Part C - Learning to Escape

SWEN30006, Semester 2 2018

Overview

After seeing your design report on the Automail simulation system, the *Robotic Mailing Solutions Inc. (RMS)* software developers were outraged at your implied criticism of their design. Seeking 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, the *RMS* developers 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, while avoiding+ the traps.

The Map

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

- Roads
- Walls
- Start (one only)
- Exit (requires a set of keys 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 only one speed (in both forward and reverse) and can only turn towards one of the four compass points. 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, the *RMS* developers win 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 critical software design reports.

The Traps

There are 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; the only way to retrieve the key is to enter the lava. You need a copy of each distinct key to make it out; the car has

slots for the keys so you know how many you need to collect. In stereotypical fashion, the *RMS* developers are confident of their plan, so confident they may even have left duplicates of some keys.

- *Grass*: bit slippery, so can't steer. The grass won't damage the car, but what comes after the grass might.
- *Mud*: bit sticky. In fact, the car can't drive out of it. So basically, drive on it and you lose.
- *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 be capable of safely:

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

A key element is that your design should be modular and extensible, clearly separating out elements of behaviour/strategy that the autocontroller deploys (see Design Rationale below). In particular, consideration should be given to additional trap types, and the potential to vary behaviour (possibly in a future version) in circumstances where options are available. You will not be assessed on how fast-traversing your algorithms are, just on the design in which they are embedded and on whether they work to get the car to the exit.

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.

Following are more details on specific aspects of your task.

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 relevant aspects of the simulation behaviour.

You will then specify your own design for a car autocontroller. As always, you should carefully consider the responsibilities you are assigning to your classes, and in particular, you should apply relevant principles/patterns covered 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") or start afresh; either way, you will need to justify the end result (see Design Rationale 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 reduce your 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 should refer to GRASP patterns, GoF patterns, or any other techniques that you have learnt in the subject, where relevant, 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 (in addition 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.

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

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 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 list of all items required for submission for this project. Please ensure you have reviewed your submission for completeness against this list. *Instructions for submitting will appear on the LMS.*

1. A file called group.txt listing your group number and group members' names (documents/diagrams should also include your group number)
2. 1 Design Class Diagram (pdf or png)
3. 1 Communication Diagram (pdf or png)
4. 1 Design Rationale (pdf: 1000-2000 words)
5. A folder called "mycontroller" containing your car AutoController:
   a. all the Java source files for your implementation of "MyAIController"
   b. 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. It will be assessed against the following rubric:

- H1+ [14-15] Solves all test maps. Good extended or extendable design/implementation with very consistent, clear, and helpful diagrams. Very clear design rationale in terms of patterns. Very few if any minor errors or omissions (no major ones).
- H1 [12-13.5] Solves all test maps. Good extended or extendable design/implementation with generally consistent, clear, and helpful diagrams. Clear design rationale in terms of patterns. Few if any minor errors or omissions (no major ones).
- H2(AB) [10-11.5] Solves most test maps. Extended or extendable design/implementation with generally consistent, clear, and helpful diagrams. Fairly clear design rationale in terms of patterns. Few errors or omissions.
- P-H3 [7.5-9.5] Solves some test maps. Reasonable design/implementation with generally consistent and clear diagrams. Fairly clear design rationale with some reference to patterns. Some errors or omissions.
- F+ [3.5-7] Plausible implementation and reasonable design with related diagrams. Plausible attempt at design rationale with some reference to patterns. Includes errors and/or omissions.
- F [0-3] No plausible implementation. Design diagrams and design rationale provided but not particularly plausible/coherent.

We also reserve the right to award or deduct marks for clever or particularly poor implementations on a case by case basis.

> **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 Date

- This project is due at **11:59 p.m. on Friday 19th of October.**.

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 Marko at marko.mihic@unimelb.edu.au before the submission deadline.