Group 14 - Design Specification

Authors: Thomas Wilkinson [tjw21], Jac Lingard [jal74], Jay Staddon [jas130], Sebastian Sogan [ses35], Ali Bajwa [ALB113]
Config Ref: SE.GP14.DesignSpec
Date: 22 March 2022
Version: 1.3
Status: Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1. Purpose Of This Document

The purpose of this document is to provide a design for the group project and explain many of its features. It explains the inner workings of much of the project such as the coding aspect, relationships, how to use the system and an overall design for how the project will work.

1.2. Scope

This document will include descriptions on many aspects of the project. It will go over significant classes needed for the programs and the classes shared between them, component diagrams for the programs, class interfaces and algorithms / data structures. The References section includes much of the different coding work.

1.3. Objectives

This document aims to describe how the final program will work and the functionality that it should have along with any other possible functions that would improve it.

2. DECOMPOSITION DESCRIPTION

2.1. Programs in System

The system for this project is one program which will allow users to play the game 'buccaneer' while meeting all of the functional requirements. It will include classes that will create and run the interface that are then linked to front end methods that will alter data in the back end classes.

2.2. Significant Classes

2.2.1. GameLoop

Manages the turn mechanics of the system as well as the movement of ships, trading and attacking between ships and ports on the board.

2.2.2. MapSquare

Serves as the bridge between front-end and back-end code. The Main functions are holding key data and updating the stackpane that each instance of this class extends.

2.2.3. ObjectLoader

Manages the loading of game data, meaning that it ensures everything is loaded in the correct order and added to the map in the appropriate positions. Includes functions which will create the islands and assign the player ports a random player and load the crew cards in the correct places.

2.2.4. Buccaneer

Runnable class which sets up the resources required for the project and loads the .fxml assets.

2.2.5. BoardController

Where the board itself is initialised, it also contains the events for the movement button presses and the instructions and inventory tab selections.

2.2.6. InputController

Includes the functions that insert the player names into the system. This includes the validation of the names.

2.2.7. InstructionController

Controller for the instructions menu which is accessible from the main game menu. Includes the button press event that is needed for the back button.

2.2.8. MenuController

A controller for the main game menu which includes the events for the user attempting to start a game or access the instructions.

2.2.9. Ship

Contains all the data and the functions associated with the ships on the board. Also, it contains the data about the player which is associated with it.

2.2.10. Island

Contains the data and functions associated with each island as well as the data for the docks and a way to distinguish between docks and islands

2.2.11. CrewCard

An implementation of the ICommodity interface which contains the functions and data which apply to the CrewCards and not the Treasure class such as colour.

2.2.12. Treasure

An implementation of the ICommodity interface which contains the functions and data which apply to the Treasure and not the CrewCard class such as the name of the treasure.

2.2.13. ChanceCardBase

Contains the data and functions which are common for most of the crew cards as even though many of them have unique functions, most do one of 2 basic functions on top of it.

2.2.14. ChanceCard 1-28

Individual sub-classes for each chance card that contain the unique functions that they have (if any).

2.2.15. WinController

Controller for the win screen displaying winner and ending options.

2.2.16. SaveData

Data class that handles storage, saving and loading of essential data to facilitate restoring game state.

2.3. Mapping From Requirements To Classes

Requirement	**Classes providing requirement**
FR1 Player Setup	ObjectLoader, MenuController, MapSquare, InputController
FR2 Port Assignment	ObjectLoader
FR3 Crew card management	CrewCard, ObjectLoader
FR4 Chance card management	ChanceCardBase, ChanceCard 1 - 28, ObjectLoader
FR5 Treasure management	Treasure, ObjectLoader, Island, Ship
FR6 Player management	ObjectLoader, CrewCard, ChanceCardBase, ChanceCard 1 - 28, Treasure, GameLoop, Ship
FR7 Port management	Island, ObjectLoader, Treasure, CrewCard, ChanceCardBase, ChanceCard 1 - 28
FR8 Flat Island management	Island, Treasure, CrewCard, ChanceCardBase, ChanceCard 1 - 28
FR9 Board display	ObjectLoader, MapSquare, Island, GameLoop, Treasure, ChanceCardBase, ChanceCard 1 - 28, Ship, BoardController
FR10 Game setup	ObjectLoader, Island, CrewCard, Treasure, Ship
FR11 Taking turns	GameLoop, Ship, Island, MapSquare, BoardController
FR12 Attacking Rules	GameLoop, CrewCard, Ship, Treasure, Island, ChanceCardBase, ChanceCard 1 - 28, MapSquare, BoardController
FR13 Treasure Island	Island, GameLoop, ChanceCardBase, ChanceCard 1 - 28
FR14 Flat Island	Island, GameLoop, Treasure, Ship, CrewCard, ChanceCardBase, ChanceCard 1 - 28
FR15 Arriving at a port	GameLoop, Island, Ship, Treasure, CrewCard, ChanceCardBase, ChanceCard 1 - 28
FR16 Anchor Bay	ChanceCardBase, ChanceCard25, ChanceCard26,

