МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ "РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени В.Ф. УТКИНА"

Кафедра САПР

К защите
Руководитель работы:

_____
дата, подпись

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
по дисциплине
«Компьютерная графика»
Тема:
«Разработка графического приложения с помощью фреймворка LibGDX»

Выполнил студент группы 246
Шапоренко А. С.

_____
дата сдачи на проверку, подпись

Руководитель работы
доц каф. САПР
Митрошин А. А.

_____ _____
оценка дата защиты, подпись

Рязань 2024

Оглавление

1. Введение

В данной курсовой работе спроектирована и реализована пиксельная 2D игра «GoldMines» с использованием возможностей фреймворка LibGDX.

LibGDX - это мощный кроссплатформенный фреймворк для разработки игр на языке Java. Он предоставляет разработчикам широкий набор функциональных возможностей, которые позволяют создавать игры для различных платформ, включая Windows, macOS, Linux, Android и iOS. Вот основные функциональные возможности LibGDX:

- Кроссплатформенность: разработка игр с использованием одного кода, который может запускаться на различных устройствах и операционных системах.
- Возможность реализации 2D и 3D графики: поддержка 2D и 3D графики, с использованием спрайтов, анимаций, текстур и трёхмерных моделей.
- Физический движок: интеграция с Box2D для создания реалистичной физики в 2D играх, включая столкновения и динамическое взаимодействие объектов.
- Аудио: поддержка различных аудиоформатов для воспроизведения звуковых эффектов и музыки, включая 3D-аудио.
- Сцены и UI (User Interface): система управления сценами и встроенные инструменты для создания пользовательского интерфейса, включая кнопки, текстовые поля и другие элементы.
- Сетевое взаимодействие: возможности для создания многопользовательских игр с использованием TCP и UDP сетевых протоколов.
- Хранение данных: поддержка сохранения данных на устройстве пользователя, включая использование файловой системы и баз данных.
- Инструменты разработки: набор инструментов для отладки, профилирования и оптимизации производительности игры.

- Расширяемость: Легкость в интеграции сторонних библиотек и модулей для расширения функциональности.
- Сообщество и документация: активное сообщество разработчиков и обширная документация, что облегчает процесс обучения и решения проблем.

2. Постановка задачи

Необходимо спроектировать и разработать пиксельную 2D игру в жанре аркада с элементами рогалика. Для реализации использовать возможности фреймворка LibGDX. В игре необходимо реализовать процедурную генерацию игровых уровней, главного героя, противников, объекты окружения, объекты взаимодействия. У главного героя должны быть здоровье, выносливость и инвентарь. Противники преследуют главного героя, если он располагается в их поле видимости, иначе передвигаются по хаотичному маршруту по игровому полю, в случае приближения к главному герою наносят ему урон и могут убить. Целью игры заключается сбор игровых монет, после сбора всех монет на уровне открывается дверь, которая позволяется перейти на другой уровень. Некие драгоценные камни позволяют открывать сундуки с бонусами, улучшающими главного героя.

3. Подключение фреймворка к проекту. Подтверждение работы фреймворка

3.1. Подготовка

Для начала работы с фрейморком LibGDX необходимо скачать libGDX Project Setup Tool (gdx-liftoff). Для этого переходим на официальный сайт фреймворка LibGDX (https://libgdx.com/) представлен на рисунке 1.

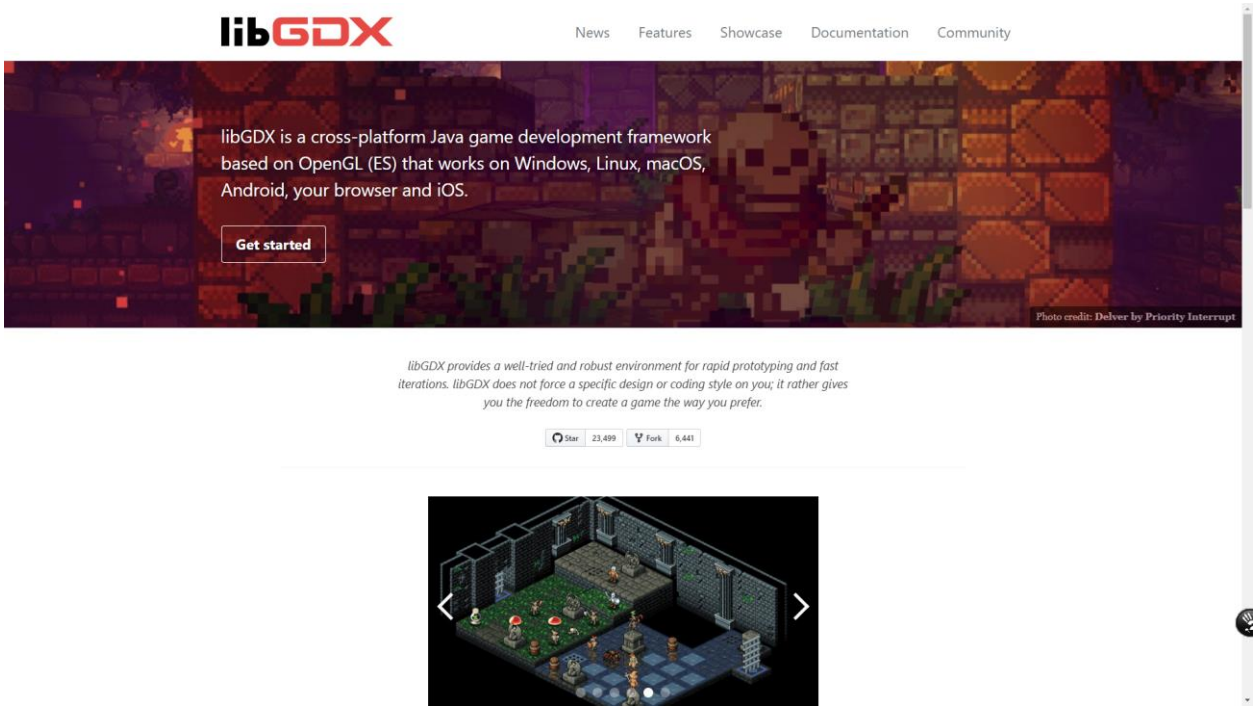

Рисунок 1 – Главная страница официального сайта LibGDX

После нажимаем кнопку “Get started” и нас переносит на другую страницу, на которой необходимо найти ссылку “setup tool”, которая в свою очередь перенаправит на сайт (рисунок 2), где располагается кнопка для скачивания программы, это кнопка “ Download gdx-liftoff”.

