Part C - Learning to Escape

SWEN30006, Semester 1 2018

Overview

After seeing your review of their Automail 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 across the map to safety, as quickly as possible*, while avoiding+ the traps.

The Map

The area you find yourself in is a constructed in the form of a simple grid of tiles consisting only of:

- Roads
- Walls
- Start (one only)
- Exit (requires key #1 to open)
- Traps

Some roads may lead to dead-ends or be impassible due to traps and the health of your vehicle.

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 four tiles in all directions (that's right, it can see through walls), ie. a 9x9 area around the car. 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. If however, your software takes the car from the start to a succesful exit, you win and live on to write more damning software design reviews.

*The time it takes the car to reach the exit will provide a score for your escape.

The Traps

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

- *Lava*: damages your vehicle, but holds the key to your escape (pun intended).

A lava trap can destroy your vehicle, resulting in failure. However, some of the lava traps contain a key in a locked box; the only way to retrieve the key is to enter the lava. In stereotypical fashion, the villains are so confident of their plan that they have provided you the key to the highest numbered box. Finding and retrieving the keys in sequence is the only way out.

- *Health*: repairs your vehicle.

The villains provided these just to give you false hope.

+ Some traps might be avoidable (i.e. can be by-passed), but others will have to be traversed (e.g. those which contain keys) 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 map and its traps.

It must also be capable of safely:

1. exploring the map and locating the keys
2. retrieving the keys in order
3. making its way to the exit

A key element is that your design should be modular, clearly separating out elements of behaviour/strategy that the autocontroller deploys (see Design Rationale below). 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 map
- A simple car autocontroller that can navigate a maze but ignores traps and can get stuck in a loop if the map is not a 'perfect maze' (if it has circuits).
- Sample maps (created using the map 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.

Your task has two main parts. The first is your design. The second is your implementation. A more detailed description 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 ("AIController.java") design or start afresh; either way, you will need to justify the end result (see below).

Static Model

You must provide a Design Class Diagram (DCD) 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.

You may use a tool to reverse engineer a design class model as a basis for your submission. However, your diagram model must contain all relevant associations and dependencies (few if any tools do this automatically) and be readable; a model that is reverse engineered and submitted unchanged is likely to receive a low mark.

Behavioural Model

In order to fully specify your software, you must specify the behaviour along with the static components. To do this, you must produce a Communication Diagram (CD) showing how the elements of your subsystem interact, that is, a communication diagram that shows all communication between the components within your subsystem. You do not need to show message ordering on this communication diagram.

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, GoF 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. You may include diagrams (additional to the DCD and CD) if that helps with your explanation.

Implementation

You will submit a working implementation of your subsystem "MyAIController" (stub provided) in the Java package "mycontroller". You must work to the interfaces provided within the simulation package. Please also note that the simulation and the car controller interface must not be modified to support your submission (though you may wish to add trace or make other similar changes during development to support your testing). Your implementation should be consistent with your submitted final design.

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

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 and for Bitbucket, and there are cloud-based bitbucket servers freely available.

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

We will be testing your application using the Windows desktop environment, and need to be able to build and run your program automatically. The entry point must not change, and you must work to the provided interface. You must not change the names of properties in the provided property file or require the presence of additional properties.

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.

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 (pdf or png)

b. 1 Communication Diagram (pdf or png)
c. 1 Design Rationale (pdf: 1000-2000 words)
d. A folder called "mycontroller" containing:
   i. all the Java source files for your implementation of "MyAIController"
   ii. only elements which are defined in the package "mycontroller"

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:

Design

The Part I Design Submission counts for 9 of the 15 marks available for this project, broken down as follows:

Criterion	Mark
Class Diagram	2 Marks
Communication Diagram	2 Mark
Design Rationale	5 Marks

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

Implementation

The Implementation Submission counts for 6 of the 15 marks available for this project, broken down as follows:

Criterion	Mark
Functional Correctness	3 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 and other forms of academic integrity very seriously in this subject. You are not permitted to submit the work of those not in your group.

Submission

All document submissions must be made to the LMS using the provided link on the project page. Every item you submit **must** contain your LMS Group number for identification purposes.

Your design documents must be in the specified format; documents in other formats *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.

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

- This project is due at **11:59 p.m. on Friday 25th 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 Peter at peter.eze@unimelb.edu.au, before the submission deadline.