Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління

Зображення, що містить текст

Автоматично згенерований опис

**Звiт**

до лабораторної роботи № 3

з дисципліни

# “ Прикладне програмування”

Виконав: студент групи ОІ-11сп

**Москалюк Назарій**

Прийняв: Мельник Р. В.

**Лабораторна робота № 3. Гра «Битва дроїдів»**

Напишіть свою реалізацію консольної гри “Битва дроїдів”. Лабораторна робота розрахована на 3 заняття.

**Мінімальний набір вимог до програми:**

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.

Виконання завдання

Menu.java

package labThree;  
import components.BattleSystems;  
import units.\*;  
import java.util.ArrayList;  
import java.util.Scanner;  
public class Menu {  
  
 public static Droid choseDroidForDuel(ArrayList<ArrayList<Droid>>teams){  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.printf("Enter team number to chose the droid: ");  
 int enteredMenuCommand2 = Integer.*valueOf*(scanner.nextLine());  
 ArrayList<Droid> firstTeamForDuel = teams.get(enteredMenuCommand2-1);  
 System.*out*.println("Picked team: \n" + firstTeamForDuel);  
  
 System.*out*.printf("Enter droid number to chose the droid: ");  
 enteredMenuCommand2 = Integer.*valueOf*(scanner.nextLine());  
 Droid DroidForDuel = firstTeamForDuel.get(enteredMenuCommand2-1);  
 return DroidForDuel;  
 }  
  
 public static ArrayList<Droid> choseTeamFroBattle(ArrayList<ArrayList<Droid>>teams){  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.printf("Enter team number to chose: ");  
 int enteredMenuCommand2 = Integer.*valueOf*(scanner.nextLine());  
 ArrayList<Droid> teamToReturn = teams.get(enteredMenuCommand2-1);  
 return teamToReturn;  
 }  
  
 public static void main(String[] args) {  
  
 ArrayList<Droid> defaultTeamRed = new ArrayList<>();  
 defaultTeamRed.add(new Assasin("r2d2", 100, 10, 0, 0));  
 defaultTeamRed.add(new Droid("r32d25", 70, 15, "Droid", 0, 0));  
 defaultTeamRed.add(new Droid("rss", 200, 5, "Droid", 0, 0));  
  
 ArrayList<Droid> defaultTeamBlue = new ArrayList<>();  
 defaultTeamBlue.add(new Tank("r2111d2", 88, 8, 10));  
 defaultTeamBlue.add(new Droid("dds22", 71, 5, "Droid", 0, 0));  
 defaultTeamBlue.add(new Droid("2ds25", 210, 6, "Droid", 0, 0));  
  
 ArrayList<ArrayList<Droid>> teams = new ArrayList<>();  
 teams.add(defaultTeamRed);  
 teams.add(defaultTeamBlue);  
  
  
  
  
 System.*out*.print("\t\tDroid battle\t\t\n");  
 System.*out*.println("\nTo see all commands type /help");  
 System.*out*.println("To exit the program type /exit");  
 String enteredMenuCommand = "";  
 Scanner scanner = new Scanner(System.*in*);  
 do{  
 System.*out*.print("Enter command: ");  
 enteredMenuCommand = scanner.nextLine();  
 switch (enteredMenuCommand){  
 case "/exit":  
 System.*exit*(0);  
 case "/help":  
 System.*out*.println("\nTo see all commands type /help");  
 System.*out*.println("To see all teams with droids type /teams");  
 System.*out*.println("To begin a duel battle between droids type /duel");  
 System.*out*.println("To begin a team battle between droids type /3v3");  
 System.*out*.println("To create a new team type /createTeam (3 droids)");  
 System.*out*.println("To exit the program type /exit");  
 break;  
 case "/teams":  
 for (int i = 0; i < teams.size(); i++) {  
 System.*out*.println("Team №" + (i+1) + teams.get(i));  
 }  
 break;  
 case "/duel":  
 Droid firstDroid = *choseDroidForDuel*(teams);  
 Droid secondDroid = *choseDroidForDuel*(teams);  
 BattleSystems.*duelBattle*(firstDroid, secondDroid);  
 break;  
 case "/3v3":  
 System.*out*.println("First chosen team will be set as team Blue, second chosen as team Red");  
 ArrayList<Droid> firstTeam = *choseTeamFroBattle*(teams);  
 ArrayList<Droid> secondTeam = *choseTeamFroBattle*(teams);  
  