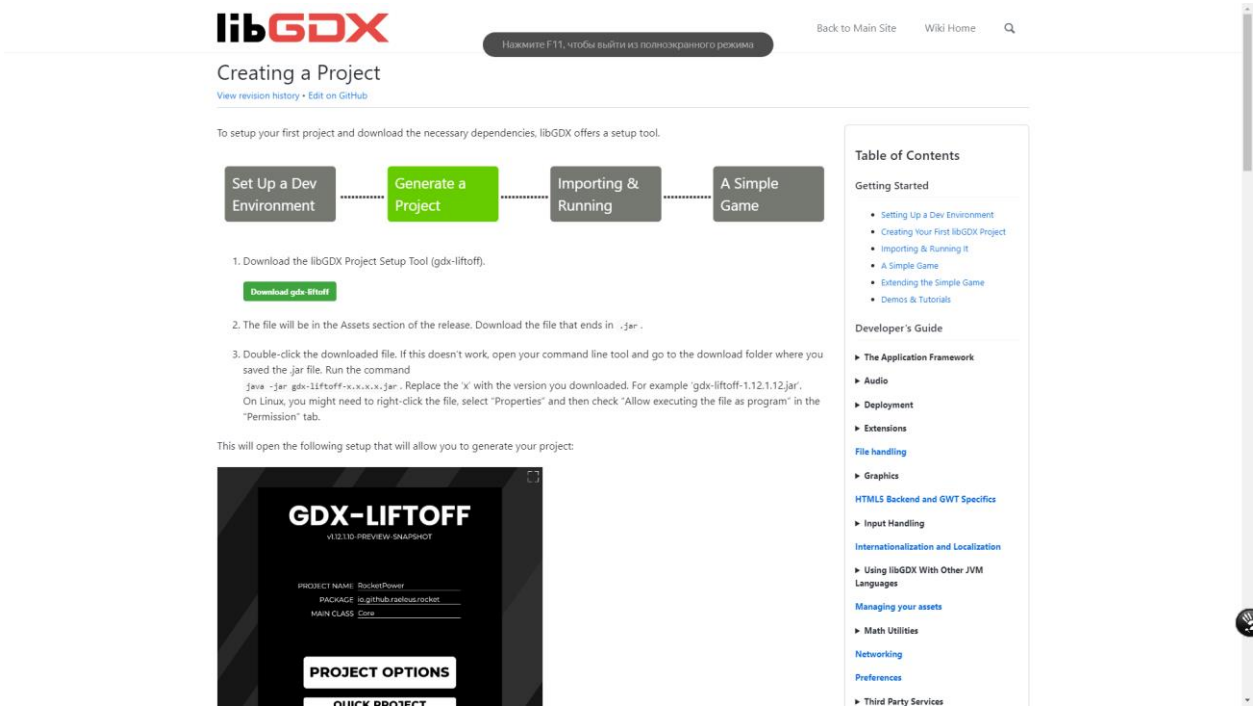

Рисунок 2 – Страница с кнопкой для скачивания

Открывается сайт GitHub, где располагается репозиторий проекта LibGDX, и предоставляется выбор для какой операционной системы необходимо скачать, в нашем случае это Windows, поэтому можно выбрать gdx-liftoff-winX64.zip, но мы выберем универсальный файл с расширением jar - gdx-liftoff-1.13.0.2.jar (рисунок 3).

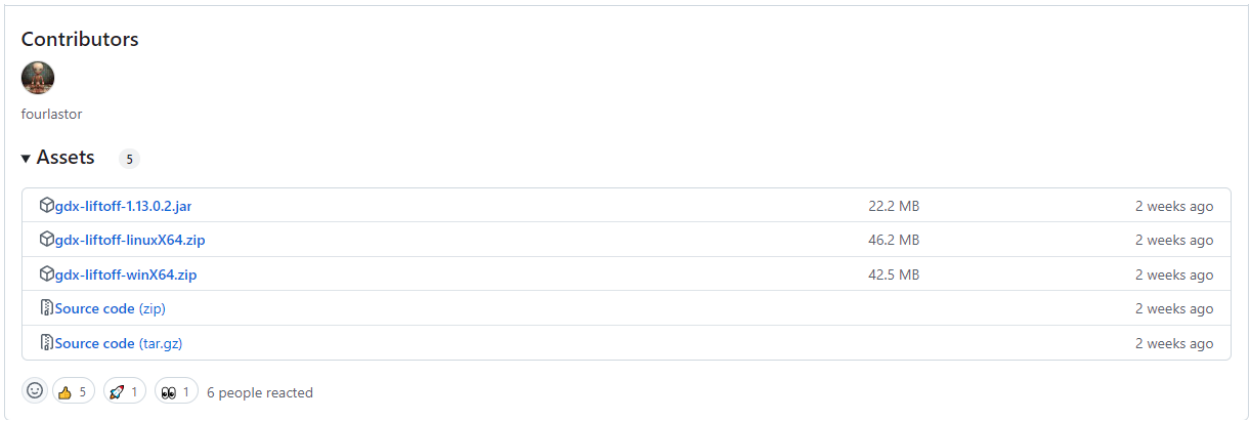

Рисунок 3 – Репозиторий фреймворка с файлами для скачивания

3.2. Создание проекта

Далее необходимо открыть файл gdx-liftoff-1.13.0.2.jar (например, с помощью Java™ Platform SE binary или OpenJDK Platform binary). В результате будет запущено приложение GDX-LIFTOFF (рисунок 4).

