COSC 3P91 – Assignment 1 – Question 2

DESIGN DESCRIPTION DOCUMENT, Brock University, Canada

This document presents an object-oriented design for a strategic territory conquest simulation system. The architecture employs abstraction hierarchies, polymorphic behaviors, and modular packaging to create an extensible framework. Seven distinct modules organize functionality: territorycore for domain management, fortifications for defensive capabilities, dwellers for population entities, assaultunits for offensive forces, gamecontrol for orchestration logic, playerspace for participant data, and frontend for interaction mechanisms.

1 NETWORK SECURITY QUESTIONS

Add your assignment answers here... Include tables, algorithms, and figures if needed...

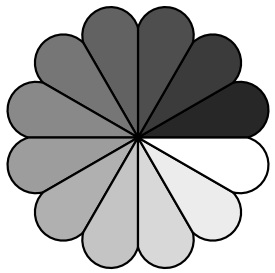

Fig. 1. Example of an image. (you can construct more sophisticated layouts using subfigs)

Table 1. Example of a table (nicely looking tables with booktabs).

Name	ID	Subtotal	Total
John Doe	jd2001	75	89

1.1 Question 1

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie

Author's address: Design Description Document, Brock University, 1812 Sir Isaac Brock Way, St. Catharines, ON, L2S 3A1, Canada.

ALGORITHM 1: Example of algorithm

Data: X_i; P_{a_i}; R; A_i; γ; ϵ
Result: π_i; v^*
$V = 0$; $\pi_i = 0$;
do
  $\Delta = 0$;
  for $x \in X$ **do**
    $Av = 0$;
    for $a \in A$ **do**
      $x_n = A(x)$;
      $Av[a] = P_a[x][x_n] * (R[x_n] + \gamma * V[x_n])$;
    $av_{best} = max(Av)$;
    $\Delta = max(\Delta, |av_{best} - V[x]|)$;
    $V[x] = av_{best}$;
    $\pi_i[x] = argmax(Av)$;
while $\Delta < \epsilon$;

vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

2 SECURED JAVA APPLICATION

The description of your Java code is here... Include any auxiliary text object for aiding your description (tables, algorithms, figures...)

Do not forget to cite sources [**?**] if any!

2.1 Performance Analysis

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim.

Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

2.2 Conclusions

A RESOURCES PACKAGE - RESOURCETYPE ENUM

```
package resources;

public enum ResourceType {
    GOLD,
    IRON,
    WOOD,
    FOOD
}
```

B RESOURCES PACKAGE - COST CLASS

```
package resources;

public class Cost {
    private int gold;
    private int iron;
    private int wood;
    private int time;

    public Cost(int gold, int iron, int wood, int time) {
        this.gold = gold;
        this.iron = iron;
        this.wood = wood;
        this.time = time;
    }

    public int getTotalValue() {
        return gold + iron + wood;
    }

    public Cost multiply(double factor) {
        return new Cost(
            (int)(gold * factor),
            (int)(iron * factor),
            (int)(wood * factor),
            (int)(time * factor)
        );
    }

    public int getGold() { return gold; }
    public int getIron() { return iron; }
    public int getWood() { return wood; }
    public int getTime() { return time; }
}
```

C RESOURCES PACKAGE - RESOURCEMANAGER CLASS

```
package resources;

public class ResourceManager {
    private int gold;
    private int goldCapacity;
    private int iron;
    private int ironCapacity;
    private int wood;
    private int woodCapacity;
    private int foodProduction;
    private int foodConsumption;

    public void addGold(int amount) {
        gold = Math.min(gold + amount, goldCapacity);
    }

    public void addIron(int amount) {
        iron = Math.min(iron + amount, ironCapacity);
    }

    public void addWood(int amount) {
        wood = Math.min(wood + amount, woodCapacity);
    }

    public boolean consumeResources(Cost cost) {
        if (hasEnoughResources(cost)) {
            gold -= cost.getGold();
            iron -= cost.getIron();
            wood -= cost.getWood();
            return true;
        }
        return false;
    }

    public boolean hasEnoughResources(Cost cost) {
        return gold >= cost.getGold() &&
               iron >= cost.getIron() &&
               wood >= cost.getWood();
    }

    public Cost getAvailableLoot() {
        return new Cost(gold / 2, iron / 2, wood / 2, 0);
    }

    public boolean canSupport(int population) {
        return foodProduction >= foodConsumption + population;
    }
}
```