	Island, Ship, Treasure
FR17 Detection of end of game	GameLoop, Treasure, Ship, Island, MapSquare, MenuController

3. DEPENDENCY DESCRIPTION

3.1. Component Diagrams

The program is split into 3 major packages:

- Buccaneer
- Controllers
- Backend
- Custom
- Models

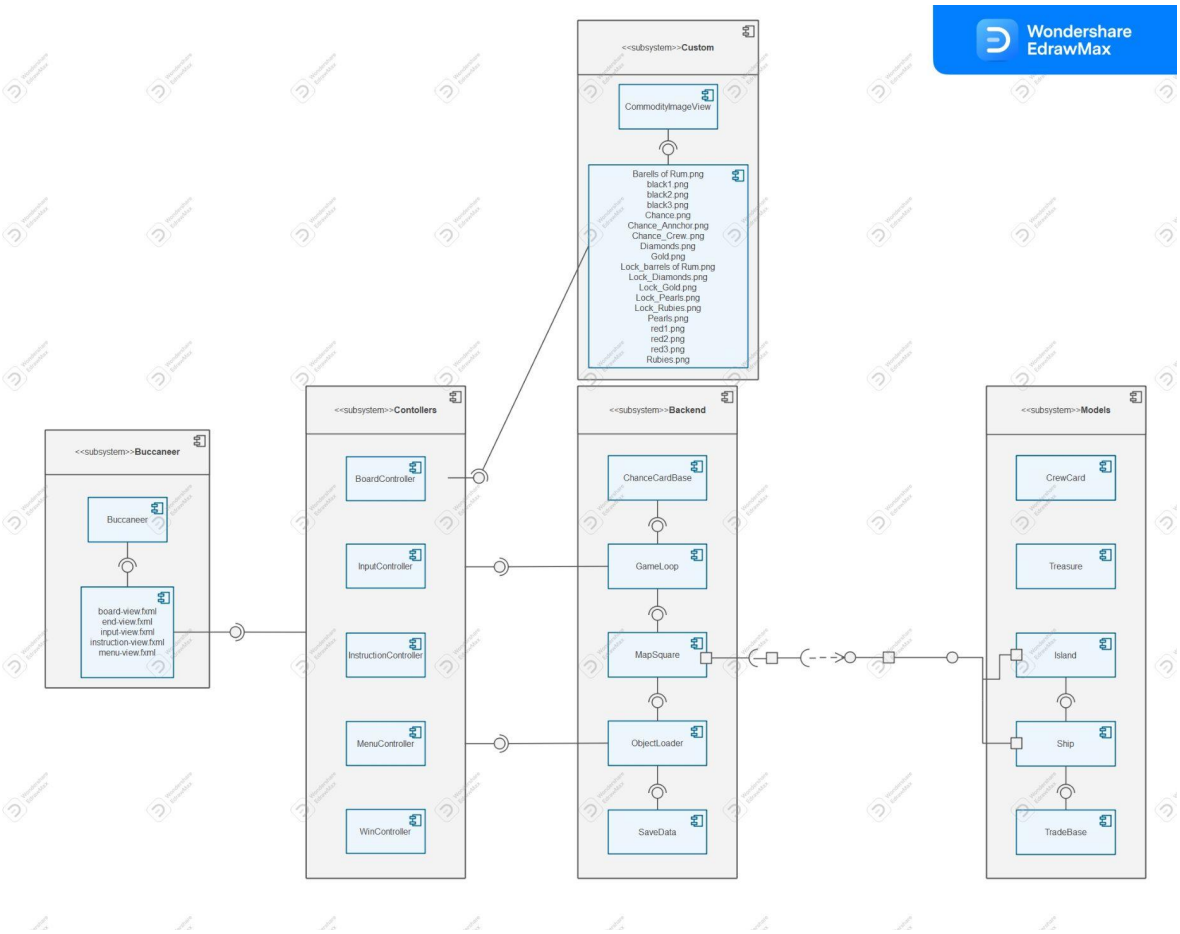
Buccaneer contains a single java class and can be seen at the root of the component diagram. Via Javafx calls this will load all the static “fxml” user interface code. Allowing for code in the linked controllers to be executed.