 BattleSystems.*threevsthree*(firstTeam, secondTeam);  
 break;  
 case "/createTeam":  
 ArrayList<Droid> teamToCreate = new ArrayList<>();  
 for (int i = 0; i < 3; i++) {  
 System.*out*.println("Please, chose what type your first droid will be created\n" +  
 "\tType 1 if: Assasin (Special Ability: 'runs fast behind enemy's back \nand stabs them 3 times')\n" +  
 "\tType 2 if: Tank (Special characteristic: has 0% evation rate \nSpecial Ability: 'increases it's outer defence layer \nand becomes twice armored \nbut loses half of atack power')\n" +  
 "\tType 3 if: Summoner (Special Ability: summons companion that atacks enemy with half atack power')\n)");  
 System.*out*.print("Enter command: ");  
 enteredMenuCommand = scanner.nextLine();  
 String droidName;  
 int health;  
 int damage;  
 int rate;  
 int armorValue;  
 switch (enteredMenuCommand){  
 case "1":  
 System.*out*.println("\tCreating new Assasin\n");  
 System.*out*.println("Enter droid's name: ");  
 droidName = scanner.nextLine();  
 System.*out*.println("Enter droid's health: ");  
 health = Integer.*valueOf*(scanner.nextLine());  
 System.*out*.println("Enter droid's atack damage: ");  
 damage = Integer.*valueOf*(scanner.nextLine());  
 System.*out*.println("Enter droid's evation rate: ");  
 rate = Integer.*valueOf*(scanner.nextLine());  
 System.*out*.println("Enter droid's armor value: ");  
 armorValue = Integer.*valueOf*(scanner.nextLine());  
  
 teamToCreate.add(new Assasin(droidName, health, damage, rate, armorValue));  
  
 break;  
 case "2":  
 System.*out*.println("\tCreating new Tank\n");  
 System.*out*.println("Enter droid's name: ");  
 droidName = scanner.nextLine();  
 System.*out*.println("Enter droid's health: ");  
 health = Integer.*valueOf*(scanner.nextLine());  
 System.*out*.println("Enter droid's atack damage: ");  
 damage = Integer.*valueOf*(scanner.nextLine());  
 System.*out*.println("Enter droid's armor value: ");  
 armorValue = Integer.*valueOf*(scanner.nextLine());  
  
 teamToCreate.add(new Tank(droidName, health, damage, armorValue));  
  
 break;  
 case "3":  
 System.*out*.println("\tCreating new Summoner\n");  
 System.*out*.println("Enter droid's name: ");  
 droidName = scanner.nextLine();  
 System.*out*.println("Enter droid's health: ");  
 health = Integer.*valueOf*(scanner.nextLine());  
 System.*out*.println("Enter droid's atack damage: ");  
 damage = Integer.*valueOf*(scanner.nextLine());  
 System.*out*.println("Enter droid's evation rate: ");  
 rate = Integer.*valueOf*(scanner.nextLine());  
 System.*out*.println("Enter droid's armor value: ");  
 armorValue = Integer.*valueOf*(scanner.nextLine());  
  
 teamToCreate.add(new Summoner(droidName, health, damage, rate, armorValue));  
  
 break;  
  
 }  
  
   
 }  
 teams.add(teamToCreate);  
 break;  
  
 }  
 }while(enteredMenuCommand != "/exit");  
  
  
  
 }  
  
}

Droid.java

package units;  
import java.util.Objects;  
import java.util.Random;  
  