D BUILDINGS PACKAGE - BUILDING ABSTRACT CLASS

```java
package buildings;

import resources.Cost;

public abstract class Building {
    protected int level;
    protected int maxLevel;
    protected int hitPoints;
    protected int maxHitPoints;
    protected int positionX;
    protected int positionY;
    protected Cost buildCost;
    protected int buildTime;
    protected boolean isUpgrading;
    protected long upgradeStartTime;

    public Building(int maxLevel, int maxHitPoints, Cost buildCost) {
        this.maxLevel = maxLevel;
        this.maxHitPoints = maxHitPoints;
        this.hitPoints = maxHitPoints;
        this.buildCost = buildCost;
        this.level = 1;
        this.isUpgrading = false;
    }

    public void upgrade() {
        if (!isMaxLevel() && !isUpgrading) {
            isUpgrading = true;
            upgradeStartTime = System.currentTimeMillis();
        }
    }

    public Cost getUpgradeCost() {
        return buildCost.multiply(level * 1.5);
    }

    public boolean isMaxLevel() {
        return level >= maxLevel;
    }

    public void takeDamage(int damage) {
        hitPoints = Math.max(0, hitPoints - damage);
    }

    public void repair() {
        hitPoints = maxHitPoints;
    }

    public void completeUpgrade() {
        if (isUpgrading) {
            level++;
            isUpgrading = false;
            maxHitPoints = (int)(maxHitPoints * 1.2);
            hitPoints = maxHitPoints;
        }
    }

    public int getLevel() { return level; }
    public int getHitPoints() { return hitPoints; }
}
```

E BUILDINGS PACKAGE - VILLAGEHALL CLASS

```
package buildings;

import resources.Cost;

public class VillageHall extends Building {
    private int villageLevel;

    public VillageHall() {
        super(10, 5000, new Cost(1000, 500, 500, 600));
        this.villageLevel = 1;
    }

    public int getAllowedBuildingLevel() {
        return villageLevel;
    }

    public int getVillageLevel() {
        return villageLevel;
    }

    @Override
    public void completeUpgrade() {
        super.completeUpgrade();
        villageLevel = level;
    }
}
```

F BUILDINGS PACKAGE - FARM CLASS

```
package buildings;

import resources.Cost;

public class Farm extends ProductionBuilding {
    private int foodPerHour;
    private int populationSupport;

    public Farm() {
        super(8, 1000, new Cost(100, 50, 200, 120));
        this.foodPerHour = 50;
        this.populationSupport = 10;
        this.maxWorkers = 3;
        this.productionRate = 20;
    }

    public int getFoodProduction() {
        return foodPerHour + getProduction();
    }

    public int getPopulationSupport() {
        return populationSupport * level;
    }
}
```

G BUILDINGS PACKAGE - ARCHERTOWER CLASS

```
package buildings;

import resources.Cost;

public class ArcherTower extends DefenseBuilding {
    private int arrowCount;

    public ArcherTower() {
        super(8, 1500, new Cost(300, 200, 100, 240));
        this.damage = 50;
        this.range = 10;
        this.attackSpeed = 1.0;
        this.arrowCount = 100;
    }

    public void reload() {
        arrowCount = 100;
    }
}
```

