cordX = cordX;
        this.cordY = cordY;
    }
    public Point ConvertPoint(){
        double tempNewX = cordX * scale;
        double tempNewY = cordY * scale;
        double[] newValues = Scale(tempNewX,tempNewY);
        int tempOutputX = (int) (50 + newValues[0]);
        int tempOutputY = (int) (50 + newValues[1]);
        return new Point(tempOutputX,tempOutputY);
    }
    public static double[] Scale(double tempX, double tempY) {
        double dist = Math.sqrt(Math.pow(tempX,2) + Math.pow(tempY,2));
        double theta = Math.atan2(tempY,tempX);
        double[] outValues = new double[2];
        outValues[0] = dist * Math.cos(theta);
        outValues[1] = dist * Math.sin(theta);
        return outValues;
    }
    public void Rotate(boolean CW, double degree){
        double radius = Math.sqrt(Math.pow(cordX,2) + Math.pow(cordY,2));
        double theta = Math.atan2(cordY, cordX);
        if (CW) { theta += 2 * Math.PI / 360 * degree * -1; }
        else { theta += 2 * Math.PI / 360 * degree; }
        cordX = radius * Math.cos(theta);
        cordY = radius * Math.sin(theta);
    }

}
```

```
package Custom_Polygons;
import Custom_Enums.PreMadeType;

import java.awt.*;
import java.awt.geom.Rectangle2D;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
public class CPolygon implements Serializable {
    public List<CPoint> Points;
    public double offsetX;
    public double offsetY;
    public CPolygon(){
        this.offsetX = -100;
        this.offsetY = -100;
        this.Points = new ArrayList<>();
    }

    public void Copy(CPolygon source) {
        this.offsetX = source.offsetX;
        this.offsetY = source.offsetY;

        this.Points.clear();
        for (CPoint temp : source.Points)
            this.Points.add(new CPoint(temp.cordX, temp.cordY));
    }

    public Polygon GetPoly(){
        Polygon tempPolygon = new Polygon();

        for (CPoint tempCPoint : Points) {
            Point tempPoint = tempCPoint.ConvertPoint();
            tempPolygon.addPoint((int) (tempPoint.x + offsetX), (int) (tempPoint.y + offsetY));
        }
        return tempPolygon;
    }

    public void Render(Graphics2D graphics2D, Color color) {
        Polygon tempPolygon = new Polygon();

        for (CPoint tempCPoint : Points) {
            Point tempPoint = tempCPoint.ConvertPoint();
            tempPolygon.addPoint((int) (tempPoint.x + offsetX), (int) (tempPoint.y + offsetY));
        }

        graphics2D.setColor(color);
        graphics2D.fillPolygon(tempPolygon);
    }
    public void UpdateOffsets(double newOffsetX, double newOffsetY) {
        this.offsetX = newOffsetX;
        this.offsetY = newOffsetY;
    }
    public void Rotate(boolean CW, double _VectorX) {
        for (CPoint temp : Points)
            temp.Rotate(CW,_VectorX);
    }
    public void InitPreMade(PreMadeType preMadeType, double size) {

        Points.clear();

        switch (preMadeType) {
            case SQUARE -> {
                Points.add(new CPoint(-5 * size, -5 * size));
                Points.add(new CPoint(5 * size, -5 * size));
                Points.add(new CPoint(5 * size, 5 * size));
                Points.add(new CPoint(-5 * size, 5 * size));
            }
            case TRIANGLE -> {
                Points.add(new CPoint(-5 * size, -5 * (size / 2)));
                Points.add(new CPoint(5 * size, -5 * (size / 2)));
```

```
                Points.add(new CPoint(0, 5 * size));
            }
            case PENTAGON -> {
                Points.add(new CPoint(0 * size, 8 * size));
                Points.add(new CPoint(8 * size, 2 * size));
                Points.add(new CPoint(4 * size, -7 * size));
                Points.add(new CPoint(-4 * size, -7 * size));
                Points.add(new CPoint(-8 * size, 2 * size));
            }
        }

    }
}
```

Klasy odpowiadają za tworzenie i generowanie przeciwników, graczy oraz pozostałych elementów gry.

Tu znajdują się funkcje odpowiedzialne za kształt obiektów, ich obracanie oraz rozmiar i rysowanie.

Enum'y użyte do łatwiejszej interpretacji kodu:

Enum „ObjectType":

```
package Custom_Enums;

public enum ObjectType {
    ENEMY,
    BULLET_LARGE,
    BULLET_NORMAL,
    BULLET_SMALLER,
    PLAYER,
    TRAP,
    BOOSTER,
    GAME,
}
```

Enum „PreMadeType":

```
package Custom_Enums;
public enum PreMadeType {
    NONE,
    SQUARE,
    PENTAGON,
    TRIANGLE,
}
```

Enum „WeaponType"

```
package Custom_Enums;
```

```
public enum WeaponType {
    SHOTGUN,
    SNIPER_RIFLE,
    PISTOL,
    RIFLE,
}
```

Pozostałe klasy programu:

Klasa „CButton”:

```
package Custom_Elements;

