Reflection

Diana Ruth (851465)

Implementation Changes

Improvements to the Player Class

The original Player class in my UML did not inherit from any other class and had no attributes and lacked many methods. This was done by accident, and the Player class inherits from Unit in Project2B. This allows the Player class to inherit all the attributes from the Unit class and all of its methods, as well as implementing methods specific to the Player class.

Render Method in Unit Class

The original Unit class did not have a render method, and the new implementation fixed this problem. The Unit class has a concrete render method that renders the Unit's image to the correct place on the screen as well as its health bar and name. This render method is used for every Unit in the game except Player, which overrides the render method and provides a different implementation since the Player does not have a health bar above its head.

Specific Attributes for Items

Originally, each Item had a healthBonus, damageBonus, and cooldownBonus attribute. Only one of these is nonzero for each item, so each Item had two unnecessary attributes. In Project2B this was changed to implement a specific method for each item that adds the single bonus to the Player based on which item the Player is picking up. This makes the pickUp method for Item much more specific based on which Item is being picked up.

Protected Attributes

In Project2A, all attributes in abstract classes were protected to provide easy access to members. However, this was changed in Project2B to provide a higher level of security between classes. Protected attributes made it too easy for any class to freely modify those attributes, so these attributes were changed to private and were given getter and setter methods to provide extra security. With this change, the encapsulation of the classes is improved and the attributes are sufficiently hidden from outside classes.

Implementing the Interactive Interface

Originally the Item and Unit classes implemented the Interactive interface so that monsters and items could interact with the Player. This was changed so that the Villager, Item, and Monster classes implement the Interactive interface because the Player does not need to interact with itself. So every Villager, Item, and Monster implements a method called interact that passes the Player object so that the player can interact with each Monster, Item, and Villager in the world.

Lessons Learned

This project taught me the importance of careful planning. If I had spent more time planning my original UML diagram, I would have saved much more time on the implementation of the final game. Initially, I believed that the abstract classes having protected attributes would provide sufficient information hiding, but protected attributes are too easily modified by outside classes. I spent a lot of time changing these attributes to private and providing getters and setters for them, and that extra work could have been avoided if I had practiced good information hiding principles from the start. This project taught me the importance of careful planning and good object-oriented principles in creating an effective software system. I also learned that information hiding is one of the most important aspects of good object-oriented design because it protects private attributes of a class from outside classes and prevents outside classes from modifying those attributes. If I could do this project again, I would take the time to plan out the software system from the start so that the actual implementation is easier and takes less time.