# 

Object Oriented Programming Lab Final Project  
Spring-2025

## **StrongHold Game**

| **Submission Instructions:**   * This is a group assignment, with a maximum of 2 students allowed per group. (No more or less students are allowed per group) * Cross section groups are not allowed. * Divide the work equally as the marking will be based on individual work. * Submit your own original work. Plagiarism will not be tolerated at all, either done from the internet or from any fellow. * Any submission that does not adhere to these instructions will not be evaluated. * Submission might be on **Github,** so be ready for that too. * Viva for the project is inevitable. * Deadline:28-4-25   **NOTE:**  This is only the phase/module 1 of the complete project. There will also be a phase/module 2 of this project too. It will be announced soon. So be prepared. |
| --- |

**Stronghold Project - Module 1: Core Kingdom Engine**

**Project Overview:** The Stronghold project is a console-based multiplayer strategy simulation game that challenges students to implement a highly dynamic medieval kingdom using advanced Object-Oriented Programming concepts in C++. The game consists of two major modules:

1. **Module 1: Core Kingdom Engine** – focuses on internal kingdom mechanics such as population, economy, army, politics, and resource systems.
2. **Module 2: Multiplayer & Diplomacy Engine** – handles player-to-player interactions, alliances, diplomacy, trade systems, and real-time communication.

Both modules will be merged at the end for a fully functional, complex OOP game simulation. The actual split between modules will not be explicitly defined. Instead, you must analyze the project as a whole, identify system boundaries, and implement accordingly. Not everything is as it seems—some parts are tightly coupled, others intentionally vague. Use your engineering intuition. Beware: the deeper you go, the more you realize how little you know. The kingdom is vast, but the path to control it is treacherous.

**Objective:** This module focuses on implementing the internal mechanics of the Stronghold kingdom. You are responsible for simulating population dynamics, military systems, economic functions, leadership and political systems, banking/corruption, and resource management using C++ OOP principles. This module forms the backbone of the game engine. Which systems belong here and how they interact is for you to explore. Nothing is handed to you. Interpret the balance of power. Follow the flow of gold. Sense the tension in the classes. And ask yourself—what holds a stronghold together?

### **🔹 Key Responsibilities**

Implement the following systems as independent, modular C++ classes. Ensure each class adheres to the principles of **Object-Oriented Programming**, including:

* **Encapsulation**
* **Inheritance & Polymorphism**
* **Templates (where applicable)**
* **Exception Handling**
* **File Handling**
* **Dynamic Memory Management using new/delete or smart pointers**

Some system boundaries may blur. Your design choices will be evaluated. Trust no default. Question every assumption.

### 

### 

### 

### **📊 Required Functionalities**

Your Stronghold engine must support a wide range of realistic, interdependent functionalities reflecting life in a medieval kingdom. These functionalities are not grouped or labeled for you—dissect them, connect them, and design accordingly.

#### **🏘️ Social Structure**

* A society with multiple interacting social classes, each with unique traits and conflicts

A kingdom is made up of various social classes—peasants, merchants, nobles, and others. Each class has its own expectations and behaviors. You'll need to simulate how these classes interact, how inequality and favoritism affect peace, and what happens when the system becomes unstable.

#### **👥 Population Dynamics**

* A living population that grows, shrinks, falls ill, or revolts based on supply and conditions

The population is not a static value. It should increase or decrease based on food supply, employment, shelter, and external threats like war or disease. Revolts may break out if the population is mistreated or neglected.

#### **🛡️ Military System**

* A military that requires training, pay, food, and is sensitive to morale and corruption

Armies don't appear magically. You must simulate recruitment based on population, training that consumes time, resources such as food and weapons, and morale that depends on leadership and success in battle. Corruption and payment issues can destabilize military power.

#### **👑 Leadership Mechanics**

* Leadership mechanics including elections, coups, and diverse leadership behaviors

In a dynamic kingdom, rulers are not always permanent. Elections, coups, or death in battle can shift leadership. Each ruler should bring different policies and behaviors to the kingdom, affecting how other systems operate.

