FIT2099

Assignment 3:

Recommendations for extensions to the game engine

Prepared by:
Vladislav Pikulin (30935679)
Daiki Kubo (30523346)
Abdalla Abdellatif (30247500)

Introduction

First and foremost, it is in order to state that the engine package is very well designed, robust, and generally very well thought through. Having said that, although it has a lot of useful features that we have made sure to make use of as much as possible in our assignment projects, we believe there are some features that could be added to the engine package that would not only allow the engine to be more extensible by allowing others to make use of those features, but also significantly help other programmers who are working on extending the program to follow programming object oriented design such as 'Don't repeat yourself'.

As such, this file aims to identify the possible improvements that could have been made to the engine package in terms of additional features, as well as analyze their advantages, disadvantages, and elaborate on how having the features would have helped us achieve our goal in the project.

Addition 1: calculateDistance public method in Location class

The first addition that we would recommend to the engine is the addition of a method that calculates the distance between one location and another. Before going any further into detail, this method has been already implemented in the FollowBehaviour. This implementation, however, should be moved to the Location class. As a team, we have had to design ThirstBehaviour and BreedBehaviour which both make use of this method. Having the method implemented in the FollowBehaviour rather than the Location class means that we had to repeat ourselves when writing the code in other classes that require it. Having it implemented in Location class instead would have allowed us to follow the DRY principle more strictly as we could have found a location of any actor and checked their location against another actor or item without having to implement anything in the Behaviour class.

Advantages:

- Allows us to follow DRY principle
    - Having it implemented in the Location class as a public method would no longer require us to repeat the code to have it in classes that require it. Instead it can be used on any instance of Location.
- Allows for abstraction of its implementation
    - Having it implemented in the engine seems to be the more ideal method of its implementation. When operating with the game, the programmer may not need to know the mathematical model for calculating the distance between two locations. Hence, implementing it in the Location class can reduce the cognitive complexity and load on the programmers.

Disadvantages:

- None

Addition 2: Interaction with ground from the same block

When having to allow the player to interact with ground types such as Grass for harvesting grass, or with trees for collecting fruits, it would have been significantly beneficial for our design to be able to interact with the ground from the same block. Currently, giving allowable actions to the ground only allows it to be called when the actor is one block away from the ground. It also, however, seems logical to have it done in such a way since if a ground is not traversable by an actor, it should be able to interact with the ground type from one block away (for example: drinking water). Therefore, this could possibly be implemented by making use of the 'canActorEnter' method. If a player can enter the block, perhaps it would be possible to get the ground's actions only when the actor is standing on the ground instead of if one of the actors exits. If it were to be implemented in such a way, it would be best to do so in the processActorTurn method in the World class. On line 117, the loop through exits of the actor identifies the ground types next to them and gets their allowable actions. It would be ideal to add another condition to check if the actor can enter the ground type, if true, then only get the allowable actions if the actor is standing on it.

Having this implemented would have allowed us to reduce the number of dependencies present between actions that an actor is supposed to be able to perform and a certain ground type and the Player instance. Not only does it create dependencies but it creates dependencies with low level objects, which also disobeys the dependency inversion principle. Hence, having this feature would have allowed us to follow the dependency inversion principle more strictly.

Advantages:

- Less dependencies for player and ground types.
    - At the moment, if a player needs to interact with the ground type, we have to check the player's ground type in the player class and add the appropriate action to the player. This creates dependencies as, if there is a new action a player needs to be able to implement. It also makes less sense to give an Actor an action that he/she should be able to perform on a ground type rather than giving that ground type an allowable action. This makes the code a lot more difficult to comprehend.

Disadvantages:

- May create inconsistent behaviours between different types of actors.
    - Although this is possibly avoidable, implementing the future in a proposed manner, may create inconsistencies between how actors behave. For instance, some actor who can traverse water would only be able to drink water if they are on the block of water, whereas those which can't traverse water would be able to drink water from an adjacent block

Addition 3: Reset ArrayList of GameMaps in the World class

Being able to reset the arrayList of the stored game maps in the world would have helped us in our implementation of our design as we had to significantly refactor the application class and have to create an entire new world instance, game map instances,and player instances before invoking the world's run method. Having the ability to reset the game maps would have allowed us to reduce some of the code smells that we have collected in the application class, in the form of having 4 arguments in a method and having a long method for the generation of the maps and the world.

Advantages:

- Allow us to reduce code smells
    - Code smells such as large methods and methods that take in 4 arguments could have been avoided. Avoiding these code smells could potentially make the code less prone to bugs, and hence make it more extensible.

Disadvantages:

- None

Addition 4: Display class to read and return a String

At the moment the display class can only read a string and return a character, which is the first character of the string. At the moment, when making the main menu, we could not make full use of the display class to read in user inputs. Instead we had to import a Scanner and create an instance of Scanner, which again creates unnecessary dependencies and associations which could have been avoided by having a public method in the Display class that reads in a user input and returns it. Not to mention that a Display class could be extended to be able to display to the game to another interface, other than the console, having a separate Scanner object increases the technical debt that we are putting on the system, as in the case that if it is being ported elsewhere the Scanner object in the main menu would eventually have to be configured to work with the new interface. Doing so would be a lot simpler by adding it into the Display class to reduce dependencies throughout the code.

Advantages:

- Reduces dependencies and association in the code.
    - Instead of creating a new scanner object, we can make use of the scanner object in the Display class. And as a result, obey the ReD principle.

Disadvantages:

- None