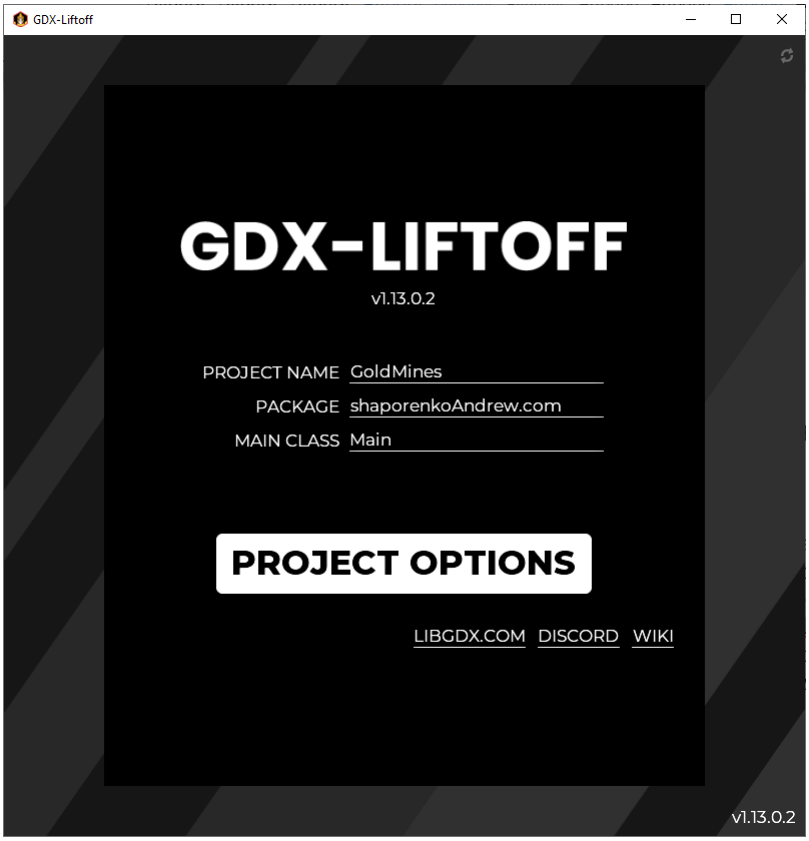

Рисунок 4 – Основное окно приложения GDX-LIFTOFF

Раскрываем на полный экран и задаём название нашего проекта, название его папки, главный класс (не соответствует классу, который запускает проект), оставляем параметры по умолчанию для ADD-ONS, во вкладке THIRD-PARTY выбираем libGDX-Oboe, libGDX Utils, libGDX Utils Box2D, PieMenu и ScreenManager. Также можно выбрать версию проекта и путь, где будет находится папка проекта.

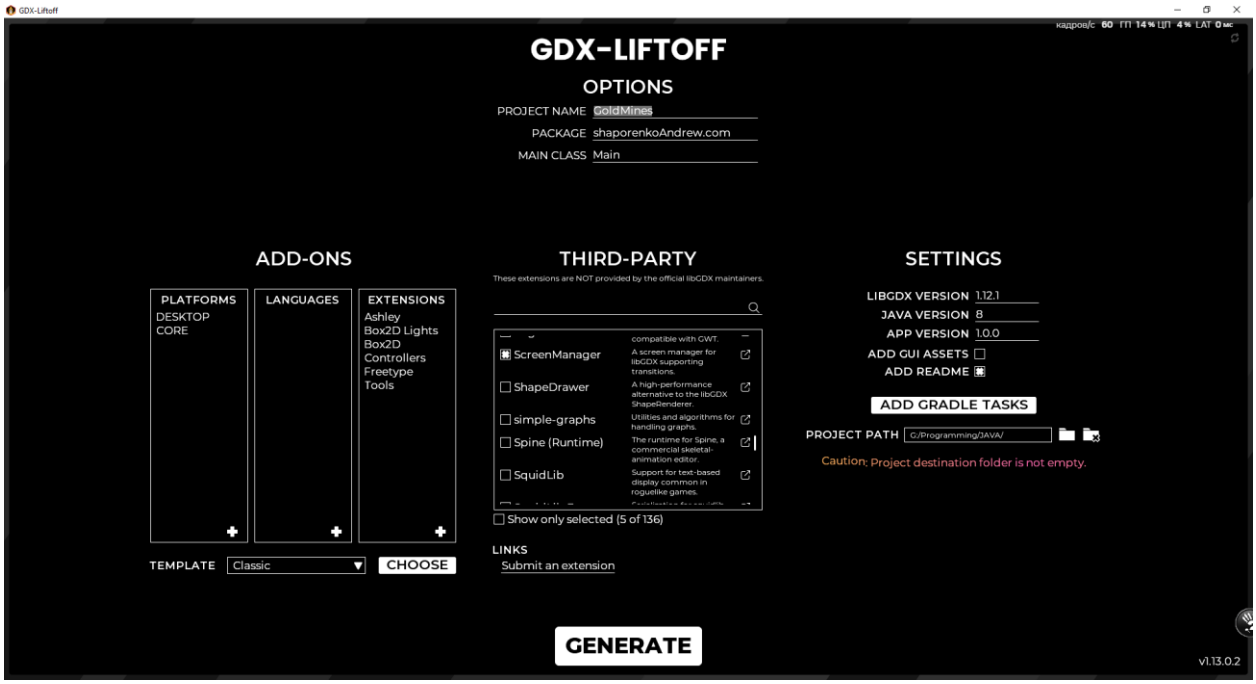

Рисунок 5 – Окно программы GDX-LIFTOFF со всеми настраиваемыми параметрами

После всех настроек нажимаем кнопку “GENERATE” и после появления окна показанного на рисунке 6 проект создан и можно продолжать дальнейшую работу.