The board controller requires dynamic user interface features such as updating player positions on a board. So therefore makes calls to “ObjectLoader” and “GameLoop” which will use code from the backend package to manipulate the “MapSquare” class. As MapSquare extends a Javafx stackpane this backend code can use the class as an interface to update the frontend.

The MapSquare as well as being an interface between the backend and frontend serves as an interface for data classes such as “Island” and “Ship”. As these are directly stored within the MapSquare class, the extended stackpane can be updated every time a setter for one of these variables is called.

Custom holds the class “CommodityImageView” which displays commodities to the inventory, extending the JavaFX class “ImageView”. This is utilised by the “BoardController” which uses this to keep the inventory up to date.

The data classes in models are used by the backend, which is passed to “MapSquare” loading in the data. The “ObjectLoader” loads in all the data at the beginning of the game and the “GameLoop” updates the data as the game goes on.


<<subsystem>>Custom
CommoditylmageView
Barells of Rum.png
black1.png
black2.png
black3.png
Chance.png
Chance_Armchor.png
Chance_Crew.png
Diamonds.png
Gold.png
Lock_barrels of Rum.png
Lock_Diamonds.png
Lock_Gold.png
Lock_Pearls.png
Lock_Rubies.png
Pearls.png
red1.png
red2.png
red3.png
Rubies.png
<<subsystem>>Buccaneer
Buccaneer
board-view.fxml
end-view.fxml
input-view.fxml
instruction-view.fxml
menu-view.fxml
<<subsystem>>Contollers
BoardController
InputController
InstructionController
MenuController
WinController
<<subsystem>>Backend
ChanceCardBase
GameLoop
MapSquare
ObjectLoader
SaveData
<<subsystem>>Models
CrewCard
Treasure
Island
Ship
TradeBase

4. INTERFACE DESCRIPTION

Buccaneer

4.1. Buccaneer Interface Specification - Public

- **public void start(Stage stage) throws IOException** - Loads the different parts of the program
- **public static void main(String[] args)** - Launches the program

Controllers

4.2. BoardController Interface Specification - Public

- **- public void loadSave(SaveData data) -** Loads the game board from a save
- **- public void startGame(String[] names) -** Loads the game board
- **- private void loadGrid(boolean save) -** Loads in the game board grid
- **- public void OnBoardPressed(MouseEvent event) -** Lets the player interact with the board
- **- void OnCompassPress(MouseEvent event) -** Changes direction of the ships
- **- void OnTurnEnded(MouseEvent event) -** Handles the end of a players turn
- **- private void updateInventory(TradeBase tradeItem) -** Used to display ship and island inventories on the UI
- **- private void tradeItemClicked(MouseEvent mouseEvent) -** Indicated item has been selected for trade
- **- void tradeClicked(ActionEvent event) -** Checks to see if it is possible to trade items
- **- void movePressed(ActionEvent event) -** Displays movement screen
- **- void inventoryPressed(ActionEvent event) -** Displays instruction screen
- **- void instructionsPressed(ActionEvent event) -** Displays instruction screen
- **- public void setWin(Scene win, WinController controller)** - Sets a win scene the controller can use.

