Game Engine Change Recommendations

Jason Leung – 29672511 – jleu0002

One of the problems that was perceived when it came to the overall design of the engine code would be related to the way the GameMap could keep track with the actors and items present on the map. The problem that a change to this specific part of the code would make the interactions outside of specific actions easier, such as spreading information of a counter present within the player to other actors and items every turn. This was made apparent when it came to the implementation of the extra task to ensure that day and night cycle could be properly implemented. Although since there was no way to directly call an actor without knowing its locations this made things more difficult, that is without looping through the entire map looking for the actor. Additionally, it would have made the dinosaurs on the other map unable to be interacted with when it came to the sleep mechanic as there would be no way to interact with the other map. In order to improve upon this so that actors can access the rest of the actors, this will be accomplished through alterations of the methods located in the GameMap. Whenever an actor is created it requires both the map and the actor to be inserted into the map, meaning that a method in the GameMap can be edited so that whenever a new actor is placed within the map, it will also be saved within GameMap as a pointer to that specific actor. This would be done by assigning it to an arraylist due to the potential exponential growth that can be experienced by the different actors on the map.

The advantages that this would have on the program as a whole is that it would make interactions that do not involve actions significantly more easier, as these basic interactions are required for the implementations of some functions, which in this case would be the implementation of a day and night system. However there would be an inherent disadvantage with this implementation and that would be the fact that any method with access to the map could potentially alter the details on the actors, however this is minimised as it does not directly hold the actor, only the methods and pointer for it. Meaning that there is a chance it could potentially violate one of the design principles, however as mentioned, this is minimised due to the nature of how they are stored being relatively protected from external interference from irrelevant classes.

Another problem that presented itself was the implementation of exits and the means that the programmer had to use and understand to fully utilise. It would have been a better idea to have it coded in a way to ensure that there is an easier usage and understanding from the programmer. This was extremely prevalent when it came to implementing functions which relied heavily on the knowledge of the locations that surrounded it, especially when creating a function to allow for a dinosaur to move 2 spaces. One of the ideal changes that could have been made is in the Exit class. Specifically, by adding a few new methods to it, these would be one that could gather the exits or locations within a provided ring into an array of locations to allow for easier utilisation from the programmer when it comes to creating new functions. Another method that could be added that would be beneficial to the programmer and design of the overall program would be the implementation of a method that would retrieve the number of surrounding exits that contained either a certain skill as defined by the game or a certain display character, this would allow for easier implementation of other methods that may rely on constant checking of surrounding exits to be implemented quickly and efficiently.

The advantages that this would provide for the program are that it would allow for easier implementation of functions that would require quick access to the exits surrounding a certain location. Additionally, it would also provide a concrete method that the programmer could use and

utilise as previously to acquire the exits. A programmer without much knowledge of the system and all of its intricacies would likely be unable to utilise it to the best that it can be. Instead it would potentially lead to inefficient code that is more difficult to understand, and therefore more difficult to troubleshoot if the issue ever arose. There would be no inherent disadvantages for the creation of these new methods as it does not violate any of the design principles that are commonly adhered to, as it is a simple and relatively isolated method that performs a single function.

Although there were certain issues that arose during the creation of the additional feature in the program, the utilisation and understandings of the application of skills onto the map and everything on it allowed for these problems to be solved easily. So, for further systems the expansion of the skills system may be a beneficial thing to consider as it is a method that did assist in much of the solving of problems as they prevented themselves. So further expansion if possible, would be highly beneficial to any further systems. This would allow for easier implementation of methods without violating any of the design principles of object-oriented programming.

However, problems aside, the program and the provided code was relatively good and there are many parts that would require little to no improvement for future iterations. This is apparent when looking at the provided starting code as it does its best to prevent itself from unnecessarily repeating itself, this adhering to DRY. This is made apparent when looking at the provided actors, players and protoceratops initially. None of which shared methods and utilised abstraction to minimise or straight up remove the unnecessary repeated code from the classes. It is to be noted that despite the ease of which the principle could be followed the actors were implemented after were unable to completely follow this design principle due to time constraints. Additionally all of the provided code was relatively isolated, which allowed for it to adhere to the principle of open closed, which ensures that each method performs a certain singular action and that further expansion of the methods in that specific class will not break the methods which are already functioning, meaning that it also followed the design principle of single responsibility, which is when a provided function or method is only utilised for one thing. This was made apparent at the start when the game was already semi-functional when it was initially provided, however when it came to implementation of the new functionality, this design principle could not be adhered to completely as some functionality would have been difficult to implement without modifying the already existing code. Generally the provided code at the start followed the majority of the design principles, however there were some principles that were unable to be followed, not at the fault of the code, but at the fault of the programmer for not being able to utilise the interface, thus leading to code which could be implemented but would ensure that the code was less flexible when it came to future change.