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

public class CButton extends JButton {
    public CButton(String text, int size, Color fontColor){
        super(text);
        setFont(new Font("Comic Sans MS", Font.BOLD, size));
        setForeground(fontColor);
        setBackground(new Color(0,0,0,0));
        setAlignmentX(Component.CENTER_ALIGNMENT);
        setBorder(BorderFactory.createEmptyBorder(5, 5, 0, 0));
    }
    public CButton(String text, int size, Color backgroundColor, Color foregroundColor){
        super(text);
        setFont(new Font("Comic Sans MS", Font.BOLD, size));
        setAlignmentX(Component.CENTER_ALIGNMENT);
        setBackground(backgroundColor);
        setForeground(foregroundColor);
        setBorder(BorderFactory.createEmptyBorder(5, 5, 0, 0));
    }
}
```

Klasa „CImagePanel”:

```
package Custom_Elements;

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

public class CImagePanel extends JPanel {
    private final Image image;
    private final int width, height;
    public CImagePanel(Image image, int width, int height) {
        this.image = image;
        this.width = width;
        this.height = height;

        this.setOpaque(false);
        this.setBackground(new Color(0,0,0,0));
    }
    @Override
    public void paintComponent(Graphics g) {
        g.drawImage(image, 0, 0,width,height, null);
    }
}
```

Klasa „CLabel”:

```
package Custom_Elements;

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

public class CLabel extends JLabel {
    public CLabel(String text, int size){
        super(text, SwingConstants.CENTER);
        setFont(new Font("Comic Sans MS", Font.BOLD, size));
        setForeground(Color.BLACK);
        setAlignmentX(Component.CENTER_ALIGNMENT);
        setBorder(BorderFactory.createEmptyBorder(5, 5, 0, 0));
    }
```

```
    public CLabel(String text, int size, Color color){
        super(text, SwingConstants.CENTER);
        setFont(new Font("Comic Sans MS", Font.BOLD, size));
        setForeground(color);
        setAlignmentX(Component.CENTER_ALIGNMENT);
        setBorder(BorderFactory.createEmptyBorder(5, 5, 0, 0));
    }
    public CLabel(ImageIcon imageIcon){
        super(imageIcon);
    }
}
```

Klasa „CSlider”:

```
package Custom_Elements;

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

public class CSlider extends JSlider {
    public CSlider(){
        setMinorTickSpacing(2);
        setMajorTickSpacing(10);
        setValue(100);
        setPaintTicks(true);
        setPaintLabels(true);
        setBackground(new Color(0,0,0,0));
        setForeground(Color.WHITE);
        setFont(new Font("Comic Sans MS", Font.PLAIN, 16));
    }
}
```

Klasa „CTextField”:

```
package Custom_Elements;

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

public class CTextField extends JTextField {
    public CTextField(String text, int size, Color fontColor){
        super(text);
        setFont(new Font("Comic Sans MS", Font.BOLD, size));
        setForeground(fontColor);
        setBackground(new Color(0,0,0,0));
        setHorizontalAlignment(JTextField.CENTER);
        setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));
    }
}
```

Klasa „Sound”:

```
package Custom_Elements;

import Config.CONFIG;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.FloatControl;
import java.net.URL;

public class Sound {

    public static void PlayOofSound(){

        URL file = null;

        try {
            file = new URL("file:src/Package_Sounds/Shoot.wav");
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }

        Clip clip = null;

        try {
            if (file != null) {
                AudioInputStream ais = AudioSystem.getAudioInputStream(file);
                clip = AudioSystem.getClip();
                clip.open(ais);

            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
```

```
        if (clip != null) {
            clip.setFramePosition(0);

            FloatControl gainControl = (FloatControl) clip.getControl(FloatControl.Type.MASTER_GAIN);
            gainControl.setValue(-CONFIG.SHOOTING_VOLUME);

            clip.start();
        }
    }
    public static void PlayBackgroundSound(){

        URL file = null;

        try {
            file = new URL("file:src/Package_Sounds/Background.wav");
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }

        Clip clip = null;

        try {
            if (file != null) {
                AudioInputStream ais = AudioSystem.getAudioInputStream(file);
                clip = AudioSystem.getClip();
                clip.open(ais);

            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }

        if (clip != null) {
            clip.setFramePosition(0);
            clip.loop(100000);

            FloatControl gainControl = (FloatControl) clip.getControl(FloatControl.Type.MASTER_GAIN);
            gainControl.setValue(-CONFIG.BACKGROUND_VOLUME);

            clip.start();
            System.out.println(clip.getLevel());
        }
    }

}
```

Klasa „CPortListFrame”:

```
package Custom_Frames;

import Config.CONFIG;
import Custom_Elements.CButton;
import Custom_Elements.CLabel;
import Online_Modules.HostModule;

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

public class CPortListFrame extends JFrame {
    public CPortListFrame(int portSearchStart, int portSearchEnd){
        super("Available Port List");
        setBounds((CONFIG.WINDOW_WIDTH / 2)- (CONFIG.PORT_WINDOW_WIDTH / 2),(CONFIG.WINDOW_HEIGHT / 2) -
(CONFIG.PORT_WINDOW_HEIGHT / 2), CONFIG.PORT_WINDOW_WIDTH, CONFIG.PORT_WINDOW_HEIGHT);

        setResizable(false);

        CONFIG.PORT_SEARCH_MIN = portSearchStart;
        CONFIG.PORT_SEARCH_MAX = portSearchEnd;

        JPanel jPanel = new JPanel(new BorderLayout());

