Age of Empires 4 Simulation Report

1. Selected Topic & Description

Topic:

Simulation of a simplified Age of Empires 4 game scenario, focusing on city development, resource management, unit training, and turn-based battles between civilizations.

Description:

This project implements a backend simulation of a strategy game inspired by Age of Empires 4. Players choose a civilization, manage resources, build structures, train units, and engage in battles against an AI opponent. The simulation is fully command-line based and models core gameplay mechanics such as villager assignment, age advancement, and combat.

2. System Design

Main Components:

Civilization:

Represents a player or AI civilization. Manages villagers, resources, buildings, units, and game progression.

Villager:

Models a villager who gathers resources. Gathering efficiency depends on civilization bonuses and age.

Unit:

Represents military units with stats (HP, attack, type, cost). Used in battles.

Building:

Base class for all buildings. Includes Town Center, House/Village (population), and Military Buildings.

Game Loop:

Handles user commands, resource gathering, building, training, battles, and saving/loading game state.

Modules & Structure:

AgeOfEmpires.py: Main game logic and command loop.

Civilization.py: Civilization class and AI logic.

Villager.py: Villager class.

Unit.py:** Unit class.

Building.py: Building classes.

Utils.py: Constants, data, helper functions, and custom exceptions.

---

3. Techniques & Course Topics Used

OOP (Classes, Objects, Inheritance, Polymorphism):

- All main entities (Civilization, Villager, Unit, Building) are modeled as classes.
- Building subclasses: TownCenter, House, MilitaryBuilding.
- Methods are overridden and extended as needed.

Functional Programming:

- Use of `map`, `filter`, and `lambda` for data processing (e.g., grouping units, filtering available units).
- Example: Grouping units for battle uses `Counter` and list comprehensions.

Modules & Package Management:

- Code is split into logical modules/files.
- Custom modules are imported and used throughout.

File Handling & Persistence:

- Game state can be saved and loaded using JSON files.

- Serialization and deserialization of game objects for persistence.

Error Handling:

- Custom exception `SimulationError` is defined.

- Extensive use of `try/except` blocks for user input and file operations.

Time & Utility Modules:

- `random` for AI decisions and battle order.

- `datetime` for timestamping saves (if extended).

- `collections.Counter` for grouping and summarizing units/buildings.

4. Notable Features

Turn-based Simulation:

Each turn, players can assign villagers, gather resources, build, train, and prepare for battles.

AI Opponent:

The AI uses a simple strategy based on civilization style, with random elements for unpredictability.

Battle System:

Units are grouped and fight in turns, with counter bonuses based on unit types.

Resource Management:

Players must balance food, gold, wood, and stone for growth and military strength.

Persistence:

Players can save and load their progress at any time.

---

5. Course Topics Checklist

Topic	**Implementation Example**
Functional Programming	Map, filter, lambda, Counter in battle/resource logic
Modules & Package Management	Code split into multiple files/modules
File Handling & Persistence	JSON save/load for game state
Error Handling	Custom exceptions, try/except for input and file ops
OOP	Classes for all main entities, inheritance for buildings
Inheritance & Polymorphism	Building subclasses, method overrides
Time & Utility Modules	Random, datetime, collection.Counter

---

6. Summary

This simulation demonstrates a modular, object-oriented approach to modeling a strategy game scenario. It incorporates functional programming, error handling, file I/O, and utility modules, fulfilling all course requirements. The code is organized, commented, and extensible for future improvements.