COSC 3P91 – Assignment 1 – StudentID

AUTHOR NAME, Brock University, Canada

This document presents the design description for Assignment 1 of COSC 3P91. The assignment focuses on the design and modeling of a village-war strategy game using UML class diagrams. This document provides a comprehensive explanation and justification of the design decisions, including the package structure, class hierarchies, and relationships that comprise the system architecture.

1 DESIGN DESCRIPTION

1.1 Question 2: System Design Explanation

This section describes and justifies the UML design decisions for the village-war strategy game system.

1.1.1 Overview of the System Design. The system is designed to support a simplified real-time village-war strategy game. The architecture emphasizes modularity, extensibility, and clear separation of responsibilities. The design is organized around several core abstractions: Village, Buildings, Villagers, Army Units, Player, and the Game Engine. Each abstraction is represented as a class or hierarchy of classes, with relationships modeled through inheritance, composition, and associations.

The goal of the design is to support the required game behaviors: building construction, villager training, army training, upgrading, attack simulation, and village generation. The system is structured to allow future extensions such as new building types, new unit types, and additional game mechanics.

1.1.2 Package Structure. To maintain clarity and modularity, the system is divided into conceptual packages:

- **village** — contains Village, Building, and all building subclasses
- **inhabitants** — contains Villager and its subclasses
- **army** — contains Army, ArmyUnit, and unit subclasses
- **gameengine** — contains GameEngine and AttackResult
- **player** — contains Player
- **ui** — contains GameUI

This structure ensures that each subsystem is isolated and can evolve independently.

1.1.3 Building Hierarchy. All buildings inherit from the abstract Building class, which defines shared attributes such as level, health, and type, as well as common operations like upgrade() and repair().

Concrete building types include:

- **Farm** — increases population capacity
- **ArcherTower** and **Cannon** — defensive structures with damage and range attributes
- **Goldmine** and **LumberMill** — resource-producing buildings
- **Warehouse** — stores resources
- **Stable** — supports cavalry units (optional extension)

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

Update date: 4 February 2026

This hierarchy supports extensibility: new buildings can be added by subclassing Building without modifying existing code.

1.1.4 Villager Hierarchy. Villagers share common attributes such as name and energy, represented in the abstract Villager class. Subclasses specialize behavior:

- **Worker** — constructs buildings
- **Miner** — extracts gold
- **Collector** — gathers resources
- **Lumberjack** — cuts wood

This design allows the village to manage different villager roles while maintaining a unified interface for training and assignment.

1.1.5 Army and Army Units. The Army class maintains a collection of ArmyUnit objects and provides operations for adding units, removing units, and attacking targets.

ArmyUnit is an abstract superclass with shared attributes (damage, health) and an attack() method. Subclasses include:

- **Knight**
- **Archer**
- **Catapult**
- **Builder** (optional extension)

This hierarchy supports future expansion of unit types and allows the Game Engine to compute attack scores generically.

1.1.6 Village Class. The Village class aggregates buildings, villagers, and an army. It provides the core gameplay operations:

- buildBuilding()
- trainVillager()
- trainArmy()

The village also maintains its name and collections of buildings and units. This class acts as the central hub for player progression.

1.1.7 Game Engine. The GameEngine coordinates the overall game flow. It maintains references to the active village and the most recent AttackResult. It provides operations such as:

- processGameSpeed()
- startGame()
- endGame()

Although simplified at this stage, the engine is designed to be extended with attack simulation, village generation, scoring, and time management in future assignments.

1.1.8 Player and UI. The Player class stores player-specific information such as name and score. It interacts with the village through the Game Engine.

The GameUI class represents the user interface layer. It provides methods for displaying menus and game screens. Although not fully implemented, its presence satisfies the requirement to acknowledge the GUI in the design.

1.1.9 Design Principles and Justification. Several object-oriented principles guided the design:

- **Encapsulation**: Each class manages its own data and behavior.

- **Inheritance**: Shared behavior is factored into abstract superclasses (Building, Villager, ArmyUnit).
- **Polymorphism**: The Game Engine and Village can operate on generic types without knowing concrete subclasses.
- **Extensibility**: New buildings, units, or villagers can be added with minimal changes to existing code.
- **Separation of concerns**: Game logic, UI, player data, and world entities are kept in separate packages.

This structure ensures the system is maintainable, scalable, and aligned with the assignment's long-term goals.

REFERENCES

[1] I. Foster, M. Ripeanu, and A. Iamnitchi. 2002. Mapping the Gnutella Network. *IEEE Internet Computing* 6 (01 2002), 50–57. DOI: http://dx.doi.org/10.1109/4236.978369