        JPanel subJPanel = new JPanel(new GridLayout(0,2));
        subJPanel.add(new CLabel("Occupied Ports",30,Color.WHITE));
        subJPanel.add(new CLabel("Available Ports",30,Color.WHITE));
        subJPanel.setBackground(Color.DARK_GRAY);

        JPanel Lists = new JPanel(new GridLayout(0,2));

        JPanel OnlinePortList = new JPanel();
        JPanel OfflinePortList = new JPanel();

        OnlinePortList.setLayout(new BoxLayout(OnlinePortList,BoxLayout.Y_AXIS));
        OfflinePortList.setLayout(new BoxLayout(OfflinePortList,BoxLayout.Y_AXIS));

        OnlinePortList.setBackground(Color.DARK_GRAY);
        OfflinePortList.setBackground(Color.DARK_GRAY);

        JScrollPane jScrollPaneOnline = new JScrollPane(OnlinePortList);
        JScrollPane jScrollPaneOffline = new JScrollPane(OfflinePortList);

        jScrollPaneOnline.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));
        jScrollPaneOffline.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));

        for (int i = CONFIG.PORT_SEARCH_MIN; i < CONFIG.PORT_SEARCH_MAX; i++) {
```

```
            if (HostModule.Available(i)) {
                OfflinePortList.add(new CLabel(i + " is FREE", 16, Color.GREEN));
            }
            else {
                OnlinePortList.add(new CLabel(i + " is TAKEN", 16, Color.RED));
            }
        }

        Lists.add(jScrollPaneOnline);
        Lists.add(jScrollPaneOffline);

        CButton closePortListFrameButton = new CButton("Close", 16, Color.RED, Color.WHITE);
        closePortListFrameButton.addActionListener(new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Dispose();
            }
        });

        jPanel.add(subJPanel,BorderLayout.NORTH);
        jPanel.add(Lists,BorderLayout.CENTER);
        jPanel.add(closePortListFrameButton,BorderLayout.SOUTH);

        setContentPane(jPanel);
        setVisible(true);

    }

    private void Dispose(){this.dispose();}

}
```

Klasa „CQuizFrame”:

```
package Custom_Frames;

import Config.CONFIG;
import Custom_Objects.QuizQuestion;

import javax.swing.*;
import java.awt.*;
import java.util.Objects;
import java.util.List;
import java.util.Random;

public class CQuizFrame extends JFrame {
    public boolean active = false;
    static List<QuizQuestion> quizQuestionList;
    public static QuizQuestion selectedQuestion;
    public JRadioButton[] jRadioButtons;
    public JButton submitButton;
    JLabel questionLabel;
    ImageIcon backgroundImage = null;
    public CQuizFrame(){
        super("Quiz Ostatniej Szansy!");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(500,500, CONFIG.QUIZ_WINDOW_WIDTH,CONFIG.QUIZ_WINDOW_HEIGHT);

        try {
            backgroundImage = new ImageIcon(Objects.requireNonNull(getClass().getResource("/Package Images/Logo-PANS.png")));
            Image tempImage = backgroundImage.getImage().getScaledInstance(420, 80,  java.awt.Image.SCALE_SMOOTH);
            backgroundImage = new ImageIcon(tempImage);
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }

        JPanel cPanel = new JPanel();
        cPanel.setLayout(new BorderLayout());

        JPanel jPanel = new JPanel(new GridLayout(7,0));
        questionLabel = new JLabel();

        jPanel.add(new JLabel("QUIZ OSTATNIEJ SZANSY!"));
        jPanel.add(new JLabel("Jeśli prawidłowo odpowiesz na wylosowane pytanie Twoja postać wróci do gry!"));
        jPanel.add(new JLabel("Pytanie: "));
        jPanel.add(questionLabel);

        jRadioButtons = new JRadioButton[3];
        ButtonGroup buttonGroup = new ButtonGroup();
        for (int i = 0; i < jRadioButtons.length; i++) {
            jRadioButtons[i] = new JRadioButton();
            buttonGroup.add(jRadioButtons[i]);
            jPanel.add(jRadioButtons[i]);
        }

        submitButton = new JButton("Wyślij");

        cPanel.add(new JLabel(backgroundImage),BorderLayout.NORTH);
        cPanel.add(jPanel,BorderLayout.CENTER);
        cPanel.add(submitButton,BorderLayout.SOUTH);

        GetQuestions();
        SelectQuestion();
        UpdateQuiz();

        setContentPane(cPanel);
        setVisible(false);
    }
```

```
    public static void GetQuestions(){
        quizQuestionList = QuizQuestion.ReadQuizQuestions();
        Random random = new Random();
        selectedQuestion = quizQuestionList.get(random.nextInt(quizQuestionList.size()));
    }
    public void SelectQuestion(){
        Random random = new Random();
        do {
            selectedQuestion = quizQuestionList.get(random.nextInt(quizQuestionList.size()));
        }
        while (selectedQuestion == quizQuestionList.get(random.nextInt(quizQuestionList.size())));
        UpdateQuiz();
    }
    public void UpdateQuiz(){
        questionLabel.setText(selectedQuestion.Question);
        jRadioButtons[0].setText(selectedQuestion.Answer_0);
        jRadioButtons[1].setText(selectedQuestion.Answer_1);
        jRadioButtons[2].setText(selectedQuestion.Answer_2);
    }
}
```

Klasa „CScoreboardFrame”:

