Learning to Escape - Design Rationale

Group number 46

The implementation of the subsystem was split up into two main subpackages within the mycontroller package – the map, navigation along with the myAIController class which uses two distinct subcontrollers, explore and escape to separate the relevant functionalities, responsibilities and concerns of the subsystem such that the design remains extensible and within each use case there is minimal overlap between the packages to ensure that they are focused and maintain high cohesion within the overall system.

The map package's responsibilities are keeping track of all the relevant information of the map that the player has explored. Therefore, it would make sense for it to have a global point of access by the Singleton Pattern since it would be used by both the navigation and strategy packages so that they can access it without having to retain a reference to Mapping and due to it capturing the current state of a single map through the use of multiple data structures, multiple iterations or instances of the same map would be redundant. While the Mapping class increases coupling since it references and stores information about the world and the map tiles present within it, this relates to the responsibility of the map subpackage which is to store and retrieve information about the tiles for a particular map which encompasses the Information Expert pattern. The broad nature of the articulateViewPoint method gives more flexibility to update the current state of the map explored by having the parameter be a set of coordinates linked to map tiles so that in the future if there are additional ways to explore the map such as a probe with varying sight range there will be minimal, if any changes required to comply with this new specification.

The facilitation of a car's movement along a fixed path generated for a certain world is handled by the Navigation subpackage. The PathFinding class is a result of Indirection being applied under Information Expert and Low Coupling by being connected between the toolset that has been implemented, in particular the controllers and the map tiles such that parts of the strategy employed by the myAIController class aren't directly coupled with a particular tile set, to allow for easier modification in the future for design specification changes and to lead to a less coupled solution - it doesn't need to interact with the map tiles directly. Pathfinding also implements a representation of a car that functions exactly like a car but allows an optimal path to be formed without having to risk the car's health or safety - once the health reaches zero it doesn't matter if the correct path is realised, since the car is already dead. Although this may be not ideal in terms of performance or efficiency, this approach strongly aligns with the non functional requirements specified.

A consideration for designing all of the car movement controls within MyAIController was entertained since it inherited from CarController and with Polymorphism, however another completely new controller would need to be designed or the code refactored substantially for a new specification, even if there was only a small change in the behaviour. However, by implementing two different phases, ExploreController and EscapeController they would essentially act as two components that can be used by MyAIController to fulfill its responsibilities. This is consistent with the Protected Variations approach - MyAIController would be able to use alternative components for explore and escape in the event of a change like the car being limited on gas and myAIController needs additional or other components that can meet those requirements. Additionally, MyAIController might not be used anymore but the components it has been are reusable either by themselves or as components of other controllers. The ExploreController class is one of the two component controllers used by MyAIController, and references classes within the map and navigation packages in place of MyAIController. In this sense, it would be appropriate to label MyAIController as the Facade controller because it is a point of contact from within this subsystem and acts as the class that collaborates with the subsystem components to reduce undesirable coupling between itself and classes within other packages. As a result, the EscapeController has a much narrower focus and has less associations, with a direct coupling to only the Navigator class. This highlights the simple functionality and lack of complexity which will make easier to adapt in the future.

Finally, the answer to how will the car know where to go and it's particular journey from A to B is reflected within the Path class. Since the Path is something tangible in the Domain Model yet also abstract in nature, it should be standalone and have direct associations with minimal classes. The self referencing nature of addToPath(Path newPath) compared to addToPath(Coordinate point) makes it quite modular; the subsystem won't break with varying changes since a path can be composed of many paths. By compressing usage within the class to itself and the use of coordinates, this means that should there be a change to a 3d environment, there will be little needed to extend this to the 3d domain.