530319263

Assignment 2 Report

The overall logic for my program design was to separate the key elements of the game into classes. For example, classes were generated for game features that would need to be instantiated multiple times and/or contributed to the visual depiction of the gameplay on screen. This was most evident for creating sprites (e.g. Monsters, Fireballs, Tower) and rendering the map (Board). Elements that required drawing a shape, such as my ManaBar, HealthBar and GameplayActions classes also fell under this functionality. Since these shapes were all rectangular and had similar attributes such as width, height and coordinates, they were grouped under a Shape superclass, which would implement the rendering logic of the shapes on screen, such as colour/dimension changes.

Most classes were also required to implement some form of logic when the user interacted with the program, however these separate logics needed to somehow be combined in order to create a cohesive gaming experience, and this was primarily achieved through my BuildTower and Waves classes. The BuildTower class handled player-based inputs/events, such as purchasing/upgrading towers and shooting fireballs, whereas the Waves class focused on producing output as the opposition, such as spawning monsters and executing the waves. Interaction between classes was managed through passing a single instance of a given class such as Mana and GameplayAction subclasses as parameters for another class' method, as this allowed event changes like pressing the pause button to be communicated to all objects that the aforementioned classes were composed of.

Inheritance was also utilised through generating subclasses that have an "is-a" relationship with another class. For example, my abstract superclasses WaveElements and GameplayActions were used to generalise shared functionalities between similar game features, such as initialising attributes for a given wave from a config file and implementing button logic respectively. This as well as the use of getters and setters allowed for the encapsulation of data without sacrificing interactivity, as it prevented unwanted manipulation of data from external classes by enforcing privileged access to a specific type. Many classes also required retrieving numeric values from a JSONObject key, and to allow more flexible access to this behaviour, my RetrieveValues interface contained default methods for this purpose. These design decisions allowed me to repurpose my code and reduce the need to repeat large sections of logic in my program.

For the extension, I created a new game-play action button that would allow the user to place a hole that I designed on any path tile, and if a monster fell into that hole, it would cause immediate death. However, to prevent the action from being too overpowering, restrictions of its use were placed, such as increasing the cost of every purchase and requiring at least 5 seconds (as indicated by a cross) in between each purchase. Holes would also disappear as soon as a monster fell into it, meaning that at most only one monster can fall into any given hole.