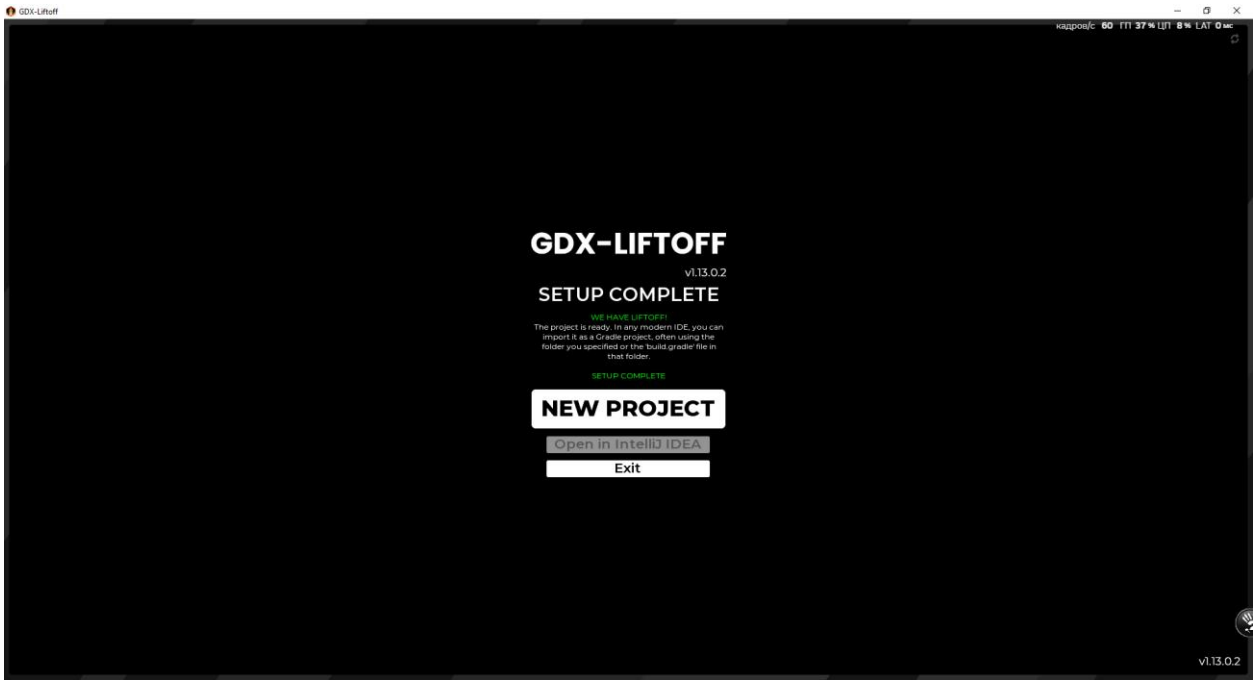

Рисунок 6 – Успешное создание проекта с фреймворком LibGDX

3.3.Открытие проекта через IntelliJ IDEA и подтверждение работы фреймворка

Запускаем IntelliJ IDEA и открываем проект, который только что создали, после чего находим в папке lwjgl3/src/main/java/shaporenko.com.lwjgl3/ класс с названием Lwjgl3Launcher (рисунок 7). Запускаем данный файл, если появляется окно с таким же изображением что и на рисунке 8, то это означает, что проект был создан правильно и фреймворк подключен к проекту.

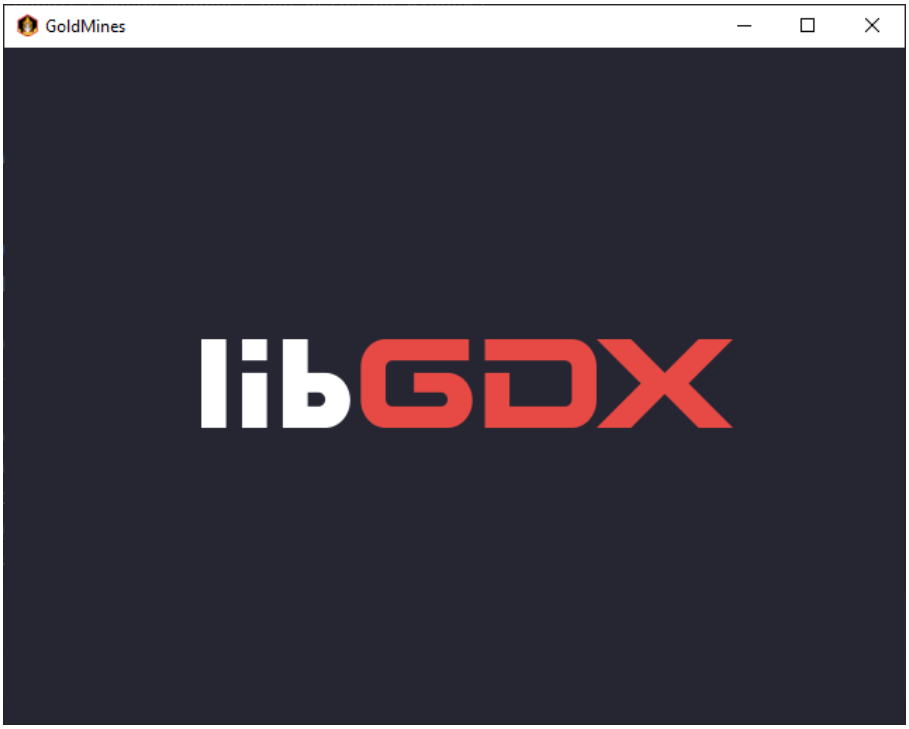

Рисунок 8 – Начальное окно запущенного проекта с фреймворком LibGDX, подтверждающее правильную работу проекта

4. Разработка программного обеспечения

Проект разработанной игры включает в себя 16 классов и имеет следующую структуру (рисунок 9). Также отдельно есть два класса, которые запускают проект с использованием LibGDX, это классы Lwjgl3Launcher и StartupHelper.