*/\*\*  
 \* The Droid class represents a combat droid with various attributes like health, attack damage,  
 \* evasion rate, and armor. It also includes methods for attacking enemies and using special abilities.  
 \*/*public class Droid {  
 protected String name;  
 protected int healthPoints;  
 protected int atackDamage;  
 protected String droidType;  
 protected int evationRate;  
 protected int armorValue;  
 protected boolean specialAbilityIsUsed = false;  
  
 static Random *rand* = new Random();  
  
 */\*\*  
 \* Default constructor that initializes a droid with default values.  
 \*/* public Droid() {  
 this.name = "Nameless droid";  
 this.healthPoints = 100;  
 this.atackDamage = 10;  
 this.droidType = "Droid";  
 this.evationRate = 0;  
 this.armorValue = 0;  
 }  
  
 */\*\*  
 \* Parametrized constructor that initializes a droid with specified values.  
 \*  
 \* @param name the name of the droid  
 \* @param healthPoints the health points of the droid  
 \* @param atackDamage the attack damage of the droid  
 \* @param droidType the type of the droid  
 \* @param evationRate the evasion rate of the droid  
 \* @param armorValue the armor value of the droid  
 \*/* public Droid(String name, int healthPoints, int atackDamage, String droidType, int evationRate, int armorValue) {  
 this.name = name;  
 this.healthPoints = healthPoints;  
 this.atackDamage = atackDamage;  
 this.droidType = droidType;  
 this.evationRate = evationRate;  
 this.armorValue = armorValue;  
 }  
  
 */\*\*  
 \* @return the name of the droid  
 \*/* public String getName() {  
 return name;  
 }  
  
 */\*\*  
 \* Sets the name of the droid.  
 \*  
 \* @param name the new name of the droid  
 \*/* public void setName(String name) {  
 this.name = name;  
 }  
  
 */\*\*  
 \* @return the health points of the droid  
 \*/* public int getHealthPoints() {  
 return healthPoints;  
 }  
  
 */\*\*  
 \* Sets the health points of the droid.  
 \*  
 \* @param healthPoints the new health points of the droid  
 \*/* public void setHealthPoints(int healthPoints) {  
 this.healthPoints = healthPoints;  
 }  
  
 */\*\*  
 \* @return the attack damage of the droid  
 \*/* public int getAtackDamage() {  
 return atackDamage;  
 }  
  
 */\*\*  
 \* Sets the attack damage of the droid.  
 \*  
 \* @param atackDamage the new attack damage of the droid  
 \*/* public void setAtackDamage(int atackDamage) {  
 this.atackDamage = atackDamage;  
 }  
  
 */\*\*  
 \* @return the type of the droid  
 \*/* public String getDroidType() {  
 return droidType;  
 }  
  
 */\*\*  
 \* Sets the type of the droid.  
 \*  
 \* @param droidType the new type of the droid  
 \*/* public void setDroidType(String droidType) {  
 this.droidType = droidType;  
 }  
  
 */\*\*  
 \* @return the evasion rate of the droid  
 \*/* public int getEvationRate() {  
 return evationRate;  
 }  
  
 */\*\*  
 \* Sets the evasion rate of the droid.  
 \*  
 \* @param evationRate the new evasion rate of the droid  
 \*/* public void setEvationRate(int evationRate) {  
 this.evationRate = evationRate;  
 }  
  
 */\*\*  
 \* @return the armor value of the droid  
 \*/* public int getArmorValue() {  
 return armorValue;  
 }  
  
 */\*\*  
 \* Sets the armor value of the droid.  
 \*  
 \* @param armorValue the new armor value of the droid  
 \*/* public void setArmorValue(int armorValue) {  
 this.armorValue = armorValue;  
 }  
  
 */\*\*  
 \* @return the random object used for attack calculations  
 \*/* public static Random getRand() {  
 return *rand*;  
 }  
  
 */\*\*  
 \* Sets the random object for attack calculations.  
 \*  
 \* @param rand the new random object  
 \*/* public static void setRand(Random rand) {  
 Droid.*rand* = rand;  
 }  
  