```
package Custom_Frames;

import Config.CONFIG;
import Custom_Elements.CButton;
import Custom_Elements.CLabel;
import Custom_Objects.SavedObject;
import Online_Modules.HostModule;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.List;

public class CScoreboardFrame extends JFrame {

    public CScoreboardFrame() {
        super("Full Scoreboard Frame");
        setBounds((CONFIG.WINDOW_WIDTH / 2) - (CONFIG.SCOREBOARD_WINDOW_WIDTH / 2), (CONFIG.WINDOW_HEIGHT / 2) -
(CONFIG.SCOREBOARD_WINDOW_HEIGHT / 2), CONFIG.SCOREBOARD_WINDOW_WIDTH, CONFIG.SCOREBOARD_WINDOW_HEIGHT);

        setResizable(false);

        JPanel jPanel = new JPanel(new BorderLayout());
        jPanel.setBackground(Color.DARK_GRAY);

        JPanel ScoreboardListPanel = new JPanel();
        ScoreboardListPanel.setLayout(new BoxLayout(ScoreboardListPanel, BoxLayout.Y_AXIS));
        ScoreboardListPanel.setBackground(Color.DARK_GRAY);

        JScrollPane scoreboardScroll = new JScrollPane(ScoreboardListPanel);
        scoreboardScroll.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));

        List<SavedObject> savedObjectList = SavedObject.Read();

        for (int i = 0; i < savedObjectList.size(); i++) {
            ScoreboardListPanel.add(new CLabel(savedObjectList.get(i).toScoreboardString(i), 18,Color.ORANGE));
        }

        CButton closePortListFrameButton = new CButton("Close", 16, Color.RED, Color.WHITE);
        closePortListFrameButton.addActionListener(new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Dispose();
            }
        });

        jPanel.add(new CLabel("TOP PLAYER LIST", 32, Color.WHITE), BorderLayout.NORTH);
        jPanel.add(scoreboardScroll, BorderLayout.CENTER);
        jPanel.add(closePortListFrameButton, BorderLayout.SOUTH);

        setContentPane(jPanel);
        setVisible(true);

    }

    private void Dispose() {
        this.dispose();
    }
}
```

Klasa „PurchaseOption”:

```
package Custom_Objects;

public class PurchaseOption {
    public String optionName;
    public int optionPrice;
    public PurchaseOption(String optionName, int optionPrice){
```

```
        this.optionName = optionName;
        this.optionPrice = optionPrice;
    }
}
```

Klasa „QuizQuestion”

```
package Custom_Objects;

import java.io.File;
import java.io.FileInputStream;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class QuizQuestion {
    public String Question, Answer_0, Answer_1, Answer_2, CorrectAnswer;
    public QuizQuestion(String question, String answer_0, String answer_1, String answer_2, String correctAnswer) {
        this.Question = question;
        this.Answer_0 = answer_0;
        this.Answer_1 = answer_1;
        this.Answer_2 = answer_2;
        this.CorrectAnswer = correctAnswer;
    }
    public static List<QuizQuestion> ReadQuizQuestions() {
        List<QuizQuestion> list = new ArrayList<>();
        String[] temp;
        try {
            File file = new File("Questions.txt");
            Scanner scanner = new Scanner(new FileInputStream(file.getAbsolutePath()), StandardCharsets.UTF_8);

            while(scanner.hasNextLine()){
                temp = scanner.nextLine().split("-");
                list.add(new QuizQuestion(temp[0], temp[1], temp[2],temp[3], temp[4]));
            }
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }
        return list;
    }
}
```

Klasa „CConnectionPanel”:

```
package Custom_Panels;

import Config.CONFIG;
import Custom_Elements.CButton;
import Custom_Elements.CLabel;
import Custom_Elements.CTextField;
import Online_Modules.HostModule;

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

public class CConnectionPanel extends JPanel {
    public boolean visible = false;
    public CButton JoinServerButton, StartSearchButton;
    public CTextField PortInput, AddressInput, PortSearchStartInput, PortSearchEndInput;
    public CConnectionPanel() {

        setLayout(new GridLayout(9,0));
        setBackground(new Color(0,0,0,100));

        JPanel OnlineConfigurationPanel = new JPanel(new GridLayout(6,0));
        OnlineConfigurationPanel.setBackground(new Color(0,0,0,0));

        AddressInput = new CTextField("192.168.1.2",22, Color.WHITE);
        PortInput = new CTextField("7777",22, Color.WHITE);

        PortSearchStartInput = new CTextField("7000",22, Color.WHITE);
        PortSearchEndInput = new CTextField("8000",22, Color.WHITE);

        JoinServerButton = new CButton("Join Server", 28,Color.GREEN,Color.BLACK);
        StartSearchButton = new CButton("Start Search", 28,Color.ORANGE,Color.BLACK);

        this.add(new CLabel("IP Input", 28,Color.WHITE));
        this.add(AddressInput);
        this.add(new CLabel("Port Input:", 28,Color.WHITE));
        this.add(PortInput);
        this.add(new CLabel("Action:", 28,Color.WHITE));
```

```
        this.add(JoinServerButton);
        this.add(new CLabel("Search Ports:", 28,Color.WHITE));

        JPanel jPanel = new JPanel(new GridLayout(2,2));
        jPanel.setBackground(new Color(0,0,0,0));
        jPanel.add(this.add(new CLabel("Search Start", 28,Color.WHITE)));
        jPanel.add(this.add(new CLabel("Search End", 28,Color.WHITE)));
        jPanel.add(PortSearchStartInput);
        jPanel.add(PortSearchEndInput);

        this.add(jPanel);
        this.add(StartSearchButton);

