Implementation

Module	SEPR
Year	2019/20
Assessment	4
Team	York Fire Marshalls
Members	Samuel Whitehead, Andrew Connor, Alex Dawson, Finn Jackson, Fred Dodd, Phuong Kha
Deliverable	Implementation

For our game, we have implemented the three different features that were required - five different types of powerups, three different modes of difficulty, and the ability to save and load the game at any point while playing. We will try to explain how we implemented those changes in the software we were handed by the previous group below.

Powerups

For the requirement to add five new powerups, we decided to add them spawning across the map at regular intervals, while each providing the player's current firetruck with either a stock of resources or a temporary ability. Each powerup functions differently, with its own visually distinct sprite while still maintaining a similar look so the player understands its purpose within the game. The five powers we have added are the following:

Refill Water - Refills the player's current truck back to full water capacity

Refill Health - Heals the player's current truck back to maximal health

Temporary Shield - Gives the player's current truck immunity to damage for a five seconds

Destroy Patrols - Gives the player's current truck the ability to destroy patrols on contact for about six seconds

Increased Range - Gives the player's current truck a greatly increased range for about six seconds

To implement the powerups, we needed to add a *powerupsprite* class based on the minigame class that was already present in the game. This class signifies a powerup of any kind, containing variables for its location, type and sprite. On contact with a player, the powerup removes itself, giving the player's truck the relevant powerup.

The actual spawning of the *powerupsprites* are all done in the GameScreen class. An arraylist contains the powerups that currently exist on the map, and every 10 seconds *spawnpowerup*() creates a new one on one of the junctions of the world map chosen at random. If there are already 15 powerups on the map, no more can be spawned, so as to prevent overflow or multiple powerups spawning on top of each other.

The effects of the powerups are handled by the *powerupsprite* class. For the first two, they simply take the firetruck that came into contact with them, subtract the amount of water or health they have from their respective maxima, and add that amount of the resource, instantly restoring the truck's ammo of that type.

The others call functions within the firetruck class, that in turn give that truck the ability for the allocated amount of time. The first two simply add an if statement to damage dealing/on contact with an enemy patrol, while the final powerup increases the range stat.

Difficulty Levels

We decided to add three different levels of difficulty to our code, which can be selected by the player after the first screen in the game. Instead of directly going to the story screen, then the tutorial, and then the main game, the player is instead brought to the *DifficultySelectScreen*. On this screen, three options are presented, as well as the opportunity to return to the main screen. Below are the three difficulty levels, described in detail.

On easy mode, the game brings the player through the story screen into the main GameScreen. As per usual, the tutorial runs - but we used a float called *difScale* to appropriately scale down certain statistics used within the game. The list of variables affected is:

- The health of the ET bases is multiplied by 0.8
- The healing per second of the ET bases is multiplied by 0.8
- The range of the ET bases is multiplied by 0.8
- The range of the ET patrols is multiplied by 0.75

Additionally, the amount of time it takes for the FireStation to be destroyed on easy difficulty is increased from 3 minutes to 6 minutes, to allow for a more relaxed playing experience.
On normal difficulty, the game skips over the tutorial by immediately calling finishTutorial() after the gamescreen is accessed. By setting *difScale* to 1, the game runs exactly as it did before, which we believed was challenging but fair.
On hard difficulty, *difScale* is set to 1.5 for both the ET bases and patrols. This makes the game considerably harder, as it often is impossible to entirely destroy a base even with a full water tank, and means the player must strategically use powerups and other resources.

Saves

In order to implement the functionality of saving we decided to use the Json library which allows you to save objects and other data to a file easily. So initially we thought about just saving the game object created at the start but there is a lot of data in here which does not need to be saved as it is constant for the game such as the camera, map etc; just the entities within the ghame need to be saved so that they can be loaded once again.
Furthermore when saving entities such as the firetrucks of the game not all of the data contained in the object needs to be saved so for an efficient save process that was easier to work with we first created some new classes in which we would save the important data as well as make methods for getting and setting these values for the save and the loads.
The main addition to the code was the SaveManager file, contained in this are methods which save the gameData to a json file and also load the data from that file.
In order to make the gameData class and populate it with the data from the game smaller more precise classes such as TruckData and ETFortressData which contains just the information about the trucks such as their health, water level and location; then this data is added to an array list of these objects. The score, time and difficulty of the game are also saved into gameData.
As part of the implementation new buttons had to be added into the pause screen, this was very simple to do due to the way it was already coded; the only difference being how the buttons react when clicked. They call the saveGame function from the SaveManager class with the appropriate path, that being save 1, 2 or 3, they also pass the gameScreen here as it contains the data to be saved. There has been another change to the menu system which is the addition of a new screen, the load game screen, which is linked from the main menu. On this screen there are 3 buttons which relate to the three load files that the user can choose from, if the user selects one without a save nothing will happen. Another problem with our code for this is that when a game is complete the save file is not removed, this is not a serious problem but is not ideally how we would have it.
When using the save and load functions a path is needed to select the correct file so this was added to the file as a variable so it could be used to load the correct data.