4.3. InputController Interface Specification - Public

- **void playPressed(MouseEvent event)** - Checks the names entered by the players to see if there are any errors with them
- **public void setPlay(Scene play, BoardController boardController)** - Sets a play zone in which the player can move

4.4. InstructionController Interface Specification - Public

- **void onBackPressed(MouseEvent event)** - Loads back to the main menu from instructions

- **public void setMenu(Scene menu)** - Sets what scene is the menu

4.5. MenuController Interface Specification - Public

- **void playPressed(MouseEvent event)** - Checks if the user clicks the play button
- **void instructionsPressed(MouseEvent event)** - Checks if the user clicks the instructions button
- **public void setInput(Scene input)** - Sets what scene is the input page
- **public void setInstructions(Scene instructions)** - Sets what scene is the instruction page
- **public void setBoard(Scene play, BoardController boardController)** - Sets what scene is the board page

4.6. WinController Interface Specification - Public

- **public void setWinner(String winner)** - Sets win
- **public void setMenu(Scene menu)** - Sets menu
- **public void exit(ActionEvent event)** - Exits the game

Backend

4.7. GameLoop Interface Specification - Public

- **public GameLoop(ArrayList<Island> islands, ArrayList<int[]> outOfBounds, MapSquare[][] MapGrid, AnchorPane move_scene)** - Constructor of the class, uses board data to setup game state.
- **public void endTurn(MapSquare[][] MapGrid, AnchorPane move_scene)** - Checks if current player has moved and starts the next turn
- **public void run(MapSquare[][] MapGrid, AnchorPane move_scene)** - Main initialisation of player turn, shows potential move spots.
- **private void calculateMove(MapSquare[][] MapGrid, int value, IntBinaryOperator x, IntBinaryOperator y)** - Calculates the positions a ship can move to using direction data of the ship.
- **private void calculateAll(MapSquare[][] MapGrid, int value)** - Calculates all possible movements in all directions in situations where a ship is leaving a port.
- **public void direction(String direction, MapSquare[][] MapGrid)** - Changes the direction of a ship if it isn't in a port.
- **public boolean move(int[] newCoords, MapSquare[][] MapGrid)** - Allows a ship to move providing the square picked isan allowed coordinate.
- **public ChanceCardBase chanceCard(MapSquare[][] MapGrid, int[] newCoords, ArrayList<ChanceCardBase> chanceCards)** - Used to handle Chance Card scenarios and execution

- **public Ship fight(MapSquare[][] MapGrid, int[] newCoords, AnchorPane move_scene)** - Called when two ships are on the same map square.
- **public void moveStarted(MapSquare[][] MapGrid, AnchorPane move_scene)** - Populates UI with move possibilities
- **public void moveDone(MapSquare[][] MapGrid)** - Clears UI move prompts when a move is made.
- **public Island isIslandDock(MapSquare[][] MapGrid, int[] newCoords)** - Checks if coords given is an island and returns a pointer to it.
- **public boolean trade(ArrayList<ICommodity> islandTrade, ArrayList<ICommodity> shipTrade, ArrayList<ChanceCardBase> chanceCardDeck, Island island)** - Function used to check and apply ship/island trades.
- **public Ship hasWon(MapSquare[][] MapGrid)** - Checks the win condition.
- **public Ship getCurrentPlayer()** - Returns the current players turn.

4.8. MapSquare Interface Specification - Public - extends StackPane

- **public MapSquare()** - Constructor for the mapsquare
- **public Island getIsland()** - Returns a island
- **public void setIsland(Island island)** - Sets UI elements for squares where an island exists
- **public void setDock(Island island)** - Sets the docks
- **public void addShip(Ship ship)** - Ensures the position the ships on the board gets updated
- **public void removeShip(Ship ship)** - Removes a ship
- **public Ship[] getShips()** - Get a copy of all the ship data in a square
- **private void loadIslandUI(String background, Island island)** - Loads tooltip name to indicate island names when hovering
- **private static boolean isJUnitTest()** - Function to mitigate UI API calls if a test is being performed.

4.9. ObjectLoader Interface Specification - Public

- **public ObjectLoader()** - Constructor for the ObjectLoader
- **public void loadMap(String[] names, MapSquare[][] table)** - Loads all data in correct order and adds it to the map contains player names map square grid
- **public void loadSave(SaveData saveData)** - Loads the last game
- **private void loadShips(String[] names)** - Get User Data for ShipsLoad into array list names of the ships
- **private void loadIslands(MapSquare[][] table)** - Load islands with randomised ships map square grid
- **private void multiIsland(MapSquare[][] table, String name, int x, int y, int z, int w)** - Load big islands with using corner coords the map square grid names of the ships

