OOSE Assignment 2020  
Written Discussion

Jonathan Wright – 19779085

Contents

**How My Design Works**

For the characters of my game I decided that I was going to have the AI (Enemy) and Players both inherit a superclass (Character), I chose this because it would in future allow the developer to allow battles between AI and battles between Players rather than just AI vs Player. The design patterns I had to implement this was either a Strategy pattern where Character would be an interface or the Template pattern where Character would be an abstract class. The reason I chose a ***Template*** pattern is because there was a lot of code duplication that occurred by using an interface superclass, using a Template pattern allowed me to reduce the amount of code duplication by storing methods and class fields both used in the super class Character. Another consideration I made was the easy addition of AI enemies (Ogre, Slime, etc), I chose to again use the Strategy pattern for each Enemy where Enemy is the super class, this makes it easy to add a new AI Enemy by simply extending the class and calling the superclasses constructor. Something that I considered about this was using a Decorator for enemy abilities, but in the end I decided to take this out because after thinking about it not all enemies will have abilities, perhaps the future developer simply just wants to add a new enemy with the default implementation of attack, so I decided the way you add abilities was to just extend the getDamage method and add the ability in the getDamage, this also means that you could extend takeDamage to allow defensive abilities in future for enemies.  
For Battles in my game, I decided to include the data in its own separate model and controller, this is because I thought about it and I could store the data relating to the battle in a model (rather than retaining the information around battles in a controller), and I could direct it using a controller. I used a ***Observer*** pattern with the Battle model, and this is because I thought that every turn it would become annoying for a developer to have to take the data from the model, and pass it to controller, then pass it to the view, using a BattleObserver the model can simply notify the observers and the view can be updated therefore the controller does not have to handle passing the updated turn to the view.  
For Items in the game I again used a Template pattern, I could have used a Strategy Pattern but I chose a Template pattern for the purpose of eliminating code duplication. I have Enchantments in the game following a ***Decorator*** pattern, and they implement a Weapon Interface, I chose to do it this way because in future if a developer wanted to make Defensive Enchantments they could make a Defensive interface, implement it and use the same enchantment superclass, this makes it easy to add new enchantments (just extend the Enchantment superclass) and it means you can choose in the player to have his weapon as a Weapon rather than a Melee (the physical weapon is a Melee).  
Data for the Shop is loaded in using a DataLoader, the DataLoader is injected however the way I set it up increased coupling, I chose to keep it this way because it reduced the amount of effort by the developer to use the shop as all they have to do is pass the DataLoader rather then the three returns from the DataLoader. The DataLoader uses the Template Pattern in order to easily add more sources in future (you would just extend DataLoader as WebLoader for a web-based loader and implement the four load functions). The DataLoader was the perfect choice for a Strategy Pattern because all four methods were abstract, however, I chose to also use the ***Factory*** pattern and include a factory method inside of the DataLoader, this eliminates the need for the developer to know which DataLoader he needs to use and can simply pass the data e.g. shop.updateShop(DataLoader.getLoader(“myFile.csv”));  
This eliminates the developer needing to know that he has to construct a TextLoader and allows for in future many different ways of loading data and not needing to know which loader he needs for loading that data.