Содержание

[Задание на курсовую работу 2](#_Toc155370113)

[Введение 3](#_Toc155370114)

[1. Установка IntelliJ IDEA и JavaFX 4](#_Toc155370115)

[1.1. Установка IntelliJ IDEA 4](#_Toc155370116)

[1.2. Установка JavaFX 7](#_Toc155370117)

[2. Планирование интерфейса и функций. 11](#_Toc155370118)

[Основное окно игры 11](#_Toc155370119)

[Начальное меню 12](#_Toc155370120)

[Меню читера 12](#_Toc155370121)

[3. Прототипирование 13](#_Toc155370122)

[4. Описание классов 15](#_Toc155370123)

[5. Вывод 17](#_Toc155370124)

[6. Список литературы 18](#_Toc155370125)

[7. Приложение 19](#_Toc155370126)

[Класс Main 19](#_Toc155370127)

[Класс Enemy 21](#_Toc155370128)

[Класс MainCharacter 23](#_Toc155370129)

[Класс Drawer 31](#_Toc155370130)

[Класс CheatsController 33](#_Toc155370131)

[Класс MainController 36](#_Toc155370132)

[Класс MenuController 48](#_Toc155370133)

# Задание на курсовую работу

Требуется разработать графическое приложение на языке Java с использованием выбранной библиотеки в IDE IntelliJ IDEA.

Для разработки приложения была выбрана библиотека JavaFX – это библиотека, предназначенная для разработки приложений с насыщенным графическим интерфейсом. Заменяет использовавшуюся ранее библиотеку Swing.

Тема курсовой работы – игра для демонстрации возможностей выбранной библиотеки (JavaFX).

Игра должно обладать следующими свойствами:

1. Игра должна иметь графический интерфейс, созданный только из элементов выбранной библиотеки.
2. Игра должна обладать полным набором функций, для выбранного жанра игры.
3. В интерфейсе должны раскрываться возможности графической библиотеки.

# Введение

Компьютерная графика (англ. computer graphics, или CG) — способ создания графических изображений и визуальной информации с помощью специальных программ. Понятие компьютерной графики существует на стыке дизайна и искусства, объединяя классическое рисование и современные технологии. Компьютерная графика — это результат взаимодействия человека и машины.

Чтобы создать ручную графику, например, сложный градиент с помощью ватмана и аэрографа, потребуется 3-4 часа, а в Illustrator или Photoshop это займёт несколько секунд. При этом результат будет примерно одинаковый. Созданную вручную графику относят больше к изобразительному искусству, а компьютерную графику — к дизайну. Разница между двумя видами графики — в степени присутствия человека и в балансе искусства и технологий.

Вся окружающая нас графическая информация сделана при помощи компьютера: книги, журналы, упаковки, обои, плакаты, инструкции, сайты и приложения и т. д. Часто ручная графика гармонично интегрируется в компьютерную: иллюстратор рисует изображение тушью, акварелью или любым другим инструментом, а затем оцифровывает его, встраивает в макет и обрабатывает. Такая интеграция превращает ручной рисунок в компьютерную графику.

# 1. Установка IntelliJ IDEA и JavaFX

IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

## 1.1. Установка IntelliJ IDEA

1. Перейти на официальный сайт JetBrains (<https://www.jetbrains.com/>)

2. Выбрать продукт IntelliJ IDEA (<https://www.jetbrains.com/idea/>)

2.1. Разработка будет вестись на операционной системе Windows (определяется автоматически, <https://www.jetbrains.com/idea/download/?section=windows>)

3. Загрузить Community Edition (бесплатная версия ПО) (<https://download.jetbrains.com/idea/ideaIC-2023.3.2.exe>)

4. Запустить установщик IntelliJ IDEA CE

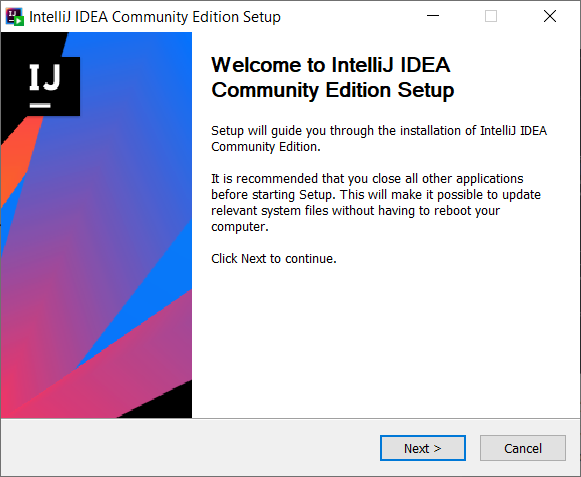


Рисунок 1. Приветствие установщика IntelliJ IDEA CE

5. Выбрать место установки

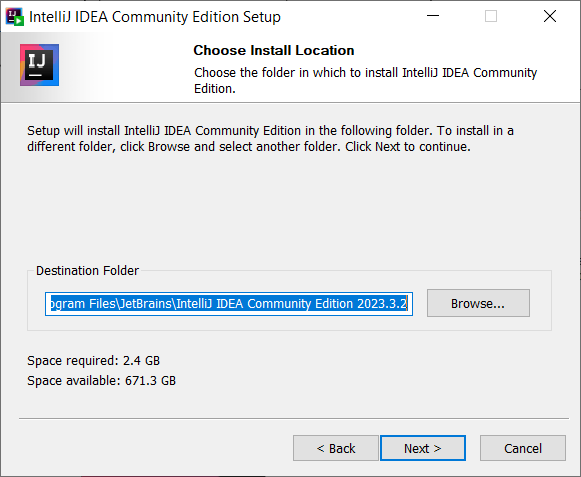


Рисунок 2. Выбор папки установки

6. Выбрать желаемые настройки (ярлык на рабочем столе, обновление контекстного меню, ассоциации форматов, обновление регистра)

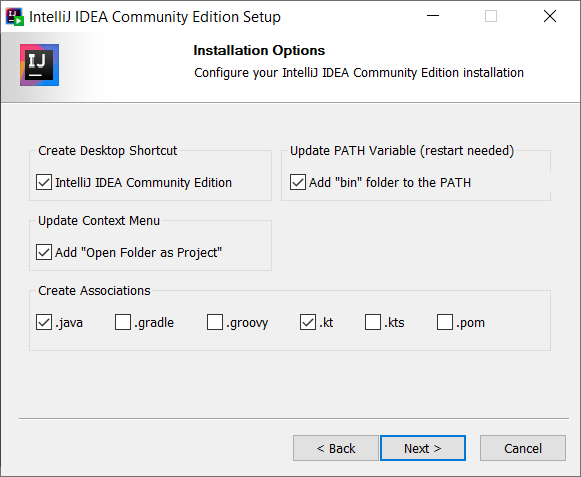


Рисунок 3. Выбор настроек установки

7. Выбор папки стартового меню

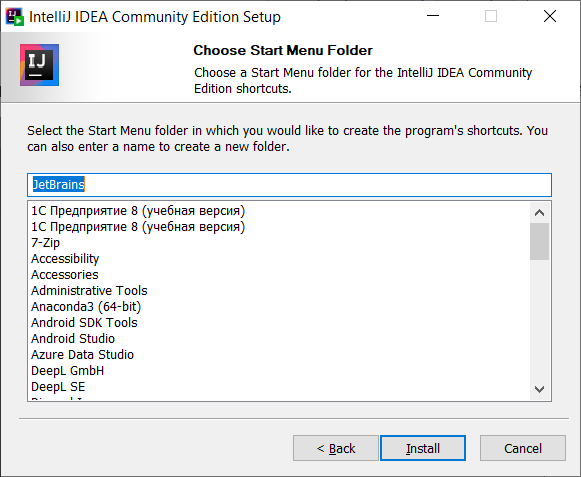


Рисунок 4. Выбор папки стартового меню

8. Установка

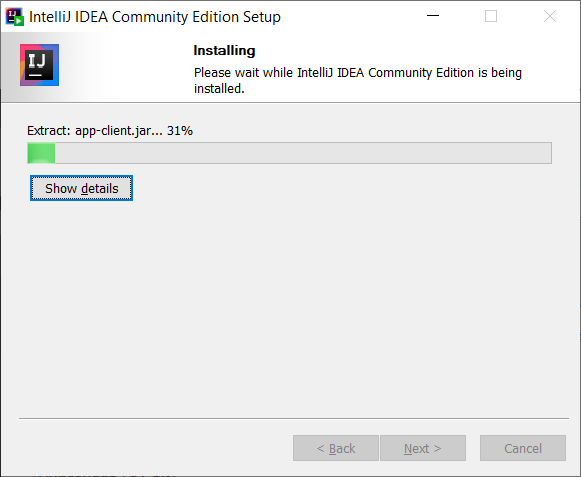


Рисунок 5. Установка

9. Установка завершена

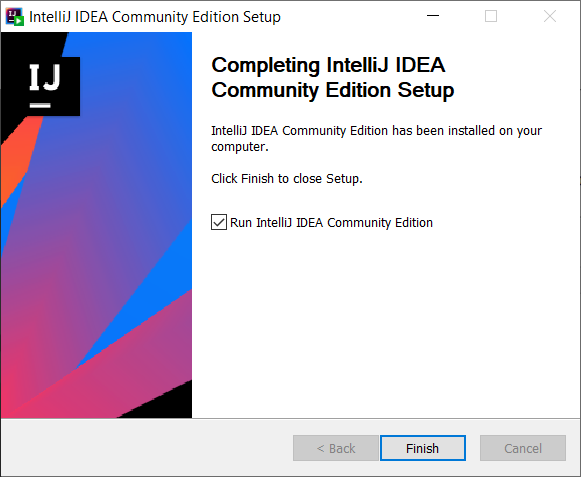


Рисунок 6. Завершение установки

## 1.2. Установка JavaFX

Для подключения библиотеки нужно создать новый проект.

