Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління

**Звіт**

до лабораторної роботи № 3

з дисципліни

Прикладне програмування

на тему:

**“** **Гра «Битва дроїдів»”**

Виконав: студент гр. ОІ-21

Коваль Максим

Прийняв: к.т.н, доцент

Римар Ю. М.

Львів - 2025

Хід виконання лабораторної роботи

1. Створіть базовий клас Droid, від якого будуть походити інші підкласи (види дроїдів), які будуть відрізнятися різними характеристиками. Мінімальний набір характеристик: name, health, damage.

2. Додайте можливість різних видів бою: 1 на 1, або команда на команду.

3. Класи потрібно грамотно розкласти по пакетах.

4. У програмі має бути консольне меню. Мінімальний набір команд:

− створити дроїда (обраного виду);

− показати список створених дроїдів;

− запустити бій 1 на 1 (вибрати дроїдів, які будуть змагатися);

− запустити бій команда на команду (сформувати команди суперників з дроїдів, яких ви створили у першому пункті);

− записати проведений бій у файл;

− відтворити проведений бій зі збереженого файлу;

− вийти з програми.

**1.Загальний план роботи та архітектура проекту**

Для дотримання вимоги щодо грамотного розподілу класів по пакетах, проект було розділено на три логічні частини (пакети):

**1) model** - Класи містять характеристики та унікальні здібності дроїдів.

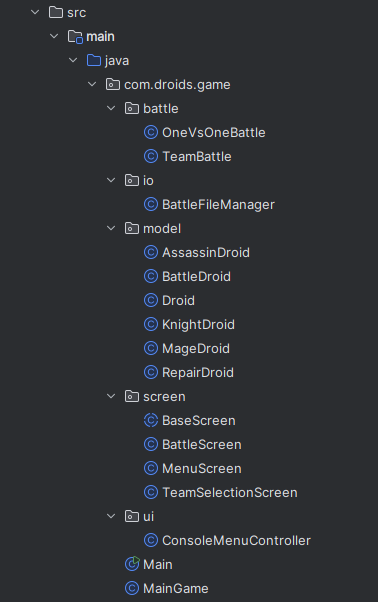
**2) battle** – Логіка бою.

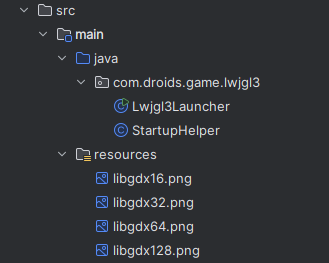
**3) іо** - Логіка роботи з файлами, що забезпечує запис та відтворення логів бою.

**4) screen** - Керування візуальними станами гри (екранами) за допомогою LibGDX.

**5) ui** - Класи для керування консольним меню та запуску програми.

**Структура проекту:**





**2. Опис файлів програми**

Пакет com.droidbattle.model (Моделі даних)

Цей пакет містить ієрархію класів, що представляють дроїдів.

* Droid.java
  + Базовий клас.
  + Визначає спільні поля: name, health, damage, maxHealth. Містить метод receiveDamage() (обробка вхідної шкоди) та isAlive().
* BattleDroid.java
  + Клас "Танк" / Бойовий Дроїд.
  + Високе HP (150). Перевизначає receiveDamage(), додаючи **ARMOR = 15**, що зменшує вхідну шкоду.
* KnightDroid.java
  + Клас "Легкий Танк".
  + Перевизначає receiveDamage(), додаючи **ARMOR = 5** для поглинання шкоди.
* AssassinDroid.java
  + Клас "Вбивця".
  + Низьке HP (90) та високий DMG (40). Перевизначає attack(), додаючи **шанс CRIT = 50%** (шкода x2).
* MageDroid.java
  + Клас "Маг" з ухиленням.
  + Середнє HP (95). Перевизначає receiveDamage(), додаючи **шанс EVASION = 35%**, що призводить до повного ухилення від вхідної шкоди.
* RepairDroid.java
  + Клас підтримки (Лікування).
  + Має поле **healingPower = 20**. Метод heal(target) використовується логікою TeamBattle для відновлення HP союзників (з 30% шансом замість атаки).

Пакет com.droids.game.battle, com.droids.game.io (Сервісний шар)

Цей шар інкапсулює логіку бою та файлові операції.

* TeamBattle.java
  + - Командний бій.
    - Має два конструктори: для **миттєвого** бою (консоль) і для **покрокового** (2D-графіка). Метод takeTurn() виконує один раунд (2.0 с затримки), включаючи логіку лікування RepairDroid.
  + OneVsOneBattle.java
    - Бій 1 на 1.
    - Використовується консольним меню для швидкої симуляції.
  + BattleFileManager.java
    - Управління логами.
    - Реалізує saveLogToFile() та replayBattle(), використовуючи **поточну робочу директорію** для зберігання файлів.

Пакет com.droids.game.screen та com.droids.game.ui (Презентаційний шар)

Цей шар відповідає за візуалізацію та керування програмою.

* MainGame.java
  + Менеджер LibGDX (Точка входу 2D).
  + Успадковує com.badlogic.gdx.Game. Завантажує всі ресурси (зображення, текстури) і керує перемиканням між екранами (MenuScreen, BattleScreen).
* MenuScreen.java
  + Головне графічне меню.
  + Використовує **Scene2D** для навігації до режимів 1v1 та Team vs Team.
* TeamSelectionScreen.java
  + Екран графічного вибору.
  + Дозволяє обрати **від 1 до 6 дроїдів** у кожну команду, адаптуючись до режиму (1v1 або командний).
* BattleScreen.java
  + Екран 2D-симуляції.
  + Керує покроковим відображенням бою (render з таймером). Реалізує **HP-бари** (ShapeRenderer), **дзеркальне відображення** (Team B) та **симетричне розміщення** дроїдів у сітці 3x2.
* ConsoleMenuController.java
  + Контролер консольного режиму.
  + Реалізує весь функціонал консольного меню, використовуючи Scanner.
* Main.java
  + Точка входу консольної версії.
  + Метод main() для запуску ConsoleMenuController.
* Lwjgl3Launcher.java
  + **Точка входу**.
  + Знаходиться у пакеті com.droids.game.lwjgl3. Створює та налаштовує вікно гри (розмір, заголовок, V-Sync) і запускає MainGame.

**Код програми:**

Droid.java

package com.droids.game.model;  
  
public class Droid {  
 private final String name; // Ім'я дроїда  
 private int health; // Поточне здоров'я  
 private final int maxHealth;  
 private int damage; // Сила атаки  
  
 // Конструктор  
 public Droid(String name, int health, int damage) {  
 this.name = name;  
 this.health = health;  
 this.maxHealth = health;  
 this.damage = damage;  
 }  
  
 // Дроїд атакує ціль  
 public void attack(Droid target) {  
 if (this.isAlive()) {  
 target.receiveDamage(this.damage);  
 }  
 }  
  
 // Отримання шкоди  
 public void receiveDamage(int incomingDamage) {  
 if (incomingDamage > 0) {  
 setHealth(getHealth() - incomingDamage);  
 }  
 }  
  
 // Перевірка, чи живий дроїд  
 public boolean isAlive() {  
 return health > 0;  
 }  
  
  
 public String getName() {  
 return name;  
 }  
  
 public int getHealth() {  
 return health;  
 }  
  
 public int getMaxHealth() {  
 return maxHealth;  
 }  
  
 public int getDamage() {  
 return damage;  
 }  
  
 // Встановлює здоров'я, мінімум 0  
 public void setHealth(int health) {  
 this.health = Math.*max*(0, health);  
 }  
  
 // Повертає тип (назву класу)  
 public String getType() {  
 return this.getClass().getSimpleName();  
 }  
  
 // Перевизначення для зручного виведення  
 @Override  
 public String toString() {  
 return String.*format*("%s (HP: %d, DMG: %d)", name, health, damage);  
 }  
}

BattleDroid.java

package com.droids.game.model;  
  
public class BattleDroid extends Droid {  
 private final int armor = 15; // Броня: зменшує вхідну шкоду на 15  
  
 public BattleDroid(String name) {  
 // HP 150, DMG 20  
 super(name, 150, 20);  
 }  
 // Перевизначення методу отримання шкоди  
 @Override  
 public void receiveDamage(int incomingDamage) {  
 int damageAfterArmor = Math.*max*(0, incomingDamage - armor);  
 super.receiveDamage(damageAfterArmor);  
 }  
}

AssassinDroid.java

package com.droids.game.model;  
  
import java.util.Random;  
  
public class AssassinDroid extends Droid {  
 private static final Random *RANDOM* = new Random();  
 private final double criticalChance = 0.5; // 50% шанс криту  
 private final int criticalMultiplier = 2; // Шкода x2  
  
 public AssassinDroid(String name) {  
 super(name, 90, 40);  
 }  
 // Перевизначення методу атаки з логікою критичного удару  
 @Override  
 public void attack(Droid target) {  
 if (this.isAlive()) {  
 int damage = this.getDamage();  
 boolean isCritical = *RANDOM*.nextDouble() < criticalChance; // Випадкове число від 0 до 1  
  
 if (isCritical) {  
 damage \*= criticalMultiplier;  
 System.*out*.printf("%s ЗАВДАЄ КРИТИЧНИЙ УДАР! Шкода: %d\n", this.getName(), damage);  
 }  
 target.receiveDamage(damage);  
 }  
 }  
}

KnightDroid.java

package com.droids.game.model;  
  
public class KnightDroid extends Droid {  
 private final int armor = 5; // Броня: зменшує вхідну шкоду  
  
 public KnightDroid(String name) {  
 super(name, 120, 25);  
 }  
 // Перевизначення методу отримання шкоди з урахуванням броні  
 @Override  
 public void receiveDamage(int incomingDamage) {  
 int damageAfterArmor = Math.*max*(0, incomingDamage - armor);  
 super.receiveDamage(damageAfterArmor);  
 }  
}

MageDroid.java

package com.droids.game.model;  
  
import java.util.Random;  
  
public class MageDroid extends Droid {  
 private static final Random *RANDOM* = new Random();  
 private final double evasionChance = 0.35; // 35% шанс ухилення  
  
 public MageDroid(String name) {  
 super(name, 95, 20);  
 }  
 // Перевизначення методу отримання шкоди з логікою ухилення  
 @Override  
 public void receiveDamage(int incomingDamage) {  
 boolean evaded = *RANDOM*.nextDouble() < evasionChance;  
  
 if (evaded) {  
 System.*out*.printf("%s УХИЛИВСЯ від шкоди!\n", this.getName());  
 // Шкода не приймається  
 } else {  
 // Приймаємо повну шкоду  
 super.receiveDamage(incomingDamage);  
 }  
 }  
}

