Part C - Amazing Escape

SWEN30006, Semester 1 2017

Overview

After seeing your review of their metro simulation design, *Shoddy Software Development Company* were outraged and sought revenge. They organised to have you kidnapped, and abandoned in a dangerous location, with the expectation that you would not survive your attempt to return to civilisation.

Awaking in a strange vehicle in an unfamiliar location surrounded by dangerous traps, you quickly realised that attempting to drive out manually would soon prove fatal. However, *Shoddy* did not account for your software design and development skills. You quickly connected your laptop (conveniently left with you) to the vehicle, coupled it with the vehicles sensors and actuators, and integrated your tailor-designed vehicle auto-drive system. In no time at all (well, before the due date at least) you were on your way, the vehicle automatically navigating its way out of the maze, as quickly as possible*, while avoiding+ the traps.

The Maze

The maze you find yourself in is a 'perfect maze' (no loops) constructed in the form of a simple grid of tiles consisting only of:

- Roads
- Walls
- Start (one only)
- Exit (one only)
- Traps

As your would expect in a maze, some roads will lead to dead-ends. As the roads will not all be the same width, you'll need to be on the lookout for narrow dead-ends where it is hard to get back out.

The Vehicle

The car you find yourself in includes a range of sensors for detecting obvious properties such as the car's speed and direction, and actuators for accelerating, braking and turning. It has a maximum speed (which is slower in reverse) and a limited turning rate. The car also includes a sensor that can detect/see three tiles in all directions (that's right, it can see through walls). The car can be damaged by events such as running into walls, or through traps (see below). It has limited health; if the car's health hits zero, *Shoddy* wins and you lose. Similarly, if the car is stuck in a trap and can't continue, you lose. If however, your software takes the car from the start to the exit, you win and live on to write more damning software design reviews.

*The time it takes the car to reach the exit and the final health of the car will be combined to provide a score for your amazing escape.

The Traps

There are different types of maze traps which have different effects on your vehicle; the effect continues as long as the vehicle is in the trap:

- *Lava*: damages your vehicle.
- *Mud*: slows your vehicle down.
- *Grass*: causes your vehicle to slide (no change in direction possible).

A lava trap may destroy your vehicle and a mud trap may stop and hold your vehicle, either way resulting in failure. A grass trap is not a problem in its own right, but where you end up when you leave the grass trap may be a problem.

+ Some traps will be avoidable (i.e. can be by-passed), but others will have to be traversed with some consideration as to the resulting damage and to where the car ends up.

The Task

Your task is to design, implement, integrate and test a car autocontroller that is able to successfully traverse the maze and its traps.

It must also be able to choose between and apply three different approaches to handling a dead-end:

1. U-turn
2. 3-point turn
3. Reverse out

A key element is that your design should be extensible to allow for it to deal with other trap types and additional or faster traversal strategies. You will not be assessed++ on how sophisticated or fast-traversing your algorithms are, beyond whether they work to get the car to the exit (++other than for the bonus mark for the top 5% based on total score).

The package you will start with contains:

- The simulation of the world and the car
- The car controller interface
- A car manual controller that you can use to drive around a maze
- A working but basic car autocontroller that can navigate a maze but ignores traps and can only u-turn in dead-ends
- Sample mazes (created using the maze editor Tiled)

If, at any point, you have any questions about how the simulation should operate or the interface specifications are not clear enough to you, it is your responsibility to seek clarification from your tutor, or via the LMS forum. Please endeavour to do this earlier rather than later so that you are not held up in completing the project in the final stages.

To achieve your task, you will be completing two submissions. The first will be your design. The second will be your implementation. The detailed task break down for these submissions follows.

System Design

You have been provided with a design class diagram for the interface to the car controller and other simulation elements on which it depends. Your first task is to understand the provided interface and specify your own design for a car autocontroller. As always, you should carefully consider the responsibilities you are assigning to your classes.

This design should consider all relevant software design principles used in the subject thus far. You are required to provide formal software documentation in the form of Static and Behavioural Models. Note that you are free to use UML frames to help manage the complexity of any of the design diagrams.

You may build on the existing autocontroller design or start afresh; either way, you will need to justify the end result (see below).

Static Models

You must provide a Design Class Diagram for your implementation of the car autocontroller subsystem, including the interface. This design diagram must include *all* classes required to implement your design including all associations, instance attributes, and methods. It should *not* include elements of the simulation, other than your autocontroller and the interface.

Behavioural Models

In order to fully specify your software, you must specify the behaviour along with the static components. To do this, you must produce sequence diagrams and a communication diagram showing how elements of your subsystem interact. For your sequence diagrams, you must present the following two use-cases.

