COSC 3P91
Assignment 1

Due date: February 4th, 2022 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 Sakai.
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 a traffic simulation game, using a UML class diagram.
Also, a Java code should be provided. 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 traffic simulator where vehicles move around in a urban traffic network.

The **traffic network**, urban environment, is basically a map coded in the form of a graph, where edges are road segments and nodes are intersections. The map can be defined statically but is kept in memory as objects. A file can define the structure of the graph and be loaded at the beginning of the game (at its startup). Each road segment maybe composed of more than 0 lanes.

The **mobile entities** in the simulator are vehicles, which can be cars, trucks, or buses. As defined as a game, certain vehicles can be controlled by users while other are just controlled by the Game Engine.

The game follows a **time-stepped simulation**, where from time to time, the state of moving entities are updated accordingly and the user is prompted for a some decision. Therefore, vehicles take movement/routing decisions, such as turning left, turning right, and going straight in intersections, whenever approaching a "decision position" in the map.

As the **main objective** of the game, **users** need to conduct their vehicle properly over the traffic network to avoid collisions.

Assignment Details

The assignment is to design a system which facilitates a game of traffic simulation 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 that **there are many feasible designs and that you will be evaluated based on the merits of your design**. 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:

- **Looking** – the player should be able to request information about its vehicle's surroundings.

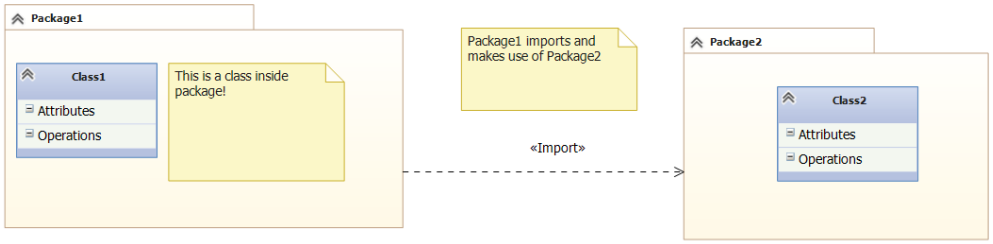

Figure 1: Example showing two packages where one package imports the other.

- **Moving** – the player should select a decision about next moving decision: changing lanes (left, right), turning in intersections, or keeping straight.
- **Asserting** – the decision of the player should be verified so that the decision means a consistent state. A problem could trying to enter a road segment in the wrong way or that is full of vehicles.
- **Gambling** – together with asserting, it allows the player to take chances. It gives risky opportunities to the player. Thus, by rolling dices, it infers if the player was lucky or not. For instance, the player decides to turn into a road segment that is full but there is a chance that, in next turn, the segment will vacant a spot.

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. 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 gambling now. However, be aware when designing your system that gambling functionality may become necessary in future work.
3. While we will be implementing a graphical user interface (GUI) in later phases, it is not necessary to design the 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 Vehicles, Map, 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: Vehicle, Map, RoadSegment, Intersection, MainGame, etc.

Movement Decisions Several situations in the simulation trigger moving objects/entities for decisions that might force them to change their direction or allow them to keep the same moving trend there were following. Such situations maybe be clustered in two major occasions, spatially represented on the simulation Map: **intersections** and **middle of lanes** (lane changing). The following aspects should be taken in consideration when designing your simulator:

1. **Intersection Decision**. When at an intersection, the vehicle may opt for moving straight ahead, turn to the left or turning to the right. The decisions need to follow the physical constraints of the map:

vehicles are not allowed to turn left a some intersections.

2. **Lane Changing**. Vehicles while moving in a road segment, they will have the chance to be change lanes in case there is too much traffic ahead slowing its movement. In case there is a traffic jam, the player will have the chance to change lanes several times while it has not covered the whole road segment yet. I some other situations, the player will be prompted for intersection decisions in case there is low traffic density and the player's vehicle can cover the road segment really fast.
3. **Challenge**. Vehicles at the intersection will compete for their right to go straight or turn. A player may feel lucky and try to move without caring about other vehicles. This action will be lead to a Challenge, which is somehow Gambling, and the Game Engine will roll dices and tell how is the winner in the Challenge.
4. **Reputation**. Each vehicle has an overall reputation in the game. It grows as right, conservatives decisions are made. It decreases as the player looses Challenges or when it gambles. All vehicles controlled by the Game Engine may present the same Reputation during all the simulation; they may only take conservative decisions all times. The Reputation can range from predefined MIN to a MAX values.
5. **Reputation Value**. With higher Reputation, a player has a better chance to win Challenges and win when Gambling.
6. **Damage**. Any movement decision may lead to an accident, which necessarily will damage the vehicle. The damage may be determined probabilistically (rolling dices) and according to several factors: status of vehicle, speed of vehicle, movement situation, etc.

Submission Material

The submission for this assignment will consist of two parts:

1. A **complete UML class diagram** 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**. 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. 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.

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 Sakai.
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 Sakai.

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

Late Assignment Policy

A penalty of 25% will be applied on late assignments. Late assignments are accepted until the Late Assignment Date, three days after the Assignment Due Date. No excuses are accepted for missing deadlines. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion.

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/`