RepairDroid.java

package com.droids.game.model;  
  
public class RepairDroid extends Droid {  
 private int healingPower;  
  
 public RepairDroid(String name) {  
 super(name, 80, 5);  
 this.healingPower = 20; // Сила лікування  
 }  
  
 // Нова здібність: лікування  
 public void heal(Droid target) {  
 if (this.isAlive()) {  
 target.setHealth(target.getHealth() + healingPower);  
 }  
 }  
  
 public int getHealingPower() {  
 return healingPower;  
 }  
}

OneVsOneBattle.java

package com.droids.game.battle;  
  
import com.droids.game.model.Droid;  
import java.util.ArrayList;  
import java.util.List;  
  
public class OneVsOneBattle {  
 // Список для зберігання логу бою  
 private List<String> battleLog = new ArrayList<>();  
  
 // Запускає бій між двома дроїдами  
 public List<String> startBattle(Droid droid1, Droid droid2) {  
 battleLog.clear();  
 battleLog.add("=== Бій 1 на 1 розпочато! ===");  
 battleLog.add(String.*format*("Учасники: %s vs %s", droid1.getName(), droid2.getName()));  
  
 int turn = 1;  
 // Цикл триває, доки обидва дроїди живі  
 while (droid1.isAlive() && droid2.isAlive()) {  
  
 // Логування стану перед ходом  
 battleLog.add(String.*format*("\n--- Хід %d ---", turn));  
 battleLog.add(String.*format*("Статус: %s | %s", droid1.toString(), droid2.toString()));  
  
 Droid attacker, defender;  
  
 // Визначення черговості атаки  
 if (turn % 2 != 0) { // Непарний хід: droid1 атакує  
 attacker = droid1;  
 defender = droid2;  
 } else { // Парний хід: droid2 атакує  
 attacker = droid2;  
 defender = droid1;  
 }  
  
 // Виконання атаки  
 if (attacker.isAlive()) {  
 attacker.attack(defender);  
 // Примітка: тут логується базова шкода. Фактична шкода (критична/броня) логується у класах Droid.  
 battleLog.add(String.*format*("%s атакує %s і завдає %d шкоди.",  
 attacker.getName(), defender.getName(), attacker.getDamage()));  
 }  
  
 turn++;  
 }  
  
 // Визначення переможця та логування результату  
 Droid winner = droid1.isAlive() ? droid1 : droid2;  
 battleLog.add("\n\*\*\* Переможець: " + winner.getName() + " \*\*\*");  
  
 return battleLog; // Повернення логу  
 }  
}

TeamBattle.java

package com.droids.game.battle;  
  
import com.droids.game.model.Droid;  
import com.droids.game.model.RepairDroid;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
  
// Клас для симуляції бою команда на команду  
public class TeamBattle {  
  
 private final List<String> battleLog = new ArrayList<>(); // Список для логування  
 private final Random random = new Random();  
 private List<Droid> teamA;  
 private List<Droid> teamB;  
 private int round = 1;  
   
 // КОНСТРУКТОРИ  
 // Порожній конструктор  
 public TeamBattle() {  
 }  
  
 // Конструктор для 2D-версії (Покроковий бій)  
 public TeamBattle(List<Droid> teamA, List<Droid> teamB) {  
 this.teamA = teamA;  
 this.teamB = teamB;  
 battleLog.add("=== Бій Команда на Команду розпочато! ===");  
 battleLog.add(String.*format*("Учасники: Команда A (%d) vs Команда Б (%d)", teamA.size(), teamB.size()));  
 }  
   
 // МЕТОДИ ДЛЯ 2D   
  
 // Виконати один раунд бою  
 public boolean takeTurn() {  
 if (isFinished()) {  
 logFinalResult();  
 return false; // Бій завершено  
 }  
  
 battleLog.add("\n--- Раунд " + round + " ---");  
  
 teamAttack(teamA, teamB, "Команда А"); // Команда А атакує  
  
 if (!teamB.isEmpty()) {  
 teamAttack(teamB, teamA, "Команда Б"); // Команда Б атакує  
 }  
  
 round++;  
 return true; // Бій триває  
 }  
  
 // Перевірка, чи завершено бій  
 public boolean isFinished() {  
 return teamA.isEmpty() || teamB.isEmpty() || round > 100; // Або одна з команд знищена, або ліміт раундів  
 }  
  
 public List<String> getBattleLog() {  
 return battleLog;  
 }  
   
 // МЕТОД ДЛЯ КОНСОЛІ (МИТТЄВИЙ РЕЖИМ)  
   
 public List<String> startBattle(List<Droid> teamA, List<Droid> teamB) {  
 List<String> instantLog = new ArrayList<>();  
  
 List<Droid> aliveTeamA = new ArrayList<>(teamA);  
 List<Droid> aliveTeamB = new ArrayList<>(teamB);  
 int turn = 1;  
  
 instantLog.add("=== Бій Команда на Команду розпочато (КОНСОЛЬ) ===");  
  
 // Основний цикл миттєвого бою  
 while (!aliveTeamA.isEmpty() && !aliveTeamB.isEmpty() && turn < 100) {  
 instantLog.add("\n--- Хід " + turn + " ---");  
  
 // Команди атакують по черзі  
 performInstantAttack(aliveTeamA, aliveTeamB, instantLog, "Команда А");  
 performInstantAttack(aliveTeamB, aliveTeamA, instantLog, "Команда Б");  
  
 turn++;  
 }  
  
 // Лог фінального результату  
 if (!aliveTeamA.isEmpty()) {  
 instantLog.add("\n\*\*\* Переможець: Команда А \*\*\*");  
 } else if (!aliveTeamB.isEmpty()) {  
 instantLog.add("\n\*\*\* Переможець: Команда Б \*\*\*");  
 } else {  
 instantLog.add("\n\*\*\* Нічия \*\*\*");  
 }  
  
 return instantLog;  
 }  
  
 // Допоміжний метод для виконання миттєвої атаки  
 private void performInstantAttack(List<Droid> attackers, List<Droid> defenders, List<String> log, String teamName) {  
 for (Droid attacker : new ArrayList<>(attackers)) {  
 if (!attacker.isAlive() || defenders.isEmpty()) continue;  
 // Вибір випадкової цілі  
 Droid defender = defenders.get(random.nextInt(defenders.size()));  
 attacker.attack(defender);  
 log.add(String.*format*("%s: %s атакує %s.", teamName, attacker.getName(), defender.getName()));  
 if (!defender.isAlive()) {  
 log.add(String.*format*("❌ %s знищено!", defender.getName()));  
 defenders.remove(defender); // Видалення знищеного дроїда  
 }  
 }  
 attackers.removeIf(droid -> !droid.isAlive()); // Видалення мертвих атакувальників  
 }  
   
 // ЛОГІКА ЛІКУВАННЯ (ДЛЯ 2D-версії)  
   
 // Спроба лікування союзника RepairDroid'ом  
 private boolean attemptHeal(Droid healer, List<Droid> team, String teamName) {  
 if (!(healer instanceof RepairDroid) || team.isEmpty()) {  
 return false;  
 }  
  
 if (random.nextDouble() > 0.70) { // 30% шанс на лікування  
 return false;  
 }  
  
 RepairDroid repairDroid = (RepairDroid) healer;  
  
 // Знаходимо всіх живих союзників  
 List<Droid> potentialTargets = new ArrayList<>();  
 for (Droid d : team) {  
 if (d.isAlive()) {  
 potentialTargets.add(d);  
 }  
 }  
  
 if (potentialTargets.isEmpty()) return false;  
  
 Droid target = potentialTargets.get(random.nextInt(potentialTargets.size())); // Випадкова ціль  
  
 repairDroid.heal(target); // Виконання лікування  
  
 battleLog.add(String.*format*("➕ %s: %s ЛІКУЄ %s (+%d HP). HP %s: %d",  
 teamName, repairDroid.getName(), target.getName(),  
 repairDroid.getHealingPower(), target.getName(), target.getHealth()));  
  
 return true;  
 }  
   
 // Логіка атаки однієї команди по іншій (для 2D-версії)  
 private void teamAttack(List<Droid> attackingTeam, List<Droid> defendingTeam, String teamName) {  
 List<Droid> attackers = new ArrayList<>(attackingTeam);  
  
 for (Droid attacker : attackers) {  
 if (!attacker.isAlive() || defendingTeam.isEmpty()) continue;  
  
 // --- ПЕРЕВІРКА ЛІКУВАННЯ ---  
 if (attacker instanceof RepairDroid) {  
 if (attemptHeal(attacker, attackingTeam, teamName)) {  
 continue; // Лікування відбулося  
 }  
 }  
 // ---------------------------  
  
 Droid defender = defendingTeam.get(random.nextInt(defendingTeam.size())); // Вибір цілі  
  
 attacker.attack(defender);  
  
 battleLog.add(String.*format*("%s: %s атакує %s.", teamName, attacker.getName(), defender.getName()));  
  
 // Перевірка, чи ціль знищена  
 if (!defender.isAlive()) {  
 battleLog.add(String.*format*("❌ %s знищено!", defender.getName()));  
 defendingTeam.remove(defender);  
 }  
 }  
 attackingTeam.removeIf(droid -> !droid.isAlive()); // Очищення від мертвих  
 }  
  
 // Логування фінального результату  
 private void logFinalResult() {  
 if (teamB.isEmpty() && !teamA.isEmpty()) {  
 battleLog.add("\n\*\*\* Переможець: Команда А \*\*\*");  
 } else if (teamA.isEmpty() && !teamB.isEmpty()) {  
 battleLog.add("\n\*\*\* Переможець: Команда Б \*\*\*");  
 } else if (round > 100) {  
 battleLog.add("\n\*\*\* Бій завершено за обмеженням раундів (100) \*\*\*");  
 } else {  
 battleLog.add("\n\*\*\* Нічия або бій зупинено \*\*\*");  
 }  
 }  
}

BattleFileManager.java

package com.droids.game.io;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.util.List;  
  
public class BattleFileManager {  
  
 // Зберігає лог бою у файл  
 public static void saveLogToFile(List<String> log, String filename) {  
 String fullPath = filename;  
  
 try (FileWriter writer = new FileWriter(fullPath)) {  
 for (String line : log) {  
 writer.write(line + System.*lineSeparator*()); // Запис кожного рядка  
 }  
 System.*out*.println("Лог бою успішно записано у файл: " + filename);  
 } catch (IOException e) {  
 System.*err*.println("Помилка при записі файлу: " + e.getMessage());  
 }  
 }  
  
 // Зчитує та виводить лог бою з файлу  
 public static void replayBattle(String filename) {  
 try {  
 List<String> lines = Files.*readAllLines*(Paths.*get*(filename));  
 System.*out*.println("\n--- ВІДТВОРЕННЯ БОЮ З ФАЙЛУ: " + filename + " ---");  
 for (String line : lines) {  
 System.*out*.println(line);  
 }  
 System.*out*.println("--- КІНЕЦЬ ВІДТВОРЕННЯ ---");  
 } catch (IOException e) {  
 System.*err*.println("Помилка при читанні файлу або файл не знайдено: " + e.getMessage());  
 }  
 }  
}

BaseScreen.java

package com.droids.game.screen;  
  
import com.badlogic.gdx.Screen;  
import com.droids.game.MainGame;  
  
// Базовий абстрактний клас для всіх екранів гри  
public abstract class BaseScreen implements Screen {  
 protected final MainGame game; // Посилання на головний клас гри  
  
 // Конструктор  
 public BaseScreen(MainGame game) {  
 this.game = game;  
 }  
  
 // Методи інтерфейсу Screen, які є обов'язковими, але реалізація  
 // залишається порожньою для більшості базових функцій.  
  
