Implementation:

Team 23

Tom Rosewell

Sam Redgewell

Tom Keeler

Jake Stogden

John Limb

Adham Noureldin

Completion of requirements

Previously the only requirements not met from the previous implementation were user requirement UR_COMBAT and functional requirement FR_ENEMY. Both of these have now been successfully implemented and all original requirements have been satisfied.

On top of this extra features have been implemented to improve the quality of the game such as UR_POWERUP and UR_BAD_WEATHER. It should be stated that under certain conditions in the game, bugs may occur that affect the quality and delivery of requirements UR_COMBAT and FR_ENEMY to the user. However, through our thorough testing methods, we were unable to detect any case where these requirements may not be satisfied, which also applies to all other requirements (old or new).

PowerUps (PowerUp.java)
We Implemented power ups by creating non-standard entities within the screen that had their own form of interaction with the playerShip on GameScreen.
This allowed us to build entities onto the map that could be visible to the player and could be interacted with by player ship without the standard values that our entities created with entity() have of Health. As well as easier deletion of them when they are hit by the player.
We also built a new formula for randomly accessing a location on the map which was under PowerUp.PowerUp().

Most PowerUps Operated by changing values by changing set values for the Ship entity however, freeze and shield operated differently.
Shield required the altering of the Ship damage method, with a new Bool being implemented for shield status. The Ship damage method was made to check ShieldHP before Ship HP.
Freeze had to prevent ships moving, so a new Bool was created within EntityAI to allow for the freeze update to occur which would stop any movement as well as shooting functionality.

Obstacles (Obstacles.java)
Obstacles were implemented in a similar manner to PowerUps where they were non-standard entities, they also implemented the same new random location getter, minor changes were added to create collision damage however no new methods.

Combat (EntityAi.java)
Combat was implemented by creating the methods for aiming at players, that used the getting of player locations and then firing at them, this was done by giving the ships a cooldown for shots that can be fired and then allowing them to fire at the player location, if they are within range.

Weather (Weather.java)
Weather was spawned in the same way as obstacles and PowerUps with the random algorithm; however a separate algorithm was created to define the movement of this entity as it moved in a random direction. This was all done within Weather.Weather()
Weather was also an entity with hitbox collision off, which deviates from our standard builds

allowing a player ship to be inside.
Updates were used to check whether the player was inside and deal appropriate damage.

Save States
Save states were implemented using already in-built functionality inside GameScreen() where information was stored in local variables, only one save state was available that used the built in screens variables to save the information, however if the screen was accessed by creating a new game the values would reset

Plunder
Most of this was created inside the Ship() entity and deviated little from standard practice.

Difficulties (GameDifficultyScreen.java)
Difficulties were split into easy, intermediate and hard, we altered difficulties by changing pre-existing values from other modules within the new GameDificultyScreen(). Health, cannonball damage, College health were all altered from this build.
This saved us from further complicating the GameScreen() file with more values for it to store and dealt with all of the respective issues, and if difficulty was needed to be worked out it could be reverse engineered from the values that the affected variables had been set to.