ENG1 Team Project
Assessment 2

Deliverable 3: Implementation

Team 13 – Team Unlucky

LILLY BROWN
ROSCOE GILLATT
ADAM JOHNSON
YUMIS ZYUTYU
BRANDON WEST
AYMAN ZAHIR

a)https://github.com/apj520/ENG1-Team-13-Assessment-2/tree/testing_Lillyv2

b)
The architecture is implemented as an Entity/Component where all game objects such as Ships and Monsters have their functionality defined with the help of components such as Renderable to handle rendering of game objects onto the screen or Transform to manipulate the positioning of all entities within the game.

The game itself is defined as a PirateGame 'parent' which creates all the screens which implement the Page abstract class and loads resources and game assets. These screens are responsible for handling all the UI elements and functionality outside of the gameplay itself. UI elements and our changes to the UI were easily defined in these classes.

The GameManager class is responsible for most of the game functionality including initialising all the game entities such as Array Lists containing ships or cannonballs. It also defines getters and setters to access all the game entities through the GameManager. To update or manipulate any behaviour of these objects we simply get the specified component and call its function. For example, to ensure an entity is re-rendered after restarting the game we used the Renderable components show() method.

The game also has an AI package which we extended to implement new features for Monsters and weather obstacles.

Much of the code was already well implemented and only facilitated the main classes such as GameManager and the MenuScreens. Hence most of the changes were done within those classes apart from major modifications the the screens, AI package and selective Entity and Component classes to extend functionality to new and existing game objects. Most of these significant changes are documented below.

Systematic Report of Changes:

REQUIREMENT ID: UR_SPEND_MONEY
GITHUB ISSUE #19
CHANGE 1: Change was made to the UI of the GameScreen to implement an interactive Shop Window to buy extra Ammo and Health with Plunder. This was implemented using the existing body of code such as Tables and Windows within GameScreen. Within the GameScreen constructor exists a new Window 'shopWindow' with Labels describing the items and cost along with TextButtons that allow the player to purchase the items. The buttons are initially disabled, only unlocked when enough Plunder is acquired. This is implemented in the GameScreen's update method with simple conditional statements. The methods addAmmo() and addArmor() were created within the Pirate class, to purchase extra ammo and health respectively.
SIDE-EFFECTS: None to note

REQUIREMENT ID: FR_GAME_DIFFICULTY:
GITHUB ISSUE #21
CHANGE 2: To implement loading the game with varied difficulties we had to significantly restructure the sequence of when the three screens are created. Originally GameScreen is created under the PirateGame class' create method along with all the other screens. This prevented us from being able to load a different JSON file with different difficulty settings.
To fix this we created the GameScreen only after a difficulty is selected from the MenuScreen button click events so that the GameManager can load the selected JSONfile under the Initialize(), method rather than returning a NullPointerException for initialising the

game without having selected a JSONfile. Hence, a new PirateGame method startGame() which creates the GameScreen is called under MenuScreen once a difficulty is selected. To keep track of user's selection we utilised the Preference data-type in the MenuScreen.
SIDE-EFFECTS: We included and renamed 3 new JSON files under assets to represent the different game difficulty settings to be loaded.

REQUIREMENT ID: FR_SAVE_POINT
GITHUB ISSUE #22
CHANGE 3: As the game initialises by reading the settings from the JSON files, to implement the saving the hardocodedLIBGDXJSON() method under GameScreen utilises the JSON and JsonWriter API's to write a hardcoded JSON String object to store the current game stats. This also required writing additional getters for every class that required its attributes saved. This JSON object is saved as a JSON file by using FileHandle class and writeString method within the save buttons event listener. When the user clicks continue in the MenuScreen the game now initialises with the saved JSON settings.
SIDE-EFFECTS: We included a new JSON file 'GameSettingsSaved.Json' to save the current game state. This JSONfile required a modified format to load more entity attributes such as colleges' kill status and thus the other three JSON files for difficulty had to be reformatted as well. Developers need to be aware that when the structure of the JSON files changes, the hardocodedLIBGDXJSON() method also needs to be refactored to match the file structure.

REQUIREMENT ID: FR_GAME_RESET
GITHUB ISSUE #22, #23
CHANGE 4: Due to the game's original structure we could not simply reinitialize a new game without exiting the application as the GameScreen restarted on top of the previous game. Finding no ideal solution to this, instead we implemented the restartGame() function within GameManager which resets Entity attributes to their initial values. It manually resets all player stats, repositions, uncaptures and revives all ships
SIDE-EFFECTS: Manually restarting had outliers such as colleges remaining destroyed upon restart. Hence restart remained partially implemented due to time constraints. it was the next best option after failing to reinitialize the entire game by creating a new game screen. It was implemented late into production, hence we had no time to produce a more elegant solution given how the code was originally structured.

REQUIREMENT ID: UR_POWER_UPS
GITHUB ISSUE #18
CHANGE 5: In GameManager, 5 power-ups are initialised. The power-ups are positioned on the map in the same method in which they are initialised (CreatePowerUps()). The power-ups each have a transform, rigidbody and renderable component, and depending on what the name of the power-up is, will apply a different effect to the player. Three of the power-ups affect player speed temporarily, whether the player takes damage, the amount of plunder the player will receive, if the player fires two bullets instead of one (at the cost of only one bullet). One power-up restores player health to 200. All the values aside from player health return to normal after a predetermined time specified by the game difficulty.
SIDE-EFFECTS: Collision with NPC ships causes the power-ups to move. Collision with a player removes the power-ups from the map.