 @Override  
 public void render(float delta) {} // Основний цикл оновлення/малювання  
  
 @Override  
 public void resize(int width, int height) {} // Обробка зміни розміру вікна  
  
 @Override  
 public void pause() {} // Пауза гри  
  
 @Override  
 public void resume() {} // Відновлення гри  
  
 @Override  
 public void show() {} // Викликається при встановленні екрана (видалено у вашому коді, але я додав його)  
  
 @Override  
 public void hide() {} // Викликається, коли екран приховано  
  
 @Override  
 public void dispose() {} // Звільнення ресурсів  
}

BattleScreen.java

package com.droids.game.screen;  
  
import com.badlogic.gdx.Gdx;  
import com.badlogic.gdx.graphics.Color;  
import com.badlogic.gdx.graphics.Texture;  
import com.badlogic.gdx.graphics.g2d.SpriteBatch;  
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;  
import com.badlogic.gdx.utils.ScreenUtils;  
import com.droids.game.MainGame;  
import com.droids.game.battle.TeamBattle;  
import com.droids.game.model.\*;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
// Екран для візуалізації командного бою (покроковий режим)  
public class BattleScreen extends BaseScreen {  
  
 private final List<Droid> teamA;  
 private final List<Droid> teamB;  
 private final boolean isOneVsOneMode; // Режим 1 на 1  
  
 // --- Логіка Бою ---  
 private final TeamBattle teamBattle;  
 private float timeSinceLastTurn = 0f;  
 private final float TURN\_DELAY = 2.0f; // Затримка між ходами (2 секунди)  
 private boolean battleRunning = true; // Стан бою  
 private int lastLogIndex = 0;  
  
 // --- Логіка завершення ---  
 private float timeAfterBattleEnd = 0f;  
 private final float EXIT\_DELAY = 3.0f; // Затримка перед виходом до меню  
  
 // --- Візуалізація ---  
 private ShapeRenderer shapeRenderer; // Для малювання HP-барів  
 private final Map<Droid, Integer> maxHealthMap; // Зберігає початкове HP для розрахунку бару  
  
 // Параметри розміщення  
 private final float DROID\_DRAW\_SIZE = 80f; // Розмір спрайта дроїда  
 private final float DROID\_PADDING\_Y = 20f;  
 private final float DROID\_PADDING\_X = 50f;  
 private final float GRID\_OFFSET\_X\_TEAM\_A = 150f;  
 private final float GRID\_OFFSET\_X\_TEAM\_B = 150f;  
 private final float GRID\_CENTER\_OFFSET\_Y;  
 private final float ONE\_VS\_ONE\_CENTER\_OFFSET = 150f;  
  
 // Конструктор  
 public BattleScreen(MainGame game, List<Droid> teamA, List<Droid> teamB) {  
 super(game);  
 this.teamA = teamA;  
 this.teamB = teamB;  
 this.maxHealthMap = new HashMap<>();  
 this.isOneVsOneMode = (teamA.size() == 1 && teamB.size() == 1);  
  
 // Обчислення центрального зміщення Y  
 float totalTeamHeight = (DROID\_DRAW\_SIZE \* 3) + (DROID\_PADDING\_Y \* 2);  
 GRID\_CENTER\_OFFSET\_Y = (Gdx.*graphics*.getHeight() - totalTeamHeight) / 2f;  
  
 initializeMaxHealths(teamA); // Ініціалізація максимального HP  
 initializeMaxHealths(teamB);  
  
 this.teamBattle = new TeamBattle(teamA, teamB); // Ініціалізація логіки бою  
  
 // Логування початкового статусу  
 List<String> initialLog = teamBattle.getBattleLog();  
 for (String line : initialLog) {  
 Gdx.*app*.log("Battle Setup", line);  
 }  
 lastLogIndex = initialLog.size();  
 }  
  
 // Допоміжний метод для визначення максимального HP дроїда  
 private void initializeMaxHealths(List<Droid> team) {  
 for (Droid droid : team) {  
 Droid tempDroid;  
 String className = droid.getClass().getSimpleName();  
  
 // Використовуємо тимчасовий екземпляр для отримання maxHealth  
 switch (className) {  
 case "KnightDroid": tempDroid = new KnightDroid("temp"); break;  
 case "AssassinDroid": tempDroid = new AssassinDroid("temp"); break;  
 case "RepairDroid": tempDroid = new RepairDroid("temp"); break;  
 case "MageDroid": tempDroid = new MageDroid("temp"); break;  
 case "BattleDroid": tempDroid = new BattleDroid("temp"); break;  
 default: tempDroid = new Droid("temp", 100, 10); break;  
 }  
 maxHealthMap.put(droid, tempDroid.getHealth());  
 }  
 }  
  
 @Override  
 public void show() {  
 shapeRenderer = new ShapeRenderer();  
 }  
  
 // Основний цикл рендерингу (малювання та оновлення)  
 @Override  
 public void render(float delta) {  
 if (battleRunning) {  
 timeSinceLastTurn += delta;  
 // Перевіряємо, чи настав час для наступного ходу  
 if (timeSinceLastTurn >= TURN\_DELAY) {  
 battleRunning = teamBattle.takeTurn(); // Виконуємо хід  
  
 // Вивід нового логу в консоль  
 List<String> currentLog = teamBattle.getBattleLog();  
 for (int i = lastLogIndex; i < currentLog.size(); i++) {  
 Gdx.*app*.log("Battle Log", currentLog.get(i));  
 }  
 lastLogIndex = currentLog.size();  
  
 timeSinceLastTurn = 0f;  
 }  
 } else {  
 // Логіка затримки після закінчення бою  
 timeAfterBattleEnd += delta;  
 if (timeAfterBattleEnd >= EXIT\_DELAY) {  
 Gdx.*app*.log("BATTLE END", "Delay finished. Returning to Menu.");  
 game.setScreen(new MenuScreen(game)); // Повернення до меню (потрібен MenuScreen)  
 return;  
 }  
 }  
  
 ScreenUtils.*clear*(0.1f, 0.1f, 0.15f, 1); // Очищення екрана  
  
 game.batch.begin();  
 game.batch.draw(game.arenaBackground, 0, 0, Gdx.*graphics*.getWidth(), Gdx.*graphics*.getHeight()); // Малювання фону  
  
 // Малювання дроїдів  
 if (isOneVsOneMode) {  
 drawOneVsOne();  
 } else {  
 drawTeam(teamA, true); // Малюємо Команду А  
 drawTeam(teamB, false); // Малюємо Команду Б (віддзеркалену)  
 }  
  
 game.batch.end();  
  
 drawHealthBars(); // Малювання HP-барів (ShapeRenderer)  
 }  
  
 // Малювання в режимі 1 на 1  
 private void drawOneVsOne() {  
 if (teamA.isEmpty() || teamB.isEmpty()) return;  
  
 float center\_X = Gdx.*graphics*.getWidth() / 2f;  
 float center\_Y = Gdx.*graphics*.getHeight() / 2f;  
  
 // Малюємо дроїдів зі зміщенням  
 drawSingleDroid(teamA.get(0), center\_X - ONE\_VS\_ONE\_CENTER\_OFFSET, center\_Y, true);  
 drawSingleDroid(teamB.get(0), center\_X + ONE\_VS\_ONE\_CENTER\_OFFSET, center\_Y, false);  
 }  
  
 // Малювання одного дроїда (використовується в 1vs1)  
 private void drawSingleDroid(Droid droid, float x, float y, boolean facingRight) {  
 SpriteBatch batch = game.batch;  
 String droidKey = getCorrectedDroidTextureKey(droid);  
 Texture texture = game.droidTextures.get(droidKey);  
  
 // Не малюємо мертвих дроїдів після закінчення бою  
 if (!droid.isAlive() && !battleRunning) {  
 return;  
 }  
  
 float size = DROID\_DRAW\_SIZE;  
  
 // Зміна кольору для мертвих/пошкоджених  
 if (!droid.isAlive()) { batch.setColor(0.5f, 0.5f, 0.5f, 0.5f); }  
 else { batch.setColor(Color.*WHITE*); }  
  
 // Малювання спрайта (віддзеркалення, якщо потрібно)  
 if (facingRight) {  
 batch.draw(texture, x, y, size, size);  
 } else {  
 batch.draw(texture, x + size, y, -size, size);  
 }  
 batch.setColor(Color.*WHITE*);  
 }  
  
 // Малювання цілої команди (в режимі сітки)  
 private void drawTeam(List<Droid> team, boolean facingRight) {  
 SpriteBatch batch = game.batch;  
 float screenWidth = Gdx.*graphics*.getWidth();  
 float startX;  
 float teamWidth = DROID\_DRAW\_SIZE + DROID\_PADDING\_X;  
  
 // Визначення стартової позиції X  
 if (facingRight) {  
 startX = GRID\_OFFSET\_X\_TEAM\_A;  
 } else {  
 startX = screenWidth - GRID\_OFFSET\_X\_TEAM\_B - teamWidth;  
 }  
  
 float startY = GRID\_CENTER\_OFFSET\_Y;  
 int rowCount = 3;  
  
 for (int i = 0; i < team.size(); i++) {  
 Droid droid = team.get(i);  
  
 String droidKey = getCorrectedDroidTextureKey(droid);  
 Texture texture = game.droidTextures.get(droidKey);  
 if (texture == null) continue;  
  