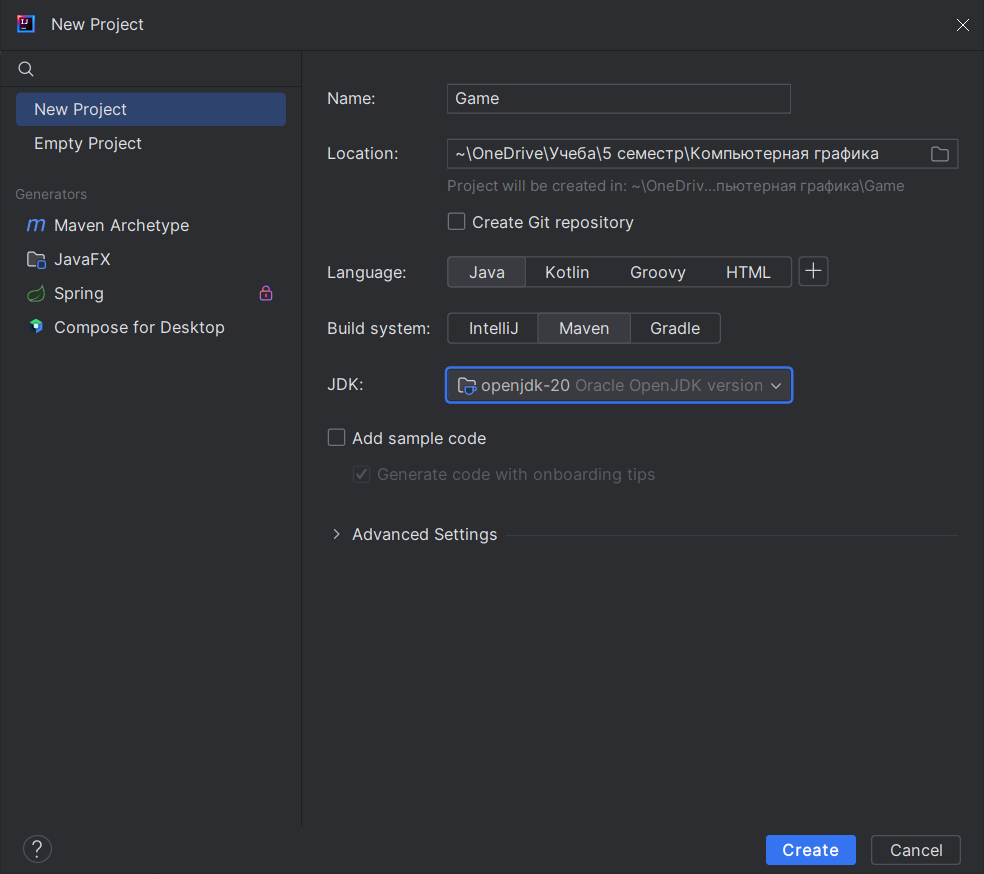


Рисунок 7. Окно создания нового проекта

При создании проекта можно указать предустановленный генератор кода (при создании будут сгенерированы примеры кода JavaFX).

Если нет предустановленных JDK (в данном проекте используется openjdk-20), можно установить нужную через интерфейс IDE.

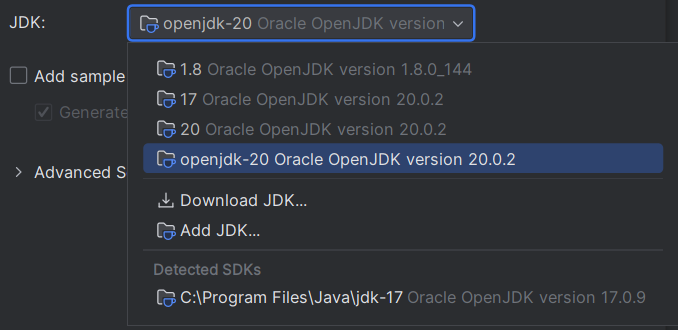


Рисунок 8. Выбор JDK

Выбран пункт Download JDK…

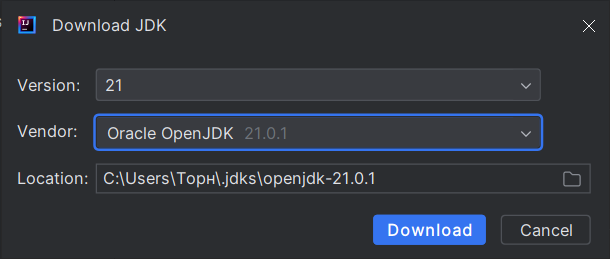


Рисунок 9. Загрузка JDK с использованием интерфейса IDE

1. После создания проекта File -> Project Structure -> Libraries -> +

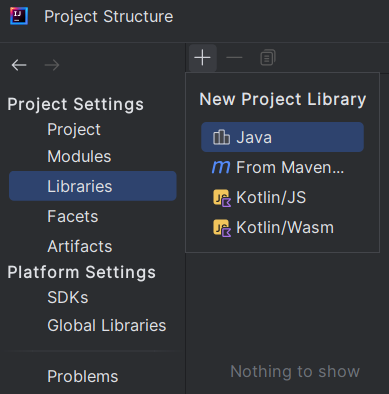


Рисунок 10. Подключение библиотеки к проекту

1.1. Если библиотека была скачана заранее, выбираем Java, после чего выбираем папку, где находится библиотека. Если библиотека не была скачана, можно воспользоваться интерфейсом IDE и скачать нужную библиотеку с репозитория Maven (пункт From Maven…).

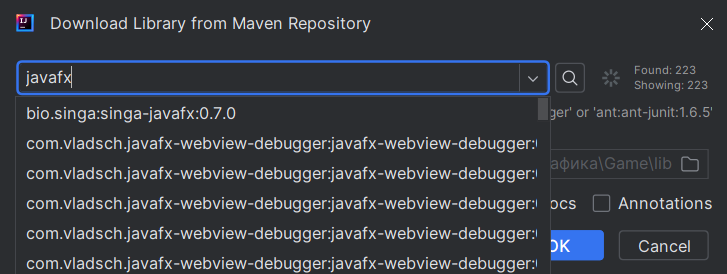


Рисунок 11. Поиск библиотеки в репозитории Maven

2. Для скачивания библиотеки переходим на официальный сайт JavaFX (<https://openjfx.io/>)

3. При нажатии на кнопку Download происходит перенаправление на сайт Gluon (<https://gluonhq.com/products/javafx/>)

4. Пользуясь фильтрами, выбираем нужную версию библиотеки (выбрана версия 21, как последняя выпущенная)

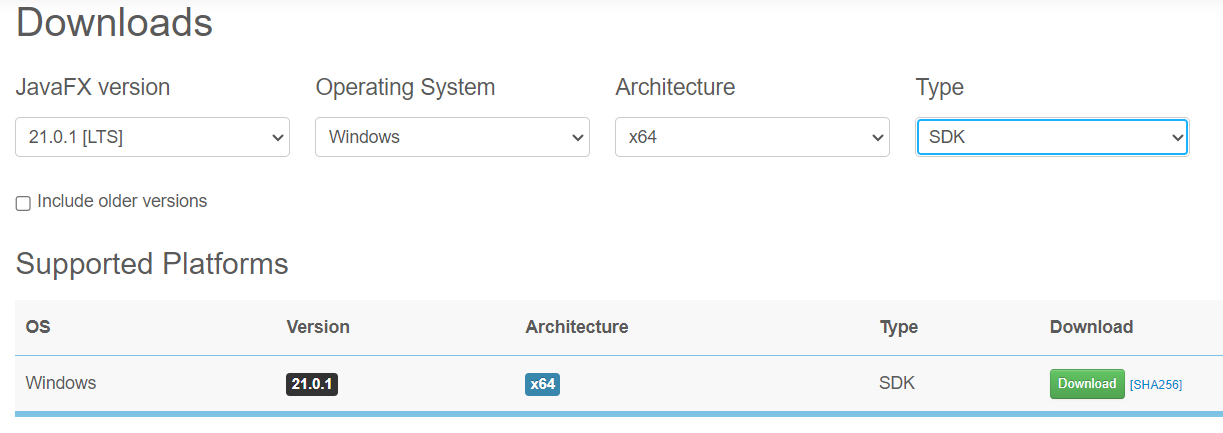


Рисунок 12. Загрузка библиотеки с сайта Gluon

5. Скачиваем библиотеку (<https://download2.gluonhq.com/openjfx/21.0.1/openjfx-21.0.1_windows-x64_bin-sdk.zip>)

