Implementation2

Cohort 1, Team 1:

Connor Blenkinsop
David Luncan
Emily Webb
Muhidin Muhidin(MO)
Sam Laljee
Sooyeon Lee

We used the concrete architecture as a base for our abstract architecture so it corresponds to the structure of the project directory as the previous abstract was a lot simpler than the end result with their concrete architecture.

We have also implemented missing features Shopping, Ship combat and obstacles portrayed in the updated architecture. This means that the code and the architecture are consistent as they have been updated.

The new code implements the requirements as seen below. And explains the other parts that were not implemented into the code

Changes Made to Implementation:

1. GoldShop *- fully implemented*
   In accordance with requirements UR_GAME_CURRENCY and respectively FR_SHOPPING we have added a screen where you can spend your gold for permanent or temporary upgrades by changing the screen to GoldShop whilst passing a reference of the player in world to allow the screen to change the amount of gold that the player has. The GoldShop allows the user to spend their gold using text buttons that checks whether the user has enough money to buy the specific upgrade and prevent the purchase if insufficient money. The GoldShop also displays the cost of the upgrade on the button that changes when a purchase has been made for that specific item. The three items that you can purchase are health, damage and speed. To help add this function a decreaseGold method was added to the player class to allow the gold to update in the world screen and not just affect the goldShop screen
   GoldShop has been implemented with the other menus as it is only called in the mainRunner. Library for text buttons, skins and texture atlas has been used as well as adding a new json file to format the textures of the button and the text inside the buttons.

2. Ship combat *- fully implemented*
   In accordance with requirements UR_SHIP_COMBAT, UR_AI_PIRATE, and their respective FRs, ship-to-ship combat via AI NPCs has been implemented under the pre-existing class of NPC entities. The basic AI simply mirrors the player's movements, while at the same time moving in the shortest possible path towards him. While this works, it creates some unwanted side effects, both identified as bugs in testing. FIrstly, when the player is stationary and the NPC ships must take a diagonal path to the player, the NPCs travel in a "wiggling" motion. While this does not affect the playability of the game, it could affect the user experience and reliability, under UR_RELIABILITY. This issue wasn't an easy fix, as it was an effect of the AI selecting the fastest route to the player, and unless this was directly up, down, left or right, a diagonal line (the hypotenuse of a right angle triangle between player and NPC) would have to be traversed, causing the ship to need to change direction on every crame (as to appear to go diagonally). All ideas to fix this were time consuming and involved fundamentally changing the way the AI we had just implemented was basing its motion off. The second issue comes from the NPC's lack of NPC to NPC collision. PLayer to NPC and Payer to College collision had already

been implemented from Assessment 1, however NPC to NPC had not (as these were static so could never overlap). This was an overlooked detail until testing, when the issue became apparent that multiple ships would overlap and travel in “one unit”. This would make it impossible to see how many NPCs were travelling together, further adding to issues with the UR_RELIABILITY requirement, as a player may not understand that multiple NPCs are following him and that that is the reason why so many bullets are needed to destroy what would appear to be only one boat. This is not a complicated fix, but takes time (which we didn’t have much) and would have required adding another type of collision under the Update method of the World class, where regular collisions are checked for and processed on a frame by frame case.

3. Obstacles- *fully implemented*
   In accordance with UR_NEW_OBSTACLES and its respective FR_POINT_EARNING_WEATHER and FR_BAD_WEATHER, obstacles have been added in 3 forms: Shipwreck, Storm and Rocks. Between them, the 3 obstacle types cover each FR, by both introducing bad weather conditions (Storm) but also by allowing such obstacles to have a potentially positive impact, such as a Shipwreck, that allows a chance of earning gold by travelin onto it. The effects of the other two are to deal damage, and to cause the player to travel at half the speed. The Obstacle Class was added to game.world.content, next to other classes that extend the Entity class (such as Bullet and NPC), and works not too dissimilarly from how a player collision would be processed- if the rectangle that contains the player overlaps the rectangle that contains the Obstacle (size of which pre defined by us in the entities.XML), the Obstacles class would be invoked. This all happens in the update method of the worldCycle in the World class. Invoking an obstacle will cause its accessory functions to apply, taking damage every 2 seconds or having a reduced speed for the amount of frames that the player overlaps the obstacle for. This is fully implemented with no known bugs, though it is possible (however not easily, and testing has established that this is not a bug that requires fixing) to apply an obstacle while being entirely out of the obstacle area, due to one or two of the ship’s encapsulating rectangle pixels overlapping with the obstacle rectangle area. As mentioned, this is hard to achieve and would not happen accidentally, therefore, the Obstacle functionality has been added with no issues.

4. Powerups - *partially implemented*
   In accordance with UR_POWERUP and its respective FR_POWERUP, powerups have been added, three different types which are damage, xp and speed. The damage powerup doubles the player damage for a set duration, the xp powerup gives you xp and the speed powerup doubles the player's speed for a set duration. As reference in the architecture the powerups class inherits from entity. The Data for all of the powerups is in the entity.xml file. Powerup is similar to how obstacles were implemented but the powerup is disposed of and removed from the set when the player overlaps with it and the player gets the power. This all occurs in the update method of the world class. The three powerups do not fully cover FR_POWERUP as there aren't 5 different types of powerups and use a temporary png for each of the powerups. This is due to the fact that other sections of the game take longer than necessary like GoldShop and save Menu which produced large amounts of errors.

5. pause / save - *Pause implemented, save not implemented*
   In accordance with UR_SAVE_STATE and FR_SAVING saving has not been implemented, what should have occurred was that the user goes to the pause screen and it presses the text button that would call a method called write in XMLLoader, however the problem occurred in the implementation of the save state, we used the XMLWriter Library in libgdx to implement this method, this allowed us to write xml however we struggled to find where the file we wanted to write to was this created an error with parsing as if the method did not know where the file was it would be unable to write any of the xml to that file. The Pause screen is fully implemented and changes the world screen to the pause screen with all the entities that would be necessary if the save method worked.

6. Difficulty - *not implemented*
   FR_GAME_DIFFICULTY
   According to our functional requirements there should have been options(options screen) to change how challenging the game would be based on the speed of movement of enemy ships.
   We were not able to implement the options screen itself due to complications we run into in the code base. However, the difficulties class(under enums module) was implemented with its constructor using java enums with instant variables(levels EASY, MEDIUM , HARD).
   The game's difficulty level is set currently on Medium manually.