COSC 3P91
Assignment 1

Due date: February 6th, 2026 at 23:59 (11:59 pm).
Team: The assignment can be completed in groups of up to two students.
Delivery method: the student needs to delivery the assignment only through D2L.
Delivery contents: document with answers and [Java, C, C++] codes if applicable (see Submission instructions).
Attention: check the Late Assignment Policy.

Assignment Overview

To design and model an strategy game, using a UML class diagram.
This is a design assignment only. No game logic, simulation, or functional implementation is required. The code should contain the classes, which respectively matches with your UML class diagram. These classes only need to have the members (non-instantiated variables) and operations (empty methods) properly mapping the relationships and functionalities.

The Game

- The **game** is a simplified village war strategy game where each player must build, develop, and upgrade a village. Also, the village must be able to defend from attacks and perform attacks. The concept is very similar to several online mobile games.
- Each village contains the following buildings:
  - A **Village Hall**;
    - The village hall may have levels/stages that condition the upgrades available to all other buildings.
  - Food production: **Farms**;
    - Each farm is responsible for feeding a population size. The number of farms restricts the population size;
    - Suggestion (extra): Farms may have levels that allow to have a larger population size fed per farm.
  - Mining: **Gold Mines**, **Iron Mines**, and **Lumber Mills**;
    - Suggestion (extra): You may just assume that the mining/extraction buildings are placed in areas with plenty of ore and lumber;
    - Suggestion (extra): Even those buildings have costs to be built. You may define precedence of requirements: (i) lumber mill can be built from scratch; (ii) iron mines need lumber to be build; and (iii) gold mines need lumber and iron to be built;
    - Suggestion (extra): You can assume resources are unlimited, or you may define a limit for extracting resources – after that, the mine/mill depletes.
  - Defences: **Archer Towers** and **Cannons**.
    - They can inflict damage to attacking armies, or army units. Depending on the level to details when simulating attacks. For instance, towers may have different damage than cannons do and may have larger defence range;
    - Suggestion (extra): Defence unities may be upgraded to inflict more damage;
    - Suggestion (extra): You may add dependencies for enabling defence unities or upgrading them, such as a blacksmith (likewise for army units).
    - Suggestion (extra): workers, miners, and collectors may help with defence but much lower damage and hitpoints.

- Each village is limited by a maximum number of Inhabitants, and they can be the following:
  - **Workers**: food production and construction;
    - A village must have idling builders for constructions and upgrades to start;
  - **Miners/Collectors** (gold, iron, and wood);
    - Each miner and collector has an average or maximum "production" capacity;
    - Each mine or mill has a maximum number of miners and collectors.
  - Army: **Soldiers**, **Archers**, **Knights**, and **Catapults**.
    - They can inflict damage to any building, including defence units.
    - Each army unit has a different damage and attack range.
    - Suggestion (extra): You may simulate the attack at a fine grain level – each army unit, depending on the level to details when simulating attacks. For instance, soldiers and archers may be more effective against defence units while catapults may inflict more damage against non-defence unities.;
    - Suggestion (extra): You may impose an army size or army composition limitations. For instance, an army can only be succeed in an attack if it is composed of at least one type of attack unit.
- Every building, worker, miner, collector, and army unit has a production cost. All of them require feeding from farms (population size). Any construction, upgrade, and army unit requires amounts of Gold, Iron, and Wood - the quantity resources differ based on each building type and army unit type. We define those values in your implementation.
- The game follows a real-time time pace, where building, producing, or upgrading take some game time to complete. You can define a guard time, in terms of ours, where your village is not exposed to attacks so that it can recover from a previous attack or freely upgrade without concerns.
- When attacking, your army can pillage some loot from the village your attacked. If your village is attacked, it can be pillaged and loose gold, iron, and wood. Each member of army has hitpoints and inflict damage when attacking. All your buildings have hitpoints, and defence buildings inflict damage to attacking armies. The success in an attack depends on the size of the army, including the level of its units, against the defence capacity of the attacked village. The defence capacity comes from the number and level of defence units, the size of the village, level of its buildings, and number of workers, miners, and collectors.
  - Suggestion (extra): an attack can be simulated in a fine grain level. That adds more realism to the attacks, turning the game interesting.
- Suggestion (hint): when designing your class diagram, think of the extensibility of your application/game. Think about your game having new levels (home village extensions or new levels), new units, new buildings, and new upgrades.
- Limits:
  - **Map**: your village can grow up to an certain area or to certain number of buildings;
  - **Upgrade**: All village elements can be upgraded to certain max level (damage and hitpoints);
  - **Mining**: There is a maximum (upgradeable) capacity for gold, wood, and iron;
  - **Loot**: There is a maximum loot (proportional to the size pillaged village). Loot also depends on the "success level" of an attack.
- A **Game Engine** controls all game actions, deciding if upgrades, attacks, and production are allowed. It also determines the termination (time) of upgrades and production. When a user decides to attack another village, the Game Engine must “randomly” generate possible villages, define the success of attacks, and the loot of attacks. The Game Engine also “randomly” generates an army to attack the

user's village whenever the village is in the non-guard period.
Note: *the Game Engine may be modeled as one or more coordinating classes responsible for orchestration only; detailed algorithms are not required at this stage.*

- As the **main objective** of the game, **users** need to max-upgrade their village, keep an army to attack other villages, and rank up.
  - It is up to you to define the ranking of a village. It might be related to the amount of attack wins, defence wins, and/or accumulated loot.

