Design Rationale

Project Group 13
Christopher Chapman 760426, Liam Whittle 832406, Raymond Sun 831681

Introduction

The main design complexity in a solution to this project is focused around decision-making. This includes higher-level decisions, such as what our current objective is or what area of the map to explore next, and lower level decision making such as when to apply acceleration and turning such as to achieve our next destination. Our design approach therefore considered the need to be able to make such decisions given a variety of constantly changing information and how we could best encapsulate information and delegate responsibilities. The main components of our system include MyMap, an internal representation of the map, allowing for a constantly updateable view of the car's current and previous surroundings. A chooseDestination strategy interface, allowing flexible implementation of different strategies for comparison and for different situations. A search class, containing different strategies in order to get from A to B and finally, a drive class in charge of the low level driving operations. Through this division, our team achieved our focus of achieving high flexibility and good delegation.

Class Design: MyMap

In order to track changes as we explore more of the map, store important information to help us to complete a level and provide helpful methods for traversing the map (eg. getDistance*)*, we created the MyMap class. MyMap, contains a HashMap of coordinates and tiles (map) which holds our current view of the world (i.e. is constantly updated) as well as one called providedMapTiles which remains as a reference for the initial map data provided. We also store and update a list of key locations. While not strictly necessary to keep a separate list of keys, it significantly reduces code complexity, has better delegation, and accessing keys supports the non-functional requirement of efficiency. Other classes frequently need to query various map data, including subsets of the map such as unseen and adjacent coordinates, key locations and tile values. Since all these functions require only the query input and data stored in MyMap, we chose to keep MyMap's internal lists and HashMaps as private attributes and gave these querying responsibilities to MyMap by the Information Expert GRASP principle. This also supports data hiding and encapsulation and means MyMap's implementation of its update and query functions are hidden from outside code and manipulation.

Decision Making Design Approach: Composite Strategy

We decided to base our solution around a simple set of questions: **1**. What **coordinate** should we drive to next, **2**. What **path** will we take to get there and **3**. What **drive action** should we currently take to begin that path. For answering question 1, we considered that we will need to account for different circumstances while driving (such as needing to explore, collecting keys, restoring health etc), and that for each stage, we should have the ability to substitute and test different implementations of the decision making. This led to us utilising a combination of two Gang of Four patterns: Strategy Pattern and Composite Pattern. In our design, MyAIController asks a strategy interface (IChooseDest) which coordinate to next drive to, where the decision is abstracted away from myAIController. The chooseDest interface then utilises a CompositeStrategy, weighing several strategies against one another, choosing the best destination for the current circumstance. This solution is flexible and more reusable, allowing one to easily switch and add strategies for varying approaches. This implementation further allows the AI to use different strategies for different phases, where for example, the objective may be to find a key, or to run and restore health.

Search: Pure Fabrication

Once a destination is found, our team requires a high level path to reach it, involving finding a short path while maximising speed and circumventing walls and lava. Since this

functionality is purely algorithmic and functional, having no internal state, we created the class Search, employing the GRASP principle of **Pure Fabrication.** Search utilises the Map, start and end points as arguments and returns a list of coordinates as the path to be taken. In order to continue to support high flexibility, Search contains a global static function getPath, which supports easily substitutable search methods without changing the base function call, adhering to the Open-Closed principle. Furthermore, this encapsulates the rest of the methods, which are private. Because the utility class Search does not contain any instance variables, we chose to make it a static class, allowing better accessibility while not affecting encapsulation.

Drive

Finally, our team approached the difficult problem of driving, facing low level problems of how to coordinate speed, direction and location to follow the set path. We removed these responsibilities from the initial implementation of MyAIController to Drive. This leads to better cohesion, where one class determines path and the other what action should be taken, similar to a driver and a navigator. This delegation of tasks follows the single responsibility principle that states that every module or class should have responsibility over a single part of the functionality, in this case, driving. In turn, this leads to more flexible and reusable code, where for example, if we were to implement another AI with a different strategy, the driving would likely remain the same, allowing reuse of code.

Test driven development:

Initially we chose to use test driven development to guide the creation of our system. Before creating classes such as MyMap, Junit tests were made, testing the correctness of functions as we went. This was important because it ensured that the tests were written and that we could design higher level artificial intelligence on a solid system. However, as the project progressed, jUnit tests became much more difficult to implement, requiring the running of the simulation in order to test other functionality such as driving. Hence, we dropped the JUnit tests, instead testing with various maps created through the tile editor, to test functionality.

Conclusion:

In conclusion, by assigning tasks away from MyAIController, and into single purpose, flexible classes, our team is able to achieve good delegation with cohesive classes, and a clean, easy to comprehend project despite the complexity of the task at hand.