umcrest.png
Department of Computer Science
Faculty of ICT
University of Malta

Mr Clyde Vassallo
cvass05@um.edu.mt

CPS2004 Object-Oriented Programming

Study-Unit Assignment

October 3, 2022

Preamble: This document describes the assignment for *CPS2004: Object Oriented Programming*. This assignment is worth **100%** of the total, final mark for this unit. You are expected to allocate approximately 80 hours to complete the assignment. The deadline is available on VLE.

You may **not** copy code from internet sources or your colleagues. The Department of Computer Science takes a very serious view on plagiarism. For more details refer to plagiarism section of the Faculty of ICT website[1].

Important Notice

This coursework assessment **may** include an interview with an examination board, so that we can make sure you have clearly understood how to solve the problems in this coursework.

You are to regularly commit all of your code to a GitHub or GitLab private repo and give me (clydevassallo) read access. In the documentation, please state the **last commit hash** and **repository URL** on the front page.

1 Deliverables

The code and documentation is to be submitted on the VLE website. Failure to submit this in the required format will result in your assignment not being graded. Please give me (clydevassallo) read accessrepo as soon as you start working on this. Marks will be deducted if you fail to commit and push regularly to this repo.

[1] https://www.um.edu.mt/ict/Plagiarism

Only one file is required for electronic submission, a zip archive containing your assignment code and the report. This needs to be uploaded to VLE. The code of each task should be located in a top level directory in the archive, named `task1` and `task2`. A single report named `report.pdf` (for all tasks) should be placed in the top level directory.

The report should **not** be longer than 14 pages (including figures and references) and, for each task, should contain the following:

- UML Diagram showing the classes' makeup and interactions
- A textual description of the approach (highlighting any extras you implemented)
- Test cases considered (and test results)
- Critical evaluation and limitations of your solution
- Answers to any questions asked in the task's description

2 Technical Specification

C++ code will be compiled using the `g++` compiler (version 12.2.0) from the GNU Compiler Collection. Please make sure to compile C++ using the command line:

```
g++ -Wall -std=c++20 -march=native *.cpp
```

Your Java submissions will be compiled and run using the OpenJDK (version 17). While you are free to use any IDE, make sure that your code can be compiled (and run) from the command line.

Failure to do so will have an impact on your grade. As a requirement, add a bash or batch script (`compile.sh` or `compile.bat`) in each task directory to compile your task. Also make sure to add a script (`run.sh` or `run.bat`) in each task directory to execute your task. In these scripts, include any command line arguments and test data files – if applicable. Each task should have a `Launcher` file (either `.java` or `.cpp`) which contains a `main(...)` method used to run your solutions. **Note that you should not have any absolute paths hardcoded in your programs/bash/batch scripts.**

3 Tasks

This assignment consists of two programming tasks. **One of these tasks must be completed in C++ while the other must be completed in Java**. It is up to you to choose which task to complete in either programming language.

3.1 Village War Game

Design a simple turn-based village war game, emphasising the concepts in **boldface**.

A fully fledged game would take months to develop, so in this exercise you will be assessed based on your ability to apply the design principles discussed in class. You do not need to implement a complex GUI (graphical user interface), a CLI (command line interface) would suffice. You do **NOT** need persistence (save game), nor networking support (online multiplayer).

In this game, players control **VILLAGES** that are scattered across a **MAP**. Each **VILLAGE** has different properties including its location (x-y coordinate), health and owner. Using **RESOURCES**, players must build and upgrade different **BUILDINGS**. Some **BUILDINGS** generate **RESOURCES** while others are used to train **TROOPS**. Different types of **TROOPS** have different **RESOURCE** costs and stats. Apart from building up their VILLAGE, players can send **ARMIES** of **TROOPS** to attack other **VILLAGES**, reducing their health and stealing **RESOURCES** if the attack is successful. Once the health of a **VILLAGE** reaches 0, the **VILLAGE** is destroyed and its owner is eliminated from the game. The player with the last surviving **VILLAGE** wins the game.