6. Распаковываем скачанный архив и подключаем его к проекту

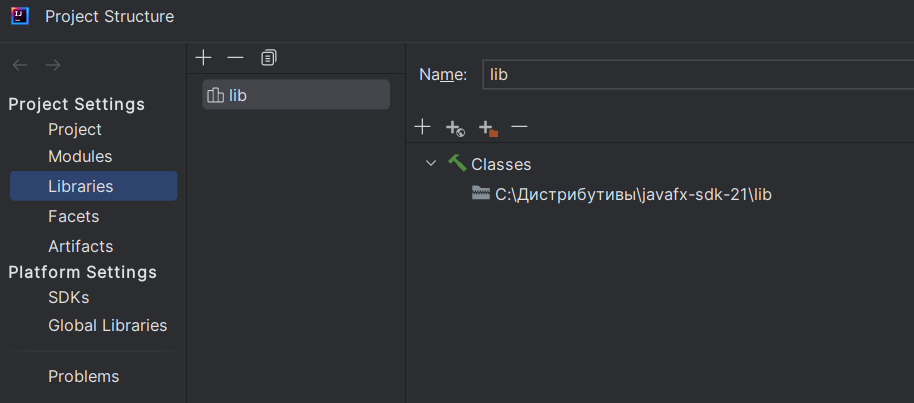


Рисунок 13. Окно библиотек проекта после подключения библиотеки

7. Установка библиотеки завершена

# 2. Планирование интерфейса и функций.

Для разработки игры был выбран жанр Idle (Инкрементальная игра).

Смысл данного жанра игр в минимальном участии игрока в игровом процессе. Игрок участвует только в «прокачке» персонажа, чтобы он стал сильнее.

## Основное окно игры

Таким образом, интерфейс игры будет делиться на две части: окно «прокачки» и игровое поле.



Рисунок 14. Планирование областей основного окна игры

В левой части экрана будут отображаться:

* Информация о персонаже
* Элементы усиления
* Настройки
* Дополнительные элементы, не являющиеся непосредственно игрой, но нужные для некоторых функций

В правой части экрана будут отображаться:

* Непосредственно игровое поле (персонаж, враги)
* Индикаторы здоровья и опыта
* Счетчик игровой валюты
* Меню взаимодействия (возможно дублирование функций с другой части экрана)

Таким образом, в левой части экрана рационально будет использовать вкладки (слишком много элементов и информации, стоит разделить на группы). Получены следующие группы-вкладки:

* Level – меню уровней персонажа и врага
* Upgrades – меню усиления персонажа за игровую валюту
* Stats – статистика персонажа (только отображение, без взаимодействия)
* Settings – меню настроек (сохранение и загрузка, скорость игры, выход, меню читера)

Планирование основного окна игры завершено.

## Начальное меню

В начальном меню должны быть возможности:

* Старт новой игры
* Загрузка сохранения
* Выход

Таким образом, интерфейс меню не предполагает никаких интересных решений.

Планирование начального меню завершено.

## Меню читера

Для демонстрации возможностей игры потребуется много времени, потому было принято решение разработать меню читера, позволяющее получить результат сразу.

Это меню должно иметь следующие функции:

* Увеличение уровня персонажа и очков развития (отдельно)
* Увеличение количества игровой валюты

Примечание: очки развития можно получить только путем повышения уровня (за исключением меню читера)

Для удобства требуется возможность выбрать предустановленное значение инкремента или ввести свое.

Планирование меню читера завершено.

# 3. Прототипирование

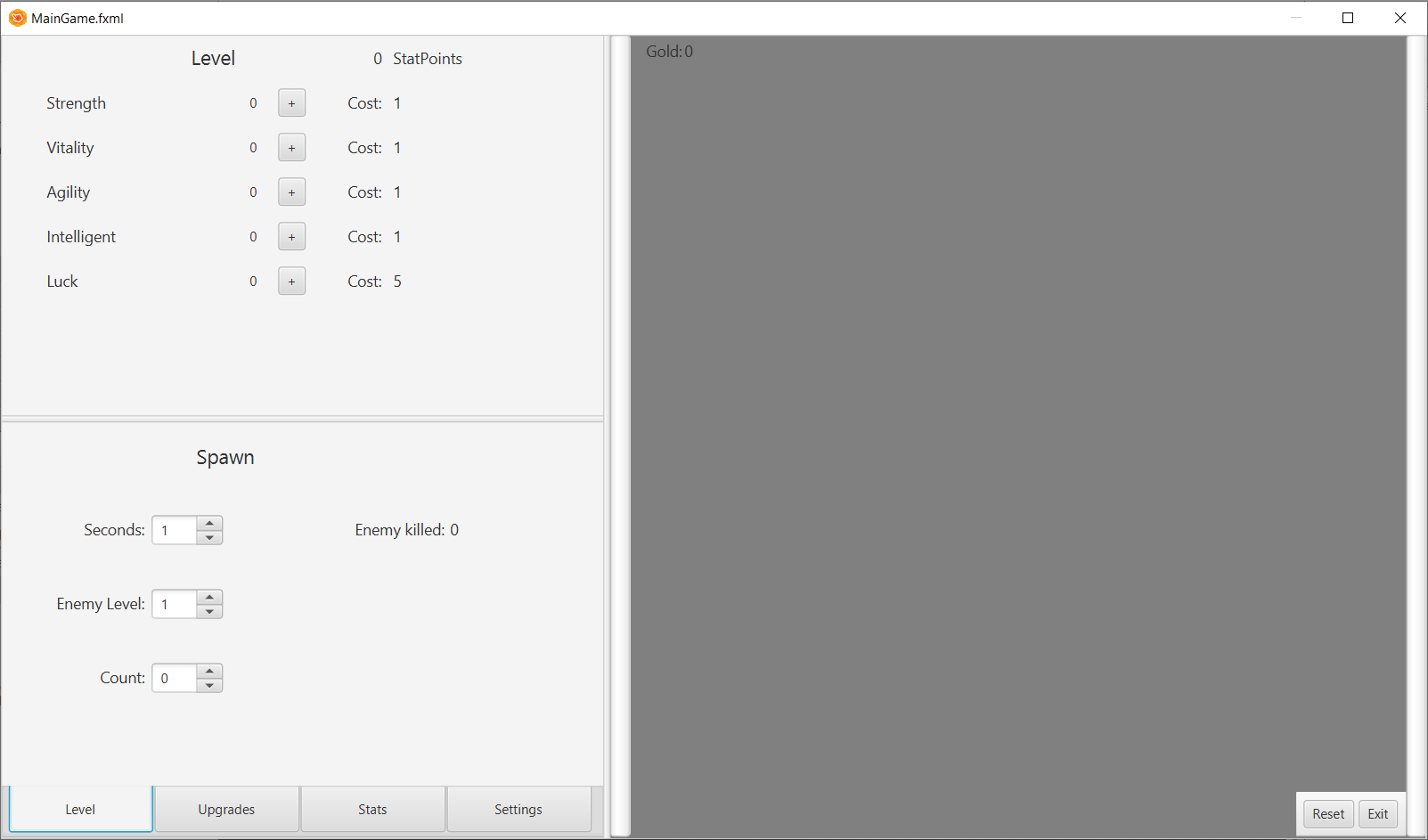


Рисунок 15. Основное окно игры, вкладка Level

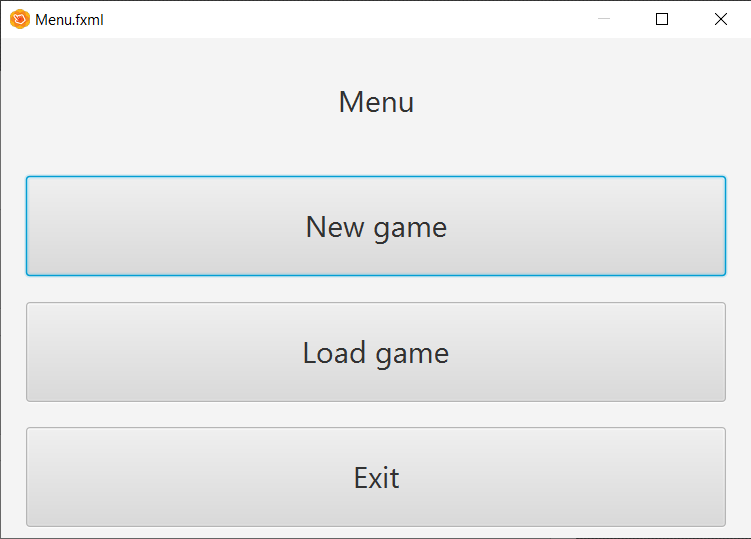


Рисунок 16. Начальное меню игры

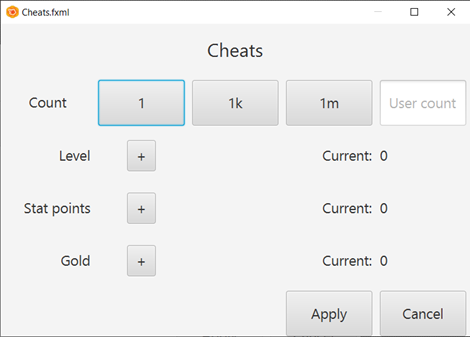


Рисунок 17. Меню читера

# 4. Описание классов

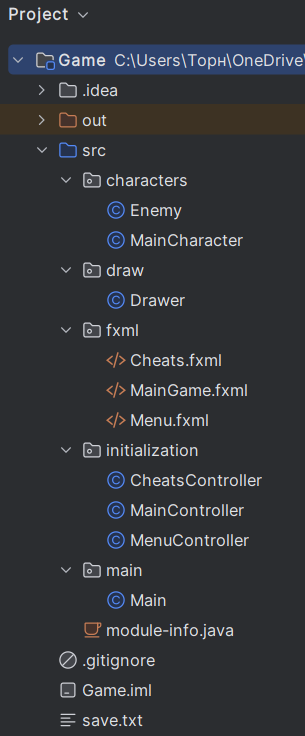


Рисунок 18. Структура проекта

Описание классов:

1. Enemy – класс врага, включает функции по его инициализации и взаимодействиям.
2. MainCharacter – класс основного персонажа, включает функции по его инициализации (в том числе загрузки из файла сохранения), взаимодействиям, сохранению
3. Drawer – класс прорисовки игрового поля и сопутствующих функций.
4. CheatsController – контроллер меню читера (связанный fxml-файл – Cheats.fxml)
5. MainController – контроллер основного окна игры (связанный fxml-файл –MinGame.fxml)
6. MenuController – контроллер начального меню игры (связанный fxml-файл – Menu.fxml)
7. Main – этот класс наследует класс Application из библиотеки JavaFX и является основным классом, в котором происходит запуск программы и открытие основного окна.
   1. main – метод, который запускает программу.
   2. start – метод, который создает основное окно программы.

# 5. Вывод

При работе с данной курсовой работой, мне удалось изучить возможности библиотеки JavaFX и освоить инструмент для разработки графического интерфейса Scene Builder. Резюмируя, JavaFX – удобная библиотека, позволяющая с относительно небольшими усилиями получить то, что тебе нужно. Набор доступных элементов широк и может использоваться для разных задач, особенно учитывая Scene Builder. Данный инструмент позволяет наглядно и в реальном времени изучить большинство элементов и удобен даже для новичков. Также стоит отметить, что JavaFX не требует большого количества кода для реализации функций, так как многое уже заложено в библиотеку или ее расширения.

# 6. Список литературы

1. Официальный сайт Oracle, документация JavaFX. URL: <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>

2. Stackoverflow, решение спорных моментов и багов. URL: <https://stackoverflow.com/>

# 7. Приложение

## Класс Main

package main;

import characters.MainCharacter;

import draw.Drawer;

import initialization.MainController;

import initialization.MenuController;

import javafx.animation.KeyFrame;

import javafx.animation.Timeline;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.geometry.Rectangle2D;

import javafx.scene.Scene;

import javafx.stage.Screen;

import javafx.stage.Stage;

import javafx.util.Duration;

import java.awt.\*;

import java.io.File;

import java.io.IOException;

import java.net.URL;

public class Main extends Application {

public static Stage stage;

private static FXMLLoader loader;

public static Timeline timeline;

public static Timeline attackTimeline;

public static Timeline spawnTimeline;

public static MainController mainController = new MainController();

public static MenuController menuController = new MenuController();

public static boolean attack = true;

private static int k = 0;

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage stage) {

try {

Main.stage = stage;

URL url = new File("src/fxml/Menu.fxml").toURI().toURL();

loader = new FXMLLoader(url);

loader.setController(menuController);

Scene scene = new Scene(loader.load());

stageStart(scene);

} catch (IOException exception) {

exception.printStackTrace();

}

}

private static void centerWindow() {

Rectangle2D screenBounds = Screen.getPrimary().getVisualBounds();

stage.setX((screenBounds.getWidth() - stage.getWidth()) / 2);

stage.setY((screenBounds.getHeight() - stage.getHeight()) / 2);

}

public static void stageStart(Scene scene) {

stage.setTitle("Game");

stage.setResizable(false);

stage.setScene(scene);

stage.show();

centerWindow();

}

public static void startGame(String s) {

try {

URL url = new File("src/fxml/MainGame.fxml").toURI().toURL();

loader = new FXMLLoader(url);

MainCharacter mainCharacter = new MainCharacter();

if (s == "load") {

mainCharacter = mainController.loadMaincharacter();

}

mainController.setMainCharacter(mainCharacter);

startMainTimeline(mainController);

startAttackTimeline(mainCharacter);

startSpawnTimeline(mainCharacter);

loader.setController(mainController);

Scene scene = new Scene(loader.load());

stageStart(scene);

} catch (IOException e) {

throw new RuntimeException(e);

}

}

public static void exit() {

stage.close();

}

public static void startMainTimeline(MainController mainController) {

timeline = new Timeline(new KeyFrame(Duration.seconds((double) 1 / 20 / mainController.getSpeed()), event -> {

mainController.move();

mainController.updateBars();

k++;

if (k > 20) {

mainController.updateMain();

k = 0;

}

}));

timeline.setCycleCount(Timeline.INDEFINITE);

timeline.play();

}

public static void startAttackTimeline(MainCharacter mainCharacter) {

attackTimeline = new Timeline(new KeyFrame((Duration.seconds((double) 1 / ((double) mainCharacter.getAttackSpeed() / 100) / mainController.getSpeed())), event -> {

attack = true;

}));

attackTimeline.setCycleCount(Timeline.INDEFINITE);

// System.out.println((double) 1 / ((double) mainCharacter.getAttackSpeed() / 100));

attackTimeline.play();

}

public static void startSpawnTimeline(MainCharacter mainCharacter) {

spawnTimeline = new Timeline(new KeyFrame((Duration.seconds(mainCharacter.getSpawnTime() / mainController.getSpeed())), event -> {

mainController.addEnemy();

}));

spawnTimeline.setCycleCount(Timeline.INDEFINITE);

spawnTimeline.play();

}

}

## Класс Enemy

package characters;

import draw.Drawer;

import java.util.Random;

import static main.Main.mainController;

public class Enemy {

private int level = 1;

private int hp = 10;

private int attack = 1;

private double speed = 1;

private int x;

private int y;

private int[] distance;

private int[] pos;

public Enemy(int x, int y, MainCharacter mainCharacter) {

this.level = mainCharacter.getEnemyLevel();

this.hp \*= level;

this.attack \*= level;

this.speed += (int) (level \* 0.01);

this.x = x;

this.y = y;

this.distance = new int[]{mainCharacter.getX() - x, mainCharacter.getY() - y};

this.pos = new int[]{(int) Math.signum(x - mainCharacter.getX()), (int) Math.signum(y - mainCharacter.getY())};

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public int getDistance() {

return (int) calcGyp(this.distance[0], this.distance[1]);

}

public int[] getPos() {

return pos;

}

public void setPos(int x, int y) {

this.x = x;

this.y = y;

}

public void slay(MainCharacter mainCharacter) {

Random random = new Random();

if (random.nextInt(100) < mainCharacter.getDodgeChance()) {

} else if (random.nextInt(100) < mainCharacter.getBlockChance()) {

mainCharacter.dealDamage((int) (getAttack() - mainCharacter.getBlockPower()));

} else {

mainCharacter.dealDamage((int) (getAttack() - mainCharacter.getArmor()));

}

}

public void moveEnemy(MainCharacter mainCharacter) {

if (this.intersects(mainCharacter)) {

slay(mainCharacter);

} else {

x -= (int) (calcDegree() \* speed \* 10 \* mainController.getSpeed()) \* pos[0];

y -= (int) (calcDegreeRe() \* speed \* 10 \* mainController.getSpeed()) \* pos[1];

reCalcDistance(mainCharacter);

reCalcPos(mainCharacter);

}

}

public void reCalcDistance(MainCharacter mainCharacter) {

this.distance = new int[]{Math.abs(mainCharacter.getX() - this.x), Math.abs(mainCharacter.getY() - this.y)};

}

public void reCalcPos(MainCharacter mainCharacter) {

this.pos = new int[]{(int) Math.signum(this.x - mainCharacter.getX()), (int) Math.signum(this.y - mainCharacter.getY())};

}

public boolean intersects(MainCharacter mainCharacter) {

return Math.abs(mainCharacter.getX() - getX()) < 30 && Math.abs(mainCharacter.getY() - getY()) < 30;

}

public double calcDegree() {

return (double) distance[0] / calcGyp(distance[0], distance[1]);

}

public double calcDegreeRe() {

return (double) distance[1] / calcGyp(distance[0], distance[1]);

