Group Project 03 –

Design Specification

Author:	S.Davies-John [std36], A.Sadkowski [ols21], C.Harper [cjh26], L.Purnell [lup35] J.Falkner [jaf43] M. Ciobanu[mnc6]
Config Ref:	SE.GP03.DESIGNSPEC
Date:	11 May 2022
Version:	1.3
Status:	Released

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 specify the implementations of the system and why those decisions have been made, as well as how various parts of the system work and interact with each other.

1.2 Scope

This document details the program, its significant classes, and the modules it uses., as well as providing a description of the module's dependencies and a description of the interface. This document should be read by all project members.

1.3 Objectives

The main objective of this document is to specify the overall design of the program. To outline major features of the project and to help with the implementation. It should specify how the program should be decomposed. It should also provide visual aids through component and interface diagrams.

2. DECOMPOSITION DESCRIPTION

2.1 Programs in system

The Buccaneer program is a board game emulator that allows exactly 4 players to interact with the various elements of the grid board.

2.2 Significant Classes

2.2.1 **Treasure**

Manages the attributes of a piece of treasure, it describes the type of treasure and its value. These objects will be stored on islands, in ports and on ships.

2.2.2 **Treasure Manager**

Initialises the correct number of treasures and allows distribution of treasure to islands, ports and players.

2.2.3 **Card Manager**

Initialises the Chance and Crew cards with the correct number of each and their attributes, it then assigns them to the correct place at the start of the game. It continues to manage the distribution of cards throughout the game.

2.2.4 **Card (super)**

Holds all the attributes that are mutual between both types of card. The Crew and Chance Card classes inherit from this.

2.2.5 **Chance Card**

Stores the information about a chance card, its ID, the card's description and whether it can be held by a player or not. This is used by the Card Manager class and can be stored inside data structures within an island or in a player's hand. This inherits from Card.

2.2.6 **Chance Card Enforcer**

Holds an array of classes to be called when a specific chance card is to be executed. Is implemented by all chance card classes. Also contains a bunch of helper methods to deal with different chance card behaviours.

2.2.7 **Crew Card**

Stores the information about a crew card, its value and the colour (red or black). This is used by the Card Manager class and can be stored inside data structures within an island or in a player's hand. This inherits from Card.

2.2.8 **Game Logic**

Handles many of the possible events that can happen while the game is running. For example, it uses MoveAssistant to highlight possible moves and then handle those moves, to do this it also utilises the states that the game can be in.

2.2.9 **BearingType**

A public enum to handle the possible bearing of a ship. It also holds a method for calculating bearing from two tiles.

2.2.10 **Game State**

Allows Game Logic to set the current state of the game, for example, "rotate" or "movement" states

2.2.11 **Port**

Holds information pertaining to a port: its name, the treasure it holds and its current player owner.

2.2.12 **Island (super)**

Stores the name of the Island and the coordinates of two opposing corners. This is used on the board and by the three other island classes which inherit from it.

2.2.13 **Treasure Island**

Container for chance cards, inherits from Island

2.2.14 **Flat Island**

Container for chance and crew cards, inherits from Island

2.2.15 **Pirate Island**

Container for chance and crew cards, inherits from Island

2.2.16 **Bay**

Stores the name of a bay, it will be contained within the Tile class.

2.2.17 **Battle**

Handles the sequence of events and actions during a battle.

2.2.18 **Trade**

Handles the sequence of events and actions during a trade including calculating the trade value and its success.

2.2.19 **MovementAssistant**

Uses static methods to facilitate possible move calculations by returning a set of possible tiles a player can move to.

2.2.20 **Ship**

Stores all the information about a ship, such as its bearing, colour, starting port and the tile it is currently on. It also stores the player information such as name, number, and colour. The ship class keeps track of amount of treasure it can hold and what treasure it currently holds. The ship class also holds methods for rotating, moving the ship and transferring treasure.

2.2.21 **Tile**

Depicts an individual tile on the board. Holds attributes such as its coordinates, whether it is land or sea, a port if the tile contains one and its JavaFX graphical element.

2.2.22 **Board**

Contains all the tiles for the game board, as well as the locations of the ports and which tiles the three islands span.

2.2.23 **Scene Control**

Utility class for launching and initializing JavaFX scenes and loading FXML files into them.

2.2.24 **XML Utility**

Provides utility methods for writing to and reading from XML files – particularly relating to loading in the cards at the start of the game and handling persistent storage of the game state.

2.2.25 **Startup**

Provides the main method which launches the game and initializes relevant objects.

2.3 Classes shared between programs

There are no classes that are being shared between programs, as we are only using one program.

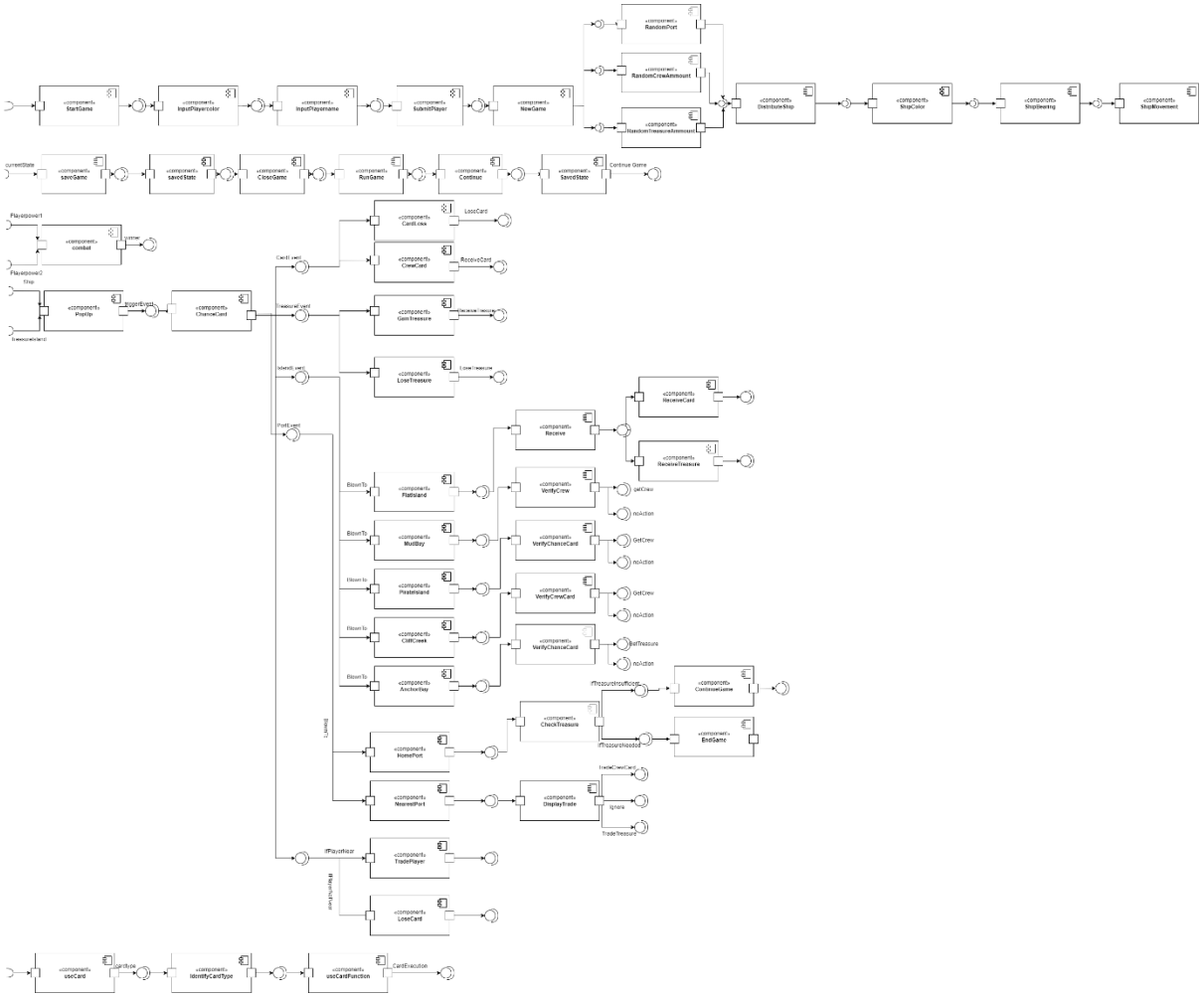
2.4 Mapping from requirements to classes