Assignment Details

The assignment is to design a system which facilitates an strategy game of village battles as described above and present the model using a UML class diagram. You will probably find it beneficial to have multiple packages which each handle their own distinct aspect of the game. A package is typically represented in UML by a folder. Thus, classes which are part of a package are typically drawn inside of the folder. See Figure 1 for an example.
(*Note: if the tool you chose to use does not offer package representation in the class diagram, at least include a text description in your submission, defining the packages and grouping classes in them.*)

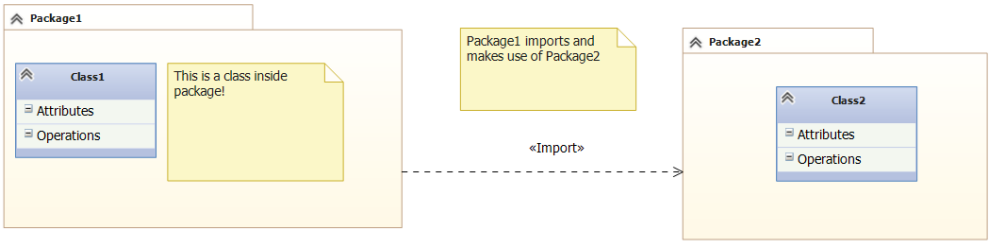

Figure 1: Example showing two packages where one package imports the other.

Your UML class diagram should include:

- Attributes and operations
- Visibility modifiers
- Inheritance and interfaces where appropriate
- Associations with multiplicities
- Packages grouping related classes

Note that **there are many feasible designs, and you will be evaluated based on the merits of yours**. Therefore, your design does not necessarily need to be perfect; it is OK to re-evaluate your design during implementation, but it must demonstrate that you have thought out and planned the overall design of the system using object-oriented principles. Also, keep in mind that you will actually be implementing the system later; thus, a well-planned design will be beneficial in the future.
Your design should facilitate, at minimum, the following behaviours:

- **Building** – the player should be able to construct any new building, following the allowed limits of the game;
- **Training** – the player should be able to train/produce habitants provided that they have the supporting structures and up to a maximum limit.
- **Upgrading** – the player should be able to upgrade any building and habitant up to a maximum level limit;

- **Generating Village** – the should be able to generate a complete village for the player to attack. This village should in similar overall compatible level as the attackers' village and army;
- **Attack Exploring** – the player should be able to choose to attack a village that the game provides. The player can keep choosing until a suitable village is found.
- **Attacking** – the game should be able to generate overall scores from the attacker's army and the attacked village. The score can be used to determine if the attack succeeds, which can be in terms of percentage; it can also give a hint for a proportional pillage loot.

Some useful notes pertaining to the assignment:

1. Given that this is only the initial design phase, **do not worry too much about supporting multiple players now**. We are mainly concerned with having a system which facilitates the overall logic of the game. However, we will, eventually, be adding multiple players in future assignments so do not pigeon-hole the design into only supporting a single player.
2. For simplicity, **do not worry much about sophisticated "generating" and "attacking" now** - they can be very simple now. However, be aware when designing your system that both "generating" and "attacking" functionalities are necessary and will be explored further in future assignments.
3. While we might be implementing a graphical user interface (GUI) in later phases, **it is not necessary to design the GUI specifics at this point**. However, you are required to acknowledge the GUI as part of your diagram. Thus, your design should include an appropriately named user interface package which interacts with your library package(s), but do not worry about the specific classes/methods it will contain at this point.
4. Recall that this course is about advanced object-oriented programming. Thus, you will want to **ensure that your design makes use of object-oriented constructs**, such as interfaces, inheritance, enumerations, generics, etc. where appropriate.

Hints and suggestions:

- **Packages**. You may want to divide your project (UML design) in packages, such as VillageMap, Game, Player(s), Main, etc.
- **Classes**. From the description and as a suggestion, the following major classes/interfaces may end up in your diagram and code: Building, Inhabitant/Person, Army, Defence, MainGame, etc.