 // Розрахунок позиції в сітці  
 int columnIndex = i / rowCount;  
 int rowIndex = i % rowCount;  
  
 float drawX = startX + columnIndex \* (DROID\_DRAW\_SIZE + DROID\_PADDING\_X);  
 float drawY = startY + rowIndex \* (DROID\_DRAW\_SIZE + DROID\_PADDING\_Y);  
  
 // Налаштування кольору для мертвих дроїдів  
 if (!droid.isAlive()) {  
 batch.setColor(0.5f, 0.5f, 0.5f, 0.5f);  
 } else {  
 batch.setColor(Color.*WHITE*);  
 }  
  
 // Малювання спрайта  
 if (facingRight) {  
 batch.draw(texture, drawX, drawY, DROID\_DRAW\_SIZE, DROID\_DRAW\_SIZE);  
 } else {  
 batch.draw(texture, drawX + DROID\_DRAW\_SIZE, drawY, -DROID\_DRAW\_SIZE, DROID\_DRAW\_SIZE);  
 }  
  
 batch.setColor(Color.*WHITE*);  
 }  
 }  
  
 // Отримання ключа текстури з назви класу (наприклад, AssassinDroid -> assassin)  
 private String getCorrectedDroidTextureKey(Droid droid) {  
 return droid.getClass().getSimpleName().toLowerCase().replace("droid", "");  
 }  
  
 // Головний метод для малювання HP-барів  
 private void drawHealthBars() {  
 drawHealthBarsForTeam(teamA, true);  
 drawHealthBarsForTeam(teamB, false);  
 }  
  
 // Малювання HP-барів для однієї команди  
 private void drawHealthBarsForTeam(List<Droid> team, boolean isTeamA) {  
 if (shapeRenderer == null) return;  
  
 // \*\*ВИПРАВЛЕННЯ:\*\* Оголошення локальної змінної screenWidth  
 float screenWidth = Gdx.*graphics*.getWidth();  
  
 shapeRenderer.setProjectionMatrix(game.batch.getProjectionMatrix());  
 shapeRenderer.begin(ShapeRenderer.ShapeType.*Filled*);  
  
 float barWidth = DROID\_DRAW\_SIZE;  
 float barHeight = 8f;  
  
 if (isOneVsOneMode) {  
 // Логіка малювання HP-барів 1 на 1  
 if (team.isEmpty()) {  
 shapeRenderer.end();  
 return;  
 }  
 Droid droid = team.get(0);  
  
 if (!droid.isAlive() && !battleRunning) { // Не малюємо бар, якщо дроїд мертвий і бій закінчено  
 shapeRenderer.end();  
 return;  
 }  
  
 float center\_X = screenWidth / 2f;  
 float center\_Y = Gdx.*graphics*.getHeight() / 2f;  
  
 // Розрахунок позиції бару  
 float droidX = isTeamA ? center\_X - ONE\_VS\_ONE\_CENTER\_OFFSET : center\_X + ONE\_VS\_ONE\_CENTER\_OFFSET;  
 float hpBarX = droidX;  
 if (!isTeamA) {  
 hpBarX = droidX + DROID\_DRAW\_SIZE - barWidth;  
 }  
  
 float hpBarY = center\_Y + DROID\_DRAW\_SIZE + 5f;  
  
 int maxHealth = maxHealthMap.getOrDefault(droid, 100);  
 float healthRatio = (float) droid.getHealth() / maxHealth;  
  
 shapeRenderer.setColor(Color.*RED*); // Фон (залишок HP)  
 shapeRenderer.rect(hpBarX, hpBarY, barWidth, barHeight);  
 shapeRenderer.setColor(Color.*GREEN*); // Поточне HP  
 shapeRenderer.rect(hpBarX, hpBarY, barWidth \* healthRatio, barHeight);  
  
 } else {  
 // Логіка малювання HP-барів командного бою  
 float startY = GRID\_CENTER\_OFFSET\_Y;  
 int rowCount = 3;  
  
 // Визначення стартової позиції X  
 float startX;  
 if (isTeamA) {  
 startX = GRID\_OFFSET\_X\_TEAM\_A;  
 } else {  
 float teamWidth = DROID\_DRAW\_SIZE + DROID\_PADDING\_X;  
 // Ось тут була помилка  
 startX = screenWidth - GRID\_OFFSET\_X\_TEAM\_B - teamWidth;  
 }  
  
 for (int i = 0; i < team.size(); i++) {  
 Droid droid = team.get(i);  
 if (!maxHealthMap.containsKey(droid) || droid.getHealth() <= 0) continue; // Пропуск мертвих  
  
 int columnIndex = i / rowCount;  
 int rowIndex = i % rowCount;  
  
 // Розрахунок позиції бару над дроїдом  
 float drawX = startX + columnIndex \* (DROID\_DRAW\_SIZE + DROID\_PADDING\_X);  
 float drawY = startY + rowIndex \* (DROID\_DRAW\_SIZE + DROID\_PADDING\_Y) + DROID\_DRAW\_SIZE + 5f;  
  
 int maxHealth = maxHealthMap.get(droid);  
 float healthRatio = (float) droid.getHealth() / maxHealth;  
  
 shapeRenderer.setColor(Color.*RED*); // Фон  
 shapeRenderer.rect(drawX, drawY, barWidth, barHeight);  
 shapeRenderer.setColor(Color.*GREEN*); // Поточне HP  
 shapeRenderer.rect(drawX, drawY, barWidth \* healthRatio, barHeight);  
 }  
 }  
  
 shapeRenderer.end();  
 }  
  
  
 @Override  
 public void dispose() {  
 if (shapeRenderer != null) {  
 shapeRenderer.dispose();  
 }  
 }  
}

MenuScreen.java

package com.droids.game.screen;  
  
import com.badlogic.gdx.Gdx;  
import com.badlogic.gdx.graphics.Color;  
import com.badlogic.gdx.graphics.g2d.BitmapFont;  
import com.badlogic.gdx.scenes.scene2d.Stage;  
import com.badlogic.gdx.scenes.scene2d.ui.Label;  
import com.badlogic.gdx.scenes.scene2d.ui.Skin;  
import com.badlogic.gdx.scenes.scene2d.ui.Table;  
import com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;  
import com.badlogic.gdx.scenes.scene2d.InputEvent;  
import com.badlogic.gdx.utils.viewport.ScreenViewport;  
import com.droids.game.MainGame;  
  
// Екран головного меню  
public class MenuScreen extends BaseScreen {  
  
 private Stage stage; // Сцена для керування елементами UI (кнопки, написи)  
 private Skin skin; // Скін для визначення стилів елементів UI  
  
 // Конструктор  
 public MenuScreen(MainGame game) {  
 super(game);  
 stage = new Stage(new ScreenViewport());  
  
 skin = new Skin();  
 BitmapFont font = new BitmapFont();  
 skin.add("default-font", font); // Додавання стандартного шрифту  
  
 // Створення стилю для кнопок  
 TextButton.TextButtonStyle textButtonStyle = new TextButton.TextButtonStyle();  
 textButtonStyle.font = skin.getFont("default-font");  
 textButtonStyle.fontColor = Color.*WHITE*;  
 skin.add("default", textButtonStyle);  
  
 // Створення стилю для написів (Label)  
 Label.LabelStyle labelStyle = new Label.LabelStyle(font, Color.*WHITE*);  
 skin.add("default", labelStyle);  
  
 createTable(); // Створення та додавання елементів до таблиці  
 }  
  
 // Створює таблицю, яка містить усі елементи меню  
 private void createTable() {  
 Table table = new Table();  
 table.setFillParent(true); // Заповнити всю сцену  
  
 // 1. Заголовок  
 Label title = new Label("DROID BATTLE ARENA", skin);  
 table.add(title).padBottom(50).row();  
  
 // 2. Кнопка "Бій 1 на 1"  
 TextButton oneVsOneButton = new TextButton("1 vs 1 Battle", skin);  
 oneVsOneButton.addListener(new ClickListener() {  
 @Override  
 public void clicked(InputEvent event, float x, float y) {  
 // Перехід до екрана вибору команди, встановлюємо режим 1 на 1  
 game.setScreen(new TeamSelectionScreen(game, true));  
 }  
 });  
 table.add(oneVsOneButton).width(300).height(50).pad(10).row();  
  
 // 3. Кнопка "Команда на команду"  
 TextButton teamButton = new TextButton("Team vs Team Battle", skin);  
 teamButton.addListener(new ClickListener() {  
 @Override  
 public void clicked(InputEvent event, float x, float y) {  
 // Перехід до екрана вибору команди, режим Команда на Команду  
 game.setScreen(new TeamSelectionScreen(game, false));  
 }  
 });  
 table.add(teamButton).width(300).height(50).pad(10).row();  
  
 // 4. Кнопка "Вихід"  
 TextButton exitButton = new TextButton("Exit Game", skin);  
 exitButton.addListener(new ClickListener() {  
 @Override  
 public void clicked(InputEvent event, float x, float y) {  
 Gdx.*app*.exit(); // Вихід з програми  
 }  
 });  
 table.add(exitButton).width(300).height(50).pad(10).row();  
  
 stage.addActor(table);  
 }  
  
 @Override  
 public void show() {  
 Gdx.*input*.setInputProcessor(stage); // Обробка вводу сценою  
 }  
  
 @Override  
 public void render(float delta) {  
 // Малювання фону  
 game.batch.begin();  
 game.batch.draw(game.arenaBackground, 0, 0, Gdx.*graphics*.getWidth(), Gdx.*graphics*.getHeight());  
 game.batch.end();  
  
 stage.act(Gdx.*graphics*.getDeltaTime()); // Оновлення сцени (кнопки, тощо)  
 stage.draw(); // Малювання сцени  
 }  
  
 @Override  
 public void dispose() {  
 stage.dispose();  
 skin.dispose(); // Звільнення ресурсів  
 }  
}

TeamSelectionScreen.java

package com.droids.game.screen;  
  
import com.badlogic.gdx.Gdx;  
import com.badlogic.gdx.graphics.Color;  
import com.badlogic.gdx.graphics.Texture;  
import com.badlogic.gdx.graphics.g2d.BitmapFont;  
import com.badlogic.gdx.scenes.scene2d.InputEvent;  
import com.badlogic.gdx.scenes.scene2d.Stage;  
import com.badlogic.gdx.scenes.scene2d.ui.\*;  
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;  
import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;  
import com.badlogic.gdx.utils.viewport.ScreenViewport;  
import com.droids.game.MainGame;  
import com.droids.game.model.\*;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Map;  
  