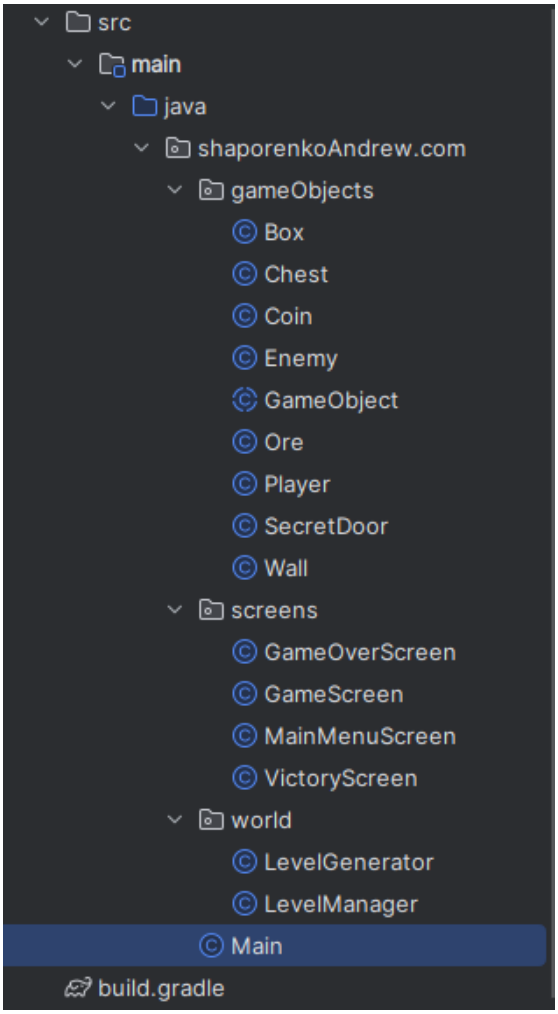

Рисунок 9 – Иерархия проекта после создания игры

Кратко опишем каждый класс проекта:

Класс **Main** (наследуется от класса Game) является основным классом, который инициализирует основные параметры приложения. Он отвечает за размеры окна приложения, установку заголовка окна, а также инициализацию главного меню игры.

Класс **MainMenuScreen** реализует интерфейс Screen из фреймворка libGDX. Он отвечает за отображение главного меню игры. В нём находится функционал, который позволяет отображать кнопки для начала игры и выхода

из него, управлять переходами между экранами и отображать фон и название игры в главном меню.

Класс **GameScreen** также реализует интерфейс Screen. Главный класс самой игры, он отрисовывает игровой мир и все его объекты, обрабатывает логику и физику объектов, управляет состоянием игры, отображается пользовательский интерфейс (UI) и обрабатывает пользовательский ввод с клавиатуры и мыши.

Класс **VictoryScreen** реализует интерфейс Screen. Является классом, который отвечает за отображение экрана в случае победы. В нём прописан функционал, с помощью которого отображается экран с поздравление игрока с победой в игре, а также отображением кнопок для новой игры или выхода в меню.

Класс **GameOverScreen** реализует интерфейс Screen. Класс, который отвечает за отображение экрана поражения в игре. Его функционал реализует отображение сообщения о проигрыше и кнопки для перезапуска игры или выхода в главное меню.

Класс **LevelGenerator** отвечает за процедурную генерацию игрового мира на уровнях. Он создаёт случайные лабиринты на уровнях, которые представлены комнатами и коридорами между ними, также создаёт и размещает игровые объекты (монеты, сундуки, ящики) и противников.

Класс **LevelManager** отвечает за управление уровнями игры. Его функционал реализует хранение информации о текущем уровне, подсчёт очков и прогресса игрока и управляет переходом между уровнями.

Класс **GameObject** (абстрактный) представляет собой основные свойства и методы, которые являются общими для всех игровых объектов. В нём находятся позиция объекта, размеры и текстура объекта.

Класс **Box** (наследуется от GameObject) отвечает за игровой объект ящика. Сам объект представляет собой неподвижный ящик, с которым может взаимодействовать игрок. Используется как препятствие и элемент окружения.

Класс **Chest** (наследуется от GameObject) отвечает за игровой объект сундука. Сам объект представляет собой неподвижный сундук, с которым может взаимодействовать игрок. Функционалом сундука является хранение ценных предметов (бонусов), взаимодействие с игроком в момент открытия, анимация открытия, выдача бонуса игроку.

Класс **Coin** (наследуется от GameObject) отвечает за игровой объект монеты. Сам объект представляет собой неподвижную монету, которую игрок может поднять, что увеличивает прогресс игры, после подбора объект пропадает с игрового поля и добавляет очки игроку.

Класс **Enemy** (наследуется от GameObject) представляет противника в игре. Реализует искусственный интеллект противников, который позволяет преследовать главного героя и наносить ему урон. Также противник имеет систему патрулирования территории и обработку столкновения с игроком и отображение текстуры.

Класс **Ore** (наследуется от GameObject) отвечает за игровой объект ценного ресурса, который игрок может добывать. Позволяет игроку открывать сундуки и получать бонусы.

Класс **Player** (наследуется от GameObject) отвечает за главного героя игры. Функционалом являются управление движением главного героя, обработка коллизий (столкновений и взаимодействия с объектами), управление инвентарём и счётом игрока, текстура персонажа.

Класс **SecretDoor** (наследуется от GameObject) отвечает за игровой объект двери. Позволяет переходить с уровня на уровень, открывается при сборе всех монет на уровне, содержит анимацию открытия двери при выполнении условия.

Класс **Wall** (наследуется от GameObject) отвечает за стены игрового мира. Это базовый неподвижный объект, формирующий структуру уровня и блокирующий передвижение главного героя и противников, а также других объектов.

Опишем подробнее как работают некоторые классы:

В классе LevelManager устанавливаются параметры генерации уровней, такие как минимальное количество комнат для каждого уровня, минимальный и максимальный размеры комнат и т. д. Метод generateLevel возвращает объект типа LevelGenerator, который создаёт новый уровень. Метод hasNextLevel проверяет наличие следующего уровня игры, а метод nextLevel осуществляет переход к следующему уровню. Также есть методы getCurrentLevel и getTotalLevels, которые возвращают номер текущего уровня и общее количество уровней соответственно. В классе удобно настраивается количество уровней и параметры, задаваемые для уровня.

В классе LevelGenerator имеет список констант и параметров генерации игрового мира. Конструктор принимает параметры для генерации уровня, сама генерация уровня осуществляется в методе generateLevel, в котором сначала игровой мир (представленный матрицей) заполняется стенами, а далее используются методы generateRooms для создания комнат, при этом одна из них становится «секретной», то есть имеет дверь для перехода на следующий уровень, и предусмотрена проверка на пересечение комнат, connectRooms для создания коридоров между комнатами, populateRooms для размещения объектов в комнатах (сундуки, монеты, ценные ресурсы и дверь). Также имеются методы carveRoom, который вырезает стены в тех местах, в которых должны располагаться игровые комнаты, createGameObject, который преобразует условные игровые объекты в реальные, то есть задаёт им текстуры (представляют собой файлы в основном png формата), количество и анимацию. Метод createEnemies позволяет создавать противников на уровне, предусматривая то, что они не должны появляется близко к игроку.