 */\*\*  
 \* @return true if the droid's special ability has been used  
 \*/* public boolean isSpecialAbilityIsUsed() {  
 return this.specialAbilityIsUsed;  
 }  
  
 */\*\*  
 \* Checks if the conditions for using the special ability are met, based on the current round.  
 \*  
 \* @param round the current round  
 \* @return true if the special ability can be used, false otherwise  
 \*/* public boolean checkConditionsForSpecialAbilityUse(int round){  
 if (round == 10 || round == 11){ return true; }  
 return false;  
 }  
  
 */\*\*  
 \* Sets whether the droid's special ability has been used.  
 \*  
 \* @param specialAbilityIsUsed the new special ability usage state  
 \*/* public void setSpecialAbilityIsUsed(boolean specialAbilityIsUsed) {  
 this.specialAbilityIsUsed = specialAbilityIsUsed;  
 }  
  
 */\*\*  
 \* Returns a string representation of the droid's attributes.  
 \*  
 \* @return a string containing the droid's attributes  
 \*/* @Override  
 public String toString() {  
 return "Droid{" +  
 "name='" + name + '\'' +  
 ", healthPoints=" + healthPoints +  
 ", atackDamage=" + atackDamage +  
 ", droidType=" + droidType +  
 ", evationRate=" + evationRate +  
 ", armorValue=" + armorValue +  
 '}';  
 }  
  
 */\*\*  
 \* Checks if two droids are equal based on their attributes.  
 \*  
 \* @param o the object to compare to  
 \* @return true if the droids are equal, false otherwise  
 \*/* @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Droid droid = (Droid) o;  
 return healthPoints == droid.healthPoints && atackDamage == droid.atackDamage && evationRate == droid.evationRate && armorValue == droid.armorValue && Objects.*equals*(name, droid.name) && Objects.*equals*(droidType, droid.droidType);  
 }  
  
 */\*\*  
 \* Generates a hash code for the droid based on its attributes.  
 \*  
 \* @return the hash code of the droid  
 \*/* @Override  
 public int hashCode() {  
 return Objects.*hash*(name, healthPoints, atackDamage, droidType, evationRate, armorValue);  
 }  
  
 */\*\*  
 \* Calculates the damage affected by the enemy droid's armor value.  
 \*  
 \* @param damageGiven the initial damage  
 \* @param enemyDroidArmorValue the armor value of the enemy droid  
 \* @return the final damage after armor reduction  
 \*/* protected static int calculateDamageAffectedByArmor(int damageGiven, int enemyDroidArmorValue) {  
 if (enemyDroidArmorValue != 0) {  
 *// Calculate the effective damage after applying armor reduction.* double damageMultiplier = 1 - (enemyDroidArmorValue / 100.0);  
 return (int) Math.*round*(damageGiven \* damageMultiplier);  
 } else {  
 return damageGiven;  
 }  
 }  
  
 */\*\*  
 \* Executes a special ability on an enemy droid. To be overridden in subclasses.  
 \*  
 \* @param enemyDroid the enemy droid  
 \*/* protected void specialAbility(Droid enemyDroid){ }  
  
 */\*\*  
 \* Sets the special ability as used.  
 \*/* public void useSpecialAbility(){ this.specialAbilityIsUsed = true; }  
  
 */\*\*  
 \* Attacks an enemy droid and calculates the damage based on the attack damage, critical hit, and armor.  
 \*  
 \* @param enemyDroid the enemy droid to attack  
 \* @param outputLogs whether to output log messages about the attack  
 \* @return the amount of damage dealt to the enemy droid  
 \*/* public int atackEnemy(Droid enemyDroid, boolean outputLogs) {  
  
 *// regular damage* int damageGiven = *rand*.nextInt(this.getAtackDamage());  
  
