**OOP PROJECT PROPOSAL**

**Introduction**

****

The Stronghold game is an ultra-challenging, console-based multiplayer strategy game designed to push C++ OOP students to their absolute limits. Players manage a medieval kingdom, dealing with war, politics, economy, corruption, diplomacy, natural disasters, and real-time strategic decisions.

This version of Stronghold enforces brutal OOP constraints, requiring students to master pointers, inheritance, polymorphism, templates, exception handling, file handling, dynamic memory management, interdependent mechanics, randomized events, and time-based functions (sleep()). The game is intentionally designed to be extremely difficult, ensuring that only students with a deep understanding of C++ and OOP principles can successfully complete it.

**2. Extreme Game Mechanics & Dependencies**

**2.1 Population & Social Structure (Heavily Interdependent)**

Population affects tax revenue, workforce, army recruitment, and stability.

Social classes include peasants, merchants, nobility, and military, each with unique needs.

Class conflicts occur if taxes are unfairly distributed, triggering riots.

Population growth depends on food supply, healthcare, and job availability.

Plagues, famine, and war can wipe out large portions of the population.

**2.2 Army System (No Easy Wars, Hardcore Realism)**

Army size is strictly dependent on population and economic conditions.

Training takes real-time seconds (sleep()), forcing long-term planning.

Unpaid soldiers rebel or desert, causing civil wars.

Army morale drops during prolonged wars, requiring food, pay, and rest cycles.

Generals are elected or appointed, but corruption can weaken the army.

Blacksmiths must produce weapons, and soldiers must be trained in barracks.

War depletes resources, weakens the economy, and impacts population stability**.**

**2.3 King Election System (Political Chaos Mechanic)**

The king is elected through votes, and different factions influence the outcome.

A weak king can be overthrown via a coup, causing civil unrest.

Bribery, assassination, and blackmail are possible tools in elections.

Each king has a different leadership style, affecting tax rates, diplomacy, and military actions.

If the king dies in battle, the kingdom falls into political instability until a new ruler is chosen.

**2.4 Economy & Taxation System**

The economy is dynamic, impacted by war, trade, taxation, corruption, and random events.

Inflation occurs if the king over-taxes or prints too much money.

A progressive tax system can be implemented, benefiting the poor but angering the nobility.

Uncollected taxes weaken the economy, but over-taxation causes revolts.

Players must manage public services (hospitals, food distribution, military funding).

Market crashes can bankrupt the kingdom, leading to loan dependency.

**2.5 Banking & Corruption System (Deep Dependencies, No Free Money)**

Loans have strict interest rates and must be repaid on time.

Defaulting on loans results in land seizure and economic collapse.

Corrupt officials can steal money, weakening the economy.

Players must audit their finances, which costs resources.

Bank heists and fraudulent activities are possible, leading to security expenses.

A treasury mismanagement system ensures that poor decisions impact future gameplay.

**2.6 Resources & Supply Chain System (No Easy Survival)**

Every resource (wood, stone, iron, gold, food, weapons) must be managed.

Food is essential for population survival, and shortages cause starvation, disease, and revolts.

Trading resources requires secure trade routes, which can be attacked by enemy factions.

Weather conditions (droughts, harsh winters) affect food production, increasing reliance on stockpiles.

War consumes massive amounts of resources, making prolonged conflicts unsustainable.

**2.7 Trade, Market & Diplomacy System (Players Must Adapt to an Unpredictable Economy)**

The market system is fully dynamic, with fluctuating supply and demand.

Players can engage in trade agreements, but alliances can be broken at any time.

Smugglers can bring illegal goods, but getting caught results in fines.

Enemy kingdoms can impose trade sanctions, crippling the economy.

Merchant guilds demand fair prices, and refusing them can lead to boycotts.

**2.8 Multiplayer System & Text Communication (Deception & Alliances Allowed)**

Players can form alliances or betray each other at any time.

A text-based communication system allows players to negotiate, lie, or manipulate.

Trade requests can be fake, leading to trust-based strategy.

Treaties can be broken at will, adding layers of unpredictability.

**---**

**3. Implementation Constraints (OOP Requirements)**

**3.1 Mandatory Three-File Structure**

Header file (.h) → Class definitions and function prototypes.

Implementation file (.cpp) → Function implementations.

Main file (main.cpp) → Game execution and user interaction**.**

**3.2 OOP Concepts Enforcement**

Every major system must be implemented as a class:

Kingdom, Army, Bank, Market, Map, Resources, Politics, etc.

**Use of inheritance & polymorphism:**

Leader (Base Class) → King, Commander, Merchant Guild Leader (Derived Classes).

Virtual functions for flexible governance (e.g., different kings have different policies).

**3.3 Pointers & Dynamic Memory Must Be Used**

All objects must be dynamically allocated using new and properly deallocated using delete.

Smart pointers (unique\_ptr, shared\_ptr) must be implemented where applicable.

**3.4 Exception Handling is Compulsory**

Invalid inputs (e.g., negative army size, illegal actions) must trigger exceptions.

File handling errors must be caught using try-catch blocks.

**3.5 Template Classes for Generalized Data Handling**

Templates must be used for inventory, market pricing, and army recruitment.

**template <typename T>**

**class Resource {**

**T quantity;**

**public:**

**void setQuantity(T q) { quantity = q; }**

**};**

**3.6 File Handling for Score & Game State**

Score tracking and resource logs must be stored in score.txt.

Game state must be saved and loaded from a file.

**---**

**4. Advanced Mechanics for Extreme Difficulty**

**4.1 Time-Based Actions (sleep()) for Realism**

Training an army takes real-time seconds (prevents instant military buildup).

Building construction and market transactions require wait times.

**4.2 Unpredictable Events System (rand())**

Random wars, natural disasters, assassinations, and revolts keep players on edge.

**void randomEvent() {**

**srand(time(0));**

**int event = rand() % 5;**

**if (event == 0) cout << "A plague has spread! Population decreases.\n";**

**}**

**4.3 Tax & Political Instability System**

Raising taxes too high will cause a revolution.

Choosing the wrong general can lead to an internal military coup.

**“BEST OF LUCK!!”**