Implementation

Cohort 1 Team 9

Dominic Hall
Firas Marzouk
Ben Morrison
Harry Whittaker
Amelia Wigglesworth
Zehang Li

3. Implementation [25 marks]:

   a) Provide documented code for a working implementation of the game that meets the remit, requirements and concrete architecture for Assessment 2. Your code comments should highlight new or extended sections of code, and should be consistent with your change report. Code can be submitted in the zipfile, or via a link to a repository with a verifiable date before the hand-in deadline. An executable JAR of the game, that includes all external dependencies, must also be included in the zipfile. (15 marks)

   b) Explain how your code implements your architecture and requirements (incorporating your recorded changes for Assessment 2). Briefly explain any significant new features, e.g. non-primitive data types, significant algorithms or data structures. Give a systematic report of any significant changes made to the previous software, clearly justifying each change, and relating it to the requirements and architecture by pointing to relevant class names and requirement IDs. Note that, if a change has significant side effects, it needs a solid software engineering justification. State explicitly any of the features required for Assessment 2 that are not (fully) implemented. (10 marks, ≤ 4 *pages*)

3. (a)
https://github.com/yorkpirates/Assessment-2-Game
https://assessment2.yorkpirates.uk/game/Pirate-Game-Group-9.jar

3. (b)

Implementation Architecture

Our code implements the architecture by sticking primarily to the hierarchy that is laid out in the abstract.This can be seen primarily by how entities are composed. Entities ,which would be classified as Game Objects ,are composed from components giving basic functionality which can be later built upon. Such as the Renderable component which holds sprite and other graphics related things. This can be seen in figure 3.1.3 where you can see the composition of entities and which components they use.

When we have made changes to the code they have primarily been changes to inherited classes which already conform to the architecture and have endeavoured to keep them inline with it. When we have added classes we have kept to the architectural design by adding in classes that stick to the predefined abstraction and fitting them into the architecture.

Implementation of Requirements

Our code implements the requirements by constructing first the foundational layer which is the game world and then populating it with actors/entities. The code gives them the ability to interact with the world and these interactions are mediated by Managers. It also gives the player control over one of them thus allowing the player to interact with other entities and the world. This allows the code to meet the requirements as it provides the functionality required by it primarily for the player.

We made additions to the inherited code as well as modifications to meet the requirements such as adding monsters and weather and their interactions with the player and the world.

Changes to Previous Software

We added a save game file that was XML along with methods for both reading and writing this format. We chose to use the STAX parser because of its speed and memory efficiency. The structure of how it parses the file in a linearly sequential manner also suited the functionality we wanted of loading a game.

Another significant new feature was the Shop. The requirement ‘UR_SPEND_MONEY’ requires a way for the player to spend the plunder that they have earnt during game play. We implemented a shop feature, with buttons a user can press to exchange money for goods, such as ammo and power ups.

The previous game had not yet implemented the firing of cannonballs from a college to a ship. To implement this we had to write a new class, as the previous cannonball classes only allowed a cannonball to be fired from a ship. Our version of the cannonball - which would fire from colleges only - would give more damage, and fire directly at the user in regular intervals to increase the difficulty of attacking the college - the user will always have to be on the move.

The table below outlines the significant changes that we made to the previous software, including our justification for doing so, the requirements it met and the architecture it impacted.

Implemented Features

Change	Justification	Requirement	Architecture
Changes to college for saving	Is alive now returns a bool, so when saving we know whether or not the college is alive. Added a method to destroy all college buildings effectively killing it, which is used when loading and a college is dead	UR_SAVELOAD	College class
Change to building	The destroy function has been made public, so it is now able to be called externally when the college is attacked	UR_HOSTILE_BUILDING_COMBAT	College class
End screen wincheck	Points as a second win condition were added, so we wrote a function to check if points requirements based on player difficulty setting were met before win	UR_GAME_WIN	EndScreen class
Game difficulty (edit UI, change starting stats)	To meet the requirement of multiple difficulty levels we had to add a selector to the start screen, and based on the user choice change the starting levels of health.	UR_DIFFICULTY	MenuScreen Class
College death implementation	The college now gives rewards, points and plunder when destroyed.	UR_HOSTILE_BUILDING_CAPTURE	College class
Increase cannonball life	Cannon balls were not travelling far enough and would often miss close targets.	UR_FIRE_WEAPONS	Cannonball Class
Added points	Adding points required a few changes to the old game, for example adding a row in the in game stats, adding a point field within pirate and making game events (time passing, destruction of a college) give points.	FR_POINTS_UPDATE UR_EARN_POINTS FR_POINTS_TRACKING	
Remove physics manager	Due to the fact that objects in the physics world ID's are stored sequentially in an arraylist we cannot simply remove them. We instead must set them as inactive meaning that they do not interact with the physics world. This must be done after the world steps through and not while in as this will cause an error. Hence if an object must be removed from the world its id is stored in a list. After every update the physics manager will loop through the items and set them as inactive.		
Game screen check player health	The game checks that the player is alive in the game screen so it knows whether or not to switch to the end screen		GameScreen class

Added removeFromPhysicsWorld to rigid body	This allowed us to remove entities when they were no longer needed. Previously they had just been teleported off screen. Now they should no longer react with the physics world, so should have less of a resource impact.		RigidBody class
Remove static methods	The health functions in pirate were static, which made having multiple different pirates with different healths difficult to implement. Other methods such as adding plunder were made non-static too.		Pirate class
Make array list of ships public	This allows other methods to access the list of ships, for example when setting targets.		
Writing new getters and setters	To save and load a previous game, we needed to be able to access key stats (ammo, position) to save and set them to load. We wrote getters and setters accordingly.	UR_SAVELOAD	Player class
Change the zoom of the game	The requirement was to fit a 13 inch screen, but when playing on one we noticed that in order to see an enemy college on an island you would have to run the boat against the shore. Zooming out, so that more of the map could fit on a screen makes the game more accessible for this screen size.	FR_VIEWPORT_SCALING	Change made in the constants file and used within the GameScreen
Added a quit confirmation screen	Originally if ‘esc’ was pressed the game would quit. This was not very user friendly, so we added a screen to confirm the user’s intent to quit.	UR_QUIT	EndScreen class

Unimplemented Features

Feature ID	Reason
UR_FRIENDLYSHIPENCOUNTER	We ran into issues getting the NPCships to target other factions which would have been the primary function of these ships and thus they were not implemented.
UR_GAME_WIN	We could not implement an “Ultimate objective” (the previous group has suggested a final boss battle to win the game), but instead we implemented a series of quests to reach a win screen.