// Екран для вибору дроїдів перед початком бою  
public class TeamSelectionScreen extends BaseScreen {  
  
 private Stage stage;  
 private Skin skin;  
  
 private final int MAX\_DROIDS\_PER\_TEAM\_CAP = 6;  
 private final boolean isOneVsOneMode; // Режим 1 на 1 чи Команда на Команду  
  
 private List<Droid> teamA = new ArrayList<>(); // Дроїди Команди А  
 private List<Droid> teamB = new ArrayList<>(); // Дроїди Команди Б  
  
 private boolean selectingTeamA = true; // Прапорець, яку команду обираємо  
 private Label selectionStatusLabel; // Напис зі статусом вибору  
 private Table teamPreviewTableA; // Панель для відображення Команди А  
 private Table teamPreviewTableB; // Панель для відображення Команди Б  
  
 // Конструктор за замовчуванням (Команда на Команду)  
 public TeamSelectionScreen(MainGame game) {  
 this(game, false);  
 }  
  
 // Основний конструктор  
 public TeamSelectionScreen(MainGame game, boolean isOneVsOneMode) {  
 super(game);  
 this.isOneVsOneMode = isOneVsOneMode;  
 stage = new Stage(new ScreenViewport());  
  
 setupSkin();  
 createUI();  
 updateStatusLabel();  
 }  
  
 // Налаштування стилів UI  
 private void setupSkin() {  
 skin = new Skin();  
 BitmapFont font = new BitmapFont();  
 skin.add("default-font", font);  
  
 // Стиль кнопок  
 TextButton.TextButtonStyle textButtonStyle = new TextButton.TextButtonStyle();  
 textButtonStyle.font = skin.getFont("default-font");  
 textButtonStyle.fontColor = Color.*WHITE*;  
 skin.add("default", textButtonStyle);  
  
 // Стиль написів  
 Label.LabelStyle labelStyle = new Label.LabelStyle(font, Color.*WHITE*);  
 skin.add("default", labelStyle);  
 }  
  
 // Створення всіх елементів інтерфейсу  
 private void createUI() {  
 Table rootTable = new Table(skin);  
 rootTable.setFillParent(true);  
 rootTable.defaults().pad(10);  
  
 // 1. Статус вибору  
 selectionStatusLabel = new Label("", skin);  
 rootTable.add(selectionStatusLabel).colspan(3).padTop(20).row();  
  
 // 2. Панелі попереднього перегляду команд  
 teamPreviewTableA = new Table(skin);  
 teamPreviewTableB = new Table(skin);  
  
 rootTable.add(new Label("Team A:", skin)).left().padRight(50);  
 rootTable.add(new Label("Team B:", skin)).left().row();  
  
 rootTable.add(teamPreviewTableA).top().width(Gdx.*graphics*.getWidth() / 2f);  
 rootTable.add(teamPreviewTableB).top().width(Gdx.*graphics*.getWidth() / 2f).row();  
  
 // 3. Роздільник  
 rootTable.add(new Label(isOneVsOneMode ? "--- Select 2 Fighters ---" : "--- Select Droids ---", skin)).colspan(3).padTop(40).row();  
  
 // 4. Панель для вибору доступних дроїдів (кнопки)  
 Table droidSelectionTable = new Table(skin);  
  
 // Створення кнопок для кожного типу дроїда  
 for (Map.Entry<String, Texture> entry : game.droidTextures.entrySet()) {  
 final String droidKey = entry.getKey();  
  
 ImageButton.ImageButtonStyle style = new ImageButton.ImageButtonStyle();  
 style.imageUp = new TextureRegionDrawable(entry.getValue());  
 ImageButton button = new ImageButton(style);  
  
 button.addListener(new ClickListener() {  
 @Override  
 public void clicked(InputEvent event, float x, float y) {  
 addDroidToTeam(droidKey); // Обробка натискання  
 }  
 });  
 droidSelectionTable.add(button).size(game.SMALL\_DROID\_SIZE).pad(10);  
 droidSelectionTable.add(new Label(droidKey, skin)).padRight(20);  
 }  
 rootTable.add(droidSelectionTable).colspan(3).padBottom(40).row();  
  
 // 5. Кнопка Переходу/Завершення  
 TextButton nextButton = new TextButton(isOneVsOneMode ? "Start 1 vs 1 Battle" : "Next Team / Finish Selection", skin);  
 nextButton.addListener(new ClickListener() {  
 @Override  
 public void clicked(InputEvent event, float x, float y) {  
 if (isOneVsOneMode) {  
 // Перехід до бою 1 на 1  
 if (teamA.size() == 1 && teamB.size() == 1) {  
 game.setScreen(new BattleScreen(game, teamA, teamB));  
 } else {  
 selectionStatusLabel.setText("ERROR: Select exactly one droid for each team!");  
 selectionStatusLabel.setColor(Color.*RED*);  
 }  
 } else {  
 switchTeam(); // Перехід до вибору Команди Б або до бою  
 }  
 }  
 });  
 rootTable.add(nextButton).width(300).height(60).padBottom(40).colspan(3).row();  
  
 stage.addActor(rootTable);  
 }  
  
 // Зміна команди, яку обираємо, або старт бою  
 private void switchTeam() {  
 if (selectingTeamA) {  
 // Перевірка мінімального розміру Команди А  
 if (teamA.isEmpty()) {  
 selectionStatusLabel.setText("ERROR: Team A must have at least one droid!");  
 selectionStatusLabel.setColor(Color.*RED*);  
 return;  
 }  
 selectingTeamA = false;  
 updateStatusLabel();  
 } else {  
 // Перевірка мінімального розміру Команди Б  
 if (teamB.isEmpty()) {  
 selectionStatusLabel.setText("ERROR: Team B must have at least one droid!");  
 selectionStatusLabel.setColor(Color.*RED*);  
 return;  
 }  
 // Старт бою (Команда на Команду)  
 game.setScreen(new BattleScreen(game, teamA, teamB));  
 }  
 }  
  
 // Додавання обраного дроїда до поточної команди  
 private void addDroidToTeam(String droidKey) {  
 int maxLimit = isOneVsOneMode ? 1 : MAX\_DROIDS\_PER\_TEAM\_CAP;  
  
 // Перевірка, чи не заповнені обидві команди  
 if (teamA.size() >= maxLimit && teamB.size() >= maxLimit) {  
 updateStatusLabel();  
 return;  
 }  
  
 Droid newDroid = createNewDroidInstance(droidKey); // Створення нового екземпляра  
  
 // Додавання до Команди А  
 if (selectingTeamA) {  
 if (teamA.size() < maxLimit) {  
 teamA.add(newDroid);  
 if (isOneVsOneMode) switchTeam(); // Для 1 vs 1 одразу переходимо до вибору Команди Б  
 } else {  
 if (!isOneVsOneMode) selectionStatusLabel.setText("Team A is full (max 6). Click 'Next Team'.");  
 }  
 // Додавання до Команди Б  
 } else {  
 if (teamB.size() < maxLimit) {  
 teamB.add(newDroid);  
 } else {  
 selectionStatusLabel.setText(isOneVsOneMode ? "Both teams are ready!" : "Team B is full (max 6). Click 'Finish Selection'.");  
 }  
 }  
 updateStatusLabel();  
 updateTeamPreview(); // Оновлення відображення іконок  
 }  
  
 // Оновлення відображення іконок обраних дроїдів  
 private void updateTeamPreview() {  
 teamPreviewTableA.clear();  
 teamPreviewTableB.clear();  
  
 for (Droid droid : teamA) {  
 String key = droid.getType().toLowerCase().replace("droid", "");  
 Texture texture = game.droidTextures.get(key);  
 if (texture != null) {  
 teamPreviewTableA.add(new Image(texture)).size(game.SMALL\_DROID\_SIZE / 2f).pad(2);  
 }  
 }  
 for (Droid droid : teamB) {  
 String key = droid.getType().toLowerCase().replace("droid", "");  
 Texture texture = game.droidTextures.get(key);  
 if (texture != null) {  
 teamPreviewTableB.add(new Image(texture)).size(game.SMALL\_DROID\_SIZE / 2f).pad(2);  
 }  
 }  
 }  
  
 // Оновлення тексту та кольору статусу  
 private void updateStatusLabel() {  
 String status = selectingTeamA ? "SELECTING TEAM A" : "SELECTING TEAM B";  
 int max = isOneVsOneMode ? 1 : MAX\_DROIDS\_PER\_TEAM\_CAP;  
 selectionStatusLabel.setText(status + " | A: " + teamA.size() + "/" + max + " | B: " + teamB.size() + "/" + max);  
 selectionStatusLabel.setColor(selectingTeamA ? Color.*GREEN* : Color.*BLUE*);  
 }  
  
 // Створення нового екземпляра дроїда за ключем  
 private Droid createNewDroidInstance(String droidKey) {  
 String name = droidKey + (selectingTeamA ? " A" : " B") + (selectingTeamA ? teamA.size() + 1 : teamB.size() + 1);  
  
 switch (droidKey.toLowerCase()) {  
 case "knight": return new KnightDroid(name);  
 case "assassin": return new AssassinDroid(name);  
 case "repair": return new RepairDroid(name);  
 case "mage": return new MageDroid(name);  
 case "battle": return new BattleDroid(name);  
 default: return new BattleDroid("Default Droid");  
 }  
 }  
  
 @Override public void show() { Gdx.*input*.setInputProcessor(stage); } // Встановлення обробника вводу  
 @Override public void render(float delta) {  
 // Малювання фону  
 game.batch.begin();  
 game.batch.draw(game.arenaBackground, 0, 0, Gdx.*graphics*.getWidth(), Gdx.*graphics*.getHeight());  
 game.batch.end();  
 stage.act(Gdx.*graphics*.getDeltaTime()); // Оновлення UI  
 stage.draw(); // Малювання UI  
 }  
 @Override public void dispose() { stage.dispose(); skin.dispose(); } // Звільнення ресурсів  
}

ConsoleMenuController.java

package com.droids.game.ui;  
  
import com.droids.game.battle.OneVsOneBattle;  
import com.droids.game.battle.TeamBattle;  
import com.droids.game.io.BattleFileManager;  
import com.droids.game.model.AssassinDroid;  
import com.droids.game.model.Droid;  
import com.droids.game.model.BattleDroid;  
import com.droids.game.model.RepairDroid;  
import com.droids.game.model.KnightDroid;  
import com.droids.game.model.MageDroid;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
// Клас для керування консольним інтерфейсом гри  
public class ConsoleMenuController {  
  
 private final List<Droid> allDroids = new ArrayList<>(); // Список усіх створених дроїдів  
 private final Scanner scanner = new Scanner(System.*in*);  
 private List<String> lastBattleLog = new ArrayList<>(); // Зберігає лог останнього бою  
   