- **private void loadCrewCards()** - Allocates crew cards to islands and shipsReturns the remaining deck.
- **private void loadTreasure()** - Load treasure into trade posts according to number of crew card values.Then load remaining into treasure island.
- **private void multiTreasure(Island island)** - Algorithm to handle rare instances where more than one item of treasure would be required to make the value 8.
- **private void loadChanceCards()** - Load all the chance cards into a deck.
- **public ArrayList<CrewCard> getCrewCardDeck()** - Get the crew card deck.
- **public ArrayList<ChanceCardBase> getChanceCardDeck()** - Get the chance card deck.
- **public ArrayList<int[]> getOutOfBounds()** - Get out of bound coordinates.
- **public ArrayList<Island> getIslands()** - Gets a list of the islands.
- **public ArrayList<Ship> getShips()** - Get a list of the ships.

4.10. SaveData Interface Specification - Public

- **public SaveData(MapSquare[][] mapGrid, GameLoop gameLoop, ArrayList<Island> islands, ArrayList<int[]> outOfBounds, ArrayList<CrewCard> crewCardDeck, ArrayList<ChanceCardBase> chanceCardDeck, MapSquare[][] MapGrid)** - Constructor for SaveData, initialises saveable items.
- **public static void save(SaveData save) throws IOException** - Saves game data in event of crash.
- **public static SaveData load() throws IOException** - Loads saved game data and returns objects.
- **public ArrayList<ICommodity> getTreasureIsland()** - Get treasure island static commodities.
- **public ArrayList<ICommodity> getFlatIsland()** - Get flat island static commodities.
- **public ArrayList<ICommodity> getPirateIsland()** - Get pirate island static commodities.
- **public MapSquare[][] getMapGrid()** - Get map grid.
- **public GameLoop getGameLoop()** - Get game loop.
- **public ArrayList<Island> getIslands()** - Get islands.
- **public ArrayList<int[]> getOutOfBounds()** - Get out of bound coordinates.
- **public ArrayList<CrewCard> getCrewCardDeck()** - Get crew card deck.
- **public ArrayList<ChanceCardBase> getChanceCardDeck()** - Get chance card deck.

Data

4.11. Island Interface Specification - Public

- **public Island(MapSquare[][] table, String name, int[] groundCoords, int[] dockCoords, Ship owner, boolean isTradingPost)** - Holds information about an island
- **public String getName()** - Get Island name.
- **public int[] getGroundCoords()** - Get ground coordinates of the island.
- **public Ship getOwner()** - Get owner of the island.
- **public int[][] getDockCoords()** - Get dock coords of island.
- **public boolean isTradingPost()** - Check if the island is a trading post.
- **private int[][] sortDock()** - Sorts docks into an island or a port
- **public Treasure[] getSecureStorage()** - Gets treasure securely stored in the dock
- **public void addSecureStorage(Treasure secureStorage)** - Adds treasure to the secure storage of a dock

4.12. Ship Interface Specification - Public

- **public Ship(String playerName)** - Constructor for the ship
- **public int[] getCoords()** - Returns the coordinates of a ship
- **public void setCoords(int[] newCoords, MapSquare[][] MapGrid)** - Sets the coordinates of a ship
- **public String getPlayerName()** - Returns a player's name
- **public String getDirection()** - Returns the direction of a ship
- **public void setDirection(String direction)** - Sets the direction of a ship
- **public int moveDistance()** - Distance the ship is allowed to move
- **public int attackPower()** - The attack power of a player
- **public int getIndex()** - Get index of ship on board, used to distinguish ship colour.
- **public boolean equals(Object o)** - Override for equating ships.

4.13. TradeBase Interface Specification - Public

- **public TradeBase()** - Handles data in array lists
- **public Treasure[] getTreasure()** - Gets all the treasure in array
- **public CrewCard[] getCrewCards()** - Gets crew cards in array
- **public ChanceCardBase[] getChanceCards()** - Gets chance cards in array
- **public void addCommodity(ICommodity commodity)** - Adds data to array
- **public void removeCommodity(ICommodity commodity)** - Removes data from array
- **private ArrayList<Treasure> sortTreasure()** - Sorts treasure array
- **private ArrayList<CrewCard> sortCrewCard()** - Sorts crew card array