Requirement	Classes providing requirement
FR1	Game Logic
FR2	Port, Game Logic, Board
FR3	Card Manager, Crew Card
FR4	Card Manager, Chance Card
FR5	Treasure Manager, Treasure
FR6	Player, Game Logic
FR7	Port, Game Logic
FR8	Flat Island
FR9	Board
FR10	Game Logic, Card Manager
FR11	Game Logic, Game State, Board, Battle
FR12	Crew Card, Game Logic, Battle
FR13	Treasure Island, Game Logic
FR14	Flat Island, Game Logic
FR15	Port, Game Logic, Trading
FR16	Bay, Game Logic
FR17	Game Logic

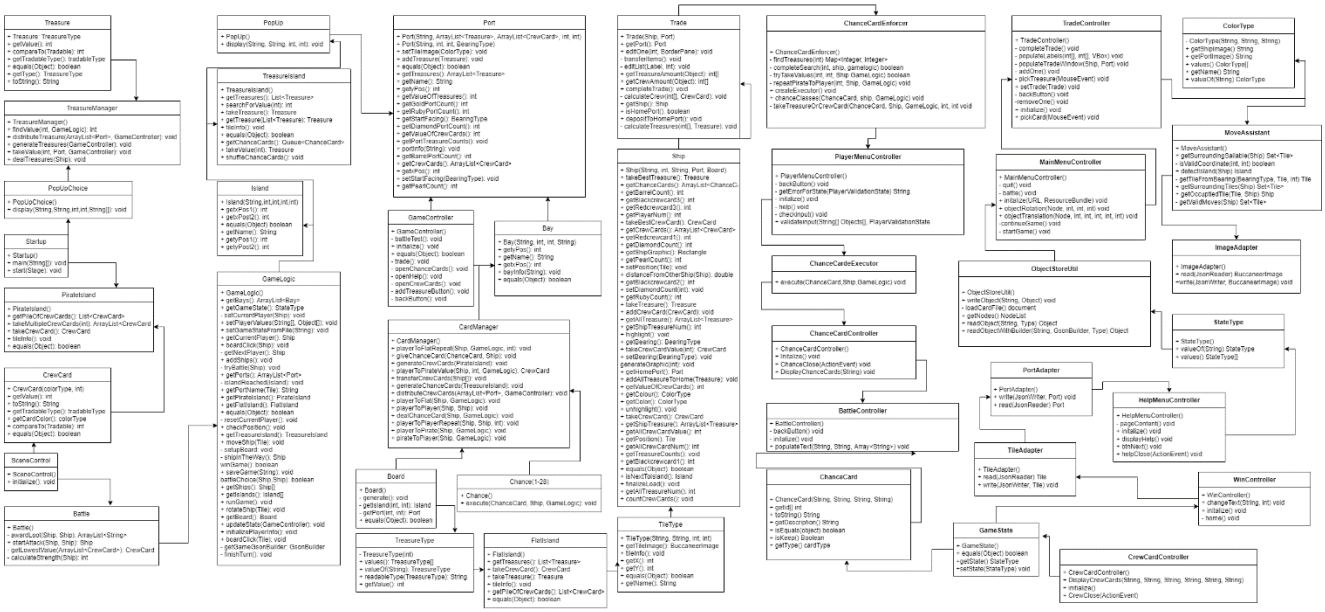
3. DEPENDENCY DESCRIPTION

3.1 Component Diagrams

3.1.1 Component Diagram for Program 1



4. INTERFACE DESCRIPTION



4.1 Startup.java

Main:

- Parameters: String[] args
- Returns: void
- Initialises the JavaFX for the game to start. Called when the program is first run.

Start:

- Parameters: Stage stage
- Returns: void
- Initialises the window of the game by giving it its name and displaying the main menu screen.

4.2 SceneControl.java

Initialise:

- Parameters: N/A
- Returns: void
- Initialises the stage and different screens that can be shown in the game.

4.3 ObjectStoreUtil.java

WriteObject:

- Parameters: String path, Object o
- Returns: void
- Writes the Object o to a JSON file at path.

writeObjectWithBuilder:

- Parameters: String path, Object o, GsonBuilder builder
- Returns: void
- Same as writeObject, except takes a GsonBuilder to help in coordination with TypeAdapters provided by Gson

readObject:

- Parameters: String path, Type token
- Returns: Object
- Reads and returns an object read from the JSON file stored at path

readObjectWithBuilder

- String path, GsonBuilder builder, Type token
- Returns Object
- Reads and returns an object from the JSON file at path using the provided GsonBuilder

getNodes:

- Parameters: N/A
- Returns: NodeList
- Uses the loadCardFile to get the chance card file and then reads the elements by the tag name and returns that list

4.4 MoveAssistant.java

getValidMoves:

- Parameters: Ship ship
- Returns: Set<Tile>
- Produces a set of the possible valid moves a player can make based on their crew card value and their bearing.

detectIsland:

- Parameters: Ship ship
- Returns: Island
- Gets the surrounding tiles around a player and checks if any of them are Island tiles, and returns the found Island or null if no Island was found.

getOccupiedTile:

- Parameters: Tile position, Ship currentShip
- Returns: Ship
- Returns the ship occupying position other than currentShip

getSurroundingTile:

- Parameters: Ship ship
- Returns: Set<Tile>
- Returns the set of tiles surrounding ship, this should return 8 tiles if all coordinates are valid positions, but returns less if there are obstacles / the edge of the board

getSurroundingSailable:

- Parameters: Ship ship
- Returns: Set<Tile>
- Returns a set of the surrounding tiles which are water (TileType=null), this is used when highlighting the surrounding tiles as we only need to highlight the water tiles.#

4.5 BuccaneerImage.java

BuccaneerImage:

- Parameters: String s
- Returns: void
- Constructs a new BucaneerImage, which can store the Url of an image stored in the resource folder, this is used in storing the state of a game / the images within it

getInputUrl:

- Parameters: N/A
- Returns: String
- Returns the Url of the image stored in the resource folder

4.6 Battle.java

startAttack:

- Parameters: Ship attacker, Ship defender
- Returns: Ship
- This method begins the attack sequence for a ship attacking another ship, it will return the loser of the battle

4.7 BattleController.java

Initialise:

- Parameters: N/A
- Returns: void
- Creates the battle screen when it is called

populateText:

- Parameters: String winnerName, String loserName, ArrayList<String> lootText
- Returns: void
- This fills in the correct information for the battle screen, such as the player names, who won and the treasure that has been looted.