REQUIREMENT ID: FR_BAD_WEATHER
GITHUB ISSUE #28

CHANGE 6: Included in initialisation of the game is a Hurricane entity. The entity is very simple, has no AI navigation or randomised movement, and no collision interaction. Damage is dealt to any ship that is within a predetermined distance to the hurricane found in the JSON settings file. Damage is dealt every 5 seconds, starting from when an entity enters the area of effect of the hurricane. If no ship enters the area of effect for 5 seconds the hurricane will deal damage immediately when a ship does enter.
SIDE-EFFECTS: No side effects as the hurricane has very limited functionality. Friendly player ships lose health or get killed when colliding with Hurricanes.

REQUIREMENT ID: FR_OBSTACLES
GITHUB ISSUE #27
CHANGE 7: Included in initialisation of the game is a monster entity. The entity is very simple, has no AI navigation or randomised movement, and no collision interaction. Damage is dealt to the player ship only, through collision with monster-specific cannonballs that have a greater damage than normal ship cannonballs, and move marginally faster. Depending on the difficulty the cannonballs deal different amounts of damage and are very powerful so pose a challenge if they aren't avoided. The area of effect is very big, again causing the player to have to navigate around and avoid being hit.
SIDE-EFFECTS: None to note

REQUIREMENT ID: FR_COMBAT
GITHUB ISSUE #26
CHANGE 8: Combat between ships was implemented from scratch. Various relationships between ships occur in the game. The EnterTrigger/ExitTrigger methods in NPCShip class are called when various interactions take place. The player can choose to fire at opponent ships, damaging them until their health is zero. At this point the ship becomes a friendly AI ship and acts as intended, attacking enemy NPC ships and following the player. NPC ships will engage in combat with each other. There is no friendly fire implemented. The EnemyState class acts as the state machine for the NPC ships and dictates what action each NPC ship has. The actions are determined by the canAttack() and isAgro() methods that determine the distance between two ships and what action is taken.
SIDE-EFFECTS: If a friendly NPC ship dies, it will be removed from the map. Combat between NPC ships from two different colleges (not including player college) is disabled. The ships just attempt to follow and attack the player and associated NPC ships.

REQUIREMENT ID: FR_COLLEGE_CAPTURE
CHANGE 9: When a player fires at college buildings, the buildings are destroyed by virtue of the destroy() method in Building Class. If all the college buildings are destroyed then the college is captured. At this point the quest to capture that particular college will be completed.
SIDE-EFFECTS: None to note

New classes were added for major functionality, whilst pre-existing classes had methods added where needed to implement the required functionalities. In particular, to fulfil FR_OBSTACLES which requires the game to have monsters as obstacles, the EnemyState class was used as inspiration to create a MonsterState class that allows for automated firing at the player. Further functionality could be implemented in this class, including following the player and firing at other entities in the game. Within the NPCShip class, the removeTargets() method is called upon a NPC ship dying, which in return removes the dead ship from the target list of all other ships that happen to contain it. This was implemented as the pre-existing method for this removal (ExitTrigger()) was not working, causing a bug to make the remaining ships that target the dead ship to travel to a part of the map where the dead ship sprites are stored.

The most complex logic in the codebase occurs in NPCShip in the EnterTrigger() method. As there are several complex relationships between the various entities considered, a series of if-else statements determines whether a particular ship becomes a target of another. This method also encapsulates the relationship between NPC ships and the player. The NPC ships actions are determined by the EnemyState class which automatically calls certain methods on the NPC ship depending on their proximity to their targets.

In addition, minor changes were implemented either to resolve low priority ‘may’ requirements or to facilitate a better user experience such as:

- Implementation of points as an earnable under the Pirate Class to resolve FR_POINTS_UPDATE and FR_POINTS_TRACKING as well as a timer under the same class to resolve UR_GAME_DURATION. Both are displayed at the end screen.
- Added tutorial text as part of feedback; made more clear that all buildings need to be destroyed to capture a college.
- General modification to the positioning of buttons and UI for improved usability
- General modification of the labels and text in the UI to make game succinct and clear
- Use of the Preference class to save player difficulty responses in MenuScreen

Changes and additions are commented with the developer name followed by the changes for easy visibility of changes.

Unimplemented/partially implemented requirements:

Requirement ID	Reasoning
FR_GAME_RESET	Partially implemented as a soft reset with all player stats being reset along with all ships being respawned to their spawn points, their health reset and all killed ships being revived and rerendered. However, since college reset could not be implemented the overall progress of the game and quest completion is only partially reset.
FR_BOSS_SPAWN	Left partially implemented as a side effect of colleges not being capturable until a quest has been assigned to it but spawning a new final boss has not been implemented
FR_GAME_WIN	Fully implemented for all scenarios except for boss encounter as boss has not been implemented. Displays players at the end of the game including timer.
FR_FRIENDLY COLLEGE	Interaction with the friendly college by the player is not implemented. The only interaction is that cannonballs from the player will hit buildings and disappear, yet no damage will be dealt to the college itself. No other interaction with the player college is possible and therefore has not been implemented fully.