Cards

4.14. ChanceCardBase Interface Specification - Public

- **protected void takeTreasureOrCard(MapSquare[][] MapGrid, Ship ship, ArrayList<CrewCard> crewCards, int crewLimit, int treasureLimit)** - Gives the player the option to pick up treasure or a crew card
- **public void takeTreasure(Ship ship, Island treasureIsland, int treasureLimit)** - Lets players pick up treasureCreates box for the player to select from
- **private void takeTreasureAction(Island treasureIsland, Label num, int finalCount, Label totalCount, int treasureLimit, boolean addOrMinus)** - Lets players pick up treasure
- **private void closeDialog(Dialog<String> dialog)** - Closes the treasure selection menu
- **public String getName()** - Returns the name of a chance card
- **public String getDescription()** - Returns the text linked to the card

4.15. Chance Interface Specification - Public

- **public ChanceCard()** - Constructor for a chance card
- **public void execute(MapSquare[][] mapGrid, ArrayList<Island> islands, ArrayList<CrewCard> crewCardDeck, Ship ship)** - Used for chance card with unique attributes

Commodities

4.16. CrewCard Interface Specification - Public

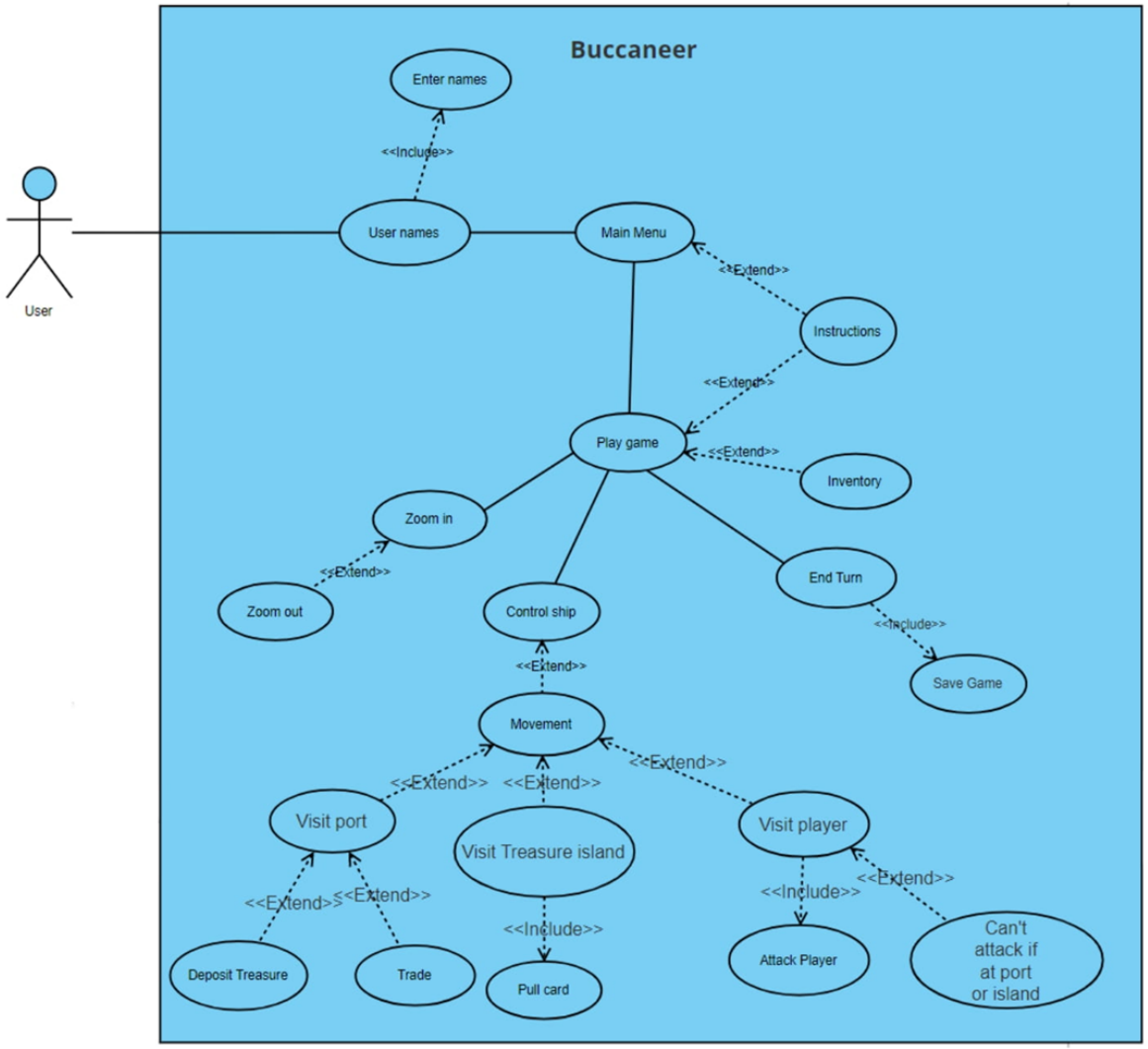
- **public CrewCard(int value, boolean isRed)** - Constructor for a crew card
- **public int getValue()** - Returns the value of the card
- **public boolean isRed()** - Sets of the crew card is red or black