 *// calculates crit damage and chance  
 // crit chance* if (*rand*.nextInt(1, 100) <= enemyDroid.getEvationRate()) {  
 *// crit damage* damageGiven \*= 2;  
 damageGiven = Droid.*calculateDamageAffectedByArmor*(damageGiven, enemyDroid.getArmorValue());  
 *// log output* if (outputLogs) {  
 System.*out*.println("Droid " + this.getName() +" deals crit damage "+ damageGiven +" to the enemy: "+enemyDroid.getName());  
 }  
 } else if (outputLogs) {  
 damageGiven = Droid.*calculateDamageAffectedByArmor*(damageGiven, enemyDroid.getArmorValue());  
 System.*out*.println("Droid " + this.getName() + " deals " + damageGiven + " damage to the droid: " + enemyDroid.getName());  
 }  
 enemyDroid.setHealthPoints( enemyDroid.getHealthPoints() - damageGiven);  
 return damageGiven;  
 }  
  
 */\*\*  
 \* Checks if the droid is still alive based on its health points.  
 \*  
 \* @return true if the droid is alive, false otherwise  
 \*/* public boolean isAlive(){ return this.getHealthPoints() > 0; }  
}

Assasin.java

package units;  
  
*/\*\*  
 \* The Assasin class represents a specialized type of Droid with unique attributes and abilities.  
 \* This droid has the ability to attack its enemy multiple times in a row using its special ability.  
 \*/*public class Assasin extends Droid {  
  
 */\*\*  
 \* Default constructor that initializes the Assasin droid with predefined attributes.  
 \*/* public Assasin() {  
 name = "Assasin";  
 healthPoints = 70;  
 atackDamage = 10;  
 droidType = "Assasin";  
 evationRate = 0;  
 armorValue = 10;  
 specialAbilityIsUsed = false;  
 }  
  
 */\*\*  
 \* Parametrized constructor that initializes the Assasin droid with custom attributes.  
 \*  
 \* @param name the name of the droid  
 \* @param health the health points of the droid  
 \* @param damage the attack damage of the droid  
 \* @param evationRate the evasion rate of the droid  
 \* @param armorValue the armor value of the droid  
 \*/* public Assasin(String name, int health, int damage, int evationRate, int armorValue) {  
 this.name = name;  
 this.healthPoints = health;  
 this.atackDamage = damage;  
 this.droidType = "Assasin";  
 this.evationRate = evationRate;  
 this.armorValue = armorValue;  
 this.specialAbilityIsUsed = false;  
 }  
  
 */\*\*  
 \* Executes the Assasin's special ability, which attacks the enemy droid three times in a row.  
 \*  
 \* @param enemyDroid the enemy droid to attack  
 \*/* @Override  
 protected void specialAbility(Droid enemyDroid) {  
 specialAbilityIsUsed = true;  
 System.*out*.println(this.getDroidType() + " " + this.getName() + " used special ability");  
  
 *// Attacks enemy 3 times in a row* for (int i = 0; i < 3; i++) {  
 this.atackEnemy(enemyDroid, true);  
 }  
 }  
}

Tank.java

package units;  
  
*/\*\*  
 \* The Tank class represents a specialized type of Droid with a focus on defense.  
 \* Its special ability doubles its armor but decreases its attack damage.  
 \*/*public class Tank extends Droid {  
  
 */\*\*  
 \* Default constructor that initializes the Tank droid with predefined attributes.  
 \*/* public Tank() {  
 name = "Tank";  
 healthPoints = 70;  
 atackDamage = 10;  
 droidType = "Tank";  
 evationRate = 0;  
 armorValue = 10;  
 specialAbilityIsUsed = false;  
 }  
  
 */\*\*  
 \* Parametrized constructor that initializes the Tank droid with custom attributes.  
 \*  
 \* @param name the name of the droid  
 \* @param health the health points of the droid  
 \* @param damage the attack damage of the droid  
 \* @param armorValue the armor value of the droid  
 \*/* public Tank(String name, int health, int damage, int armorValue) {  
 this.name = name;  
 this.healthPoints = health;  
 this.atackDamage = damage;  
 this.droidType = "Tank";  
 this.evationRate = 0;  
 this.armorValue = armorValue;  
 this.specialAbilityIsUsed = false;  
 }  
  
