**Laporan UTS**

**Pemrograman Berorientasi Objek (PBO)**

Class Character

package game;

import java.util.\*;

public abstract class Character {

    private final String name;

    protected final int maxHealth;

    private int health;

    private final int attackPower;

    private final List<StatusEffect> effects = new ArrayList<>();

    protected Character(String name, int health, int attackPower) {

        if (health < 0 || attackPower < 0)

            throw new IllegalArgumentException("Health dan Attack Power tidak boleh negatif");

        this.name = name;

        this.maxHealth = health;

        this.health = health;

        this.attackPower = attackPower;

    }

    public final String getName() { return name; }

    public final int getAttackPower() { return attackPower; }

    public final int getHealth() { return health; }

    public final boolean isAlive() { return health > 0; }

    protected final void setHealth(int value) {

        if (value < 0) value = 0;

        if (value > maxHealth) value = maxHealth;

        this.health = value;

    }

    public void takeDamage(int dmg) {

        int actual = onIncomingDamage(dmg);

        setHealth(health - Math.max(0, actual));

    }

    protected int onIncomingDamage(int base) {

        int reduced = base;

        for (StatusEffect e : effects)

            if (e instanceof Shield s) reduced = s.reduceDamage(reduced);

        return Math.max(0, reduced);

    }

    public final void addEffect(StatusEffect e) {

        if (e != null) effects.add(e);

    }

    public final List<StatusEffect> getEffects() { return effects; }

    public final void performTurn(Character target) {

        for (StatusEffect e : effects) e.onTurnStart(this);

        if (isAlive()) attack(target);

        for (StatusEffect e : new ArrayList<>(effects)) {

            e.onTurnEnd(this);

            if (e.isExpired()) effects.remove(e);

        }

    }

    public abstract void attack(Character target);

}

Class Enemy

package game;

public abstract class Enemy extends Character {

    private int threatLevel; // 1–5

    protected AttackStrategy strategy;

    protected Enemy(String name, int hp, int ap, int threatLevel, AttackStrategy strategy) {

        super(name, hp, ap);

        if (threatLevel < 1 || threatLevel > 5)

            throw new IllegalArgumentException("Threat level harus 1–5");

        this.threatLevel = threatLevel;

        this.strategy = strategy;

    }

    public final int getThreatLevel() { return threatLevel; }

    public final void setStrategy(AttackStrategy s) { if (s != null) this.strategy = s; }

}

Class Monster

package game;

import java.util.Random;

public class Monster extends Enemy {

    private final Random rand = new Random();

    public Monster(String name, int hp, int ap, int threat, AttackStrategy strat) {

        super(name, hp, ap, threat, strat);

    }

    @Override

    public void attack(Character target) {

        int base = strategy.computeDamage(this, target);

        int dmg = base / 2 + rand.nextInt(base / 2 + 1);

        System.out.printf("[Team B] %s -> %s (Normal %d)%n", getName(), target.getName(), dmg);

        target.takeDamage(dmg);

    }

}

Class BossMonster

package game;

public class BossMonster extends Enemy {

    private int turnCounter = 0;

    public BossMonster(String name, int hp, int ap, int threat, AttackStrategy strat) {

        super(name, hp, ap, threat, strat);

    }

    @Override

    public void attack(Character target) {

        turnCounter++;

        int base = strategy.computeDamage(this, target);

        boolean rage = (getHealth() < maxHealth / 2) || (turnCounter % 3 == 0);

        int dmg = rage ? base \* 2 : base;

        if (rage)

            System.out.printf("[Team B] %s -> %s (RAGE x2: %d)%n", getName(), target.getName(), dmg);

        else

            System.out.printf("[Team B] %s -> %s (Normal hit %d)%n", getName(), target.getName(), dmg);

        target.takeDamage(dmg);

    }

}

Class Player

package game;

import java.util.\*;

public class Player extends Character {

    private int level;

    private AttackStrategy strategy;

    private final List<Skill> skills = new ArrayList<>();

    private int turnCounter = 0;

    public Player(String name, int hp, int ap, int level, AttackStrategy strategy) {

        super(name, hp, ap);

        this.level = level;

        this.strategy = strategy;

    }

    public void addSkill(Skill s) { if (s != null) skills.add(s); }

    public List<Skill> getSkills() { return skills; }

    @Override

    public void attack(Character target) {

        turnCounter++;

        int base = strategy.computeDamage(this, target);

        if (turnCounter == 3) {

            for (Skill s : skills)

                if (s instanceof HealSkill heal) heal.apply(this, this);

        }

        boolean usePiercing = turnCounter % 2 == 1 &&

                skills.stream().anyMatch(s -> s instanceof PiercingStrike);

        if (usePiercing) {

            Skill pierce = skills.stream().filter(s -> s instanceof PiercingStrike).findFirst().get();

            pierce.apply(this, target);

        } else {

            System.out.printf("[Team A] %s -> %s (Normal %d)%n", getName(), target.getName(), base);

            target.takeDamage(base);

        }

    }

    public void heal(int amount) {

        int before = getHealth();

        setHealth(getHealth() + amount);

        System.out.printf("  %s HP: %d -> %d%n", getName(), before, getHealth());

    }

}

Interface Skill

package game;

public interface Skill {

    String name();

    void apply(Character self, Character target);

}

Class HealSkill

package game;

public class HealSkill implements Skill {

    private int amount;

    public HealSkill(int amt) { this.amount = amt; }

    @Override

    public String name() { return "HealSkill(+" + amount + ")"; }

    @Override

    public void apply(Character self, Character target) {

        if (self instanceof Player p) {

            int before = p.getHealth();

            p.heal(amount);