}

private double calcGyp(double a, double b) {

return Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));

}

public int getHp() {

return hp;

}

public int getAttack() {

return attack;

}

public int getLevel() {

return level;

}

public double getSpeed() {

return speed;

}

public void dealDamage(int i, MainCharacter mainCharacter) {

if (hp <= i) {

Drawer.destroyEnemy(this, mainCharacter);

} else {

hp -= i;

}

}

}

## Класс MainCharacter

package characters;

import draw.Drawer;

import java.io.File;

import java.util.Arrays;

import java.util.List;

import java.util.Random;

import static main.Main.\*;

public class MainCharacter {

private static final int LEVEL\_COST\_INCREASE = 10;

private int level;

private int levelCost = LEVEL\_COST\_INCREASE;

private int sPoints;

private int strength;

private int vitality;

private int agility;

private int intelligent;

private int luck;

private int experience;

private int hp;

private int enemyLevel = 1;

private int spawnCount = 0;

private int spawnTime = 1;

private int enemyKilled = 0;

private int x;

private int y;

private int[][] upgrades;

private static final int[][] upgradeInc = {{10, 50, 20, 10, 50, 10}, {100, 10, 20, 50, 50, 100}};

private int[][] upgradeCost = {{10, 50, 20, 10, 50, 10}, {100, 10, 20, 50, 50, 100}};

private int gold;

private boolean crit = false;

public MainCharacter() {

this.level = 0;

this.strength = 0;

this.vitality = 0;

this.agility = 0;

this.intelligent = 0;

this.luck = 0;

this.experience = 0;

this.upgrades = new int[2][6];

this.hp = this.getHealth();

this.x = 369;

this.y = 359;

}

public MainCharacter(MainCharacter mainCharacter) {

this.level = mainCharacter.getLevel();

this.strength = mainCharacter.getStrength();

this.vitality = mainCharacter.getVitality();

this.agility = mainCharacter.getAgility();

this.intelligent = mainCharacter.getIntelligent();

this.luck = mainCharacter.getLuck();

this.experience = mainCharacter.getExperience();

this.upgrades = mainCharacter.getUpgrades();

this.hp = mainCharacter.getHealth();

this.sPoints = mainCharacter.getSPoints();

this.gold = mainCharacter.getGold();

this.enemyLevel = mainCharacter.getEnemyLevel();

this.spawnCount = mainCharacter.getSpawnCount();

this.spawnTime = mainCharacter.getSpawnTime();

this.x = 369;

this.y = 359;

}

public MainCharacter(String string) {

String[] list = string.split("\n");

this.level = Integer.parseInt(list[0]);

this.sPoints = Integer.parseInt(list[1]);

this.strength = Integer.parseInt(list[2]);

this.vitality = Integer.parseInt(list[3]);

this.agility = Integer.parseInt(list[4]);

this.intelligent = Integer.parseInt(list[5]);

this.luck = Integer.parseInt(list[6]);

this.experience = Integer.parseInt(list[7]);

this.enemyLevel = Integer.parseInt(list[8]);

this.spawnCount = Integer.parseInt(list[9]);

this.spawnTime = Integer.parseInt(list[10]);

this.enemyKilled = Integer.parseInt(list[11]);

this.upgrades = fromString(list[12]);

this.upgradeCost = fromString(list[13]);

this.gold = Integer.parseInt(String.valueOf(list[14]));

this.hp = this.getHealth();

this.x = 369;

this.y = 359;

}

private int[][] fromString(String string) {

int[][] list = new int[2][6];

string = string.replaceAll("[\\Q[\\E]", "");

string = string.replaceAll("[\\Q]\\E]", "");

String[] sList = string.split(",");

int k = 0;

for (String s : sList) {

list[(int) k / 6][k - 6 \* ((int) k / 6)] = Integer.parseInt(s.trim());

k++;

}

return list;

}

public int getLevel() {

return this.level;

}

public int getSPoints() {

return this.sPoints;

}

public int getStrength() {

return strength;

}

public int getStrengthCost() {

return getStrength() + 1;

}

public boolean incStrength() {

if (this.sPoints > getStrength()) {

this.strength++;

this.sPoints -= getStrength();

return true;

}

return false;

}

public int getVitality() {

return vitality;

}

public int getVitalityCost() {

return getVitality() + 1;

}

public boolean incVitality() {

if (this.sPoints > getVitality()) {

this.vitality++;

this.sPoints -= getVitality();

return true;

}

return false;

}

public int getAgility() {

return agility;

}

public int getAgilityCost() {

return getAgility() + 1;

}

public boolean incAgility() {

if (this.sPoints > getAgility()) {

this.agility++;

this.sPoints -= getAgility();

return true;

}

return false;

}

public int getIntelligent() {

return intelligent;

}

public int getIntelligentCost() {

return getIntelligent() + 1;

}

public boolean incIntelligent() {

if (this.sPoints > getIntelligent()) {

this.intelligent++;

this.sPoints -= getIntelligent();

return true;

}

return false;

}

public int getLuck() {

return luck;

}

public int getLuckCost() {

return getLuck() \* 5 + 5;

}

public boolean incLuck() {

if (this.sPoints > getLuck() \* 5 + 4) {

this.luck++;

this.sPoints -= getLuck() \* 5;

return true;

}

return false;

}

public int getExperience() {

return experience;

}

public int getLevelCost() {

return levelCost;

}

public void addExperience(int experience) {

this.experience += experience;

if (this.experience >= levelCost) {

this.experience -= levelCost;

levelCost += LEVEL\_COST\_INCREASE;

level++;

sPoints += level / 5 + 1;

}

}

public int getUpgradeLevel(int x, int y) {

return this.upgrades[x][y];

}

public int[][] getUpgrades() {

return upgrades;

}

public int[][] getUpgradesCost() {

return upgradeCost;

}

public void addGold(int gold) {

this.gold += gold;

}

public boolean removeGold(int cost) {

if (this.gold >= cost) {

this.gold -= cost;

return true;

}

return false;

}

public boolean incUpgrade(int x, int y) {

if (removeGold(getUpgradeCost(x, y))) {

upgrades[x][y]++;

return true;

}

return false;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public void moveMain(List<Enemy> enemyList) {

if (enemyList.isEmpty()) {

return;

}

Enemy en = enemyList.get(Drawer.getNearestEnemy(enemyList) - 1);

if (en.intersects(this)) {

if (attack) {

attackTimeline.stop();

attackTimeline.play();

attack = false;

slay(en);

}

} else {

crit = false;

x -= (int) (en.calcDegree() \* getSpeed() / 10 \* (-1) \* en.getPos()[0] \* mainController.getSpeed());

y -= (int) (en.calcDegreeRe() \* getSpeed() / 10 \* (-1) \* en.getPos()[1] \* mainController.getSpeed());

}

}

public void slay(Enemy en) {

Random random = new Random();

if (random.nextInt(100) < getCritChance()) {

en.dealDamage(getAttackPower() \* getCritPower() / 100, this);

crit = true;

} else {

crit = false;

en.dealDamage(getAttackPower(), this);

}

}

public void dealDamage(int attack) {

if (hp <= attack) {

Drawer.restartBattle(this);

} else {

hp -= attack;

}

}

public void restart() {

x = 369;

y = 359;

hp = getHealth();

gold = 0;

enemyLevel = 1;

timeline.stop();

timeline.play();

spawnTimeline.stop();

startSpawnTimeline(this);

}

public void healthRegen() {

hp += (int) (getHealth() \* getHealthRegeneration() \* 0.01);

}

public int getHp() {

return hp;

}

public int getUpgradeCost(int x, int y) {

return (upgradeCost[x][y] + upgradeInc[x][y] \* getUpgradeLevel(x, y));

}

public int getAttackPower() {

return (10 + getStrength() \* 2 + getUpgradeLevel(0, 0));

}

public int getAttackSpeed() {

return (100 + 5 \* getAgility() + 2 \* getUpgradeLevel(1, 0));

}

public double getCritChance() {

return (getAgility() + 0.5 \* getIntelligent() + getLuck() + 0.5 \* getUpgradeLevel(0, 1));

}

public int getCritPower() {

return (110 + 2 \* getStrength() + getAgility() + getUpgradeLevel(1, 1));

}

public int getSpeed() {

return (100 + 2 \* getVitality() + getAgility() + getUpgradeLevel(0, 2));

}

public double getArmor() {

return 0.5 \* getVitality() + 0.5 \* getUpgradeLevel(1, 2);

}

public int getHealth() {

return 100 + 5 \* getVitality() + 10 \* getUpgradeLevel(0, 3);

}

public double getHealthRegeneration() {

return getVitality() + 0.5 \* getUpgradeLevel(1, 3);

}

public double getBlockChance() {

return getStrength() + getVitality() + 0.1 \* getUpgradeLevel(0, 4);

}

public double getDodgeChance() {

return 2 \* getAgility() + getLuck() + 0.1 \* getUpgradeLevel(1, 4);

}

public int getSkillPower() {

return 10 + getIntelligent() + getUpgradeLevel(0, 5);