 //ГОЛОВНИЙ ЦИКЛ МЕНЮ  
  
 public void start() {  
 // Додаємо стартових дроїдів  
 allDroids.add(new BattleDroid("Iron-Bender-1"));  
 allDroids.add(new RepairDroid("Heal-Bot-A"));  
  
 int choice;  
 do {  
 printMenu();  
 if (scanner.hasNextInt()) {  
 choice = scanner.nextInt();  
 scanner.nextLine(); // Очищення буфера  
 handleChoice(choice);  
 } else {  
 System.*out*.println("❌ Некоректний ввід. Спробуйте ще раз.");  
 scanner.nextLine();  
 choice = 0;  
 }  
 } while (choice != 7);  
 }  
  
 private void printMenu() {  
 System.*out*.println("\n--- 🤖 Меню Гри DroidBattle ---");  
 System.*out*.println("1. Створити дроїда");  
 System.*out*.println("2. Показати список створених дроїдів");  
 System.*out*.println("3. Запустити бій 1 на 1");  
 System.*out*.println("4. Запустити бій команда на команду");  
 System.*out*.println("5. Записати проведений бій у файл");  
 System.*out*.println("6. Відтворити проведений бій зі збереженого файлу");  
 System.*out*.println("7. Вийти з програми");  
 System.*out*.print("▶️ Виберіть опцію: ");  
 }  
  
 // Обробка вибору в меню  
 private void handleChoice(int choice) {  
 switch (choice) {  
 case 1: createDroid(); break;  
 case 2: showDroids(); break;  
 case 3: startOneVsOneBattle(); break;  
 case 4: startTeamBattle(); break;  
 case 5: saveBattleLog(); break;  
 case 6: replayBattle(); break;  
 case 7: System.*out*.println("👋 До побачення!"); break;  
 default: System.*out*.println("⚠️ Невідома команда.");  
 }  
 }  
   
 // Функція 1: Створення дроїда  
 private void createDroid() {  
 System.*out*.println("\n--- Створення Дроїда ---");  
 // [Опис типів]  
 System.*out*.println("Виберіть тип дроїда:");  
 System.*out*.println("1. BattleDroid | HP: 150, DMG: 20 | Унікальна здібність: ARMOR = 15");  
 System.*out*.println("2. RepairDroid | HP: 80, DMG: 5 | Унікальна здібність: HEAL = 15");  
 System.*out*.println("3. KnightDroid | HP: 120, DMG: 25 | Унікальна здібність: ARMOR = 5");  
 System.*out*.println("4. AssassinDroid| HP: 90, DMG: 40 | Унікальна здібність: CRIT = 50%");  
 System.*out*.println("5. MageDroid | HP: 95, DMG: 20 | Унікальна здібність: EVASION = 35%");  
 System.*out*.print("Ввід: ");  
  
 int typeChoice = scanner.nextInt();  
 scanner.nextLine();  
  
 System.*out*.print("Введіть ім'я дроїда: ");  
 String name = scanner.nextLine();  
  
 Droid newDroid = null;  
 // Створення екземпляра відповідно до вибору  
 if (typeChoice == 1) {  
 newDroid = new BattleDroid(name);  
 } else if (typeChoice == 2) {  
 newDroid = new RepairDroid(name);  
 } else if (typeChoice == 3) {  
 newDroid = new KnightDroid(name);  
 } else if (typeChoice == 4) {  
 newDroid = new AssassinDroid(name);  
 } else if (typeChoice == 5) {  
 newDroid = new MageDroid(name);  
 } else {  
 System.*out*.println("Невідомий тип дроїда.");  
 return;  
 }  
  
 allDroids.add(newDroid);  
 System.*out*.println("✅ Створено: " + newDroid.toString());  
 }  
  
 // Функція 2: Показати список  
 private void showDroids() {  
 if (allDroids.isEmpty()) {  
 System.*out*.println("🤷 Список дроїдів порожній.");  
 return;  
 }  
 System.*out*.println("\n--- Список Дроїдів ---");  
 for (int i = 0; i < allDroids.size(); i++) {  
 System.*out*.printf("%d. %s\n", i + 1, allDroids.get(i).toString());  
 }  
 }  
  
 // Функція 3: Бій 1 на 1  
 private void startOneVsOneBattle() {  
 if (allDroids.size() < 2) {  
 System.*out*.println("⚠️ Потрібно мінімум два дроїда для бою.");  
 return;  
 }  
 showDroids();  
 System.*out*.print("Виберіть номер першого дроїда: ");  
 int index1 = scanner.nextInt() - 1;  
 System.*out*.print("Виберіть номер другого дроїда: ");  
 int index2 = scanner.nextInt() - 1;  
 scanner.nextLine();  
  
 if (index1 >= 0 && index1 < allDroids.size() && index2 >= 0 && index2 < allDroids.size() && index1 != index2) {  
 // Створюємо копії дроїдів для бою, щоб не змінювати оригінали  
 Droid original1 = allDroids.get(index1);  
 Droid original2 = allDroids.get(index2);  
 Droid fighter1 = createBattleCopy(original1, " (Боєць А)");  
 Droid fighter2 = createBattleCopy(original2, " (Боєць Б)");  
  
 OneVsOneBattle battle = new OneVsOneBattle();  
 lastBattleLog = battle.startBattle(fighter1, fighter2); // Запуск і збереження логу  
  
 // Вивід логу на консоль  
 lastBattleLog.forEach(System.*out*::println);  
 } else {  
 System.*out*.println("Некоректний вибір.");  
 }  
 }  
   
 private Droid createBattleCopy(Droid original, String suffix) {  
 String newName = original.getName() + suffix;  
  
 // Використовуємо instanceof для створення правильного підтипу  
 if (original instanceof BattleDroid) {  
 return new BattleDroid(newName);  
 } else if (original instanceof RepairDroid) {  
 return new RepairDroid(newName);  
 } else if (original instanceof KnightDroid) {  
 return new KnightDroid(newName);  
 } else if (original instanceof AssassinDroid) {  
 return new AssassinDroid(newName);  
 } else if (original instanceof MageDroid) {  
 return new MageDroid(newName);  
 } else {  
 // Fallback для базового Droid  
 return new Droid(newName, original.getMaxHealth(), original.getDamage());  
 }  
 }  
   
 private List<Droid> selectTeam(String teamName) {  
 List<Droid> teamDroids = new ArrayList<>();  
 System.*out*.printf("\n--- Формування %s ---\n", teamName);  
  
 while (true) {  
 showDroids(); // Показ списку  
 System.*out*.printf("▶️ Виберіть номер дроїда для додавання до %s (або 0, щоб завершити вибір): ", teamName);  
  
 if (scanner.hasNextInt()) {  
 int selection = scanner.nextInt();  
 scanner.nextLine();  
  
 if (selection == 0) {  
 if (teamDroids.isEmpty()) {  
 System.*out*.println("⚠️ Команда не може бути порожньою. Будь ласка, оберіть хоча б одного дроїда.");  
 continue;  
 }  
 break;  
 }  
  
 int index = selection - 1;  
 if (index >= 0 && index < allDroids.size()) {  
 Droid originalDroid = allDroids.get(index);  
 Droid fighterCopy = createBattleCopy(originalDroid, " (" + teamName + ")"); // Створення копії  
 teamDroids.add(fighterCopy);  
 System.*out*.printf("✅ Додано до %s: %s\n", teamName, fighterCopy.getName());  
 } else {  
 System.*out*.println("Некоректний номер дроїда. Спробуйте ще раз.");  
 }  
 } else {  
 System.*out*.println("Некоректний ввід. Будь ласка, введіть число.");  
 scanner.nextLine();  
 }  
 }  
 return teamDroids;  
 }  
  
 // Функція 4: Бій Команда на Команду  
 private void startTeamBattle() {  
 if (allDroids.size() < 2) {  
 System.*out*.println("⚠️ Потрібно створити мінімум два дроїда для командного бою.");  
 return;  
 }  
  
 List<Droid> teamA = selectTeam("Команда А"); // Вибір Команди А  
 List<Droid> teamB = selectTeam("Команда Б"); // Вибір Команди Б  
  
 if (teamA.isEmpty() || teamB.isEmpty()) {  
 System.*out*.println("⚠️ Бій не може розпочатися, якщо одна з команд порожня.");  
 return;  
 }  
  
 // Запуск командного бою  
 TeamBattle battle = new TeamBattle();  
 lastBattleLog = battle.startBattle(teamA, teamB);  
  
 lastBattleLog.forEach(System.*out*::println);  
 }  
  
 // Функція 5: Збереження логу  
 private void saveBattleLog() {  
 if (lastBattleLog.isEmpty()) {  
 System.*out*.println("⚠️ Спочатку проведіть бій, щоб його записати.");  
 return;  
 }  
 System.*out*.print("Введіть назву файлу для збереження (наприклад, 'battle.txt'): ");  
 String filename = scanner.nextLine();  
  
 BattleFileManager.*saveLogToFile*(lastBattleLog, filename); // Виклик функції збереження  
 }  
  
 // Функція 6: Відтворення логу  
 private void replayBattle() {  
 System.*out*.print("Введіть назву файлу для відтворення: ");  
 String filename = scanner.nextLine();  
  
 BattleFileManager.*replayBattle*(filename); // Виклик функції відтворення  
 }  
}

Main.java

package com.droids.game;  
  
import com.droids.game.ui.ConsoleMenuController;  
  
public class Main {  
 public static void main(String[] args) {  
 // Створюємо та запускаємо контролер меню  
 ConsoleMenuController menu = new ConsoleMenuController();  
 menu.start();  
 }  
  
}

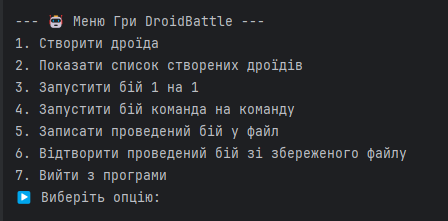
MainGame.java

// com.droids.game.MainGame (ОНОВЛЕНО)  
  
package com.droids.game;  
  
import com.badlogic.gdx.Game; // Основний клас LibGDX для керування екранами  
import com.badlogic.gdx.graphics.Texture;  
import com.badlogic.gdx.graphics.g2d.SpriteBatch;  
import com.droids.game.screen.MenuScreen;  
import java.util.HashMap;  
import java.util.Map;  
  
  
public class MainGame extends Game { // Спадкування від Game дозволяє керувати екранами  
  