3.1.1 Game Setup

At the start of a Game Session, players must be prompted for:

- The number of players that will be playing (at least one).
- The number of AIs (AI players) that should be played by the game itself (zero or more).

One VILLAGE must be generated for each player and each AI. These VILLAGES must be placed on the MAP, each with a different set of co-ordinates. Ideally, VILLAGES are spread evenly across the MAP.

3.1.2 Modelling

When designing any non-trivial application, a crucial step is Modelling. During Modelling, the problem is broken down and Domain Models are extracted. Domain Models are representations of real world concepts (in this case VILLAGE, BUILDINGS, TROOPS etc) designed to solve the problem at hand (in this case the Game). You are free to add any Domain Models you feel will help represent the concepts required to develop the Game. However, you must include at least the following concepts:

- MAP - is a 2D grid of cells, each with its own x and y coordinates.

- VILLAGE - owned by a player and is a contained for a collection of BUILDINGS and TROOPS. Each VILLAGE has its own health and location.
- RESOURCES - are spent to build/upgrade BUILDINGS and train TROOPS. Your game must support at least 3 different types of resources.
- TROOPS - are trained by the player and used to attack other VILLAGES or defend the VILLAGE in which they are stationed. Each type of TROOP has its own training RESOURCE cost, health, attack, carrying capacity and marching speed. Your game must support at least 3 different types of troops.
- BUILDINGS - are first built then upgraded to increase their efficacy. Your game must support at least 3 different RESOURCE-generating BUILDINGS (1 for each type of RESOURCE) and 3 different TROOP-generating BUILDINGS (1 for each type of TROOP). RESOURCE-generating BUILDINGS generate an amount of the respective RESOURCE every TURN. TROOP-generating BUILDINGS allow players to train TROOPS at the expense of RESOURCES.

3.1.3 Game Loop

The Game Loop represents the overall repeating flow of the game until the win condition is met. In this game, the game is split into Rounds. Each Round consists of a number of Round Phases, starting with the Turn Phase. During this Phase, every player and AI with a standing VILLAGE must take one turn. During each turn, the following Phases occur:

1. Friendly Troop Arrival - Any TROOPS owned by the turn taker that have arrived back at the VILLAGE after an attack are considered to be stationed in the VILLAGE. Any RESOURCES they are carrying are added to the RESOURCES of the VILLAGE.
2. Enemy Troop Arrival - Any attacks on the turn taker's VILLAGE by enemy TROOPS that have successfully arrived at the VILLAGE's location must be resolved.
3. Resource Earning - The turn-taker earns RESOURCES according to the upgrade level of the VILLAGE's RESOURCE-generating BUILDINGS.
4. Player Actions - The turn-taker can choose to perform any of the following actions until the VILLAGE's RESOURCES are depleted or until the player chooses to pass:
   - Build/upgrade BUILDINGS (at the expense of RESOURCES).
   - Train TROOPS (at the expense of RESOURCES).
   - Attack another VILLAGE with an ARMY of TROOPS.
   - Surrender (the VILLAGE is destroyed) and the player is eliminated from the game.

- Pass the turn to the next player.

After each surviving player and AI takes their turn within the current round, the Turn Phase is complete and the following Round Phases take place:

1. Win Condition Check - If only a single VILLAGE remains standing, the owner of that VILLAGE wins the game and the game ends.
2. Marching - Marching ARMIES' locations are updated according to their marching speed and target.
3. End Round - The current round is considered completed.
4. Start Round - The next round begins.

3.1.4 Attacking/Defending a Village

During the Player Actions Turn Phase, players can send ARMIES of TROOPS to attack VILLAGES. ARMIES will begin marching towards their respective target VILLAGE at the marching speed of the slowest type of TROOP in the ARMY. Marching speed should be expressed as cells per round. During the Marching Round Phase, if the distance between the ARMY and the target VILLAGE is less than or equal to the ARMY's marching speed, the ARMY is considered to have successfully arrived at the VILLAGE's location. Combat must be then resolved during the next Enemy Troop Arrival Phase of the target VILLAGE's owner.

