Project 5 - Group 1

Domain Model

The domain model of our project can be broken down into Monsters, Monster, Grid, and the TouchMe class.

The TouchMe Class is the main activity class. This allows the game to be initialized and set up the level which will lead to the differing amount of Monsters that are created upon the start along with the percentage that a monster will be vulnerable(removable) within a turn. In addition, the TouchMe class also deals with User Interaction(Clicking on monsters within the grid.) Finally, upon Initialization, this class triggers the creation of the Grid and triggers the start of movement for monsters. And begins/ keeps additional information in sync (clock ticker, ect.)

The Monster Class deals with individual monsters. This includes getting and setting the position of individual monsters, and dealing with individual monster movement.

The Monsters Class deals with the actual initialization of monsters into the game board, and keeps track of monsters within the Grid.

The Grid class mainly deals with the generation of the valid spaces and helps with drawing the monsters in the correct place.

Design

Whereas the Model-View-Controller (MVC) pattern allows the controller to be in charge of the application, a Model-View-Adapter (MVA) pattern allows the current view to be in charge of actions. With the MVA pattern, there is usually an interface that is created that returns a specific action based on the state of the application. This can be seen in our start/stop of the game. While the state is in an Enabled state, we have the monsters move around and change status,

and allow the user to interact with the gameboard when the monsters are vulnerable. This is also carried throughout our code.

Because we broke down our model into different parts, we were able to follow the single responsibility principle of the SOLID design. Our Monster class is only responsible for a single monster. The Monsters class allows us to have a class that is decided to a large group of monsters. Our Grid class takes responsibility of the creation of the game board itself. Lastly, our TouchMe class acts as the main, responsible for running the game itself.

We did not implement any tests for our project. That being said, I believe that each of our classes would be easily testable, as well as our main program. One example is our leveling system. Because our level system is based off a simple switch, it would be easy to implement a complex running test of running through and selecting multiple levels of our game. Our parameters for each of the levels are found in an array, which can be easily accessed, and tested for the correct number of monsters and the probability of the monster becoming vulnerable.

We used the AsyncTask to work in our Monster class. We used this method to move the monster in the background, in our case every .08 of a second. After this background task was run, we then made it report that movement, if that happened, back to the Grid class saying that the monster has moved.

EXTRA CREDIT

PROJECT 6

Using Project 5 we have found several parts that can be hosted remotely. Within things that we can host remotely and share between users for our current project may be the Level Data. If so desired, we can host this data on an online server (For example a Heroku app with a Database Connection) and we could create an api that would allow users to retrieve level Data (for example: herokudomain.com/api/GetLevels). This api would most likely return a JSON object that would store 2 int arrays where one array would store the starting amount of monsters and

the second would store the percentage chance that a monster will become vulnerable. This would become useful if we wanted to tweak the difficulty due to user requests of wither some levels being to easy/hard.

As for things that we could preserve across games, we could have a function that saves a game when the stop screen is pressed or when the game is exited and send a post request to our theoretical api with JSON data of the Grid, a list of position locations of monsters left, the value of time remaining, and the count of monsters you have removed thus far (for example: herokudomain.com/api/SaveGameplay). Upon reopening the app, there would be a call out to the api which can be an api request to the same location but a GET request instead of a POST (for example: herokudomain.com/api/LoadGameplay) which would send back this data so we can reinitialize the data and continue the game on another device. If we were to implement a score system, this would also be beneficial as we would be able to return a full list of top scores.

A third thing that may become beneficial to host online in a remote service would be analytical data of a session or an error report that would be sent at the time of an app closing/crashing. This may include a Date of the Crash if it crashed, The duration of the session, the amount of gameplays (include a ticker in the TouchMe Class that can do some logging)/ including Date of when the user opens the app. This would all be stored as a JSON object and then it could be sent to a logging endpoint as a POST rest request. (for example: herokudomain.com/api/Logging)