 */\*\*  
 \* Executes the Tank's special ability, which doubles its armor value but halves its attack damage.  
 \*  
 \* @param enemyDroid the enemy droid (not directly affected by this ability, but passed for future use)  
 \*/* @Override  
 protected void specialAbility(Droid enemyDroid) {  
 specialAbilityIsUsed = true;  
 System.*out*.println(this.getDroidType() + " " + this.getName() + " used special ability");  
  
 *// Doubles the droid's armor* this.setArmorValue(this.getArmorValue() \* 2);  
  
 *// Halves the droid's attack damage* this.setAtackDamage(this.getAtackDamage() / 2);  
 }  
}

Summoner.java

package units;  
  
*/\*\*  
 \* The Summoner class represents a type of Droid with the ability to summon a pet that assists in battle.  
 \* The summoned pet attacks alongside the Summoner but with reduced damage.  
 \*/*public class Summoner extends Droid {  
  
 */\*\*  
 \* Indicates whether the Summoner's pet has been summoned.  
 \*/* private boolean petSummoned = false;  
  
 */\*\*  
 \* Default constructor that initializes the Summoner droid with predefined attributes.  
 \*/* public Summoner() {  
 name = "Tank";  
 healthPoints = 70;  
 atackDamage = 10;  
 droidType = "Tank";  
 evationRate = 0;  
 armorValue = 10;  
 specialAbilityIsUsed = false;  
 petSummoned = false;  
 }  
  
 */\*\*  
 \* Parametrized constructor that initializes the Summoner droid with custom attributes.  
 \*  
 \* @param name the name of the droid  
 \* @param health the health points of the droid  
 \* @param damage the attack damage of the droid  
 \* @param evationRate the evasion rate of the droid  
 \* @param armorValue the armor value of the droid  
 \*/* public Summoner(String name, int health, int damage, int evationRate, int armorValue) {  
 this.name = name;  
 this.healthPoints = health;  
 this.atackDamage = damage;  
 this.droidType = "Tank";  
 this.evationRate = evationRate;  
 this.armorValue = armorValue;  
 this.specialAbilityIsUsed = false;  
 this.petSummoned = false;  
 }  
  
 */\*\*  
 \* Sets whether the Summoner's pet has been summoned.  
 \*  
 \* @param petSummoned true if the pet is summoned, false otherwise  
 \*/* public void setPetSummoned(boolean petSummoned) {  
 this.petSummoned = petSummoned;  
 }  
  
 */\*\*  
 \* Executes the Summoner's special ability, summoning a pet that assists in attacks.  
 \* The pet attacks with half of the Summoner's damage until the end of the battle.  
 \*/* protected void specialAbility() {  
 specialAbilityIsUsed = true;  
 System.*out*.println(this.getDroidType() + " " + this.getName() + " used special ability");  
  
 *// Summons a pet that attacks each round with half the Summoner's damage* this.setPetSummoned(true);  
 }  
  
 */\*\*  
 \* Attacks the enemy droid and applies damage. If a pet has been summoned, it also attacks for half damage.  
 \*  
 \* @param enemyDroid the enemy droid to attack  
 \* @param outputLogs whether to output logs for the attack  
 \* @return the total damage dealt to the enemy droid  
 \*/* @Override  
 public int atackEnemy(Droid enemyDroid, boolean outputLogs) {  
  
 *// Regular damage* int damageGiven = *rand*.nextInt(this.getAtackDamage());  
  
 *// Calculate critical hit chance and apply critical damage* if (*rand*.nextInt(1, 100) <= enemyDroid.getEvationRate()) {  
 damageGiven \*= 2;  
 damageGiven = Droid.*calculateDamageAffectedByArmor*(damageGiven, enemyDroid.getArmorValue());  
 if (outputLogs) {  
 System.*out*.println("Droid " + this.getName() + " deals crit damage " + damageGiven + " to the enemy: " + enemyDroid.getName());  
 }  
 } else if (outputLogs) {  
 damageGiven = Droid.*calculateDamageAffectedByArmor*(damageGiven, enemyDroid.getArmorValue());  
 System.*out*.println("Droid " + this.getName() + " deals " + damageGiven + " damage to the droid: " + enemyDroid.getName());  
 }  
  