4.8 GameController.java

Initialise:

- Parameters: N/A
- Returns: void
- Initialises the board and game graphics

Equals:

- Parameters: Object o
- Returns: Boolean
- Checks if two objects have the same class

4.9 GameLogic.java

GameLogic:

- Parameters: N/A
- Returns: void
- Initialises an instance of GameLogic with all of the relevant information for a game such as the Islands, Ports, Ships and Bays.

setPlayerValues:

- Parameters: String[] playerNames, Object[] playerColours
- Returns: void
- Stores the inputted names and chosen colours for the players so they can be used when the game is created.

runGame:

- Parameters: N/A
- Returns: void
- Creates a new game state and gets the first player

getBays:

- Parameters: N/A
- Returns: ArrayList<Bay>
- Gets the list of bays in the game (Mud Bay, Anchor Bay and Cliff Creek)

getIslands:

- Parameters: N/A
- Returns: Island[]
- Gets a list of the three islands (flatIsland, pirateIsland and treasureIsland)

initializePlayerInfo:

- Parameters: N/A
- Returns: void
- Called before the game scene is loaded to initialise and convert the player colours into a list of ColourTypes

getPorts:

- Parameters: N/A
- Returns: ArrayList<Port>
- Gets the list of Ports on the board

getGameState:

- Parameters: N/A
- Returns: StateType
- Gets the state of the game (used for logic of a turn)

addShips:

- Parameters: N/A
- Returns: void
- Adds all of the player's ships to the board

getShips:

- Parameters: N/A
- Returns: Ship[]
- Gets the list of player's ships

getPirateIsland:

- Parameters: N/A
- Returns: PirateIsland
- Gets the instance of PirateIsland for that game

getTreasureIsland:

- Parameters: N/A
- Returns: TreasureIsland
- Gets the instance of TreasureIsland for that game

getFlatIsland:

- Parameters: N/A
- Returns: FlatIsland
- Gets the instance of FlatIsland for that game

getBoard:

- Parameters: N/A
- Returns: Board
- Gets the instance of the board for that game

getCurrentPlayer:

- Parameters: N/A
- Returns: Ship
- Gets the current player's ship

winGame:

- Parameters: N/A
- Returns: boolean
- Checks the currentPlayer's port to see if the value of treasures is 20 or more, if it is then it shows the win screen.

moveShip:

- Parameters: Tile tile
- Returns: void
- Moves currentPlayer to the tile provided.

checkPosition:

- Parameters: N/A
- Returns: void
- Checks if a player is in a port or bay and then runs through the appropriate actions at those locations

rotateShip:

- Parameters: Tile tile
- Returns: void
- Rotates currentPlayer's ship to face tile (using the calculateBearing method)

boardClick:

- Parameters: Tile tile
- Returns: void
- Handles the actions when a tile is clicked

battleChoice:

- Parameters: Ship instigator, Ship victim
- Returns: boolean
- Checks if the user wants to begin a fight and then begins the attack sequence if they do

shipInTheWay:

- Parameters: N/A
- Returns: Ship
- Checks if there is a ship in the way of another ship's movement and returns that ship

boardClick:

- Parameters: Ship ship
- Returns: void
- Handles click events on ships

getPortName:

- Parameters: Tile tile
- Returns: String
- Gets the name of the port that occupying a tile

equals:

- Parameters: Object o
- Returns: boolean
- Compares this GameLogic object to another to check for equality, used in loading and saving the game

updateStats:

- Parameters: GameController game
- Returns: void
- Changes and updates the labels on the screen to display the statistics

saveGame:

- Parameters: String fileLocation
- Returns: void
- Save the game to a file stored in fileLocation

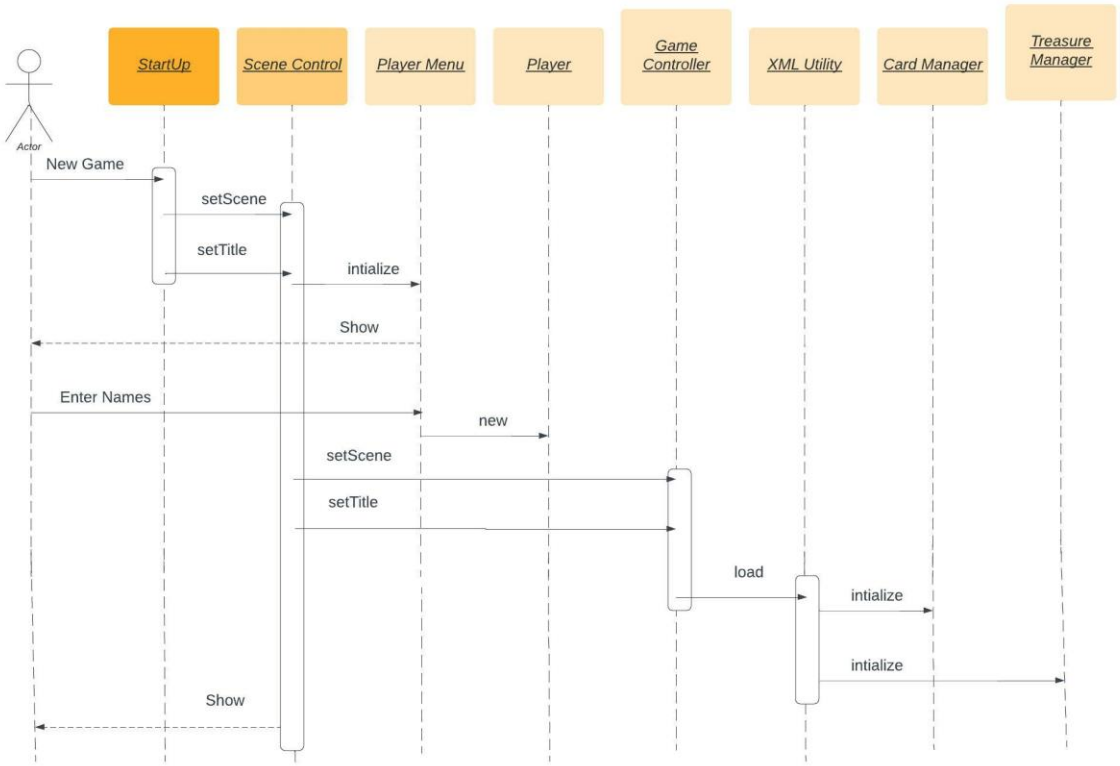
setGameStateFromFile:

- Parameters: String fileLocation
- Returns: void
- Sets the game's state from the file stored in fileLocation