4.17. Treasure Interface Specification - Public

- **public Treasure(String name, int value)** - Constructor for a piece of treasure
- **public String getName()** - Returns the piece of treasure's name
- **public int getValue()** - Returns the value of the treasure

5. DETAILED DESIGN

5.1. Sequence Diagrams



5.2. Significant Algorithms

While doing experimental programming, many parts of the project appeared harder to implement in the code than we had originally thought. This section of the design spec will show pseudo code for these parts of the system or any code that has been drafted.

We had an issue of how we would implement the chance cards as they have such different functionalities so to cover them all would be very hard with one simple class. In the end we decided on going with an abstract class that would be a super class to 28 smaller classes, one for each of the chance cards. We also decided on making an interface for the chance cards that the

abstract class will implement. Here is some pseudo code for the abstract class for the chance cards:

```
Public abstract class ChanceCardBase implements IChanceCard
        String name
        String description

        void takeCrew(Ship ship, ArrayList<CrewCard>crewCardDeck, int limit)

        void takeTreasureCard(Ship ship, ArrayList<Island> islands, String island, int limit)

        public String getName()
                return name

        Public String getDescription()
                return description
```

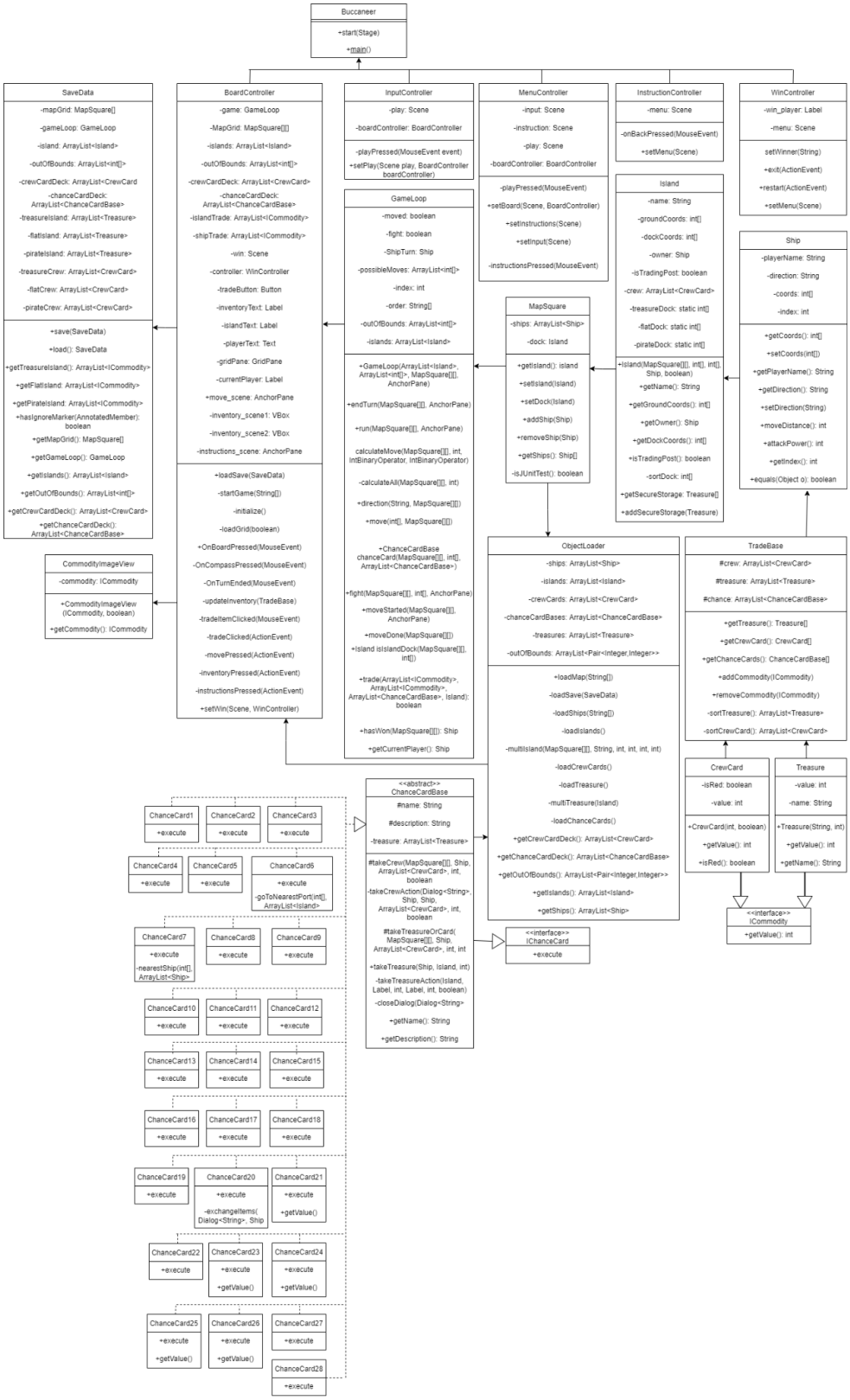
While working through how the island class would work we first decided that the ports and the islands would be in separate classes but after having a discussion with our project manager and going through possible options we decided that it would be best to cover the islands and ports in the same class as it was unnecessary to have too many classes that could be simplified. We adapted our plans for the Island class to accommodate these changes and here is the pseudo code for this new implementation:

```
Public class Island