#### 

#### 

#### 

#### 

#### **💰 Economic Management**

* An economic structure influenced by war, inflation, services, and tax decision.

Money fuels the kingdom. Simulate taxation, inflation, trade impacts, and funding for public systems. Mismanagement may benefit the rich while triggering chaos among the poor. Strategic balancing is essential.

#### **🏦 Banking & Corruption**

* Banking systems where loans, interest, fraud, and audits affect financial health.

Wealth must be managed—through loans, treasury audits, and sometimes corruption. Your system should support loans with conditions, detect embezzlement, and include consequences for failing to manage finances wisely.

#### **🪵 Resource Ecosystem**

* A dynamic environment where resources are gathered, consumed, traded, or lost.

All operations—from feeding people to producing weapons—depend on a variety of resources like food, wood, stone, and iron. Implement a system where these resources are gathered, consumed, traded, and stockpiled.

#### **⚠️ Event Handling**

* Events that emerge from chaos—famine, disease, war, weather, betrayal

Chaos is inevitable. Random events like war, famine, drought, or plagues should trigger system-wide disruptions. Your design must account for their impact and support appropriate responses from leadership.

The functionalities you must simulate include:

* A society with multiple interacting social classes, each with unique traits and conflicts
* A living population that grows, shrinks, falls ill, or revolts based on supply and conditions
* A military that requires training, pay, food, and is sensitive to morale and corruption
* Leadership mechanics including elections, coups, and diverse leadership behaviors
* An economic structure influenced by war, inflation, services, and tax decisions
* Banking systems where loans, interest, fraud, and audits affect financial health
* A dynamic environment where resources are gathered, consumed, traded, or lost
* Events that emerge from chaos—famine, disease, war, weather, betrayal

### **🔹 Implementation Constraints**

* **File Structure:**
  + There will be a **single header file** (Stronghold.h) that contains all class declarations and prototypes.
  + Each class will have its own **separate implementation file** (e.g., Army.cpp, Bank.cpp).
  + main.cpp will serve as the entry point and handle user interaction.
* **Exception Handling:**
  + Invalid inputs (negative values, illegal operations) must throw and catch exceptions
  + File handling errors must be managed
* **Templates:** Use template classes for general-purpose data handling.
* **Naming Convention:**
  + Use proper naming convention for variables.
* **Pointers:**
  + Use new and delete OR smart pointers (unique\_ptr, shared\_ptr) wherever dynamic memory is involved
* **File I/O:**
  + Save and load game state to a file (e.g., game\_save.txt)
  + Log score and resource usage to score.txt

### 

### **🎁 Bonus Challenges (Optional)**

These tasks are **not required**, but completing them will earn bonus marks and serious bragging rights.

#### **🧠 AI-Driven Decisions**

Design a simple AI that simulates decision-making for a non-player kingdom or leader. This AI should make tax decisions, mobilize armies, and handle internal conflict autonomously based on in-game conditions.

#### **📈 Analytics Logger**

Create a system that tracks and logs historical changes in key indicators like population, treasury, army strength, and resource levels. Display this as a time-based progression report after the game ends.

### **🔹 Submission Requirements**

* Modular, well-commented C++ code for all systems
* Instructions to compile and run your module
* Sample output demonstrating system functionality

### **🔹 Evaluation Criteria**

* Correct implementation of each class and feature
* Usage of required OOP features
* Code organization and modularity
* Error handling robustness
* Game realism and system interactions
* Ability to derive structure and dependencies without explicit directions
* Ability to navigate through vagueness and uncertainty

**Note:** This module will later be integrated with Module 2 (Multiplayer & Diplomacy Engine). Build your code to be modular and interface-ready—but remember, the true boundary between modules is a puzzle wrapped in power struggles, deception, and decay.

***Best of luck, commanders—and may your logic be sharper than your sword!***