В классе Enemy создающем противников находятся константы для поведения, системы патрулирования и предотвращения застревания. В конструкторе задаются начальные значения параметров и добавляется текстура. Метод update обновляет состояния противника, предоставляя информацию о том, в пределах ли видимости главный герой, определяет

движение и коллизии, а также возможность атаковать игрока. Методы updateChasing и updatePatrol отвечают за режимы преследования игрока и патрулирования территории.

В классе Player прописаны константы для настройки характеристик персонажа, для системы добычи ценных ресурсов и для системы коллизий и движения. Конструктор задаёт основные параметры и текстуру персонажа. Метод update реализуй движение главного героя, управление выносливостью и здоровьем, добычу ценных ресурсов и систему коллизий. Метод renderMiningProgress позволяет добывать ценные ресурсы и убирать их с игрового поля.

В классе SecretDoor представлен код, который реализуй дверь, позволяющую перемещаться на следующий игровой уровень, при условии, что все монеты данного уровня собраны игроком. Для этого реализованы методы checkAndOpen, interact, isOpen, render, dispose.

В классе Chest реализован код для игровых объектов сундуков, которые дают персонажу бонусы со случайным выпадением, открытие сундука происходит при наличии у игрока двух ценных ресурсов в инвентаре. Реализованы методы open для открытия сундука, isOpened и dispose.

5. Тестирование

Для корректной работы программы необходимо протестировать весь (почти весь, т. е. основной) функционал. Для начала тестирования необходимо описать управление и возможности игры для пользователя. Управление главным героем, то есть его перемещение осуществляется на клавиши WASD, подбор монет сделан автоматических, сбор ценного ресурса производится при наведении на него курсора мыши, а также близкого расположения главного героя к ценному ресурсу, переход на следующий уровень осуществляется с помощью открытой двери, для прохода в открытую дверь необходимо нажать на клавишу E. Также при наличии в инвентаре двух ценных ресурсов игрок может открыть сундук с помощью клавиши E. В игре предусмотрена пауза, осуществляемая клавишей Escape, для возобновления игрового процесса необходимо повторное нажатие. Инвентарь персонажа открывается/закрывается на клавишу Tab. Клавиша Shift позволяется бегать, при этом теряется выносливость, она может восстанавливаться если персонаж стоит или передвигается пешком. Клавиша Enter осуществляет перезапуск игры на определённых для этого экранах.

Были протестированы следующие игровые аспекты и механики:

– Отображение главного меню с кнопками start (запуск новой игры), control (меню настроек) и exit (выход из игры) (рисунок 10);

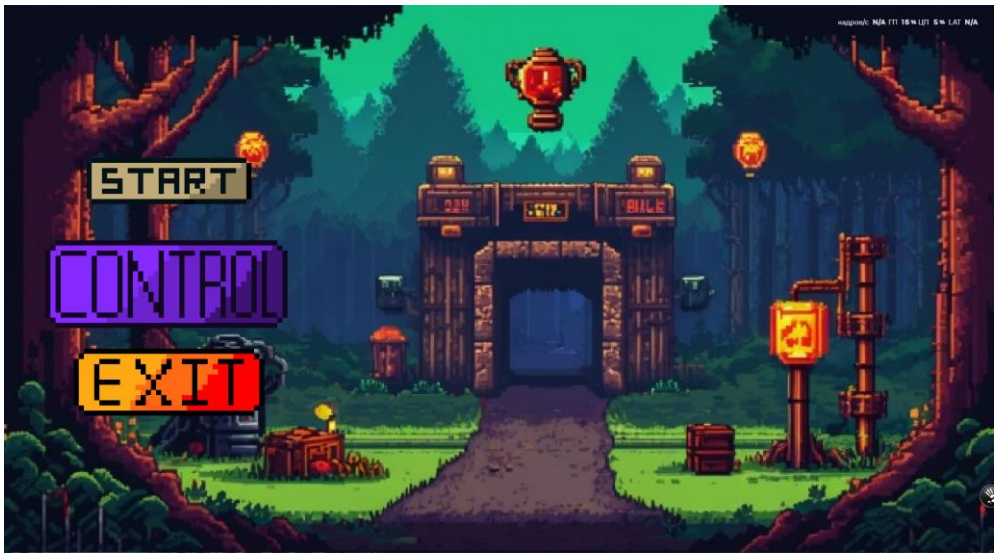

Рисунок 10 – Главное меню программы

– При нажатии на кнопку старт осуществляется генерация первого уровня и его отображение (рисунок 11);

Рисунок 11 – Первый уровень игры, после нажатия кнопки start в главном меню

– При наведении курсора мыши на ценный ресурс и при нахождении игрока на большом расстоянии не осуществляется его добыча;

– При наведении курсора мыши на ценный ресурс при близком к нему расположении игрока осуществляется добыча ценного ресурса (рисунок 12);

Рисунок 12 – Подтверждение добычи ценного ресурса при наведении на него курсора мыши

– При наличии двух (и более) ценных ресурсов в инвентаре протестировано открытие сундука (рисунок 13);

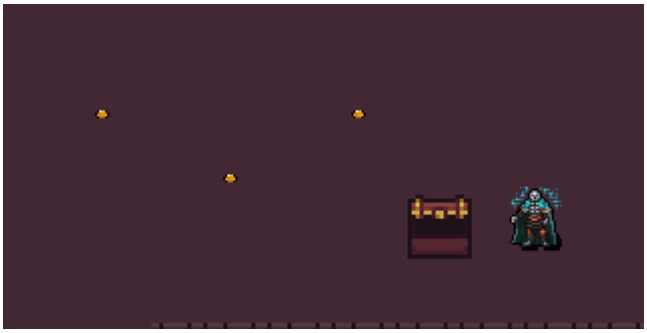

Рисунок 13 – Открытый сундук после добычи двух ценных ресурсов и нажатия клавиши E

– При сборе всех игровых монет на уровне открывается дверь (рисунок 14);

Рисунок 14 – Открытая дверь после сбора всех игровых монет на уровне

– Протестирован экран паузы;

– Протестированы экраны проигрыша и выигрыша;

– Протестированы взаимодействие и получение урона от противника;

– Протестирован переход на следующий уровень;

– Протестирован (с помощью нескольких запусков игры) процедурная генерация игрового мира.

