Implementation

Group Name: Team siKz
Group Number: 6

Ryan Bulman
Frederick Clarke
Jack Ellis
Yuhao Hu
Thomas Nicholson
James Pursglove

3b)

Significant new additions

Assessment 2 required us to implement multiple different features into the existing system. These were: Weather and obstacles (FR.GEN_OBS, FR.OBS_IMPACT), temporary upgrades through power-ups (FR.TMP_UPGRD), permanent upgrades through a shop (FR.UPGRADES), the ability to change the difficulty of the game (FR.CHANGE_DIFF), combat with AI ships (FR.SHIP_AI), and the ability save and load the gamestate (FR.SAVE, FR.LOAD).

Of these features, obstacles, shops, and power-ups could be extended from the existing GameObject object, so they did not require many significant additions.

- Obstacles are the simplest new addition, just being a game object which checks for player collisions and applies an effect when this is true, such as damage, a knockback or gold.
- Shops are assigned 1:1 to a college, and are initialised with the activated variable set to false. This means they are not visible or interactable until the college the shop was linked to is defeated. After this the player's position can be checked and awaits user input to open a shop window.
- Upgrades required creating new public variables in the Player object. These are accessible from both the shop and power-ups. These are: speed, damage, and armour.
- Power-ups were implemented as an extension from obstacles, simply providing a temporary benefit and being removed from the screen on collision.

The other feature additions could rely less on a single object to extend from and required more of their own code

Weather

- The weather sections on the map are where the player has temporarily reduced speed and some have reduced visibility.
- This was done by having a weather array and passing weather objects which were with an enum of weather type.
- There are 4 weather types: rain, snow, storm and mortar. Mortar depletes the players health by an amount every few seconds, storms reduce the visibility of the player and they all reduce the speed of the player.
- The weather class has a few static methods such as testing for which weather event the player is currently in and a disadvantage method which applies the negative effects to the player.
- Animation is added to the weather by using TimeUtils and setting a timer, which is a lot more efficient than using threads. This animation moves static textures around the screen to provide the desired effect.

Difficulty

- The UI was changed in the TitleScreen file to remove the play button and add three new buttons for 'easy', 'medium' and 'hard'.
- The amount of damage the player takes differs depending on the difficulty selected. The takeDamage function in the Player file was edited to implement this.
- A new variable was also added to the YorkPirates file which refers to a string representing the difficulty selected.

Ship AI

- The AI for the ships comprises two sections - movement and firing. The firing was simpler to implement, sharing a lot of logic with the firing for colleges; when the player is in range, fire a projectile at the player's location.
- Movement is based on simply following the player's location up to a "stop_range" in a radius around the player. Two checks were also added to ensure that the boats wouldn't pass through the ground or other boats.

- Boats are controlled through the college they are associated with in that college's update function. The list of boats for each college are iterated through, calculating their movement, checking the collision and undoing the movement if necessary, updating the hitbox of the boat and firing if in range of the player.

Save and Load

- When the "K" key is pressed the game will make a save of the game and when the "L" key is pressed this save is loaded. To discourage players from abusing saves not everything is loaded from a save. For example when you load it resets the health of enemies and removes any active power ups. That said, the save will store and load important information such as if a college has been captured and the positions of enemies and the player.
- We used libgdx's inbuilt xml library because we knew from experience with assessment one that it would be more convenient when we compiled the game.
- When a save function is called the game passes the player, colleges and obstacle arrays into the save function. It then iterates through the arrays and saves relevant information about that object into the save.xml file.
- When a load function is called the xml file is interpreted and information about the objects is passed through to the game screen. The game screen then generates new objects to replace the existing ones and replaces the content of the object arrays.

Significant changes

Change ID	Description	Justification
IMP01	Shot delay is calculated using timer.utils instead of threads (CR.LOW_SPEC)	This is a more efficient method of adding a delay between shots. The previous implementation used threads which held until the delay was complete. This meant that when the fire rate was high, other sections of code were not being addressed at the rate they should have been, leading to slowdown, especially with regards to movement.
IMP02	Functions for obstacles and weather in player (FR.OBS_IMPACT)	We added new functionality to the player based on the current weather event. The player would have reduced speed and possibly reduced visibility depending on the current weather event.
IMP03	Created new variables in the player class and made existing ones public (FR.UPGRADES)	This change was necessary to access variables in the player to enable the upgrade system we implemented.
IMP04	Replaced Texture array in GameObject with single Texture (CR.LOW_SPEC)	We replaced the texture array that the previous group used, with a single texture for each game object. The issue with using texture arrays is that if you want to draw an object with only one texture (that doesn't animate) you'd have to make a texture array with only one texture, wasting a lot of space. Using this method is more efficient and is also a lot easier to implement because we only have to pass one Texture rather than an array of them.
IMP05	The arrow keys have been reassigned from movement controls to shooting	Given that one very likely use case of the program is a laptop on open day, only having the ability to

	(CR.LOW_SPEC)	shoot with a mouse made the game nearly unplayable. This is due to how Libgdx interacts with the trackpad and button checks. While mouse aim was acceptable with a free-standing mouse, there needed to be an alternative that worked on a laptop
IMP06	Added three difficulty options to the game (FR.CHANGE_DIFF)	Depending on which difficulty level the player selects, it will change the damage dealt to the player. This is done by using a multiplier, which multiplies the damage dealt by some constant that is set based on the difficulty level.
IMP07	Functionality in college to run ships AI (FR.SHIP_AI)	Within the college object a number of functions are called. These have the effect of controlling the basic AI for each ship - enabling them to move and fire independently