We first decide to use strategy pattern for deciding the route of the car. This is because there might be multiple strategy for finding the route we want to test. If instead of using strategy pattern, but use just a couple of function, we may end up keep changing a certain part of the code for strategy change. This will bring confusion of the program and reduce the efficiency in experimenting. However, by applying strategy pattern, we can just do things like plug in a module and plug out-all we need to do is change which strategy class we want to load at the start, without changing multiple place in the program for different strategy. And we can contain the detail of the strategy in different class, keep it for future comparison. Whereas all these different strategy will share the same interface, keep the logic clear and easier to change the strategy and conduct experiment. This also reduce the coupling in the system- if we change the strategy, there is no need to change other part of the code in the system.

We also use the controller pattern in the MyAIController class, instead of putting car control, route decision function in the class like AIController did. MyAiController servers as a indirect class between the Simulation class and its Modules. To be Specific, MyAIController take the information input from the Simulation, pass to modules to make decision, then modules control the car’s action through the MyAiController. By Applying the controller pattern, though we increase the coupling of this single class, we reduce the system’s overall coupling and increase the cohesion in the system. Also, for the principle of protected variation, we put functions that are stable and unlikely to change in the controller, while put different functionality that might need to be changed frequently in its module. Using controller pattern not only increase the cohesion of other modules, it also makes us easier to separate the functionality of project and allocate the job to teammates. Each of us can do whatever change we want in the module, whereas not breaking the program. This also avoid the conflict on our repo.

As mentioned in above, there are several modules for our MyAIController. Technically, they are simple class. MyAIController has three main class, PerceptionClass, DecisionClass and ActionClass. PerceptionClass is responsible for updating the map of the maze, DecisionClass is responsible for deciding the route for car and ActionClass is responsible for how the car is actually controlled. This design took some idea of a classical robot system. We realized that this project we are not creating controller for a car, rather, it is more like a robot. Therefore, for a typical robot, it has three process: processing surrounding information, making decision, controlling its action. In this project it cooresponds to:

|  |  |  |
| --- | --- | --- |
| Processing surrounding |  | Reading the types of lava and where it is, also see whether we saw a key (PerceptionClass) |
| Making Decision |  | Deciding the route and next coordinate car need to go (DecisionClass) |
| Control body to make action |  | Optimised car speed, how it turn (ActionModule) |

This design supports our controller pattern and help reduce the representation gap.