    }
}
```

Klasa „CMenuPanel”:

```
package Custom_Panels;

import Custom_Elements.CButton;
import Custom_Elements.CLabel;
import Custom_Elements.CTextField;

import javax.swing.*;
import java.awt.*;
public class CMenuPanel extends JPanel {
    public CButton StartGameButton, CloseGameButton, SettingsButton, OnlineConfigButton, FullScoreboardButton;
    public CTextField NickInput;
    public CMenuPanel() {

        this.setLayout(new GridLayout(8,0));
        this.setBackground(new Color(0,0,0,160));

        JPanel secondPartPanel = new JPanel(new GridLayout(2,0));
        JPanel thirdPartPanel = new JPanel(new GridLayout(2,0));
        JPanel fourthPartPanel = new JPanel(new GridLayout(2,0));
        JPanel fifthPartPanel = new JPanel(new GridLayout(2,0));

        secondPartPanel.setBackground(new Color(0,0,0,0));
        thirdPartPanel.setBackground(new Color(0,0,0,0));
        fourthPartPanel.setBackground(new Color(0,0,0,0));
        fifthPartPanel.setBackground(new Color(0,0,0,0));

        NickInput = new CTextField("Type Nick", 20,Color.ORANGE);
        SettingsButton = new CButton("<- Settings <-", 18,Color.ORANGE);
        OnlineConfigButton = new CButton("-> Multiplayer ->", 18,Color.ORANGE);
        FullScoreboardButton = new CButton("<- Scoreboard ->", 18,Color.ORANGE);

        secondPartPanel.add(new CLabel("Enter Nickname", 20,Color.WHITE));
        secondPartPanel.add(NickInput);

        thirdPartPanel.add(new CLabel("Settings", 20,Color.WHITE));
        thirdPartPanel.add(SettingsButton);

        fourthPartPanel.add(new CLabel("Online Configuration", 20,Color.WHITE));
        fourthPartPanel.add(OnlineConfigButton);

        fifthPartPanel.add(new CLabel("Open Full Scoreboard", 20,Color.WHITE));
        fifthPartPanel.add(FullScoreboardButton);

        JPanel MainMenuPanel = new JPanel(new GridLayout(2,2));

        MainMenuPanel.setBackground(new Color(0,0,0,0));

        StartGameButton = new CButton("Play", 16,Color.GREEN,Color.BLACK);
        CloseGameButton = new CButton("Close",16,Color.RED,Color.BLACK);

        MainMenuPanel.add(new CLabel("Start Game!", 20,Color.WHITE));
        MainMenuPanel.add(new CLabel("Close Game", 20,Color.WHITE));
        MainMenuPanel.add(StartGameButton);
        MainMenuPanel.add(CloseGameButton);

        this.add(new CLabel("Swing Shotgun Game!", 42,Color.WHITE));
        this.add(new JLabel());
        this.add(secondPartPanel);
        this.add(thirdPartPanel);
        this.add(fourthPartPanel);
        this.add(fifthPartPanel);
        this.add(new JLabel());
        this.add(MainMenuPanel);
    }
}
```

Klasa „CPanel”:

```
package Custom_Panels;

import Config.CONFIG;

import Custom_Enums.ObjectType;
import Custom_Objects.*;

import Main_Package.Main;

import javax.swing.*;
```

```
import java.awt.*;
import java.util.ArrayList;
import java.util.List;

import static Custom_Panels.CShopPanel.*;

public class CPanel extends JPanel {
    protected List<GameObject> tempDrawingList;
    Image backgroundImage;
    public CPanel(Image backgroundImage){
        this.setLayout(null);
        this.backgroundImage = backgroundImage;
        tempDrawingList = new ArrayList<>();
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2D = (Graphics2D) g;
        g2D.drawImage(backgroundImage,0,0,CONFIG.WINDOW_WIDTH,CONFIG.WINDOW_HEIGHT,this);

        tempDrawingList.clear();
        tempDrawingList.addAll(Main.PlayerList);
        tempDrawingList.addAll(Main.BulletList);
        tempDrawingList.addAll(Main.EnemyList);
        tempDrawingList.addAll(Main.BoosterList);
        tempDrawingList.addAll(Main.TrapList);

        setFont(new Font("Comic Sans MS", Font.BOLD, 11));

        for (GameObject gameObject : tempDrawingList) {
            if (gameObject.visible) {
                if (gameObject.myType == ObjectType.ENEMY || gameObject.myType == ObjectType.PLAYER) {
                    g2D.setColor(Color.WHITE);
                    g2D.drawString(gameObject.objectName, (int) gameObject.cordX, (int) gameObject.cordY - 28);
                    g2D.setColor(Color.RED);
                    g2D.fillRect((int) gameObject.cordX, (int) gameObject.cordY - 20, 100, 20);
                    g2D.setColor(Color.GREEN);
                    g2D.fillRect((int) gameObject.cordX, (int) gameObject.cordY - 20, (int) gameObject.hpCurrent, 20);
                }

                if (CONFIG.SIMPLE_SHAPES)
                    g2D.fillRect((int) gameObject.cordX, (int) gameObject.cordY,20,20);
                else {
                    gameObject.myCPolygon.Render(g2D, gameObject.MyColor);
                }
            }
        }

        if (Main.GAME_RUNNING)
            if (Main.PlayerList.contains(Main.playerHolder))
                CQuestPanel.RenderQuest(g2D,275,525);

        if (Main.displayScoreboard)
            if (Main.PlayerList.contains(Main.playerHolder))
                CScoreboardPanel.RenderScoreboard(g2D,1280,720);

        if (Main.displayPlayerList)
            if (Main.PlayerList.contains(Main.playerHolder))
                CPlayerList.RenderPlayerList(g2D,1280,720);

        if (Main.displayShop)
            if (Main.PlayerList.contains(Main.playerHolder))
                RenderShop(g2D,1280,720);


