**Group: 77**

**Team Member: Siyi Guo, Mofan Li, Jihai Fan**

We 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 to support its action, PerceptionModule class, DecisionModule class and ActionModule class. We use robot’s AI Structure as a reference, therefore decide this modules and controller’s algorithm. A typical robot AI’s decision process has following steps: receiving the information of surrounding, making plan to achieve the goal and decide the next action, apply action using its control algorithm to move. This coorespond to the responsibility of our 3 modules, Perception, Decision and Action. By doing so, we increase the cohesion of the system, lower the representation gap between our intuition of the class,

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 corresponds the responsibility of Perception, Decision and Action class. Using this design help us lower the representation gap between our intuition and the program. We can also separate the task and even take reference of the classical Robot AI’s algorithm and implemented into our program.

At first, we thought we should apply Strategy pattern on the Decision Module. However, after some coding, we found out that Action Module need Strategy Pattern More than Decision Module need. Therefore, we decide to use Strategy Pattern on Action Module, and use Decision Module as normal class to avoid over complicate it. This is because of that, Action Module which corresponds to the Control algorithm of the car, need more adjustment, changing strategy than the Decision Module. For example, how we stop the car, how car should accelerate, how car should stop before turning, how we need to turn. Therefore, in Action Module, we create two Strategy class, StraightLine Strategy and TurningStrategy. In order to support this strategy pattern, we used Polymorphism. We create two interface, one for straight line moving, one for Turning. By doing so, we can try different control algorithm, all implements this interface, but have different core inside there own class. This increase the cohesion of the class and also adding the flexibility in changing the module during our experiment.

In the Decision Module, we applied Pure Fabrication to decrease the cost of changing program and did not use the strategy pattern. This is because we figure out there is not much algorithm we are going to apply. The best algorithm are A star for exploring the map and Dijkstra for go to a specify destination. Therefore, we just put them as a separate function in the Module. However, we did apply Pure Fabrication and created class Route, Position. Though this are not part of our domain diagram, we the package already has similar class. We found out that some of the package’s classes’ data type does not support our program. Therefore, we wrote our own class to pass the information, as well as to protect future variation – in case we need to pass more information between systems yet we do not want to change system’s interface. Therefore, by using pure fabrication., we not only customized our data, but also save future changes on multiple function’s input if we want to pass more functions.

More Over, we create a Perception Module to increase the cohesion of Controller. The responsibility of Perception Module is to get the surrounding information and update it. It serves as a database for the knowing position. Also, if other module want to know the information about the map, they can call Perception module’s method to know what type of the tile it is. Though this increase the coupling to Perception, it also increase the cohesion of the other two class Decision and Action module. And it makes sense for a robotics, if the robot want to see something about the surrounding, it will ask its sensors, instead of the decision module has a sensor connect to it directly.

Besides, there is a small detail in our class implementation of threashold numbers. Instead of all putting threashoold number at the top of class and declare as a static method, we put them as local variable in different methods. This is because, in our Action class design, there are so many parameters for different actions. Simply grouping them together will not increase the efficiency but confusion. But putting them together with their corresponding function, we know immediate what we should adjust when we see the function.

To Make conclusion, the big structure of our sub system is defined by controller pattern and strategy pattern. MyAIController class serves as a controller and creator of 3 modules: Perception, Decision and Action. We keep the stable code inside the controller to interact with the simulation, where as put different module’s strategy in its module. In order to support experiment of different strategy, we use the strategy pattern in the class that need a lost of adjustment. The strategy pattern is achieved by polymorphism, we define a common interface then different strategy class just implements it to be used by the Module. Therefore, we not only can change the strategy fast, but also keep a copy of all the strategies.