COSC 3P91
Assignment 4

Due date: April 11th, 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

The assignment consists of turning the current implementation of the Traffic Simulation Game into a networked application. The game makes use of a console as an interface for the interactions of a player. The console is basically System.in/out for I/O communication with the player. Ideally, your current implementation, obtained from Assignment 3, will form the basis of your modifications. In this updated design, the game will contain a server that runs the simulations and continuously interacts with a client. The client is responsible for running on the player/user side and permeating all the interactions with the simulation.

The Game

The **game** is a simplified traffic simulator where vehicles move around in a urban traffic network. The game is a turn-based simulator that updates the position of vehicles each "time step".

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

Redesign of the Traffic Simulation Game so that it contemplates a Client/Server paradigm. All the game functionalities, requested from Assignments 2 and 3, should be still provided in the upgraded code of Assignment 4. The Game should allow *Looking*, *Moving*, *Asserting*, and *Gambling*; the movement decisions should be performed at intersections and middle of lanes, considering *Intersection Decision*, *Lane Changing*, *Challenge*, *Reputation*, *Reputation Value*, and *Damage*.
The **Model View Controller** (MVC) pattern applied in Assignment 3 should facilitate the modifications in the code so that the client representing the player is decoupled from the server (the simulation).
The students will have to choose the type of sockets they need to use for supporting this networked game. The decisions must be always justified. Students must implement a standard method that the client will follow to interact with the server. This standard method may be serialized objects or a text-based messages (coded in JSON, for example).

On the client side, the application should authenticate the user with the server under a handshake-like protocol where the server must check in a database that the user/player exists in the system. There is no need to implement sophisticated encryption methods for this authentication, just text-based comparisons suffices this purpose.

On the server side, the game still continues to run in time steps. However, students will have to employ threads to parallize the execution of game when updating position of vehicles during a time step (round). It is up to the students to decided the suitable number of threads, use of thread pools, split of load (which vehicles) by thread, and so on. Keep in mind that all decisions must be justified in the description file.

Important Notes

1. A network-based game will require to split the current implementation in two decoupled parts: client and server. This split must be coherent, involving the minimum network communication overhead on the simulation execution.
2. Students can choose any underlying socket type (TCP or UDP) to support the communication between a client and the server. The choice must be justified.
3. There is no need to have multiple clients. It is not a requirement for this implementation. However, it would interesting if the game would allow to have multiple players at the same time.
4. Students must choose how the communication messages will happen between the client and server will happen. It can be through sending (i) serialized objects or through (ii) text-based messages.
5. Students must define a language/protocol regulates how the client talks with server. There should be a standard language, which includes all possible options a player may have in the game.
6. The game may run without a client, just as a pure simulation.
7. The server needs to target performance. Thus, it will have to use multithreading for speeding up the movement updates of vehicles during a time-step. The students need to decide how to split load among threads, as well as other implementation details. Keep in mind that the simulation must be coherent all times.
8. **Reiterating** – Recall that this course is about advanced object-oriented programming. Thus, you will want to ensure that your implementation makes use of object-oriented constructs, such as interfaces, inheritance, enumerations, and generics, where appropriate. Such constructs should have been identified during design.
9. **Reiterating** – Be sure to include sufficient comments which should consist of, at minimum, an appropriate comment for each file and method. Consider using Javadoc style comments. If you use NetBenas, Javadoc comments can be auto-created in NetBeans by placing the cursor above a method or a class that currently has no Javadoc comment, typing "/**", then pressing Enter. This will create a template Javadoc comment with some information automatically completed for you (e.g., parameters for a method).
    - Your comments, if done correctly, should facilitate writing the document that explains your code. The comments should match with what you have in your description document.
10. Besides the Object Orientation concepts, you must be able to show complete understanding of the following concepts and employ them in your code. **Adequately use them whenever possible and explain them, pointing out in your description file** (in other words, marking will consist checking the existence and proper use of each of these elements):
    (a) Coherent modification of code into a Client/Server paradigm;
    (b) Proper use of sockets;
    (c) Proper communication between client and server (user authentication and simulation execution, for ex);
    (d) Proper and consistent multithreaded execution of the server.

Submission Material

The submission for this assignment will consist of two parts:

1. A ***Description document (PDF)***. A document succinctly describing your design and implementation

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 decisions are consistent with your Java code.

- ***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.

2. The **Java code** implementing design in previous assignment or the one provided. The code should follow the UML class diagram. The code also must address the OOP concepts listed above. The classes and methods must be documented (commented). The code should compile and run properly. The compilation and execution of your code should not rely on any IDE.
    - ***Compilation***. Provide the command for compiling your source code from the command line.
    - ***Running***. Provide the command for running your compiled code from the command line.

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.

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 (description document), and text Java files (Java code). 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/`