    }
}
```

Klasa „CPlayerList”:

```
package Custom_Panels;

import Config.CONFIG;
import Custom_Objects.GameObject;
import Main_Package.Main;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;

public class CPlayerList {
    public static List<String> PlayerList = new ArrayList<>();

    public static void UpdatePlayerList(){
        PlayerList.clear();
        List<GameObject> gameObjectList = new ArrayList<>();
        gameObjectList.addAll(Main.PlayerList);
        gameObjectList.addAll(Main.EnemyList);

        for (GameObject temp : gameObjectList)
            PlayerList.add(temp.objectName + " Weapon: " + temp.myWeapon + " DMG: " + temp.damage + " K: " +
temp.statKillCount + " T: " + temp.statTrapCount + " B: " + temp.statBoosterCount);
    }

    public static void RenderPlayerList(Graphics2D graphics2D, int windowWidth, int windowHeight) {
        int newX = (CONFIG.WINDOW_WIDTH / 2) - (windowWidth/2);
        int newY = (CONFIG.WINDOW_HEIGHT / 2) - (windowHeight/2);
```

```
        graphics2D.setColor(new Color(0,0,0,100));
        graphics2D.fillRect(newX,newY,windowWidth,windowHeight);

        graphics2D.setColor(Color.WHITE);
        Font ContentFont = new Font("Comic Sans MS", Font.BOLD, 12);
        Font TitleFont = new Font("Comic Sans MS", Font.BOLD, 94);

        FontMetrics titleFontMetrics = graphics2D.getFontMetrics(TitleFont);
        int titleCordX = newX + (windowWidth - titleFontMetrics.stringWidth("Players Online")) / 2;

        FontMetrics contentFontMetrics = graphics2D.getFontMetrics(ContentFont);
        int contentCordX;

        graphics2D.setFont(TitleFont);
        graphics2D.drawString("Players Online",titleCordX,newY);

        graphics2D.setFont(ContentFont);

        for (int i = 0; i < 20; i++) {
            if (i < PlayerList.size()) {
                contentCordX = newX + (windowWidth - contentFontMetrics.stringWidth(PlayerList.get(i))) / 2;
                graphics2D.drawString(PlayerList.get(i), contentCordX, newY + contentFontMetrics.getHeight() * ((i + 1) *
2));
            }
            else {
                break;
            }
        }

    }

}
```

Klasa „CQuestPanel”:

```
package Custom_Panels;

import Config.CONFIG;
import Main_Package.*;

import java.awt.*;

import static Main_Package.Main.MyPlayerIndex;


public class CQuestPanel {
    public static void RenderQuest(Graphics2D graphics2D, int windowWidth, int windowHeight) {
        int newX = CONFIG.WINDOW_WIDTH - (windowWidth + 25);

        graphics2D.setColor(new Color(0,0,0,100));
        graphics2D.fillRect(newX,0,windowWidth,windowHeight);

        graphics2D.setColor(Color.WHITE);
        Font ContentFont = new Font("Comic Sans MS", Font.BOLD, 15);
        Font TitleFont = new Font("Comic Sans MS", Font.BOLD, 28);

        FontMetrics titleFontMetrics = graphics2D.getFontMetrics(TitleFont);
        FontMetrics contentFontMetrics = graphics2D.getFontMetrics(ContentFont);

        int titleCordX = newX + (windowWidth - titleFontMetrics.stringWidth("Active Quest")) / 2;

        graphics2D.setFont(TitleFont);
        graphics2D.drawString("Active Quest",titleCordX,titleFontMetrics.getHeight());

        graphics2D.setFont(ContentFont);
        graphics2D.setColor(Color.ORANGE);

        String tempString;
        int contentCordX;
        int skips = 0;

        for (int i = 0; i < 6; i++) {
            tempString = "";

            switch (i) {
                case 0 -> {
                    if (Main.activeQuestObject.targetStatKillCount == 0) {
                        skips++;
                        continue;
                    }
                    tempString = "Kill Enemies: " + Main.PlayerList.get(MyPlayerIndex).statKillCount + " / " +
Main.activeQuestObject.targetStatKillCount;
                }
                case 1 -> {
                    if (Main.activeQuestObject.targetStatTrapCount == 0) {
                        skips++;
                        continue;
                    }
                    tempString = "Punch Traps: " + Main.PlayerList.get(MyPlayerIndex).statTrapCount + " / " +
Main.activeQuestObject.targetStatTrapCount;
                }
                case 2 -> {
                    if (Main.activeQuestObject.targetStatBoosterCount == 0) {
                        skips++;
                        continue;
                    }
```

