Final Project Report

group3

Byambatsog Chimed , Yi Jiang, Ilona Shparii, Xiaolu(Anne) Zhao, Hongbin(Nate) Sun

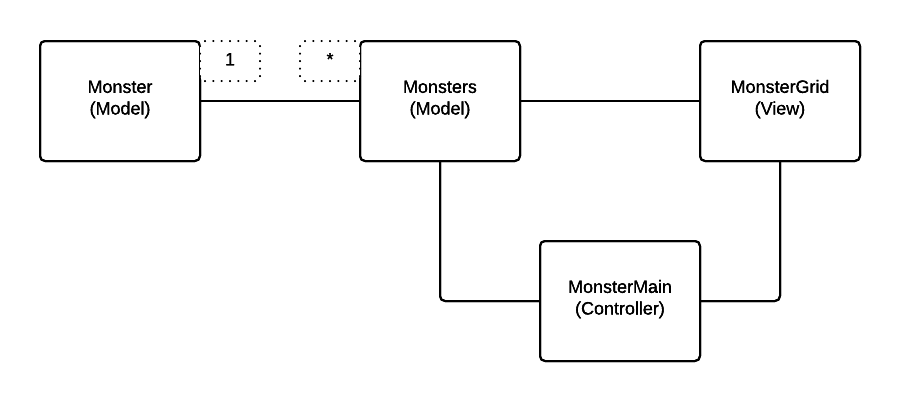
MVA(Model-View-Adapter) and MVC (Model-View-Controller) are popular architectural patterns used in software engineering with some degree of similarity. The main difference between MVA and MVC is that MVA encapsulates Model from View and acts as a listener to dynamically synchronize the events. Due to the encapsulation, the View does not need to know anything about the Model. In addition, the ability to run a task without attaching listeners and therefore, prevent event-action-event loop, and the simplicity to swap a new Model object to an old one makes MVA advantage over MVC. However, the adaptor in MVA takes too much responsibility and can be daunting complicated if the Model and View becomes long and complex. In our project, we adopted MVC because it makes easier to directly connect the Monster Model data with View upon the user commands so that the Monster Model can show up on the screen View.

During the project, we implemented some of SOLID principles. For example, we followed single responsibility principle so that each class(Monster, Monsters, MonsterGrind and MonsterMain) is responsible only for one area. Monster class defines state and behavior of object that represent single monster. Monsters class is a composite of Monster objects. MonsterGrid is a view in our case. MonsterMain class executes controller tasks. In addition, we also followed open-closed principle. We encapsulated data making most of class variables private, so other classes don’t have access to it. To use encapsulated data we created public getters and setters methods.

Testing part was quite challenging since the application has a lot of random values for variables such as random movement or random changing of vulnerability. We did automated testing of main activity and some other features that are not random. We have created one test AbstractMonsterActivityClass.java for all purposes. In addition, we tested application from user standpoint. It is especially useful for things that are random.

Moverover, we used AsyncTask during the project to move individual monster and concurrency by setting sleep time. In addition, we synchronized move for each thread of monster so that they will not collide to each other.

Domain model in our project:



In our project, The Monster and Monsters are Model, and we followed single responsible principle so that in Monster class, each monster object in Monster has individual functionality, and Monsters is a linkedlist composed of certain amount of Monster. The MonsterGrid is a View for the UI thread, there we drew the grids for the screen and rescaled the monster image to each grid. The MonsterMain is a controller between the Model and View, basically acting as listeners to respond to the comments from View or Model. However, there is still some direct connection between View and Model that is not controlled by the controller, for instance, we rescaled our image in Monster to the size of the grid in MonsterView so that we can draw Monsters in the View. Also we called monsters in MonsterView to move to clear the monsters. The advantage to adapt such model is that the controller will not take all the responsibilites between Model and View so that the code could be easier.