6. Заключение

В результате выполнения курсовой работы были получены навыки и изучена работа с фреймворком LibGDX. Для этого была создана простая пиксельная 2D игра жанра аркада с элементами рогалика под названием “GoldMines”. Были реализован и протестирован весь необходимый функционал игры.

Также есть возможность улучшения и расширения возможностей игры, стиля и визуализации игрового мира, анимаций, а также сложности и других вещей.

7. Библиографический список

1. LibGDX Wiki [Электронный ресурс] – URL: https://libgdx.com/wiki/

2. LibGDX GitHub [Электронный ресурс] – URL: https://github.com/libgdx/libgdx

3. LibGDX Tutorial 5: Handing Input-Touch and gestures [Электронный ресурс] – URL: https://gamefromscratch.com/libgdx-tutorial-5-handling-input-touch-and-gestures/

4. StackOverflow LibGDX [Электронный ресурс] – URL: https://stackoverflow.com/questions/tagged/libgdx

5. James Cook LibGDX Game Development By Example - First published: August 2015

8. Приложение

Класс Main

```
package shaporenkoAndrew.com;
import com.badlogic.gdx.Game;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Graphics.DisplayMode;
import shaporenkoAndrew.com.screens.MainMenuScreen;
public class Main extends Game {
    private int screenWidth;
    private int screenHeight;
    @Override
    public void create() {
        DisplayMode displayMode = Gdx.graphics.getDisplayMode();
        screenWidth = displayMode.width;
        screenHeight = displayMode.height;
        Gdx.graphics.setWindowedMode(screenWidth, screenHeight);
        Gdx.graphics.setTitle("Gold Mines");
        this.setScreen(new MainMenuScreen(this));
    }
    @Override
    public void dispose() {
        super.dispose();
    }
}
```

Класс LavelManager

```
package shaporenkoAndrew.com.world;
public class LevelManager {
    private static final int TOTAL_LEVELS = 3;
    private int currentLevel;
    private static final int[] MIN_ROOMS = {8, 12, 15};
    private static final int[] MAX_ROOMS = {12, 16, 20};
    private static final int[] MIN_ROOM_SIZE = {8, 10, 12};
    private static final int[] MAX_ROOM_SIZE = {12, 16, 20};
    private static final int[] CORRIDOR_WIDTH = {2, 3, 4};
    private static final int[] MAX_COINS = {30, 40, 50};
    public LevelManager() {
        this.currentLevel = 0;
    }
    public LevelGenerator generateLevel(int width, int height) {
        return new LevelGenerator(
            width,
            height,
            MIN_ROOMS[currentLevel],
            MAX_ROOMS[currentLevel],
            MIN_ROOM_SIZE[currentLevel],
            MAX_ROOM_SIZE[currentLevel],
            CORRIDOR_WIDTH[currentLevel],
            MAX_COINS[currentLevel]
        );
    }
    public boolean hasNextLevel() {
        return currentLevel < TOTAL_LEVELS - 1;
    }
    public void nextLevel() {
        if (hasNextLevel()) {
            currentLevel++;
        }
```

```
    }
    public int getCurrentLevel() {
        return currentLevel + 1;
    }
    public int getTotalLevels() {
        return TOTAL_LEVELS;
    }
}
```

