Design Rationale

Introduction

The overall requirement of our task is to implement a car auto controller to traverse the map and its traps, including exploring a map, locating the keys, retrieving the keys in order, and finding its way to exit with sufficient health. This report aims to clarify and critically analysis our design choices. In detail, this report will discuss the analysis phase and design decisions made by our team in the analysis phase.

Our task in the design

The project provided by our lecturer is provided with tiles in the map, utilities including Coordinate, peekTube, and world including car, world and world spatial. We are required to implement our own controller, enabling the car to find its way to exit with the collecting keys in order.

How we design our controller

The overall design of our controller has four classes and two strategy patterns. These class are Drive, carStatus, MapRecorder, MyAIController and OperationType, and two strategies are the strategies of PathDiscovery and strategies of PositionStrategy.

To be more specific, MyAIController is for updating the car (handling the operation). Drive is an information expert that is to determine which way to go with the information from map recorder and next position strategies. CarStatus is a class that record status of the car, its angles, positions and health. MapRecorder is for recording information from the map, and with these information, help the car to find the path by referring different path strategies in different situations. The operationType is to list different operations, including break, forward, backward, turning north, turning south, turning east and turning west.

As for the position strategies, we have exitPositionStrategy (handling the way out of the car), ExplorePositionStrategy (handling the car exploration), HealPositionStrategy (handling the situation of low HP of the car), KeyPositionStrategy (handling the way to locate the key in order)

As for the PathDiscovery strategies, it has AStarStrategy and Node, providing the strategy for the map recorder in determining the path from current position to target position. This strategy is implements by the interface of IDiscoveryStrategy. At the moment, we only have one strategy for path calculation, but in the future, it can be more than one path discovery strategies in our system, so the interface is preparing for the future usage.

Why we made this decision for controller design

Use strategy pattern with referring to GoF

When we approach this task in our discussion, we noticed that to find the way out, the car has several tasks to do. This first is to explore the map to locate the keys. Second is to get the keys in order, and finally is to make the ways out. In the whole process, the car has also to maintain a sufficient HP. In order words, the car has to make different strategies during the task. To solve this problem, we define each strategy in a separate class, with a common interface.

Therefore, we came up with position strategies, handling and determining the car's next position according to different situations as mentioned above. With introducing the strategy pattern, it is easier to modify each strategy in separate class. It is also easier to extend our performance if we come more advanced strategies for the task.

Path strategy
At the beginning, we created a class called Path to handle with the task of locating the way from one point to the other point. However, after discussion, we found that it was illogical that the Path class find the path, and it was more logical for the map recorder to find a path with its sufficient information about the map. Therefore, we came up with a new idea that the map recorder is to handle path discover, and path only provides strategy for map recorder.

To be more specific, Path strategy provided solution for the MapRecorder class to determine the path from one point to another point, such as from car current position to the key position or healing position. We believed that the MapRecorder class is able to handle how the car find its way by using function such as findPath, however, its decision can be different according to a specific situation. Therefore, we came up with the path discovery strategies pattern, handling these various situations.

Use factory pattern
With many strategies in our design, we came up with an idea that we can apply the factory strategy pattern in our design. The reason behind is that to create instance for each strategy is complicated and it’s hard to manage when strategy classed are changed in the future. With the implementation of our factory pattern called NextPositionFactory, we are able to instantiate the class that we want.

In addition, with the factory strategy, we are able to implement our composite strategy pattern. In detail, in using the strategy such as key position strategy, exit position strategy and explore position strategy, we also can apply heal position strategy as long as the HP lower than 30 and the map has health trap.

The NestPositionFactory is also a good way of decoupling, by instantiating the strategies within a specific class.

The information expert
After analyzing the AIcontroller provided by our lecturer. We notice that the controller is not good enough to because it is only for assigning the task. So, we re-designed it, and introduced an information expert called Drive. The class has the information of the keys, exits, healing traps. Also, the class can also inform the our MyAIController for the operation. The benefit of inWith this information expert, the controller need to call handleOperation only, and handle Operation will get the operation for the expert.

Controller - only assign task - HighCohesion
As menttioned above, we need a better controller that only assign tasks. The AIController, however, do everything for the car, including exploring the map and operations. This controller cannot meet the GRASP requriement of high cohesion and low coupling.

So, we create a new controller called myAIController and overwaited the update functions that handled the operation function by referring the drive for instruction. With the new controller, it only assigns tasks for the information expert - Drive. Therefore, it achieves high cohesion and low coupling.

Decoupling to achieve high cohesion
With decoupling in mind, our group try to make different classes such that each class is high corelated. For example, the   carStatus class referring the car's position, health and angle, which created by myAIController, rather in within the controller class. Moreover, the class of OperationType only handle with the car's turning, acceleration and break, which provides a high cohesion and reducing the coupling in the class of myAIController.

Conclusion
All in all, our implementation is based on our modelling and pattern design, and our design is guided by GRASP patterns and GoF patterns. In summary, we have strategy patterns for path discovery and position guidance. Factory pattern and composite strategy are also being applied in the position strategy. We have an information expert called drive, handling different situations, which assigned by myAIController. To achieve high cohesion and low coupling, we have operation type class handling various operations types, and we have map recorder class, handling all kinds of map information about keys, map tiles, exit and path to a specific point.