Module	SEPR
Year	2019/20
Assessment	4
Team	Early Bird
Members	James Little, Ryan Vint, Adam Lynch, Georgina Martin, Kheng Yeoh, Tanay Malde, Cameron Devane-Waters
Deliverable	Implementation Report

Implementation Report

To meet the new requirements found [1] for the Difficulty portion of the changes we had to amend both code and the UI. For the UI, we needed to create an intermediary screen between the Main Menu and Game Screen to allow for the User to choose a difficulty. This addition was added using the LibGDX Screen Libraries and the preexisting screens within the adopted project. This new class was named ‘DifficultyScreen’ and was placed within the Screen package. New graphics had to be created for the screen as well as passing the Chosen difficulty from the screen class through into the Main Game constructor. Next, was the implementation of changing unit stats for the new difficulties: Easy, Medium and Hard. To do this we went into the preexisting ‘DifficultyControl.java’ file which was used previously to implement the increasing of stats over time when the game is being played. We chose to amend the following stats: DifficultyChangeInterval (time between fortress upgrades, smaller the value the harder the difficulty) and modeMultiplier (combined with a fortresses AP value to calculate the damage that a fortress projectile will do). We felt that more could have been changed to add further complexity but for ease of creating a balanced game we chose to keep simple changes between each difficulty whilst maintaining clear distinctions to keep easy mode easy and hard mode very difficult. To implement this, we employed the use of an Enum, DifficultyMode, which allows for easy and set transitions between the 3 difficulties and for quick amendment of the values that they hold. Then as said earlier through the creation of a new game these values are then set within the MainGame screen constructor.

In order to fulfil the new requirements for save functionality we implemented a 'SaveManager' class that handles all saving and loading, this kept it out of the way and made it easy to reference elsewhere. In order to use the saveManager class effectively we created a new screen 'LoadScreen' that when passed a particular mode it either saves or loads. We included 3 buttons corresponding to 3 various saves to satisfy the FR_MULTIPLE_SAVES requirement. For most classes that needed to be saved such as 'Firetruck' and 'Patrol' we created a corresponding data class as there was no need to save large unimportant information such as textures. We chose to save and load from json files as they have more functionality than text files, in addition, libgdx had various classes which supported saving and loading from them. We also chose to encode the json data as a means to stop users from altering their save files.

When implementing powerups we decided to create a 'PowerUpTile' class which upon being picked up by a Truck would create a 'PowerUp' and add it to the truck. This allowed us to not have to store a texture in the 'PowerUp' class. All power ups inherit the abstract class 'PowerUP' which contains all the methods any powerUp will need. This inheritance makes it very easy to call all necessary methods of every powerUp that each truck has as well as making it easy to make new PowerUps. We also made corresponding Data classes for 'PowerUpTile' and 'PowerUp'. These saved a powerUp type ‘Power’ which enabled us to not have Data classes for each powerUp.

References

[1] Updated requirements
https://jameslittlecs.github.io/Kroy_Final_Assessment/assessment4/Req4.pdf