 *// Apply damage to enemy* enemyDroid.setHealthPoints(enemyDroid.getHealthPoints() - damageGiven);  
  
 *// If pet is summoned, pet also attacks with half damage* if (this.petSummoned) {  
 enemyDroid.setHealthPoints(enemyDroid.getHealthPoints() - damageGiven / 2);  
 }  
  
 return damageGiven;  
 }  
}

BattleSystems.java

package components;  
  
import units.Droid;  
import java.util.ArrayList;  
  
*/\*\*  
 \* The BattleSystems class provides methods to simulate battles between droids, either in a duel or team-based format.  
 \*/*public class BattleSystems {  
  
 */\*\*  
 \* Simulates a duel battle between two droids. The battle continues until one droid is no longer alive.  
 \* Droids alternate between attacking and defending. Special abilities can be activated under certain conditions.  
 \*  
 \* @param defenderDroid the droid that defends in the initial round  
 \* @param atackerDroid the droid that attacks in the initial round  
 \* @return the droid that won the battle  
 \*/* public static Droid duelBattle(Droid defenderDroid, Droid atackerDroid) {  
 int round = 0;  
  
 do {  
 System.*out*.println("\*============ROUND " + ++round + "=============\*");  
 *// Droids switch roles between attacker and defender* Droid temp = atackerDroid;  
 atackerDroid = defenderDroid;  
 defenderDroid = temp;  
  
 *// Check for special ability activation* if (!atackerDroid.isSpecialAbilityIsUsed()) {  
 if (atackerDroid.checkConditionsForSpecialAbilityUse(round)) {  
 atackerDroid.useSpecialAbility();  
 }  
 }  
  
 System.*out*.print(atackerDroid.getName() + " has " + atackerDroid.getHealthPoints() + " health points");  
 System.*out*.println(defenderDroid.getName() + " has " + defenderDroid.getHealthPoints() + " health points");  
 int damageDelt = atackerDroid.atackEnemy(defenderDroid, true);  
  
 } while (defenderDroid.isAlive());  
  
 return atackerDroid;  
 }  
  
 */\*\*  
 \* Simulates a 3vs3 battle between two teams of droids. Each droid on Team Blue fights the corresponding droid on Team Red.  
 \* The team with more victories wins the battle.  
 \*  
 \* @param droidTeamBlue an ArrayList of 3 droids representing Team Blue  
 \* @param droidTeamRed an ArrayList of 3 droids representing Team Red  
 \* @return the winning team, either droidTeamBlue or droidTeamRed  
 \*/* public static ArrayList<Droid> threevsthree(ArrayList<Droid> droidTeamBlue, ArrayList<Droid> droidTeamRed) {  
 if (droidTeamBlue.size() != 3 || droidTeamRed.size() != 3) {  
 System.*out*.println("Either droidTeamBlue or droidTeamRed contains more or less than 3 Droids;");  
 return null;  
 }  
  
 int teamBlueScore = 0;  
 int teamRedScore = 0;  
  
 *// Loop through each droid and conduct one-on-one battles* for (int i = 0; i < droidTeamRed.size(); i++) {  
 if (droidTeamRed.get(i).equals(droidTeamRed.get(i).atackEnemy(droidTeamBlue.get(i), false))) {  
 teamRedScore++;  
 } else {  
 teamBlueScore++;  
 }  
 }  
  
 *// Determine the winning team based on the score* if (teamBlueScore > teamRedScore) {  
 System.*out*.println("Team Blue won the 3vs3 battle!");  
 return droidTeamBlue;  
 } else {  
 System.*out*.println("Team Red won the 3vs3 battle!");  
 return droidTeamRed;  
 }  
 }  
}