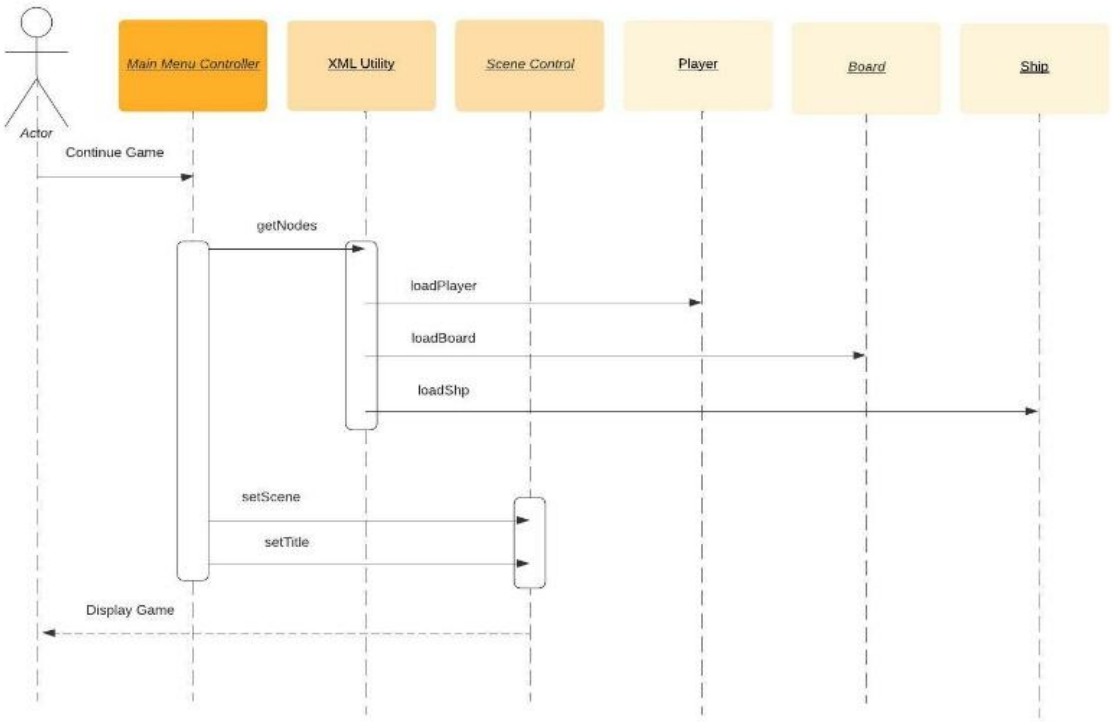
5. DETAILED DESIGN

5.1 Sequence diagrams

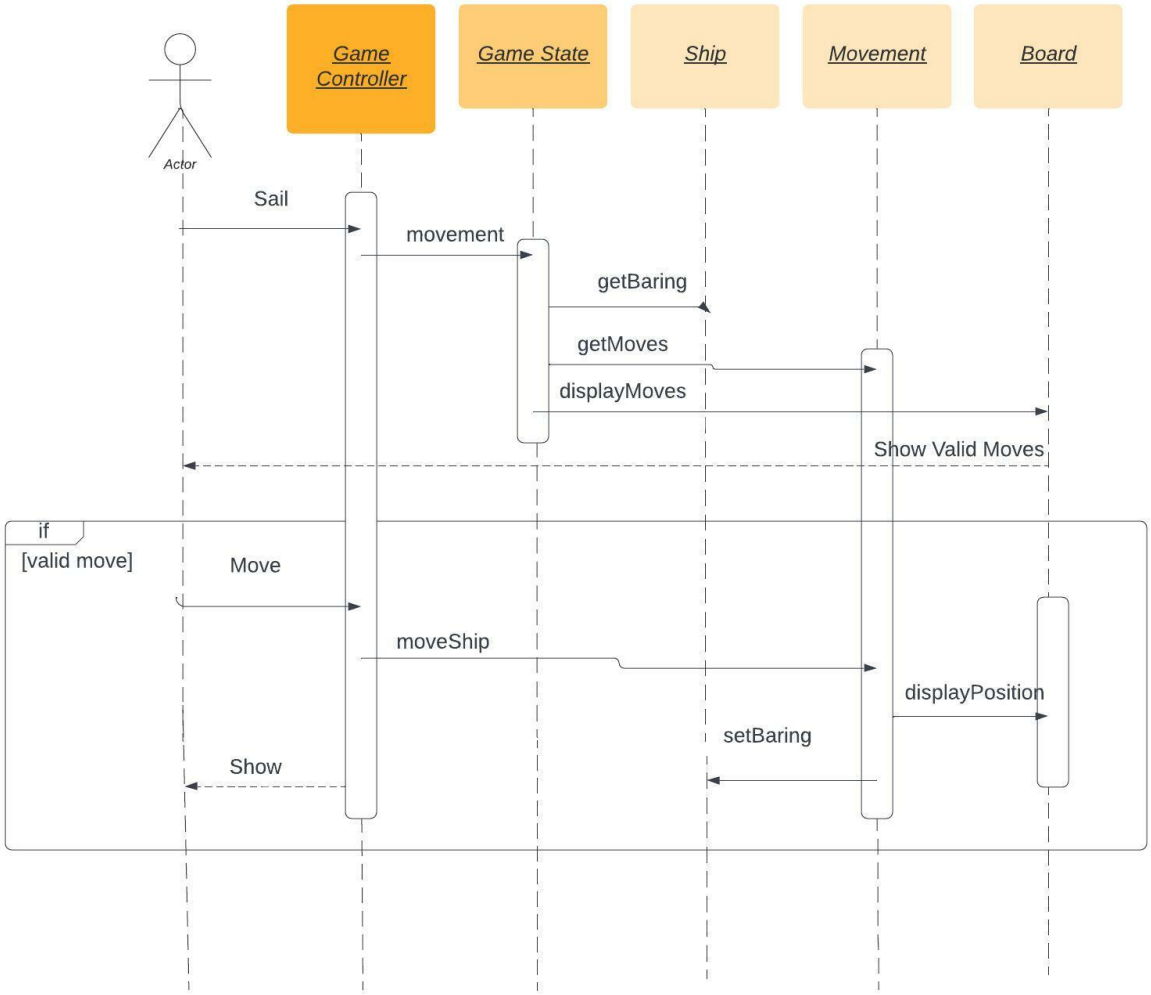
1) Start a new game



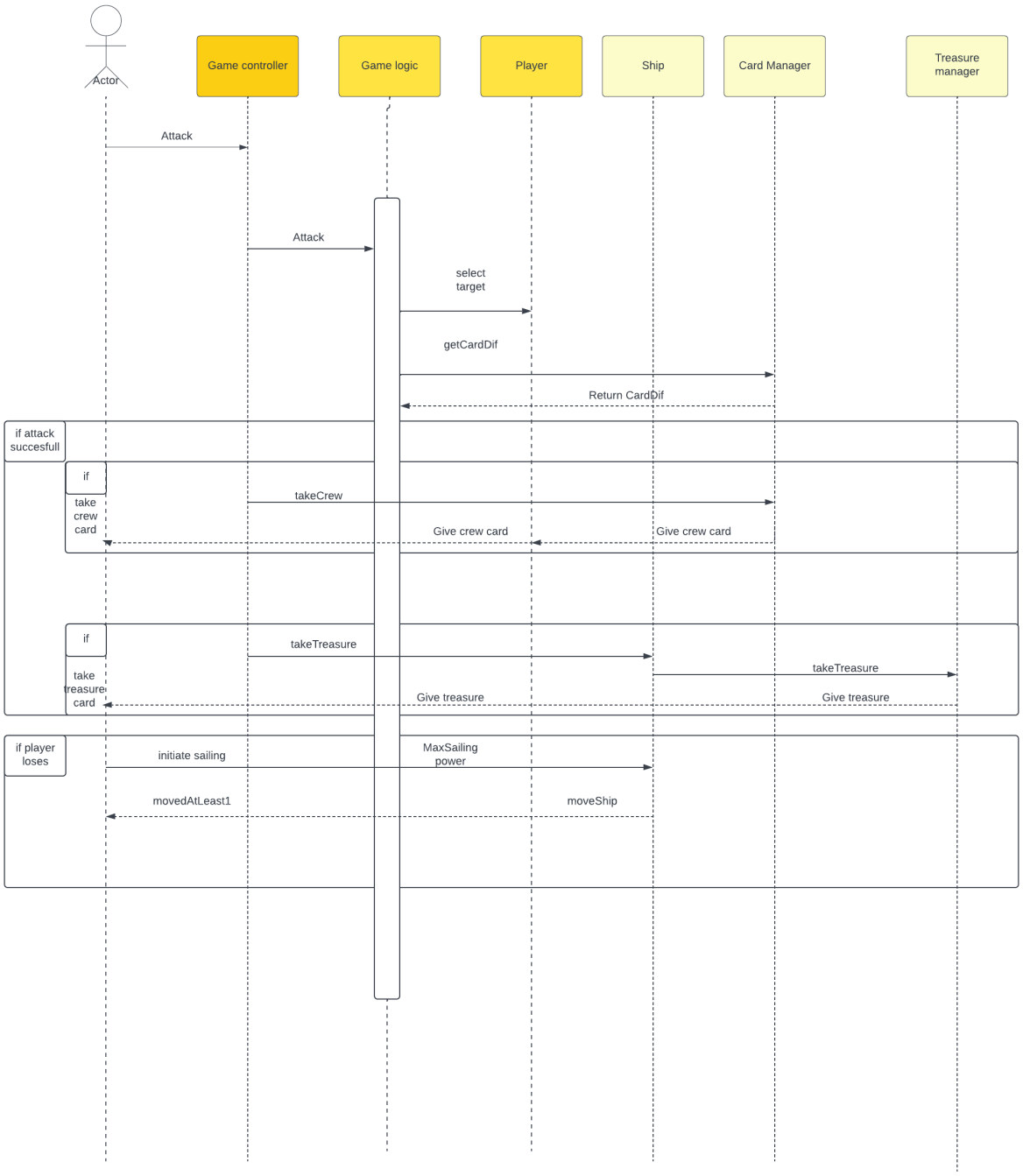
2) Continue a saved game



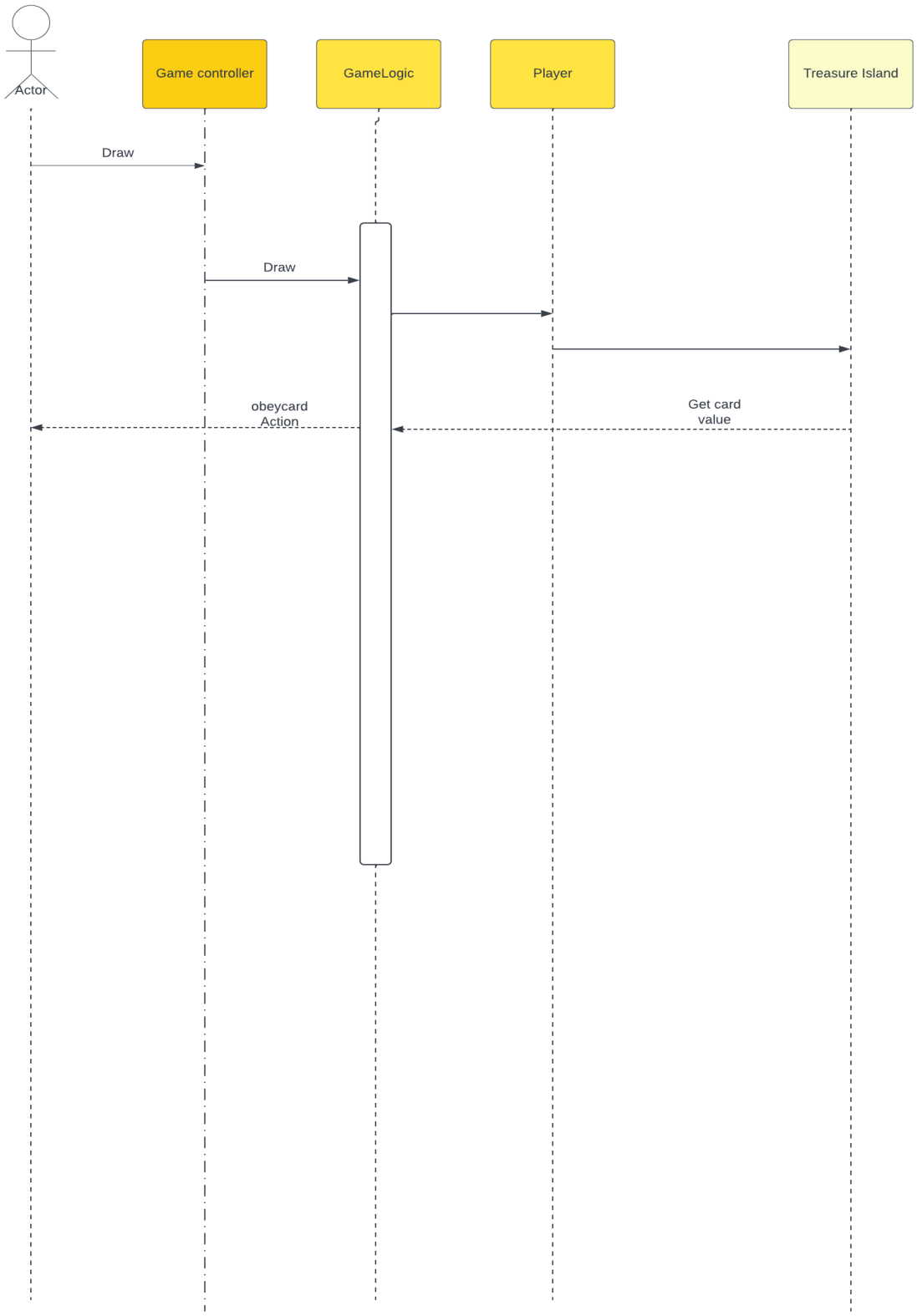
3) Sail


Actor
Game Controller
Game State
Ship
Movement
Board
Sail
movement
getBaring
getMoves
displayMoves
Show Valid Moves
if
[valid move]
Move
moveShip
displayPosition