}

public int getSkillSpeed() {

return 100 + getAgility() + 5 \* getVitality() + 2 \* getUpgradeLevel(1, 5);

}

public int getBonusGold() {

return getLuck();

}

public int getBonusExp() {

return (int) (getLuck() + getIntelligent() \* 0.5);

}

public double getBlockPower() {

return getVitality() \* 0.5 + getUpgradeLevel(1, 2);

}

public int getGold() {

return gold;

}

public boolean getCrit() {

return crit;

}

public int getEnemyLevel() {

return enemyLevel;

}

public void setEnemyLevel(int i) {

enemyLevel = i;

}

public int getSpawnCount() {

return spawnCount;

}

public void setSpawnCount(int i) {

spawnCount = i;

}

public int getSpawnTime() {

return spawnTime;

}

public void setSpawnTime(int i) {

spawnTime = i;

spawnTimeline.stop();

startSpawnTimeline(this);

}

public int getEnemyKilled() {

return enemyKilled;

}

public void incEnemyKilled() {

enemyKilled++;

}

@Override

public String toString() {

StringBuilder string = new StringBuilder();

string.append(getLevel()).append("\n");

string.append(getSPoints()).append("\n");

string.append(getStrength()).append("\n");

string.append(getVitality()).append("\n");

string.append(getAgility()).append("\n");

string.append(getIntelligent()).append("\n");

string.append(getLuck()).append("\n");

string.append(getExperience()).append("\n");

string.append(getEnemyLevel()).append("\n");

string.append(getSpawnCount()).append("\n");

string.append(getSpawnTime()).append("\n");

string.append(getEnemyKilled()).append("\n");

string.append(Arrays.deepToString(getUpgrades())).append("\n");

string.append(Arrays.deepToString(getUpgradesCost())).append("\n");

string.append(getGold());

return string.toString();

}

public void addSPoints(int i) {

this.sPoints += i;

}

}

## Класс Drawer

package draw;

import characters.Enemy;

import characters.MainCharacter;

import javafx.scene.Group;

import javafx.scene.Node;

import javafx.scene.paint.Color;

import javafx.scene.paint.Paint;

import javafx.scene.shape.Circle;

import javafx.scene.text.Text;

import main.Main;

import java.util.ArrayList;

import java.util.List;

import java.util.ListIterator;

public class Drawer {

private static List<Enemy> enemyList = new ArrayList<>();

private static final Paint enemyAttackRadiusPaint = new Color(0.5, 0, 0, 0.5);

private static final Paint mainAttackRadiusPaint = new Color(0, 0.5, 0, 0.5);

public static Group drawBattleField() {

Group group = new Group();

for (Enemy en : enemyList) {

group.getChildren().add(drawEnemy(en));

group.getChildren().add(drawEnemyAttackRadius(en));

}

return group;

}

public static List<Enemy> getEnemyList() {

return enemyList;

}

public static int getNearestEnemy(List<Enemy> enemyList) {

if (enemyList.isEmpty()) {

return 0;

}

ListIterator<Enemy> iter = enemyList.listIterator();

int old = 1000;

int index = 0;

int indexRe = 0;

while (iter.hasNext()) {

int dist = iter.next().getDistance();

index++;

if (dist < old) {

old = dist;

indexRe = index;

}

}

return indexRe;

}

public static void destroyEnemy(Enemy enemy, MainCharacter mainCharacter) {

enemyList.remove(enemy);

mainCharacter.addGold(enemy.getLevel() + mainCharacter.getBonusGold());

mainCharacter.addExperience((enemy.getLevel() + mainCharacter.getBonusExp()));

mainCharacter.incEnemyKilled();

}

public static void addEnemy(int x, int y, MainCharacter mainCharacter) {

enemyList.add(new Enemy(x, y, mainCharacter));

}

public static Node drawEnemy(Enemy enemy) {

return new Circle(enemy.getX(), enemy.getY(), 10);

}

public static Node drawMainCharacter(MainCharacter mainCharacter) {

return new Circle(mainCharacter.getX(), mainCharacter.getY(), 20);

}

public static Node drawEnemyAttackRadius(Enemy enemy) {

Circle circle = new Circle(enemy.getX(), enemy.getY(), 30);

circle.setFill(enemyAttackRadiusPaint);

return circle;

}

public static Node drawMainAttackRadius(MainCharacter mainCharacter) {

Group group = new Group();

Circle circle = new Circle(mainCharacter.getX(), mainCharacter.getY(), 40);

circle.setFill(mainAttackRadiusPaint);

Circle point = new Circle(mainCharacter.getX(), mainCharacter.getY(), 10);

if (Main.attack) {

point.setFill(mainAttackRadiusPaint);

} else {

point.setFill(enemyAttackRadiusPaint);

}

if (drawCrit(mainCharacter) != null) {

group.getChildren().add(drawCrit(mainCharacter));

}

group.getChildren().add(circle);

group.getChildren().add(point);

return group;

}

public static Node drawCrit(MainCharacter mainCharacter) {

if (mainCharacter.getCrit()) {

Text text = new Text("Crit!");

text.setLayoutX(mainCharacter.getX() - 10);

text.setLayoutY(mainCharacter.getY() - 40);

return text;

}

return null;

}

public static void restartBattle(MainCharacter mainCharacter) {

mainCharacter.restart();

while (!enemyList.isEmpty()) {

destroyEnemy(enemyList.get(0), mainCharacter);

}

}

public static String convert(double count) {

int k = 0;

int countW = (int) count;

while (countW > 1000) {

k++;

countW = countW / 1000;

}

switch (k) {

case 0:

if(count%1==0){

return String.valueOf(countW);

}

return String.valueOf(count);

case 1:

return countW + "k";

case 2:

return countW + "m";

default:

return null;

}

}

}

## Класс CheatsController

package initialization;

import characters.MainCharacter;

import draw.Drawer;

import javafx.beans.value.ObservableValue;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.geometry.Rectangle2D;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.scene.control.TextFormatter;

import javafx.scene.input.MouseEvent;

import javafx.stage.Screen;

import javafx.stage.Stage;

import main.Main;

import javax.swing.event.ChangeEvent;

import javax.swing.event.ChangeListener;

import java.io.File;

import java.io.IOException;

import java.net.MalformedURLException;

import java.net.URL;

public class CheatsController {

@FXML

public Label currentLevelLabel;

@FXML

public Label currentSPointsLabel;

@FXML

public Label currentGoldLabel;

@FXML

public Button levelAddButton;

@FXML

public Button sPointsAddButton;

@FXML

public Button goldAddButton;

@FXML

public Button count1Button;

@FXML

public Button count1kButton;

@FXML

public Button count1mButton;

@FXML

public TextField countUserField;

@FXML

public Button applyButton;

@FXML

public Button cancelButton;

private static MainCharacter mainCharacter = null;

private static Stage stage = new Stage();

public static void startCheats(MainController mainController) {

try {

URL url = new File("src/fxml/Cheats.fxml").toURI().toURL();

FXMLLoader loader = new FXMLLoader(url);

mainCharacter = new MainCharacter(mainController.getMainCharacter());

CheatsController cheatsController = new CheatsController();

loader.setController(cheatsController);

Scene scene = new Scene(loader.load());

stage.setTitle("Cheats");

stage.setResizable(false);

stage.setScene(scene);

stage.show();

Rectangle2D screenBounds = Screen.getPrimary().getVisualBounds();

stage.setX((screenBounds.getWidth() - stage.getWidth()) / 2);

stage.setY((screenBounds.getHeight() - stage.getHeight()) / 2);

} catch (MalformedURLException e) {

throw new RuntimeException(e);

} catch (IOException e) {

throw new RuntimeException(e);

}

}

public void initialize() {

redrawCurrent();

countUserField.setText("1");

countUserField.textProperty().addListener(e -> {

if (!countUserField.getText().matches("\\d\*")) {

countUserField.setText(countUserField.getText().replaceAll("[^\\d]", ""));

}

});

applyButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> apply());

cancelButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> cancel());

count1Button.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> count1());

count1kButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> count1k());

count1mButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> count1m());

levelAddButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> addLevel());

sPointsAddButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> addSPoints());

goldAddButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> addGold());

}

private void redrawCurrent() {

currentGoldLabel.setText(Drawer.convert(mainCharacter.getGold()));

currentLevelLabel.setText(Drawer.convert(mainCharacter.getLevel()));

currentSPointsLabel.setText(Drawer.convert(mainCharacter.getSPoints()));

}

private void apply() {

Main.mainController.setMainCharacter(mainCharacter);

stage.close();

}

private void cancel() {

stage.close();

}

private void count1() {

countUserField.setText("1");

}

private void count1k() {

countUserField.setText("1000");

}

private void count1m() {

countUserField.setText("1000000");

}

private void addLevel() {

for (int i = 0; i < Integer.parseInt(countUserField.getText()); i++) {

mainCharacter.addExperience(mainCharacter.getLevelCost());

}

redrawCurrent();

}

private void addSPoints() {

mainCharacter.addSPoints(Integer.parseInt(countUserField.getText()));

redrawCurrent();