H INHABITANTS PACKAGE - INHABITANT ABSTRACT CLASS

```
package inhabitants;

import resources.Cost;

public abstract class Inhabitant {
    protected int level;
    protected int maxLevel;
    protected int hitPoints;
    protected int maxHitPoints;
    protected Cost trainingCost;
    protected int trainingTime;
    protected int foodConsumption;
    protected boolean isUpgrading;

    public Inhabitant(int maxLevel, int maxHitPoints) {
        this.maxLevel = maxLevel;
        this.maxHitPoints = maxHitPoints;
        this.hitPoints = maxHitPoints;
        this.level = 1;
        this.isUpgrading = false;
    }

    public void upgrade() {
        if (!isUpgrading && level < maxLevel) {
            isUpgrading = true;
        }
    }

    public Cost getUpgradeCost() {
        return trainingCost.multiply(level * 1.5);
    }

    public void takeDamage(int damage) {
        hitPoints = Math.max(0, hitPoints - damage);
    }

    public boolean isAlive() {
        return hitPoints > 0;
    }
```

```
    public void heal() {
        hitPoints = maxHitPoints;
    }
}
```

I INHABITANTS PACKAGE - WORKER CLASS

```
package inhabitants;

import resources.Cost;
import buildings.Building;

public class Worker extends Inhabitant {
    private boolean isIdle;
    private String currentTask;

    public Worker() {
        super(5, 100);
        this.trainingCost = new Cost(50, 0, 0, 30);
        this.foodConsumption = 1;
        this.isIdle = true;
        this.currentTask = "";
    }

    public void buildStructure(Building b) {
        isIdle = false;
        currentTask = "Building";
    }

    public void produceFood() {
        isIdle = false;
        currentTask = "Producing Food";
    }

    public void repair(Building b) {
        isIdle = false;
        currentTask = "Repairing";
    }

    public void setIdle(boolean idle) {
        this.isIdle = idle;
        if (idle) {
            currentTask = "";
        }
    }
}
```

J INHABITANTS PACKAGE - SOLDIER CLASS

```
package inhabitants;

import resources.Cost;

public class Soldier extends ArmyUnit {
    private int shield;

    public Soldier() {
        super(6, 150);
        this.trainingCost = new Cost(100, 50, 0, 60);
        this.damage = 30;
        this.attackRange = 1;
        this.movementSpeed = 2;
        this.shield = 20;
        this.foodConsumption = 2;
    }

    public void block() {
        // Implementation for blocking attacks
```

```
        }
}
```

K CORE PACKAGE - VILLAGE CLASS

```
package core;

import java.util.List;
import java.util.ArrayList;
import buildings.Building;
import buildings.VillageHall;
import inhabitants.Inhabitant;
import resources.ResourceManager;

public class Village {
    private String name;
    private int level;
    private int population;
    private int maxPopulation;
    private int area;
    private int maxArea;
    private VillageHall villageHall;
    private List<Building> buildings;
    private List<Inhabitant> inhabitants;
    private ResourceManager resources;
    private int defenseScore;
    private int attackScore;
    private long guardPeriodEnd;
    private boolean inGuardPeriod;

    public Village(String name) {
        this.name = name;
        this.level = 1;
        this.population = 0;
        this.maxPopulation = 50;
        this.area = 0;
        this.maxArea = 100;
        this.buildings = new ArrayList<>();
        this.inhabitants = new ArrayList<>();
        this.resources = new ResourceManager();
        this.villageHall = new VillageHall();
        this.inGuardPeriod = true;
        this.guardPeriodEnd = System.currentTimeMillis()
            + (7 * 24 * 60 * 60 * 1000);
    }

    public boolean addBuilding(Building b) {
        if (canBuild(b)) {
            buildings.add(b);
            return true;
        }
        return false;
    }

    public void removeBuilding(Building b) {
        buildings.remove(b);
    }

    public boolean addInhabitant(Inhabitant i) {
        if (population < maxPopulation) {
            inhabitants.add(i);
            population++;
            return true;
        }
        return false;
    }

    public int calculateDefenseScore() {
```

```
        defenseScore = 0;
        for (Building b : buildings) {
            defenseScore += b.getLevel() * 10;
        }
        return defenseScore;
    }

    public boolean canBuild(Building b) {
        return area + 10 <= maxArea;
    }
}
```