setBaring
Show

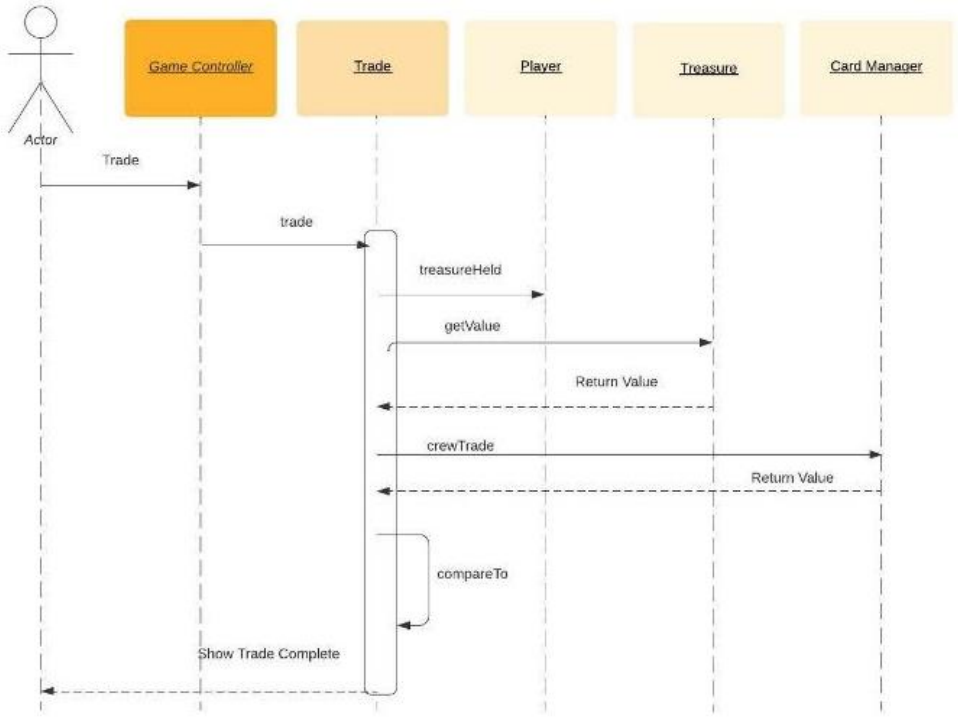
4) Attack


Actor
Game controller
Game logic
Player
Ship
Card Manager
Treasure manager
Attack
Attack
select target
getCardDif
Return CardDif
if attack succesfull
if
take crew card
takeCrew
Give crew card
Give crew card
if
take treasure card
takeTreasure
takeTreasure
Give treasure
Give treasure
if player loses
initiate sailing
MaxSailing power
movedAtLeast1
moveShip