 // Зроблено публічними, щоб екрани мали прямий доступ до ресурсів  
 public SpriteBatch batch; // Об'єкт для малювання 2D графіки  
 public Map<String, Texture> droidTextures; // Карта для зберігання текстур дроїдів  
 public Texture arenaBackground; // Текстура фону  
  
 // Константи для розмірів спрайтів  
 public final float DROID\_SIZE = 200f;  
 public final float SMALL\_DROID\_SIZE = 150f;  
  
 @Override  
 public void create() {  
 batch = new SpriteBatch();  
 droidTextures = new HashMap<>();  
  
 // --- 1. Завантаження всіх ресурсів ---  
 arenaBackground = new Texture("arena.png");  
 // Завантаження текстур дроїдів за ключем (типом)  
 droidTextures.put("assassin", new Texture("assassin\_droid.png"));  
 droidTextures.put("battle", new Texture("battle\_droid.png"));  
 droidTextures.put("repair", new Texture("repair\_droid.png"));  
 droidTextures.put("knight", new Texture("knight\_droid.png"));  
 droidTextures.put("mage", new Texture("mage\_droid.png"));  
  
 // --- 2. Встановлюємо перший екран ---  
 setScreen(new MenuScreen(this)); // Запускаємо гру з MenuScreen  
 }  
  
 @Override  
 public void render() {  
 // Делегує малювання активному екрану  
 super.render();  
 }  
  
 @Override  
 public void dispose() {  
 super.dispose();  
 batch.dispose(); // Звільнення ресурсів SpriteBatch  
 arenaBackground.dispose(); // Звільнення текстури фону  
  
 // Звільнення всіх завантажених текстур дроїдів  
 for (Texture texture : droidTextures.values()) {  
 texture.dispose();  
 }  
 }  
}

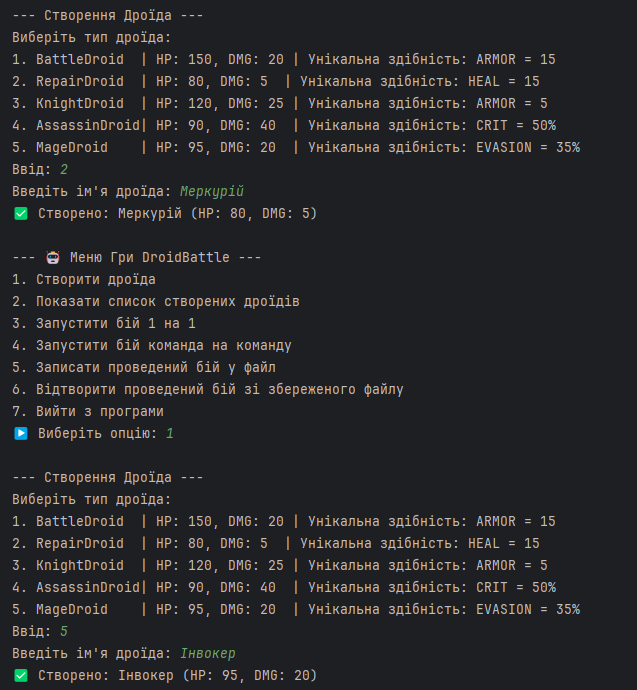
Lwjgl3Launcher.java

package com.droids.game.lwjgl3;  
  
import com.badlogic.gdx.backends.lwjgl3.Lwjgl3Application;  
import com.badlogic.gdx.backends.lwjgl3.Lwjgl3ApplicationConfiguration;  
import com.droids.game.MainGame;  
  
*/\*\* Launches the desktop (LWJGL3) application. \*/*public class Lwjgl3Launcher {  
 public static void main(String[] args) {  
 if (StartupHelper.*startNewJvmIfRequired*()) return; // This handles macOS support and helps on Windows.  
 *createApplication*();  
 }  
  
 private static Lwjgl3Application createApplication() {  
 return new Lwjgl3Application(new MainGame(), *getDefaultConfiguration*());  
 }  
  
 private static Lwjgl3ApplicationConfiguration getDefaultConfiguration() {  
 Lwjgl3ApplicationConfiguration configuration = new Lwjgl3ApplicationConfiguration();  
 configuration.setTitle("DroidBattle1");  
 *//// Vsync limits the frames per second to what your hardware can display, and helps eliminate  
 //// screen tearing. This setting doesn't always work on Linux, so the line after is a safeguard.* configuration.useVsync(true);  
 *//// Limits FPS to the refresh rate of the currently active monitor, plus 1 to try to match fractional  
 //// refresh rates. The Vsync setting above should limit the actual FPS to match the monitor.* configuration.setForegroundFPS(Lwjgl3ApplicationConfiguration.*getDisplayMode*().refreshRate + 1);  
 *//// If you remove the above line and set Vsync to false, you can get unlimited FPS, which can be  
 //// useful for testing performance, but can also be very stressful to some hardware.  
 //// You may also need to configure GPU drivers to fully disable Vsync; this can cause screen tearing.* configuration.setWindowedMode(1366, 756);  
 *//// You can change these files; they are in lwjgl3/src/main/resources/ .  
 //// They can also be loaded from the root of assets/ .* configuration.setWindowIcon("libgdx128.png", "libgdx64.png", "libgdx32.png", "libgdx16.png");  
  
 *//// This should improve compatibility with Windows machines with buggy OpenGL drivers, Macs  
 //// with Apple Silicon that have to emulate compatibility with OpenGL anyway, and more.  
 //// This uses the dependency `com.badlogicgames.gdx:gdx-lwjgl3-angle` to function.  
 //// You can choose to remove the following line and the mentioned dependency if you want; they  
 //// are not intended for games that use GL30 (which is compatibility with OpenGL ES 3.0).* configuration.setOpenGLEmulation(Lwjgl3ApplicationConfiguration.GLEmulation.*ANGLE\_GLES20*, 0, 0);  
  
 return configuration;  
 }  
}

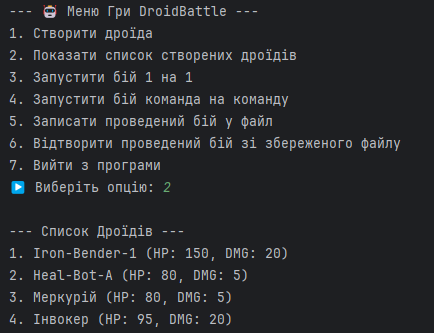
Результати виконання:



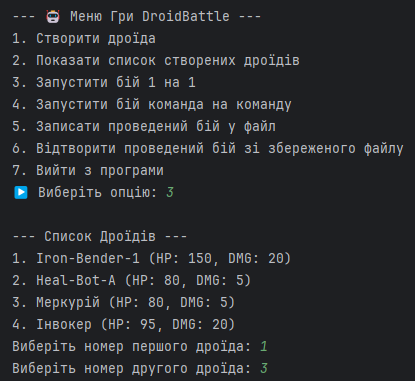
1)Створення

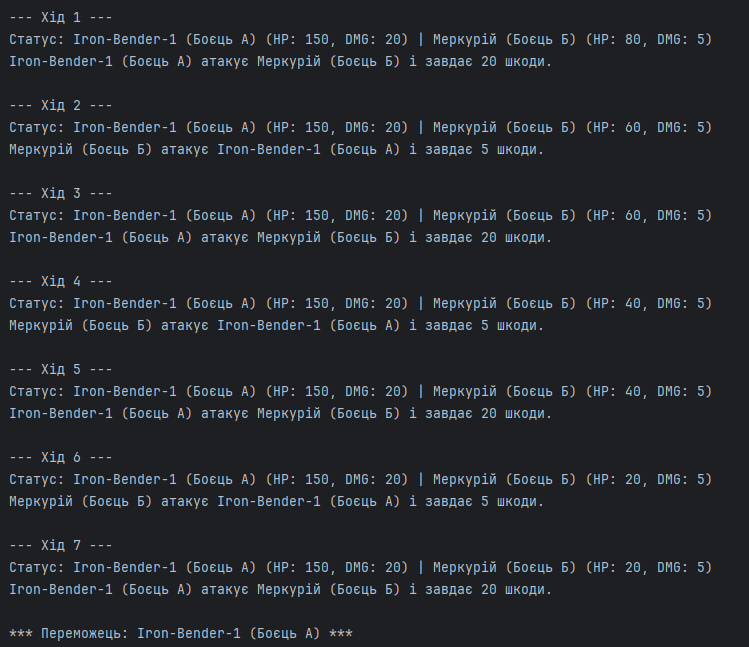


2)Список всіх

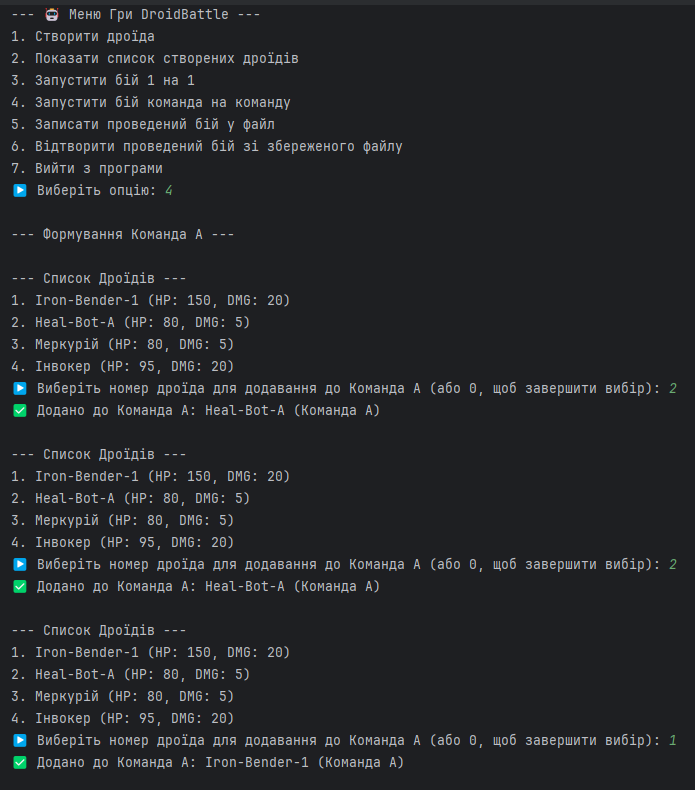


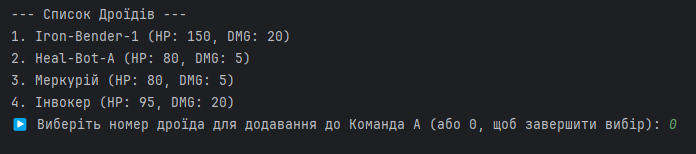
3)Бій 1 на 1

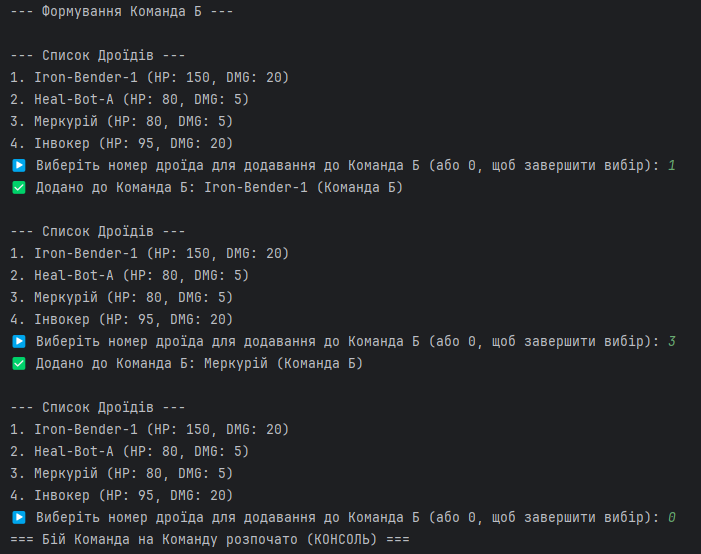


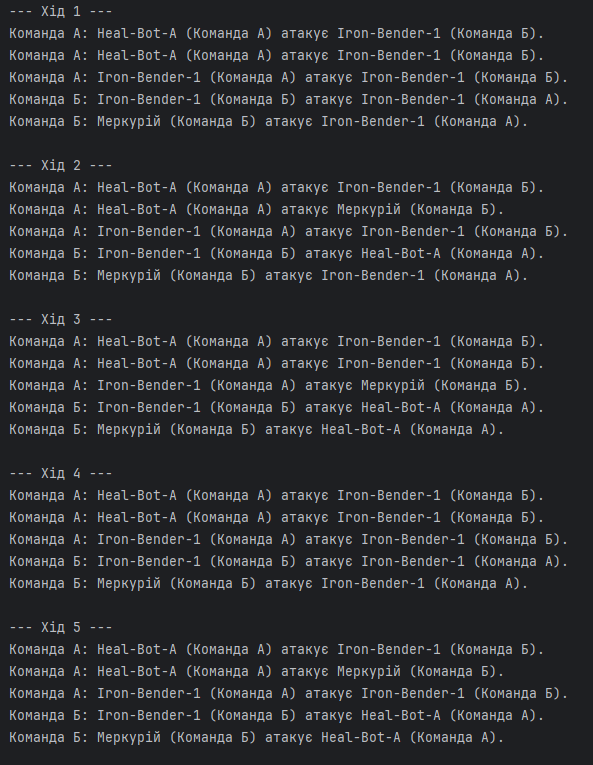


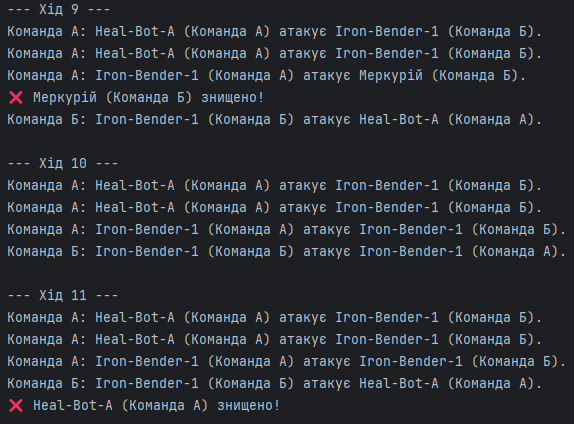
4) Командний бій

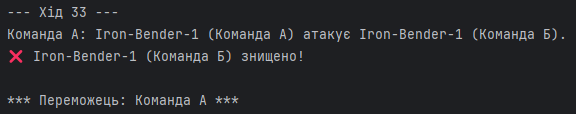




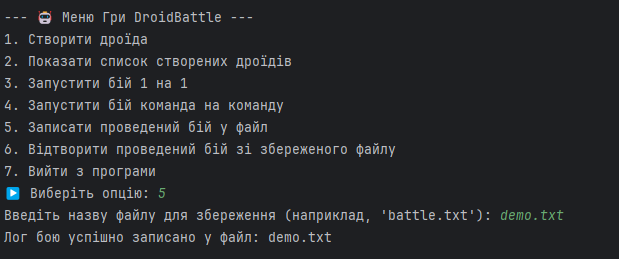


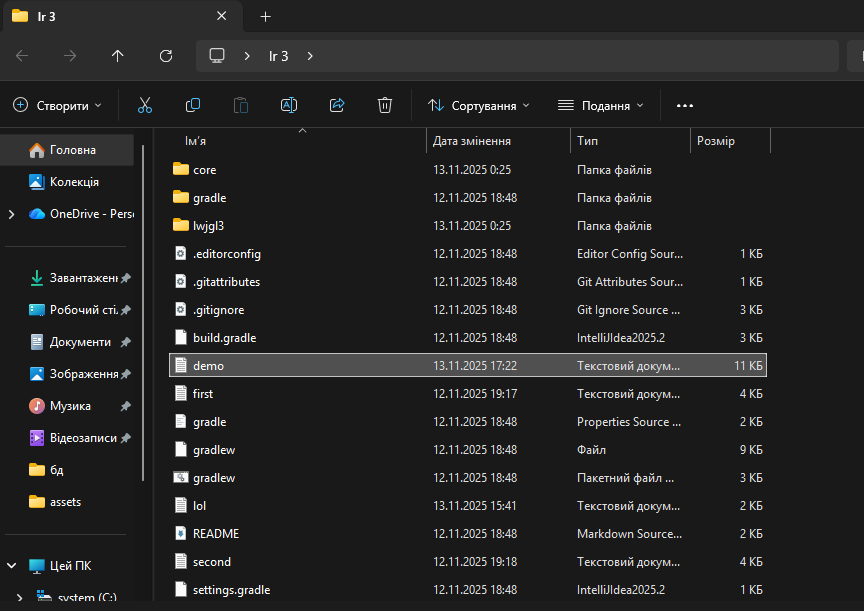




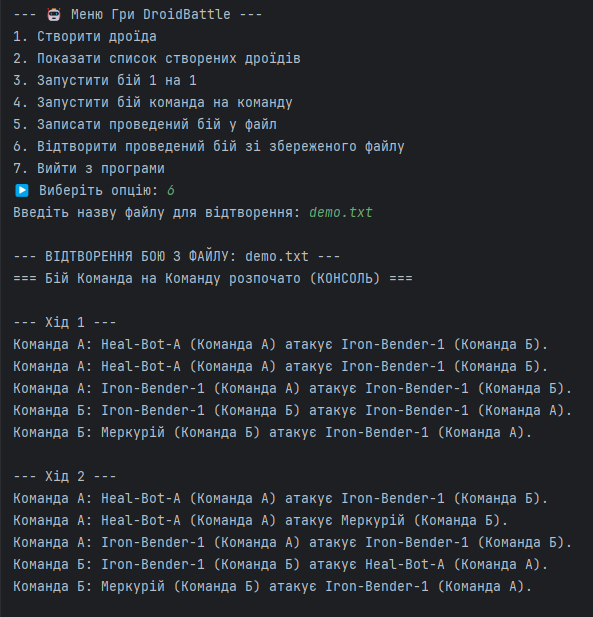


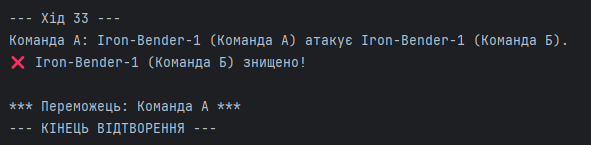
5) Запис бою у файл



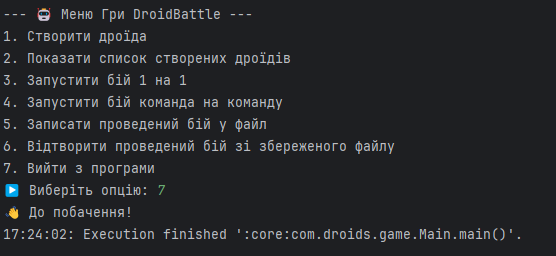


6) Вивід збереженого бою





7) Вийти з програми



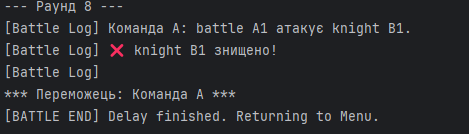
8) Графічне меню



9) Бій 1 на 1



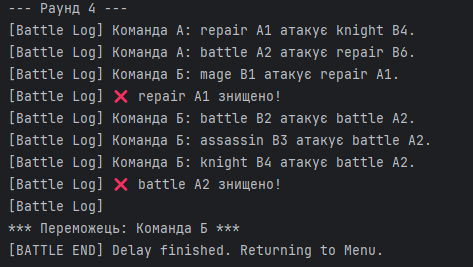




10) Бій команда на команда







Висновок

Програма **успішно та повноцінно реалізує всі мінімальні вимоги** лабораторної роботи №3. Створено чітку ієрархію класів дроїдів, забезпечено коректне рознесення коду по пакетах. Повний функціонал **консольного меню** (ConsoleMenuController), включаючи створення дроїдів, запуск **двох режимів бою** (1 на 1 та команда на команду), а також **запис та відтворення логу бою у файл** (BattleFileManager), реалізовано коректно.

**Спеціалізована Ієрархія Дроїдів**:

* Створено **6 унікальних класів-підтипів** (BattleDroid, KnightDroid, AssassinDroid, MageDroid, RepairDroid), кожен з яких має унікальну бойову механіку: **броню**, **критичні удари**, **ухилення** та **лікування** союзників.

**Складність Бою**:

* Додано **випадкові елементи** (шанс криту та ухилення) та **стратегію лікування** у командному бою, що робить результат непередбачуваним.

**Інтеграція LibGDX (GUI)**:

* Проєкт розширено повноцінним **графічним інтерфейсом** (LibGDX) через класи MainGame, MenuScreen, TeamSelectionScreen та BattleScreen. Це дозволило реалізувати **покрокову візуалізацію бою** з HP-барами та анімацією, замінивши просту консольну візуалізацію.

**Командний Бій**:

* Реалізовано клас **TeamBattle**, який використовує обидва режими бою (миттєвий для консолі та покроковий для GUI) і включає логіку атаки та лікування.