Module	ENG1
Year	20/21
Team	33 (Short Circuits)
Members	Jack Lord, Neo Metcalfe, Sam Rodgers, Mohammad Abdullah, Qi Tang
Deliverable	Implementation

The implementation of the saving feature required a new data-type, a gamestate, which is the actual object that we save onto the disk. This contains all the variables needed to reconstruct the game into the state it was in when the player saved.
This GameState is then saved to disk using the new GameStorage class, which handles the saving and loading of any GameState objects that we need to save.

The GameController class was refactored to use the factory design pattern, rather than creating new instances of the objects it needs inside of GameController itself. This was done to facilitate testing of the GameController class, since some of the objects required a libgdx game to be running when they are constructed, instead, we could mock these objects using the Mockito Java library and pass them to the GameController, meaning we could use them in a test environment.

A new screen was created where the player can select the difficulty level they want. This sets the variables in the new Difficulty class to the correct values for that difficulty level. All other classes that need to worry about the difficulty level then get their initial variables that are affected by the difficulty from the difficulty class.

The player class has had its position variables removed, since Sprite, which it extends, already has position variables for x and y that were not being used. In the previous implementation, this meant that the x and y variables in Player needed to be public, because Sprite already had getters and setters for the variables which hadn't been overridden. We made this change so that the getX() and getY() methods could be mocked for testing purposes. This also improved the encapsulation of the Player class, and since these variables already existed in the superclass, there was no need to redefine them in Player.

On the main menu, a new button has been added that loads the currently saved gameState. Whenever a new game is started, everything that changes in the gameState is loaded from the GameState class. This means that when the game doesn't have a save, the objects and variables created are the default values that represent a game starting from the beginning, but when the game does have a save, they are instead set to the values that have been loaded from the saved game and the player carries on from where they saved.

The features that haven't been implemented include all 5 power ups for the Auber. However, 3 have been implemented. This is due to the time constraint and these would have been finished otherwise. The game is still functional despite this, and the user can use the 3 power ups that have been coded, they will just not experience the entire product.

Another feature that was not included was the demo mode which was from the original team's brief. Again, this was due to the time constraint and would have been implemented using a basic script for the player making them move around. The AI for the imposters could have been used to showcase their part in the game. The impact of this on the game is that the customer will not be able to showcase the game whilst idle. However, users can still play. The customer would have to just have the main menu on display.