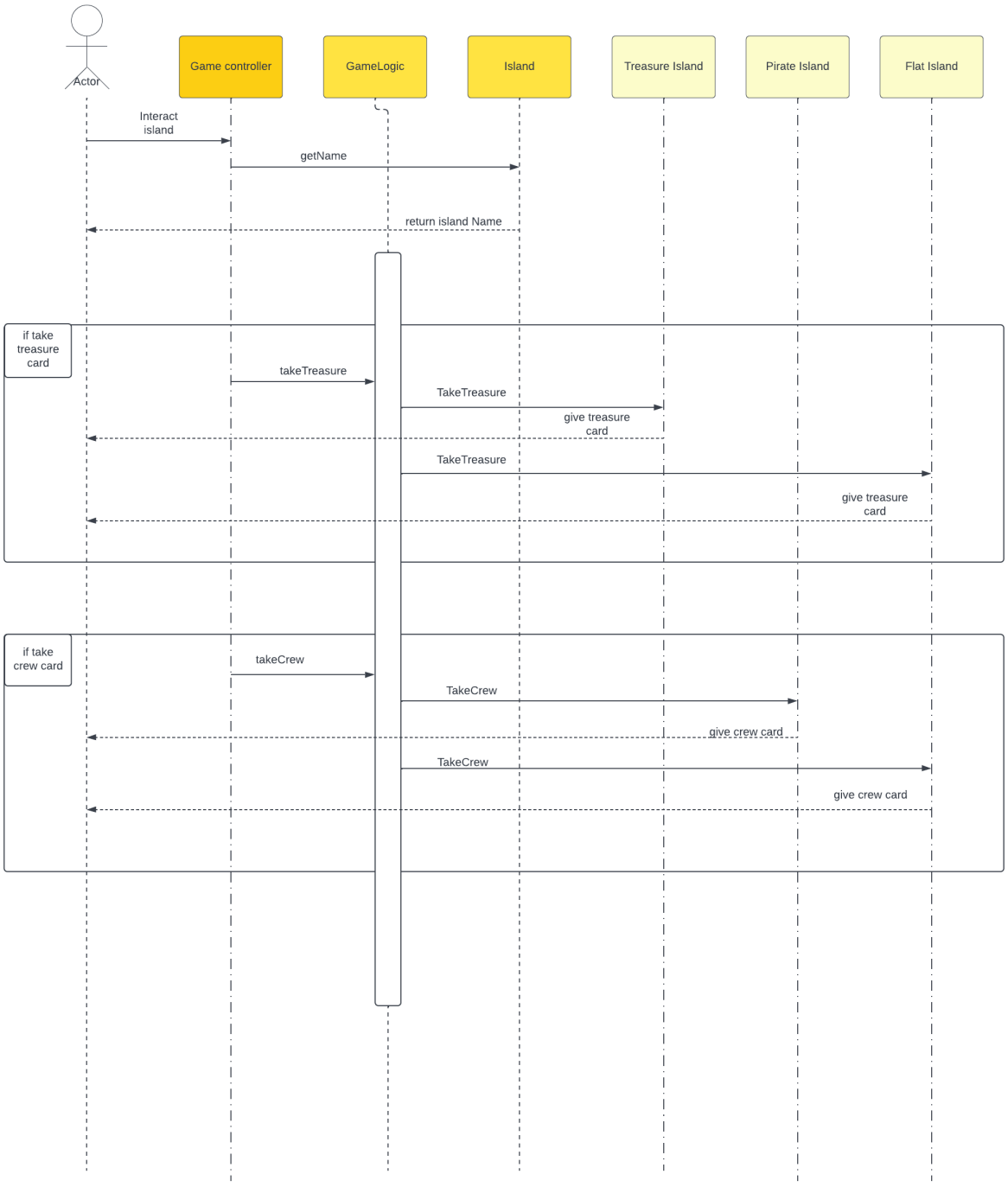
5) Draw chance card



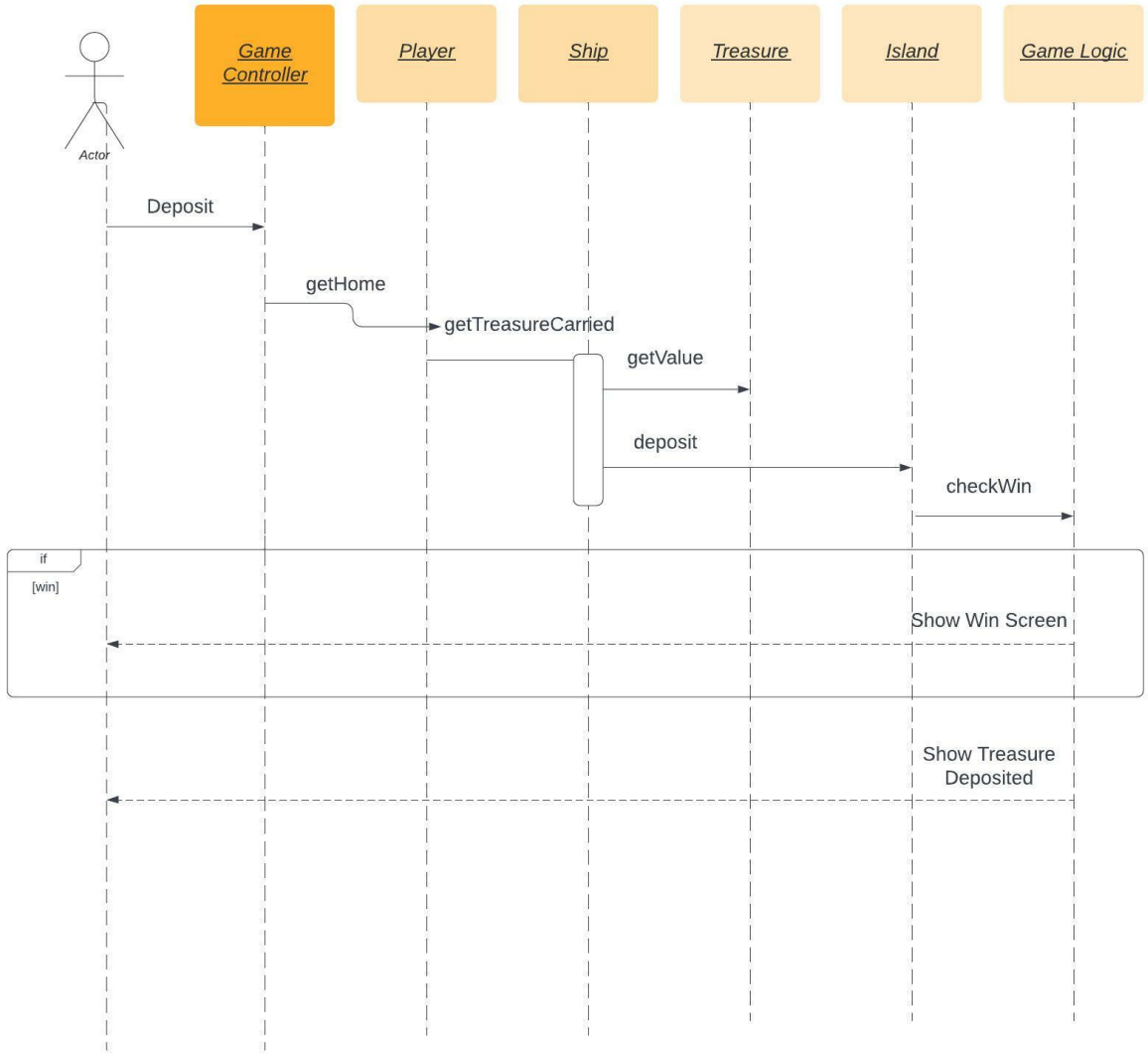
6) Trade


Actor
Game Controller
Trade
Player
Treasure
Card Manager
Trade
trade
treasureHeld
getValue
Return Value
crewTrade
Return Value
compareTo
Show Trade Complete