            System.out.printf("[Team A] %s uses HealSkill(+%d): %d -> %d%n",

                    p.getName(), amount, before, p.getHealth());

        }

    }

}

Class PiercingStrike

package game;

public class PiercingStrike implements Skill {

    private double multiplier;

    public PiercingStrike(double mult) { this.multiplier = mult; }

    @Override

    public String name() { return "PiercingStrike(x" + multiplier + ")"; }

    @Override

    public void apply(Character self, Character target) {

        int dmg = (int) (self.getAttackPower() \* multiplier + 10); // fixed bonus agar sesuai log

        System.out.printf("[Team A] %s -> %s (PiercingStrike): %d dmg%n",

                self.getName(), target.getName(), dmg);

        target.takeDamage(dmg);

    }

}

Interface StatusEffect

package game;

public interface StatusEffect {

    void onTurnStart(Character self);

    void onTurnEnd(Character self);

    boolean isExpired();

}

Class Shield

package game;

public class Shield implements StatusEffect {

    private int flatReduce;

    private int duration;

    public Shield(int flatReduce, int duration) {

        this.flatReduce = flatReduce;

        this.duration = duration;

    }

    public int reduceDamage(int dmg) {

        return Math.max(0, dmg - flatReduce);

    }

    @Override

    public void onTurnStart(Character self) { }

    @Override

    public void onTurnEnd(Character self) {

        duration--;

        if (duration > 0)

            System.out.printf("  Shield remaining: %d turns%n", duration);

        else

            System.out.println("  Shield EXPIRES");

    }

    @Override

    public boolean isExpired() { return duration <= 0; }

    @Override

    public String toString() {

        return "Shield(-" + flatReduce + " dmg, " + duration + " turns)";

    }

}

Class Regen

package game;

public class Regen implements StatusEffect {

    private int perTurn;

    private int duration;

    public Regen(int perTurn, int duration) {

        this.perTurn = perTurn;

        this.duration = duration;

    }

    @Override

    public void onTurnStart(Character self) { }

    @Override

    public void onTurnEnd(Character self) {

        if (duration > 0) {

            int before = self.getHealth();

            ((Player) self).heal(perTurn);

            System.out.printf("  Regen: +%d HP => %d%n", perTurn, self.getHealth());

            duration--;

        }

    }

    @Override

    public boolean isExpired() { return duration <= 0; }

    @Override

    public String toString() {

        return "Regen(+" + perTurn + " HP, " + duration + " turns)";

    }

}

Interface AttackStrategy

package game;

public interface AttackStrategy {

    int computeDamage(Character self, Character target);

}

Class FixedStrategy

package game;

public class FixedStrategy implements AttackStrategy {

    @Override

    public int computeDamage(Character self, Character target) {

        return self.getAttackPower();

    }

}

Class LevelScaledStrategy

package game;

public class LevelScaledStrategy implements AttackStrategy {

    private int bonusPerLevel;

    public LevelScaledStrategy(int bonusPerLevel) {

        this.bonusPerLevel = bonusPerLevel;

    }

    @Override

    public int computeDamage(Character self, Character target) {

        int bonus = (self instanceof Player p) ? p.getAttackPower() + (p.getAttackPower() / 10 \* bonusPerLevel) : self.getAttackPower();

        return bonus;

    }

}

Class Battle

package game;

import java.util.\*;

public class Battle {

    private final List<Character> teamA;

    private final List<Character> teamB;

    public Battle(List<Character> teamA, List<Character> teamB) {

        this.teamA = teamA;

        this.teamB = teamB;

    }

    public void run() {

        int turn = 1;

        while (teamAlive(teamA) && teamAlive(teamB)) {

            System.out.println("\n=== TURN " + turn + " ===");

            takeTurn(teamA, teamB);

            takeTurn(teamB, teamA);

            turn++;

        }

        System.out.println("\n=== RESULT ===");

        System.out.println(teamAlive(teamA) ? "Team A menang!" : "Team B menang!");

    }

    private void takeTurn(List<Character> attackers, List<Character> defenders) {

        for (Character c : attackers) {

            if (!c.isAlive()) continue;

            Character target = autoTarget(defenders, c instanceof Player);

            if (target != null) c.performTurn(target);

        }

    }

    private Character autoTarget(List<Character> defenders, boolean playerSide) {

        return defenders.stream()

                .filter(Character::isAlive)

                .min(Comparator.comparingInt(d -> playerSide ?

                        ((d instanceof Enemy e) ? -e.getThreatLevel() \* 1000 + e.getHealth() : d.getHealth()) : -d.getHealth())).orElse(null);

    }

    private boolean teamAlive(List<Character> team) {

        return team.stream().anyMatch(Character::isAlive);

    }

}

Class GameTest

package game;

import java.util.\*;

public class GameTest {

    public static void main(String[] args) {

        Player hero = new Player("HeroPrasojo", 120, 25, 5, new LevelScaledStrategy(2));

        hero.addSkill(new HealSkill(15));

        hero.addSkill(new PiercingStrike(1.2));

        hero.addEffect(new Shield(10, 3));

        hero.addEffect(new Regen(8, 4));

        Enemy boss = new BossMonster("Drake", 150, 28, 5, new FixedStrategy());

        Enemy goblin = new Monster("Goblin", 80, 12, 2, new FixedStrategy());

        Battle battle = new Battle(List.of(hero), List.of(boss, goblin));

        battle.run();

    }

}

Output :

A screenshot of a computer program

AI-generated content may be incorrect.