L CORE PACKAGE - PLAYER CLASS

```
package core;

import buildings.Building;
import inhabitants.Inhabitant;
import engine.AttackResult;

public class Player {
    private String playerName;
    private String playerId;
    private Village village;
    private int rank;
    private int attackWins;
    private int attackLosses;
    private int defenseWins;
    private int defenseLosses;
    private int totalLoot;

    public Player(String name) {
        this.playerName = name;
        this.playerId = generatePlayerId();
        this.village = new Village(name + "'s Village");
        this.rank = 0;
    }

    private String generatePlayerId() {
        return "P" + System.currentTimeMillis();
    }

    public boolean buildBuilding(Building b) {
        return village.addBuilding(b);
    }

    public boolean upgradeBuilding(Building b) {
        b.upgrade();
        return true;
    }

    public boolean upgradeInhabitant(Inhabitant i) {
        i.upgrade();
        return true;
    }

    public void updateRank() {
        rank = (attackWins * 10) - (attackLosses * 5)
            + (defenseWins * 8) - (defenseLosses * 3);
    }

    public Village getVillage() {
        return village;
    }
}
```

M ENGINE PACKAGE - GAMEENGINE CLASS

```
package engine;

import java.util.List;
import java.util.ArrayList;
import core.Player;
import core.Village;
import buildings.Building;

public class GameEngine {
    private long currentTime;
    private double gameSpeed;
    private AttackSimulator attackSimulator;
    private VillageGenerator villageGenerator;
    private ScoreCalculator scoreCalculator;
    private List<Player> players;

    public GameEngine() {
        this.currentTime = System.currentTimeMillis();
        this.gameSpeed = 1.0;
        this.attackSimulator = new AttackSimulator();
        this.villageGenerator = new VillageGenerator();
        this.scoreCalculator = new ScoreCalculator();
        this.players = new ArrayList<>();
    }

    public void startGame() {
        currentTime = System.currentTimeMillis();
    }

    public void updateTime() {
        currentTime = System.currentTimeMillis();
    }

    public void processUpgrades(Village v) {
        // Process building and inhabitant upgrades
    }

    public boolean allowBuild(Village v, Building b) {
        return v.canBuild(b);
    }

    public void addPlayer(Player p) {
        players.add(p);
    }
}
```

N ENGINE PACKAGE - ATTACKRESULT CLASS

```
package engine;

import resources.Cost;

public class AttackResult {
    private boolean success;
    private double successRate;
    private Cost loot;
    private int attackerLosses;
    private int defenderLosses;
    private int stars;

    public AttackResult(boolean success, Cost loot) {
```

```
        this.success = success;
        this.loot = loot;
        this.attackerLosses = 0;
        this.defenderLosses = 0;
        this.stars = success ? 3 : 0;
    }

    public boolean isSuccess() { return success; }
    public Cost getLoot() { return loot; }
    public int getStars() { return stars; }
}
```

O UI PACKAGE - GAMEUI CLASS

```
package ui;

import core.Player;
import engine.GameEngine;

public class GameUI {
    private Player currentPlayer;
    private GameEngine gameEngine;

    public GameUI(Player player, GameEngine engine) {
        this.currentPlayer = player;
        this.gameEngine = engine;
    }

    public void displayVillage() {
        System.out.println("=== Village ===");
        // Display village information
    }

    public void displayBuildMenu() {
        System.out.println("=== Build Menu ===");
        System.out.println("1. Farm");
        System.out.println("2. Gold Mine");
        System.out.println("3. Archer Tower");
    }

    public void displayAttackMenu() {
        System.out.println("=== Attack Menu ===");
        // Display attack options
    }

    public void handleUserInput() {
        // Handle user input for game actions
    }
}
```