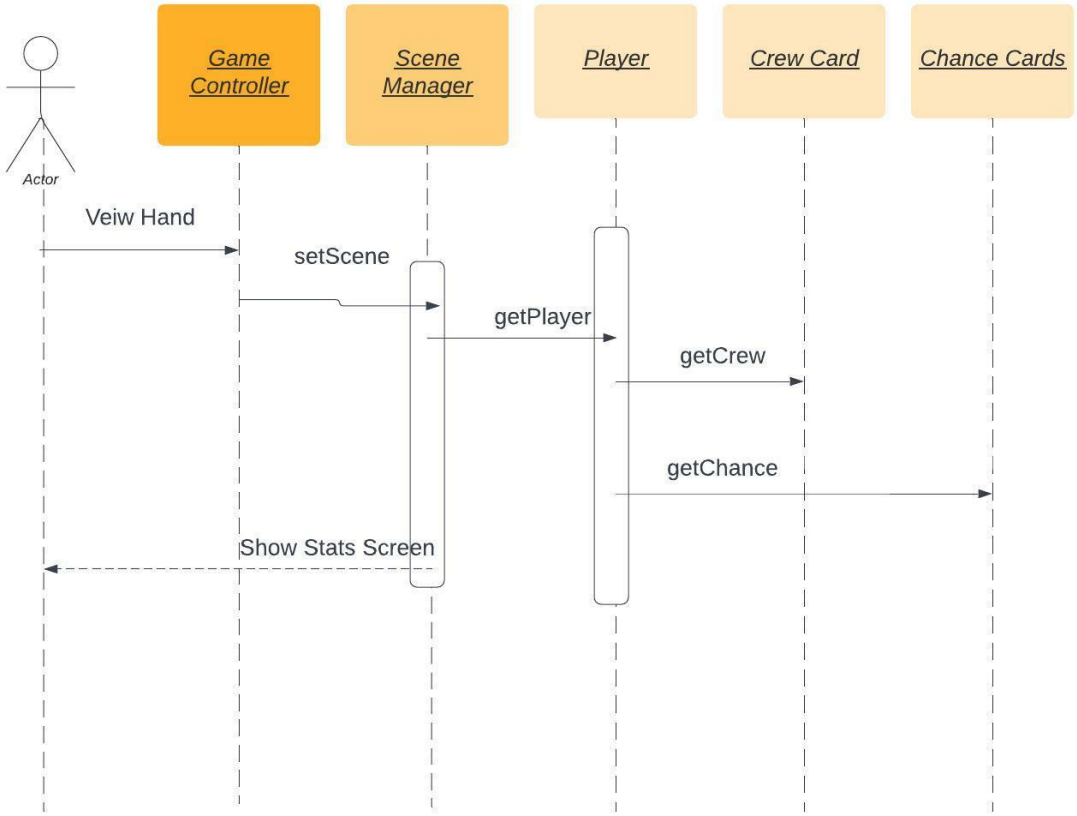
7) Interact with the Islands



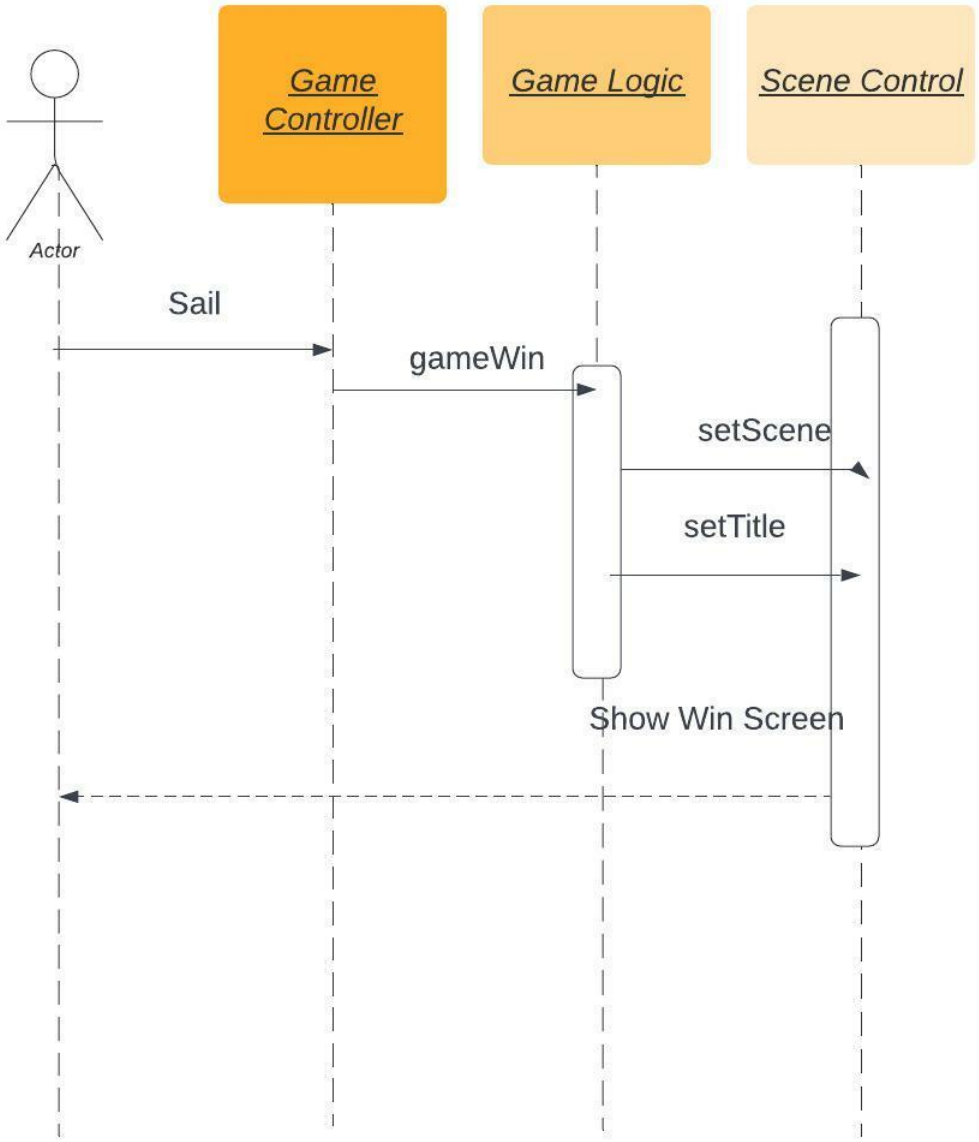
8) Deposit treasure to home port



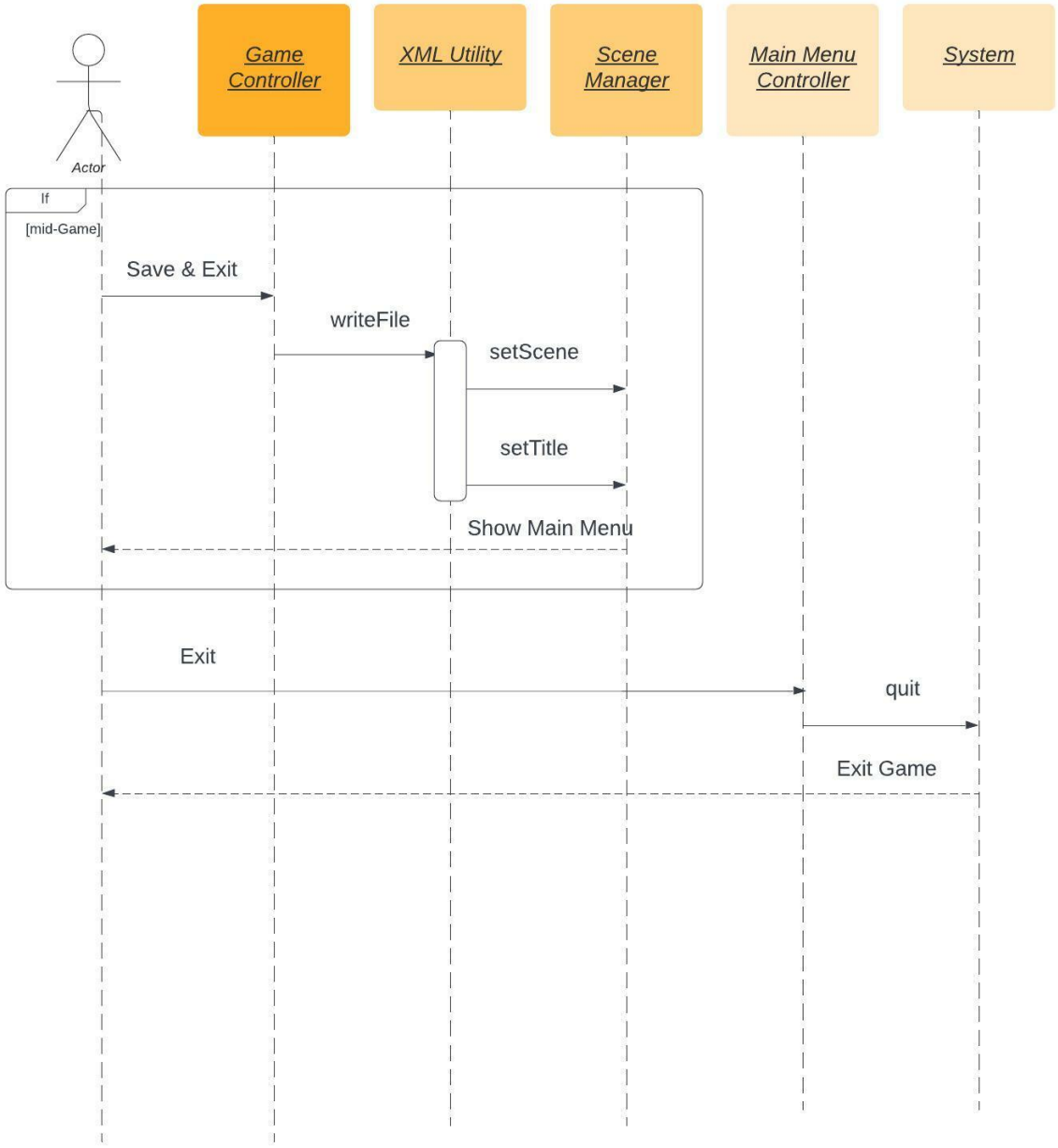
9) View chance cards in player's hand