Класс Player

```
package shaporenkoAndrew.com.gameObjects;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.utils.Array;
import shaporenkoAndrew.com.screens.GameScreen;
import com.badlogic.gdx.graphics.Pixmap;
public class Player extends GameObject {
    private static final float BASE_SPEED = 200f;
    private static final float SPRINT_MULTIPLIER = 1.5f;
    private static final float STAMINA_SPRINT_COST = 30f;
    private static final float STAMINA_MINING_COST = 20f;
    private static final float STAMINA_REGEN_RATE = 15f;
    private float currentSpeed;
    private int coins;
    private Array<Ore> inventory;
    private boolean isInventoryOpen;
    private float miningProgress;
    private float maxHealth = 100f;
    private float currentHealth;
    private float maxStamina = 100f;
    private float currentStamina;
    private boolean isStaminaRegenPaused;
    private Vector2 previousPosition;
    private Vector2 currentPosition;
    private GameScreen gameScreen;
    private Array<GameObject> objectsToRemove = new Array<>();
    private boolean isDead = false;
    private float speedMultiplier = 1.0f;
    private float miningTimer = 0;
    private Ore targetOre = null;
    private static final float MINING_DISTANCE = 70f;
    private static final float MINING_TIME = 1.0f;
    private boolean isFacingLeft = false;
    public Player(float x, float y, Texture texture, GameScreen gameScreen) {
        super(x, y, 32, 32, texture);
        this.gameScreen = gameScreen;
        this.currentSpeed = BASE_SPEED;
        this.coins = 0;
        this.inventory = new Array<>();
        this.isInventoryOpen = false;
        this.miningProgress = 0;
        this.currentHealth = maxHealth;
        this.currentStamina = maxStamina;
        this.previousPosition = new Vector2(x, y);
        this.currentPosition = new Vector2(x, y);
    }
```

```
    @Override
    public void update(float delta) {
        if (isDead) return;
        previousPosition.set(x, y);
        float moveX = 0;
        float moveY = 0;
        if (Gdx.input.isKeyPressed(Input.Keys.W)) moveY += 1;
        if (Gdx.input.isKeyPressed(Input.Keys.S)) moveY -= 1;
        if (Gdx.input.isKeyPressed(Input.Keys.A)) {
            moveX -= 1;
            isFacingLeft = false;
        }
        if (Gdx.input.isKeyPressed(Input.Keys.D)) {
            moveX += 1;
            isFacingLeft = true;
        }
        boolean isSprinting = Gdx.input.isKeyPressed(Input.Keys.SHIFT_LEFT) &&
currentStamina > 0;
        float currentBaseSpeed = BASE_SPEED * speedMultiplier;
        currentSpeed = isSprinting ? currentBaseSpeed * SPRINT_MULTIPLIER :
currentBaseSpeed;
        if (isSprinting && (moveX != 0 || moveY != 0)) {
            currentStamina = Math.max(0, currentStamina - STAMINA_SPRINT_COST
* delta);
            isStaminaRegenPaused = true;
        }
        if (targetOre != null) {
            miningTimer += delta;
            if (miningTimer >= MINING_TIME) {
                inventory.add(targetOre);
                gameScreen.removeObject(targetOre);
                targetOre = null;
                miningTimer = 0;
            }
        }
        if (!isSprinting) {
            if (isStaminaRegenPaused) {
                isStaminaRegenPaused = false;
            } else {
                currentStamina = Math.min(maxStamina, currentStamina +
STAMINA_REGEN_RATE * delta);
            }
        }
        if (moveX != 0 && moveY != 0) {
            moveX *= 0.7071f;
            moveY *= 0.7071f;
        }
        float newX = x + moveX * currentSpeed * delta;
        float newY = y + moveY * currentSpeed * delta;
        x = newX;
        bounds.setPosition(x, y);
        boolean collisionX = false;
        for (GameObject wall : gameScreen.getWallLayer()) {
            if (bounds.overlaps(wall.getBounds())) {
                collisionX = true;
                break;
            }
        }
        if (collisionX) {
            x = previousPosition.x;
            bounds.setPosition(x, y);
```

```
        }
        y = newY;
        bounds.setPosition(x, y);
        boolean collisionY = false;
        for (GameObject wall : gameScreen.getWallLayer()) {
            if (bounds.overlaps(wall.getBounds())) {
                collisionY = true;
                break;
            }
        }
        if (collisionY) {
            y = previousPosition.y;
            bounds.setPosition(x, y);
        }
        currentPosition.set(x, y);
        bounds.setPosition(x, y);
        if (Gdx.input.isKeyJustPressed(Input.Keys.TAB)) {
            isInventoryOpen = !isInventoryOpen;
        }
    }
    @Override
    public void render(SpriteBatch batch) {
        if (isFacingLeft) {
            batch.draw(texture, x + bounds.width, y, -bounds.width,
bounds.height);
        } else {
            batch.draw(texture, x, y, bounds.width, bounds.height);
        }
        if (targetOre != null) {
            float progress = getMiningProgress();
            float barWidth = 32;
            float barHeight = 4;
            float barX = targetOre.getX();
            float barY = targetOre.getY() + 36; // Немного выше руды
            Pixmap bgPixmap = new Pixmap((int)barWidth, (int)barHeight,
Pixmap.Format.RGBA8888);
            bgPixmap.setColor(0, 0, 0, 0.5f);
            bgPixmap.fillRectangle(0, 0, (int)barWidth, (int)barHeight);
            Texture bgTexture = new Texture(bgPixmap);
            batch.draw(bgTexture, barX, barY);
            bgPixmap.dispose();
            bgTexture.dispose();
            Pixmap fgPixmap = new Pixmap((int)(barWidth * progress),
(int)barHeight, Pixmap.Format.RGBA8888);
            fgPixmap.setColor(1, 1, 0, 1); // Желтый цвет для прогресса
            fgPixmap.fillRectangle(0, 0, (int)(barWidth * progress),
(int)barHeight);
            Texture fgTexture = new Texture(fgPixmap);
            batch.draw(fgTexture, barX, barY);
            fgPixmap.dispose();
            fgTexture.dispose();
        }
        if (isInventoryOpen) {
            renderInventory(batch);
        }
    }
    private void renderInventory(SpriteBatch batch) {
        // Отрисовка фона инвентаря
        // TODO: Добавить отрисовку инвентаря
    }
```

```
    public void checkCollisions(Array<GameObject> objects) {
        objectsToRemove.clear();
        Array<GameObject> tempObjects = new Array<>(objects);
        for (GameObject obj : tempObjects) {
            if (bounds.overlaps(obj.getBounds())) {
                handleCollision(obj);
            }
        }
        objects.removeAll(objectsToRemove, true);
    }
    public void handleCollision(GameObject obj) {
        if (obj instanceof Coin) {
            coins++;
            objectsToRemove.add(obj);
            gameScreen.coinCollected();
        } else if (obj instanceof Chest &&
Gdx.input.isKeyJustPressed(Input.Keys.E)) {
            Chest chest = (Chest) obj;
            if (!chest.isOpened()) {
                chest.open(this);
            }
        } else if (obj instanceof SecretDoor &&
Gdx.input.isKeyJustPressed(Input.Keys.E)) {
            SecretDoor door = (SecretDoor) obj;
            door.setGameScreen(gameScreen);
            door.checkAndOpen(gameScreen.getCollectedCoins());
            if (door.isOpen()) {
                door.interact();
            }
        } else if (obj instanceof Wall || obj instanceof Box) {
            resolveCollision(obj.getBounds());
        }
    }
    private void resolveCollision(Rectangle otherBounds) {
        if (bounds.overlaps(otherBounds)) {
            float overlapX = 0;
            float overlapY = 0;
            if (x < otherBounds.x) {
                overlapX = (x + bounds.width) - otherBounds.x;
            } else {
                overlapX = x - (otherBounds.x + otherBounds.width);
            }
            if (y < otherBounds.y) {
                overlapY = (y + bounds.height) - otherBounds.y;
            } else {
                overlapY = y - (otherBounds.y + otherBounds.height);
            }
            if (Math.abs(overlapX) < Math.abs(overlapY)) {
                x = previousPosition.x;
            } else {
                y = previousPosition.y;
            }
            bounds.setPosition(x, y);
            currentPosition.set(x, y);
        }
    }
...
    public void clearTargetOre() {
        targetOre = null;
        miningTimer = 0;  }
}
```