```
                    tempString = "Catch Boosters: " + Main.PlayerList.get(MyPlayerIndex).statBoosterCount + " / " +
Main.activeQuestObject.targetStatBoosterCount;
                }
                case 3 -> {
                    if (Main.activeQuestObject.targetStatDamageTaken == 0) {
                        skips++;
                        continue;
                    }
                    tempString = "Take Damage: " + Main.PlayerList.get(MyPlayerIndex).statDamageTaken + " / " +
Main.activeQuestObject.targetStatDamageTaken;
                }
                case 4 -> {
                    if (Main.activeQuestObject.targetStatDamageDealt == 0) {
                        skips++;
                        continue;
                    }
                    tempString = "Deal Damage: " + Main.PlayerList.get(MyPlayerIndex).statDamageDealt + " / " +
Main.activeQuestObject.targetStatDamageDealt;
                }
                case 5 -> {
                    if (Main.activeQuestObject.targetStatShootCount == 0) {
                        skips++;
                        continue;
                    }
                    tempString = "Shoot Count: " + Main.PlayerList.get(MyPlayerIndex).statShootCount + " / " +
Main.activeQuestObject.targetStatShootCount;
                }
            }
            contentCordX = newX + (windowWidth - contentFontMetrics.stringWidth(tempString)) / 2;
            graphics2D.drawString(tempString, contentCordX, 50 + contentFontMetrics.getHeight() * (2 * (i + 1 - skips)));
        }

        tempString = "Money: " + Main.activeQuestObject.rewardCash + "$";
        contentCordX = newX + (windowWidth - contentFontMetrics.stringWidth(tempString)) / 2;
        graphics2D.drawString(tempString, contentCordX, windowHeight - contentFontMetrics.getHeight() * 3);

        tempString = "Healing: " + Main.activeQuestObject.rewardHeal;
        contentCordX = newX + (windowWidth - contentFontMetrics.stringWidth(tempString)) / 2;
        graphics2D.drawString(tempString, contentCordX, windowHeight - contentFontMetrics.getHeight());

        graphics2D.setColor(Color.WHITE);
        graphics2D.setFont(TitleFont);
        tempString = "Reward";
        contentCordX = newX + (windowWidth - titleFontMetrics.stringWidth(tempString)) / 2;
        graphics2D.drawString(tempString, contentCordX, windowHeight - contentFontMetrics.getHeight() * 5);


    }
}
```

Klasa „CScoreboardPanel”:

```
package Custom_Panels;

import Config.CONFIG;
import Custom_Objects.SavedObject;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;

public class CScoreboardPanel {
    public static List<String> ScoreboardList = new ArrayList<>();
    public static void UpdateScoreboardList(){
        ScoreboardList.clear();
        List<SavedObject> savedObjectList = SavedObject.Read();
        for (int i = 0; i < savedObjectList.size(); i++) {
            ScoreboardList.add(savedObjectList.get(i).toScoreboardString(i));
        }
    }
    public static void RenderScoreboard(Graphics2D graphics2D, int windowWidth, int windowHeight) {
        int newX = (CONFIG.WINDOW_WIDTH / 2) - (windowWidth/2);
        int newY = (CONFIG.WINDOW_HEIGHT / 2) - (windowHeight/2);

        graphics2D.setColor(new Color(0,0,0,100));
        graphics2D.fillRect(newX,newY,windowWidth,windowHeight);

        graphics2D.setColor(Color.WHITE);
        Font ContentFont = new Font("Comic Sans MS", Font.BOLD, 24);
        Font TitleFont = new Font("Comic Sans MS", Font.BOLD, 94);

        FontMetrics titleFontMetrics = graphics2D.getFontMetrics(TitleFont);
        int titleCordX = newX + (windowWidth - titleFontMetrics.stringWidth("Top 10 Players")) / 2;

        FontMetrics contentFontMetrics = graphics2D.getFontMetrics(ContentFont);
        int contentCordX;

        graphics2D.setFont(TitleFont);
        graphics2D.drawString("Top 10 Players",titleCordX,newY);
        graphics2D.setFont(ContentFont);

        for (int i = 0; i < 10; i++) {
            if (i < ScoreboardList.size()) {
```

```
                contentCordX = newX + (windowWidth - contentFontMetrics.stringWidth(ScoreboardList.get(i))) / 2;
                graphics2D.drawString(ScoreboardList.get(i), contentCordX, newY + contentFontMetrics.getHeight() * ((i + 1) *
2));
            }
            else {
                break;
            }
        }
    }

}
```

Klasa „CSettingsPanel”

```
package Custom_Panels;

import Custom_Elements.CButton;
import Custom_Elements.CLabel;
import Custom_Elements.CSlider;

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

public class CSettingsPanel extends JPanel {
    public boolean visible = false;
    public CButton AnimationButton, SimplerShapesButton;
    public CSlider backgroundVolumeSlider, shootingVolumeSlider;
    public CSettingsPanel() {

        JPanel volumeSettingPanel = new JPanel(new GridLayout(4,0));
        volumeSettingPanel.setBackground(new Color(0,0,0,0));

        backgroundVolumeSlider = new CSlider();
        shootingVolumeSlider = new CSlider();

        volumeSettingPanel.add(new CLabel("Change Background Volume",20,Color.WHITE));
        volumeSettingPanel.add(backgroundVolumeSlider);
        volumeSettingPanel.add(new CLabel("Change Shooting Sound Volume",20,Color.WHITE));
        volumeSettingPanel.add(shootingVolumeSlider);