}

private void addGold() {

mainCharacter.addGold(Integer.parseInt(countUserField.getText()));

redrawCurrent();

}

}

## Класс MainController

package initialization;

import characters.Enemy;

import characters.MainCharacter;

import draw.Drawer;

import javafx.fxml.FXML;

import javafx.scene.Group;

import javafx.scene.control.\*;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.input.MouseEvent;

import javafx.stage.FileChooser;

import main.Main;

import java.io.\*;

import java.util.ConcurrentModificationException;

import java.util.Random;

public class MainController {

//Level Ups

@FXML

public Label levelCurrent;

@FXML

public Label statPointsCurrent;

@FXML

public Label upgradeStrengthLevel;

@FXML

public Label upgradeVitalityLevel;

@FXML

public Label upgradeAgilityLevel;

@FXML

public Label upgradeIntelligentLevel;

@FXML

public Label upgradeLuckLevel;

@FXML

public Button upgradeStrength;

@FXML

public Button upgradeVitality;

@FXML

public Button upgradeAgility;

@FXML

public Button upgradeIntelligent;

@FXML

public Button upgradeLuck;

@FXML

public Label upgradeStrengthCost;

@FXML

public Label upgradeVitalityCost;

@FXML

public Label upgradeAgilityCost;

@FXML

public Label upgradeIntelligentCost;

@FXML

public Label upgradeLuckCost;

//End Level Ups

//Stats

@FXML

public Label APCurrent;

@FXML

public Label CCCurrent;

@FXML

public Label SCurrent;

@FXML

public Label HCurrent;

@FXML

public Label BCCurrent;

@FXML

public Label SPCurrent;

@FXML

public Label BPCurrent;

@FXML

public Label BECurrent;

@FXML

public Label ASCurrent;

@FXML

public Label CPCurrent;

@FXML

public Label ACurrent;

@FXML

public Label HRCurrent;

@FXML

public Label DCCurrent;

@FXML

public Label SSCurrent;

@FXML

public Label BGCurrent;

//End Stats

//Upgrades

@FXML

public Button upgradeAP;

@FXML

public Label upgradeAPLevel;

@FXML

public Label upgradeAPCurrent;

@FXML

public Label upgradeAPCost;

@FXML

public Button upgradeAS;

@FXML

public Label upgradeASLevel;

@FXML

public Label upgradeASCurrent;

@FXML

public Label upgradeASCost;

@FXML

public Button upgradeCC;

@FXML

public Label upgradeCCLevel;

@FXML

public Label upgradeCCCurrent;

@FXML

public Label upgradeCCCost;

@FXML

public Button upgradeCP;

@FXML

public Label upgradeCPLevel;

@FXML

public Label upgradeCPCurrent;

@FXML

public Label upgradeCPCost;

@FXML

public Button upgradeS;

@FXML

public Label upgradeSLevel;

@FXML

public Label upgradeSCurrent;

@FXML

public Label upgradeSCost;

@FXML

public Button upgradeA;

@FXML

public Label upgradeALevel;

@FXML

public Label upgradeACurrent;

@FXML

public Label upgradeACost;

@FXML

public Button upgradeH;

@FXML

public Label upgradeHLevel;

@FXML

public Label upgradeHCurrent;

@FXML

public Label upgradeHCost;

@FXML

public Button upgradeHR;

@FXML

public Label upgradeHRLevel;

@FXML

public Label upgradeHRCurrent;

@FXML

public Label upgradeHRCost;

@FXML

public Button upgradeBC;

@FXML

public Label upgradeBCLevel;

@FXML

public Label upgradeBCCurrent;

@FXML

public Label upgradeBCCost;

@FXML

public Button upgradeDC;

@FXML

public Label upgradeDCLevel;

@FXML

public Label upgradeDCCurrent;

@FXML

public Label upgradeDCCost;

@FXML

public Button upgradeSS;

@FXML

public Label upgradeSSLevel;

@FXML

public Label upgradeSSCurrent;

@FXML

public Label upgradeSSCost;

@FXML

public Button upgradeSP;

@FXML

public Label upgradeSPLevel;

@FXML

public Label upgradeSPCurrent;

@FXML

public Label upgradeSPCost;

//End Upgrades

@FXML

public Group battleFieldGroup = new Group();

@FXML

public ProgressBar expBar;

@FXML

public ProgressBar healthBar;

@FXML

public Label goldLabel = new Label();

@FXML

public Spinner<Integer> enemyLevelSpinner = new Spinner<>();

@FXML

public Spinner<Integer> spawnTimeSpinner = new Spinner<>();

@FXML

public Spinner<Integer> spawnCountSpinner = new Spinner<>();

@FXML

public Label enemyKilledLabel;

@FXML

public Button saveButton;

@FXML

public Button loadButton;

@FXML

public Button cheatsButton;

@FXML

public Button exitButton;

@FXML

public Button exitBarButton;

@FXML

public Button resetButton;

@FXML

public Slider gameSpeedSlider;

private MainCharacter mainCharacter;

public MainController() {

}

public void setMainCharacter(MainCharacter mainCharacter) {

this.mainCharacter = mainCharacter;

}

public MainCharacter getMainCharacter() {

return this.mainCharacter;

}

@FXML

public void redrawBattlefield(Group enemyGroup) {

battleFieldGroup.getChildren().clear();

battleFieldGroup.getChildren().add(Drawer.drawMainCharacter(mainCharacter));

battleFieldGroup.getChildren().add(Drawer.drawMainAttackRadius(mainCharacter));

battleFieldGroup.getChildren().add(enemyGroup);

enemyKilledLabel.setText(Drawer.convert(mainCharacter.getEnemyKilled()));

goldLabel.setText(Drawer.convert(mainCharacter.getGold()));

}

private void redrawSpawn() {

spawnTimeSpinner.getValueFactory().setValue(mainCharacter.getSpawnTime());

enemyLevelSpinner.getValueFactory().setValue(mainCharacter.getEnemyLevel());

spawnCountSpinner.getValueFactory().setValue(mainCharacter.getSpawnCount());

}

public void initialize() {

upgradeStrength.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeStrengthClicked());

upgradeVitality.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeVitalityClicked());

upgradeAgility.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeAgilityClicked());

upgradeIntelligent.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeIntelligentClicked());

upgradeLuck.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeLuckClicked());

upgradeAP.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeAPClicked());

upgradeAS.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeASClicked());

upgradeCC.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeCCClicked());

upgradeCP.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeCPClicked());

upgradeS.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeSClicked());

upgradeA.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeAClicked());

upgradeH.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeHClicked());

upgradeHR.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeHRClicked());

upgradeBC.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeBCClicked());

upgradeDC.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeDCClicked());

upgradeSP.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeSPClicked());

upgradeSS.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> upgradeSSClicked());

spawnTimeSpinner.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> spawnTimeSpinnerClicked());

spawnCountSpinner.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> spawnCountSpinnerClicked());

enemyLevelSpinner.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> enemyLevelSpinnerClicked());

redrawSpawn();

redrawLevel();

redrawStats();

redrawUpgrades();

redrawBattlefield(Drawer.drawBattleField());

saveButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> saveDialog());

loadButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> loadDialog());

exitButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> Main.exit());

exitBarButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> Main.exit());

resetButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> Drawer.restartBattle(mainCharacter));

cheatsButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> CheatsController.startCheats(this));

gameSpeedSlider.addEventHandler(MouseEvent.MOUSE\_RELEASED, mouseEvent -> Drawer.restartBattle(mainCharacter));

}

public double getSpeed() {

if (this.gameSpeedSlider != null) {

return this.gameSpeedSlider.getValue();

} else {

return 1;

}

}

private void enemyLevelSpinnerClicked() {

mainCharacter.setEnemyLevel(enemyLevelSpinner.getValue());

}

private void spawnCountSpinnerClicked() {

mainCharacter.setSpawnCount(spawnCountSpinner.getValue());

}

private void spawnTimeSpinnerClicked() {

mainCharacter.setSpawnTime(spawnTimeSpinner.getValue());

}

private void loadDialog() {

mainCharacter = loadMaincharacter();

redrawLevel();

redrawUpgrades();

redrawStats();

redrawSpawn();

}

public MainCharacter loadMaincharacter() {

FileChooser fc = new FileChooser();

fc.setTitle("Load");

fc.setInitialDirectory(new File("C:\\Users\\Торн\\OneDrive\\Курсач\\5 семестр (Графика)\\Game"));

fc.setSelectedExtensionFilter(new FileChooser.ExtensionFilter("TXT files (\*.txt)", "\*.txt"));

fc.setInitialFileName("save.txt");

File file = fc.showOpenDialog(Main.stage);

if (file != null) {

try {

FileReader fr = new FileReader(file);

char[] buffer = new char[256];

StringBuilder string = new StringBuilder();

while (fr.ready()) {

fr.read(buffer);

}

for (char c : buffer) {

if (c != '\u0000') {

string.append(c);

}

}

fr.close();

return new MainCharacter(string.toString());

} catch (FileNotFoundException ex) {

System.out.println("Ошибка чтения из файла!");