Controller Functionality
Detecting and progressing from a dead-end
Detecting and avoiding/traversing a trap

Your sequence diagrams must be low level sequence diagrams (i.e. show methods and arguments) which are consistent with your static models. Further to this, you must provide a communication diagram that shows all communication between your components within your subsystem.

Design Rationale

Finally, you must provide a design rationale, which details the choices made when designing your autocontroller and, most critically, **why** you made those decisions. You may want to apply GRASP patterns, or any other techniques that you have learnt in the subject, to explain your reasoning. Keep your design rationale succinct; you must keep your entire rational to between 1000 and 2000 words

Design Submission

You will submit a final design which includes all elements as specified in the submission checklist.

Implementation Submission

Your final task is to provide a working implementation of your subsystem. You must implement the interfaces provided within the simulation package. This implementation must be consistent with your submitted final design; if this is not possible due to a flaw in your final design, your implementation may vary from the design to correct this flaw and this inconsistency with your design must be clearly noted in comments in your source code. Please also note that the simulation and the car controller interface must not be modified in your submission (though you may wish to add trace to it or make other similar changes to support your testing).

You should ensure that your system provided is well commented and follows good Object Oriented principles.

Gradle

Your first implementation task should be to import the project, and build and run the application, to ensure that you have the existing simulation running without issue. The provided simulation makes use of Gradle as a build and dependency management tool. Instructions for installing the Gradle plugin in Eclipse were provided for Project Part B; if you have already completed the installation of the gradle plugin for Part B, you will not need to do it again.

Version Control

It is **strongly** recommended that you use *Git* or a similar system for version control. Using a version control system makes it much easier to work as a team on a complex project. The Melbourne School of Engineering runs servers for vanilla Git (http://git.eng.unimelb.edu.au/) and for bitbucket (https://bitbucket.cis.unimelb.edu.au:8445/), and there are cloud-based servers freely available (https://bitbucket.org/).

If you do use version control, please ensure you have set your repository to **private** so that other students in the subject cannot find and copy your work.

Building and Running Your Program

Your program must be importable using the same setup we have provided you to run the project. We will be testing your application using the desktop environment.

Submission Checklist

This checklist provides a comprehensive list of all items required for submission for the each part of this project. Please ensure you have reviewed your submissions for completeness against this list.

Design

1. Ensure you have added your group number to all documents
2. Create a ZIP file including the following design elements for your Car AutoController:
    a. 1 Design Class Diagram
    b. 2 Sequence Diagrams
    c. 1 Communication Diagram
    d. 1 Design Rationale (1000-2000 words)

Implementation

1. Ensure you have added your group number to all documents
2. Zip your package folder including all required src files.

Marking Criteria

This project will account for 15 marks out of the total 100 available for this subject. These will be broken down as follows:

Final Design

The Part I Design Submission counts for 10 of the 15 marks available for this project, broken down as follows:

Criterion	Mark
Class Diagram	2 Marks
2 Sequence Diagrams	4 Marks
Communication Diagram	1 Mark
Design Rationale	3 Marks

We also reserve the right to deduct marks for incorrect UML syntax and inconsistencies within your diagrams.

Implementation

The Implementation Submission counts for 5 of the 15 marks available for this project, broken down as follows:

Criterion	Mark
Functional Correctness	2 Marks
Design Compliance and Code Quality	3 Mark

We also reserve the right to award or deduct marks for clever or poor implementations on a case by case basis outside of the prescribed marking scheme.

Further, we expect to see good variable names, well commented functions, inline comments for complicated code. We also expect good object oriented design principles and functional decomposition.

If we find any significant issues with code quality we may deduct further marks.

On Plagiarism

We take plagiarism very seriously in this subject. You are not permitted to submit the work of others under your own name. This is a **group** project. More information can be found here: (https://academichonesty.unimelb.edu.au/advice.html).

Submission

All document submissions must be made to the LMS using the provided links on the project page. Instructions for submitting the implementation will be provided seperately. Your document submissions for each part must include all items as required in the submission checklist. Your document submissions (whether they are design documents or reports) **must be pdf documents**. Documents in any format other than PDF *will not* be considered during marking.

You must submit one **zip** file for each submission containing all required items. You must not use any compression format other than **zip** for your submission. Other archive formats will not be considered for marking.

Every item you submit **must** contain your LMS Group number for identification purposes.

If you have any questions at all about submission, please contact your tutor *before* the submission due date.

Only one member from your group should submit your project.

Submission Date

- Design is due at **11:59 p.m. on the 23rd of May.**.
- Implementation is due at **11:59 p.m. on the 28th of May.**.

Any late submissions will incur a 1 mark penalty per day unless you have supporting documents. If you have any issues with submission, please email William at tiow@unimelb.edu.au, before the submission date.