        String name
        int[] groundCoords
        int[] dockCoords
        Boolean isTradingPost
        Ship owner
        ArrayList<CrewCard> crew
        ArrayList<Treasure> treasure

        …


        public Island(MapSquare[][] table, String name, int[] groundCoords, int[] dockCoords, Ship
        owner, boolean isTradingPost)
                this.name = name
                this.groundCoords = groundCoords
                this.dockCoords = dockCoords
                this.owner = owner
                this.isTradingPost = isTradingPost
                this.crew = new ArrayList<>()
                this.treasure = new ArrayList<>()

                …
```

```
public boolean isTradingPost()
        return isTradingPost

…
```

In the pseudo code above, the use of ‘ … ‘ indicates where there would need to be other code in real implementation of the code as what is shown above isn’t enough for the Island class. The Pseudo code shows that in the Island class there would be common traits between the island and the ports such as the name and coordinates but there is a variable to distinguish the two from each other, this being the isTradingPost variable.

5.3. Significant Data Structures



6. REFERENCES

[1] QA Document SE.QA.05 - Design Specification Standards

7. DOCUMENT CHANGE HISTORY

Version	Issue No.	Date	Changes made to document	Changed by
0.1	N/A	01/03/2022	New Document	TJW21
0.2	N/A	06/03/2022	Started the introduction	TJW21
0.3	N/A	08/03/2022	Edited and finished a full draft for the introduction	JAL74
0.4	N/A	12/03/2022	Worked on document structure based on UI feedback	TJW21
0.5	N/A	19/03/2022	Added component diagrams and the descriptions to section 3	SES35
0.6	N/A	21/03/2022	Added 2.3	JAL74
0.7	N/A	22/03/2022	Added 2.1 and 2.2	TJW21
0.8	N/A	22/03/2022	Added Section 4	JAS130
0.9	N/A	22/03/2022	Added Section 5.2	JAL74
1.0	N/A	09/05/2022	Added Section 5.1	ALB113
1.1	N/A	10/05/2022	Added Section 5.3	JAS130
1.2	N/A	10/05/2022	Updated Interface Description	SES35
1.3	N/A	11/05/2022	Added diagram to Section 3.1	ALB113