} catch (IOException e) {

System.out.println("Ошибка потоков чтения!");

}

}

System.out.println("Ошибка считывания персонажа, создан новый!");

return new MainCharacter();

}

private void saveDialog() {

FileChooser fc = new FileChooser();

fc.setTitle("Save");

fc.setInitialDirectory(new File("C:\\Users\\Торн\\OneDrive\\Курсач\\5 семестр (Графика)\\Game"));

fc.setSelectedExtensionFilter(new FileChooser.ExtensionFilter("TXT files (\*.txt)", "\*.txt"));

fc.setInitialFileName("save.txt");

File file = fc.showSaveDialog(Main.stage);

if (file != null) {

try {

FileWriter fw = new FileWriter(file);

fw.write(mainCharacter.toString());

fw.close();

} catch (IOException e) {

System.out.println("Ошибка записи в файл!");

}

}

}

@FXML

public void addEnemy() {

for (int i = 0; i < mainCharacter.getSpawnCount(); i++) {

Random r = new Random();

Drawer.addEnemy(r.nextInt(738), r.nextInt(2) \* 718, mainCharacter);

}

redrawSpawn();

redrawBattlefield(Drawer.drawBattleField());

}

@FXML

public void move() {

try {

for (Enemy en : Drawer.getEnemyList()) {

en.moveEnemy(mainCharacter);

}

} catch (ConcurrentModificationException ex) {

System.out.println("Ошибка потоков!");

}

mainCharacter.moveMain(Drawer.getEnemyList());

redrawBattlefield(Drawer.drawBattleField());

redrawLevel();

}

@FXML

public void upgradeStrengthClicked() {

if (mainCharacter.incStrength()) {

redrawLevel();

redrawStats();

}

}

@FXML

public void upgradeVitalityClicked() {

if (mainCharacter.incVitality()) {

redrawLevel();

redrawStats();

}

}

@FXML

public void upgradeAgilityClicked() {

if (mainCharacter.incAgility()) {

redrawLevel();

redrawStats();

Main.startAttackTimeline(mainCharacter);

}

}

@FXML

public void upgradeIntelligentClicked() {

if (mainCharacter.incIntelligent()) {

redrawLevel();

redrawStats();

}

}

@FXML

public void upgradeLuckClicked() {

if (mainCharacter.incLuck()) {

redrawLevel();

redrawStats();

}

}

@FXML

public void upgradeAPClicked() {

if (mainCharacter.incUpgrade(0, 0)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeASClicked() {

if (mainCharacter.incUpgrade(1, 0)) {

redrawStats();

redrawUpgrades();

Main.startAttackTimeline(mainCharacter);

}

}

@FXML

public void upgradeCCClicked() {

if (mainCharacter.incUpgrade(0, 1)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeCPClicked() {

if (mainCharacter.incUpgrade(1, 1)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeSClicked() {

if (mainCharacter.incUpgrade(0, 2)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeAClicked() {

if (mainCharacter.incUpgrade(1, 2)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeHClicked() {

if (mainCharacter.incUpgrade(0, 3)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeHRClicked() {

if (mainCharacter.incUpgrade(1, 3)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeBCClicked() {

if (mainCharacter.incUpgrade(0, 4)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeDCClicked() {

if (mainCharacter.incUpgrade(1, 4)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeSPClicked() {

if (mainCharacter.incUpgrade(0, 5)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void upgradeSSClicked() {

if (mainCharacter.incUpgrade(1, 5)) {

redrawStats();

redrawUpgrades();

}

}

@FXML

public void updateBars() {

healthBar.setProgress((double) mainCharacter.getHp() / mainCharacter.getHealth());

expBar.setProgress((double) mainCharacter.getExperience() / mainCharacter.getLevelCost());

}

private void redrawLevel() {

levelCurrent.setText("Level " + this.mainCharacter.getLevel());

statPointsCurrent.setText(Drawer.convert(this.mainCharacter.getSPoints()));

upgradeStrengthLevel.setText(Drawer.convert(mainCharacter.getStrength()));

upgradeStrengthCost.setText(Drawer.convert(mainCharacter.getStrengthCost()));

upgradeVitalityLevel.setText(Drawer.convert(mainCharacter.getVitality()));

upgradeVitalityCost.setText(Drawer.convert(mainCharacter.getVitalityCost()));

upgradeAgilityLevel.setText(Drawer.convert(mainCharacter.getAgility()));

upgradeAgilityCost.setText(Drawer.convert(mainCharacter.getAgilityCost()));

upgradeIntelligentLevel.setText(Drawer.convert(mainCharacter.getIntelligent()));

upgradeIntelligentCost.setText(Drawer.convert(mainCharacter.getIntelligentCost()));

upgradeLuckLevel.setText(Drawer.convert(mainCharacter.getLuck()));

upgradeLuckCost.setText(Drawer.convert(mainCharacter.getLuckCost()));

}

private void redrawStats() {

upgradeAPCurrent.setText(Drawer.convert(mainCharacter.getAttackPower()));

APCurrent.setText(Drawer.convert(mainCharacter.getAttackPower()));

upgradeASCurrent.setText(Drawer.convert(mainCharacter.getAttackSpeed()) + "%");

ASCurrent.setText(Drawer.convert(mainCharacter.getAttackSpeed()) + "%");

upgradeCCCurrent.setText(Drawer.convert(mainCharacter.getCritChance()) + "%");

CCCurrent.setText(Drawer.convert(mainCharacter.getCritChance()) + "%");

upgradeCPCurrent.setText(Drawer.convert(mainCharacter.getCritPower()) + "%");

CPCurrent.setText(Drawer.convert(mainCharacter.getCritPower()) + "%");

upgradeSCurrent.setText(Drawer.convert(mainCharacter.getSpeed()) + "%");

SCurrent.setText(Drawer.convert(mainCharacter.getSpeed()) + "%");

upgradeACurrent.setText(Drawer.convert(mainCharacter.getArmor()));

ACurrent.setText(Drawer.convert(mainCharacter.getArmor()));

upgradeHCurrent.setText(Drawer.convert(mainCharacter.getHealth()));

HCurrent.setText(Drawer.convert(mainCharacter.getHealth()));

upgradeHRCurrent.setText(Drawer.convert(mainCharacter.getHealthRegeneration()) + "%");

HRCurrent.setText(Drawer.convert(mainCharacter.getHealthRegeneration()) + "%");

upgradeBCCurrent.setText(Drawer.convert(mainCharacter.getBlockChance()) + "%");

BCCurrent.setText(Drawer.convert(mainCharacter.getBlockChance()) + "%");

upgradeDCCurrent.setText(Drawer.convert(mainCharacter.getDodgeChance()) + "%");

DCCurrent.setText(Drawer.convert(mainCharacter.getDodgeChance()) + "%");

upgradeSPCurrent.setText(Drawer.convert(mainCharacter.getSkillPower()));

SPCurrent.setText(Drawer.convert(mainCharacter.getSkillPower()));

upgradeSSCurrent.setText(Drawer.convert(mainCharacter.getSkillSpeed()) + "%");

SSCurrent.setText(Drawer.convert(mainCharacter.getSkillSpeed()) + "%");

BGCurrent.setText(Drawer.convert(mainCharacter.getBonusGold()));

BECurrent.setText(Drawer.convert(mainCharacter.getBonusExp()));

BPCurrent.setText(Drawer.convert(mainCharacter.getBlockPower()));

}

private void redrawUpgrades() {

upgradeAPLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(0, 0)));

upgradeAPCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(0, 0)));

upgradeASLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(1, 0)));

upgradeASCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(1, 0)));

upgradeCCLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(0, 1)));

upgradeCCCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(0, 1)));

upgradeCPLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(1, 1)));

upgradeCPCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(1, 1)));

upgradeSLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(0, 2)));

upgradeSCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(0, 2)));

upgradeALevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(1, 2)));

upgradeACost.setText(Drawer.convert(mainCharacter.getUpgradeCost(1, 2)));

upgradeHLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(0, 3)));

upgradeHCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(0, 3)));

upgradeHRLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(1, 3)));

upgradeHRCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(1, 3)));

upgradeBCLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(0, 4)));

upgradeBCCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(0, 4)));

upgradeDCLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(1, 4)));

upgradeDCCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(1, 4)));

upgradeSPLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(0, 5)));

upgradeSPCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(0, 5)));

upgradeSSLevel.setText(Drawer.convert(mainCharacter.getUpgradeLevel(1, 5)));

upgradeSSCost.setText(Drawer.convert(mainCharacter.getUpgradeCost(1, 5)));

}

public void updateMain() {

mainCharacter.healthRegen();

}

}

## Класс MenuController

package initialization;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.input.MouseEvent;

import java.awt.\*;

import static main.Main.exit;

import static main.Main.startGame;

public class MenuController {

@FXML

public Button newGameButton;

@FXML

public Button loadGameButton;

@FXML

public Button exitButton;

public void initialize() {

newGameButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> startGame("new"));

loadGameButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> startGame("load"));

exitButton.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEvent -> exit());

}

}