Game Decisions and Engine

Core underlying parts of this strategic game rely on an engine deciding "fate", dictating pace, and arbitrating decisions. The following aspects should be taken in consideration when designing your simulator:

1. **Time** – The game happens in real-time. The engine must control and define the progress of the game – the time to finish village constructions and upgrades. The engine also controls the guard times – the time in which the village cannot be attacked.
2. **Score** – The game engine calculates the overall scores of a village. Its defence score based on the number, configuration, and level of defence buildings. Its attack score is based on the number, composition, and level of attack units. Its loot score is based on the attack outcome and amount of gold, iron, and wood in defending village.
3. **Success** – The engine will tell the fate of attacks, coming from either the generated army (generated by the engine) or the user's army. The success is obtained from comparing defence and attack scores with some added luck: dice rolling.

SUGGESTION: Focus on modeling the core abstractions and relationships rather than exhaustively modeling every possible detail.

Submission Material

The submission for this assignment will consist of ***THREE PARTS***:

1. A **complete UML class diagram (in a known image format: JPG, PNG, EPS, etc)** of your designed system. For you to design your UML class diagram, you can use any tool available:
   - There exist plugins for Eclipse IDE and plugins for NetBeans IDE. For both, the UML design plugins are heavy, can crash, and might be incompatible.
   - There are online tools that can be used in the browser. They are usually limited in their free license.
   - There are offline tools that you can run locally in your machine. I have already selected 4 of those tools - most of them are jar files that need JVM to be run.
     (a) Violet UML Editor - a good-enough class diagram editor;
     (b) Star UML - a good-enough class diagram editor;
     (c) yEd - it is a general-purpose diagram editor (cumbersome);
     (d) Argo UML - a good class diagram editor (it allows you to generate Java code out of your design);

   Please find these four off-line tools in the enclosed tools.zip file.
2. A ***Description document (PDF)***. A document succinctly describing your design decisions is necessary. Also, you will find it beneficial to justify your design choices such that the marker does not have to reason about why you have designed your system as you have. Make sure that the description and reasoning of your design decisions are consistent with your UML class diagram.
   - ***Latex template - a must for writing your assignment***. For writing your description file, use the Latex template enclosed in this assignment (update it accordingly!). You do not need to install Latex software in your computer. You can write it through Overleaf on your browser (it is a free tool). Just upload the latex template to your Overleaf project; it should compile/render the tex file gracefully.
3. The **respective Java code of your designed classes (in a zip file)**. The classes should match with your UML class diagram. These classes only need to have the members (non-instantiated variables) and operations (empty methods) properly mapping the relationships and functionalities. In other words, your code must contain only the declaration of classes, attributes, and methods shown in your class diagram. The classes and methods must be documented (commented). The code should compile properly. The compilation of your code should not rely on any IDE.
   - ***Compilation***. Provide the command for compiling your source code from the command line.

You are free to use any modelling software you wish so long as the final diagram is provided in a reasonable format (preferably PDF). Similarly, you are welcome to complete your assignment by hand, but you need to submit the assignment electronically, so you will be responsible for ensuring that the drawings, and any accompanying documents, are scanned (and legible) and are submitted in a reasonable electronic format. If you have any questions regarding which file types are acceptable, please inquire prior to submission. Note that it is not the fault of the marker if they are unable to mark your assignment due to submitting an unreasonable or uncommon file format.

Marking Scheme

Marks will be awarded for completeness and demonstration of understanding of the material; **Obtain needed information from the game entities/things described in Assignment Overview and game inner behaviour described in Assignment Details**.
It is important that you fully show your knowledge when providing solutions in a concise manner. Quality

and conciseness of solutions are considered when awarding marks. Every code added to the originals should be well commented and explicitly indicated in the Java files; lack of clarity may lead you to loose marks, so keep it simple and clear.

Marking emphasis:

- UML design quality and correctness
- Appropriate use of OOP constructs
- Consistency between UML and code
- Clarity of design description

Submission

Submission is to be a PDF (in case of the UML diagram and description document), and text Java files. All the submission should be performed electronically through D2L.

You must guarantee that your code is legible, clear, and succinct. Keep in mind that any questionable implementation decision or copy from any source might have a negative effect on your mark. If you still have any questions regarding which file types are acceptable, please inquire prior to submission. Note that it is not the fault of the marker if they are unable to mark your assignment due to submitting an unreasonable or uncommon file format.

All content files should be organized and put together in a ZIP for the submission through D2L.

*** Do not forget to include the names and student IDs of group members.**

**** Only one student needs to submit the assignment on behalf of a group.**

Plagiarism

Students are expected respect academic integrity and deliver evaluation materials that are only produced by themselves. Any copy of content, text or code, from other students, books, web, or any other source is not tolerated. If there is any indication that an activity contains any part copied from any source, a case will be open and brought to a plagiarism committee's attention. In case plagiarism is determined, the activity will be cancelled, and the author(s) will be subject to the university regulations.

For further information on this sensitive subject, please refer to the document below:

`https://brocku.ca/academic-integrity/`