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, retriveing the keys in order, and finding its way to exit with sufficient health. This report aims to clarify and critically analysis our design choices.

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

To be more specific, MyAIController is for updating the car (handling the operation). Drive is a information expert that is to determine which way to go with the information from map recorder and next position strategies. 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)

**Why we made this decision for controller design**

**Use strategy pattern**

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.

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.

**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 its 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.

**Information expert**

After analyzing the AIcontroller provided by our lecturer. We notice that the controller is not good enough that only assign the task. So we

**Controller - only assign task - HighCohesion**

**Decoupling**