        JPanel LocalConfigurationPanel = new JPanel(new GridLayout(4,0));
        LocalConfigurationPanel.setBackground(new Color(0,0,0,0));

        AnimationButton = new CButton("Turn Off Animations",18,Color.GREEN,Color.BLACK);
        SimplerShapesButton = new CButton("Turn On Simple Shapes",18, Color.RED,Color.BLACK);

        LocalConfigurationPanel.add(new CLabel("Turn Off / On Animations:", 20,Color.WHITE));
        LocalConfigurationPanel.add(AnimationButton);
        LocalConfigurationPanel.add(new CLabel("Turn Off / On Simple Shapes:", 20,Color.WHITE));
        LocalConfigurationPanel.add(SimplerShapesButton);

        setLayout(new GridLayout(3,0));
        setBackground(new Color(0,0,0,100));
        add(volumeSettingPanel);
        add(LocalConfigurationPanel);
        add(new JLabel());
    }
}
```

Klasa „CShopPanel”:

```
package Custom_Panels;

import Config.CONFIG;
import Custom_Objects.PurchaseOption;
import Main_Package.*;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;

import static Main_Package.Main.MyPlayerIndex;

public class CShopPanel {
    public static List<PurchaseOption> ShopList = new ArrayList<>();
    public static Image shopkeeper = null;
    public static boolean shopInitialized = false;
    public static List <Rectangle> fakeButtons = new ArrayList<>();

    public static void RenderShop(Graphics2D graphics2D, int windowWidth, int windowHeight) {

        int newX = (CONFIG.WINDOW_WIDTH / 2) - (windowWidth/2);
        int newY = (CONFIG.WINDOW_HEIGHT / 2) - (windowHeight/2);

        graphics2D.setColor(new Color(0,0,0,100));
        graphics2D.fillRect(newX,newY,windowWidth,windowHeight);


        Font ContentFont = new Font("Comic Sans MS", Font.BOLD, 18);
        Font WalletFont = new Font("Comic Sans MS", Font.BOLD, 24);
```

```
        FontMetrics contentFontMetrics = graphics2D.getFontMetrics(ContentFont);
        int contentCordX_0, contentCordX_1, contentCordX_2;

        graphics2D.setColor(Color.GREEN);
        graphics2D.setFont(WalletFont);

        if (Main.PlayerList.contains(Main.playerHolder))
            graphics2D.drawString("Wallet: " + Main.PlayerList.get(MyPlayerIndex).wallet + "$", newX + 50, newY + 50);

        graphics2D.setFont(ContentFont);
        graphics2D.setColor(Color.WHITE);

        graphics2D.drawImage(shopkeeper,newX + windowWidth/2 - 200, newY,400,400,null);

        for (int i = 0; i < 10; i++) {
            if (i < ShopList.size()) {

                graphics2D.setColor(Color.WHITE);

                contentCordX_0 = newX + (windowWidth - contentFontMetrics.stringWidth(ShopList.get(i).optionName)) / 2;
                contentCordX_1 = newX + (windowWidth - contentFontMetrics.stringWidth(ShopList.get(i).optionPrice + "$")) /
2;
                contentCordX_2 = newX + (windowWidth - contentFontMetrics.stringWidth("Purchase Me")) / 2;

                graphics2D.drawString(ShopList.get(i).optionName, (int) (contentCordX_0 * 0.5), 375 + newY +
contentFontMetrics.getHeight() * ((i + 1) * 2));
                graphics2D.drawString(ShopList.get(i).optionPrice + "$", contentCordX_1, 375 + newY +
contentFontMetrics.getHeight() * ((i + 1) * 2));

                if (Main.PlayerList.get(Main.MyPlayerIndex).CanAfford(CONFIG.SHOP_PRICES[i]))
                    graphics2D.setColor(Color.GREEN);
                else
                    graphics2D.setColor(Color.RED);

                graphics2D.drawString("Purchase Me", (int) (contentCordX_2 * 1.5), 375 + newY +
contentFontMetrics.getHeight() * ((i + 1) * 2));

                if (!shopInitialized) {
                    fakeButtons.add(new Rectangle((int) (contentCordX_2 * 1.5), (375 + newY + contentFontMetrics.getHeight()
* ((i + 1) * 2)) - (int) (contentFontMetrics.getHeight()/1.5),contentFontMetrics.stringWidth("Purchase
Me"),contentFontMetrics.getHeight()));
                }

            }
        }

        shopInitialized = true;

    }
    public static void UpdateShopList(){
        ShopList.clear();
        ShopList.add(new PurchaseOption("Shotgun",CONFIG.SHOP_PRICES[0]));
        ShopList.add(new PurchaseOption("Sniper Rifle",CONFIG.SHOP_PRICES[1]));
        ShopList.add(new PurchaseOption("Pistol",CONFIG.SHOP_PRICES[2]));
        ShopList.add(new PurchaseOption("Rifle",CONFIG.SHOP_PRICES[3]));
        ShopList.add(new PurchaseOption("Healing Potion",CONFIG.SHOP_PRICES[4]));
        ShopList.add(new PurchaseOption("Damage Upgrade",CONFIG.SHOP_PRICES[5]));

    }

}
```

Damian D. bo RODO

PANS Informatyka Rok II

2022 / 2023