To resolve combat:

1. Sum the attack of the TROOPS in the attacking ARMY. The attacking ARMY is made up of the TROOPS that have successfully arrived in the VILLAGE being attacked. TROOPS outside the target VILLAGE do not contribute to this combat and are not affected by it.
2. Sum the attack of the TROOPS in the defending ARMY. The defending ARMY is made up of the TROOPS currently stationed in the VILLAGE being attacked. TROOPS outside the VILLAGE do not contribute to this combat and are not affected by it.
3. Kill TROOPS from both ARMIES until the total health of the TROOPS killed is equal to the total attack of the opposing ARMY.
4. If the attacker still has TROOPS standing, the attack is considered successful.
5. If the attacker does not have any TROOPS standing, the attack is considered unsuccessful and the combat ends.

In the case of a successful attack:

1. Damage is dealt to the defending VILLAGE equal to the total ATTACK of the surviving TROOPS from the attacking army.
2. Resources equal to the carrying capacity of surviving TROOPS from the attacking army are reduced from the defending VILLAGE's resource pool. These are considered to be carried by the attacking ARMY.
3. The surviving TROOPS from the attacking ARMY start marching back to their owner's VILLAGE at the ARMY's respective marching speed (speed of the slowest surviving TROOP).

3.1.5 AI's Turn

As highlighted throughout this document, the game you develop is expected to support AI turns. Note that you are **NOT** expected to develop a sophisticated AI.

The AI's decision making can be kept very simple as long as the actions taken:

1. Do not violate any of the game rules.
2. Somehow contribute towards the progression of the AI's village.

3.1.6 Conclusion

The task description outlines the basic structure of the game, including the basic rules and models. However, feel free to take any reasonable assumptions if you find any game rules to be missing. These assumptions should be documented in the final report.

(Total for task: 80 marks)

3.2 Minesweeper

The game will take place on a randomly generated 16×16 minesweeper grid of cells with 40 mines. The game should allow a **single player** to attempt to clear the grid without hitting any mines.

To clear the grid, the player must choose uncleared cells one at a time. When an uncleared cell is chosen, the following happens:

- If the cell contains a mine, the player loses the game.
- If the cell is the last uncleared cell without a mine, the player wins the game.
- If the cell does not contain a mine but has one or more adjacent mines, the cell is cleared and the number of adjacent mines is shown.

- If neither cell nor any of its 8 surrounding cells contain a mine, the cell and its surrounding cells are cleared, revealing the number of adjacent mines of each.

The game board should look similar to this:

```
-- 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
01 -- -- -- -- -- -- -- -- -- -- -- -- -- XX -- --
02 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
03 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
04 -- -- -- -- -- -- -- -- 01 -- -- -- -- -- -- --
05 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
06 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
07 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
08 -- -- -- -- 04 -- -- -- -- -- -- -- -- -- -- --
09 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10 -- -- -- -- -- -- -- -- -- -- -- 00 00 00 -- --
11 -- -- -- -- -- -- -- -- -- -- -- 00 00 00 -- --
12 -- -- -- -- -- -- -- -- -- -- -- 00 00 00 -- --
13 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
14 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
15 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

With:

- 'XX' representing a mine.
- '–' representing an unexplored cell on the grid.
- '00'/'01'/'02'...'08' representing the adjacent number of mines.

(Total for task: 20 marks)

4 Grading Criteria

The following criteria, described in Table 1, will be taken into consideration when grading your assignment.

Table 1: CPS2004 Assignment grading criteria. Equal consideration will be given to each.

Overall Considerations	
OO Concepts	Demonstrable and thorough understanding and application of OO concepts and design. You should make use of most of the concepts presented during lectures.
Thoughtful Design	As presented during lectures and clearly documented. Please include: design (in UML), technical approach, testing, critical evaluation and limitations.
Functionality	Completeness and adherence to tasks' specification. Correctness of solutions provided.
Quality and Robustness	Proper error handling. Proper (and demonstrable) testing of solutions. No memory leaks or wasted resources.
Programming Skill	Easy to understand code utilizing appropriate Java and C++ features.
Environment	Use of version control and reliable build environment.