10) Winning the game



11) Exit the game

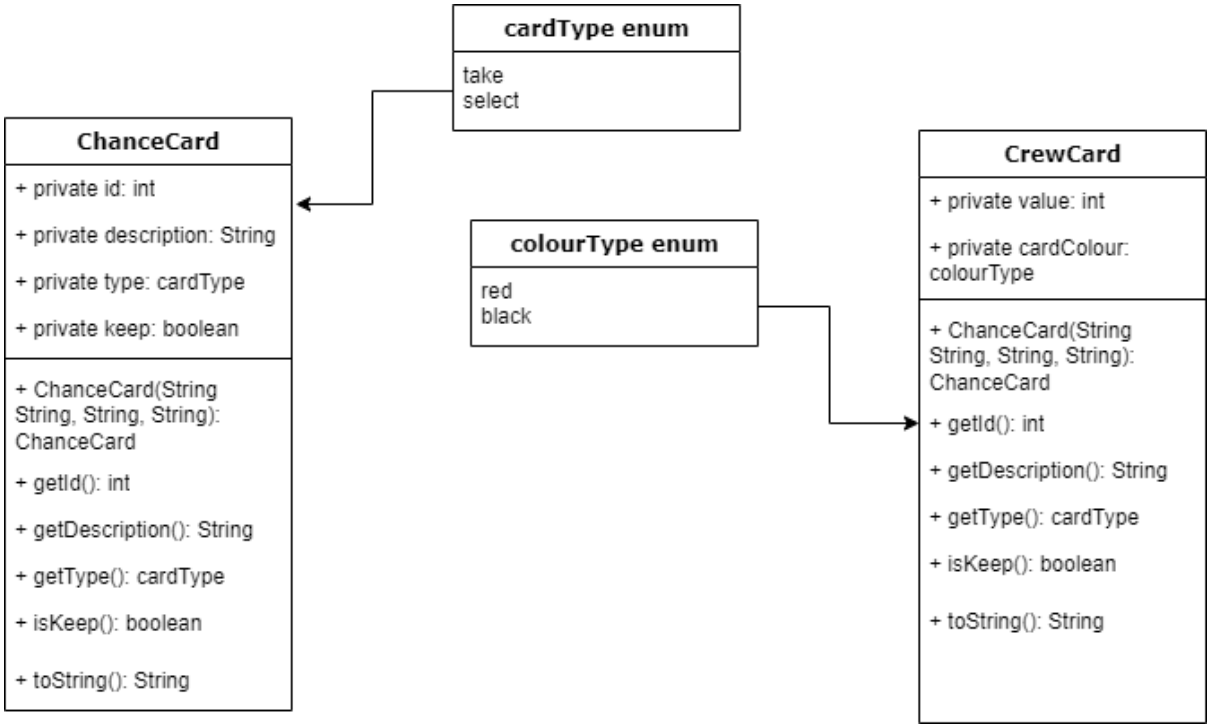


5.2 Significant algorithms

The ship orientation when clicking would be determined by comparing the difference in both the X and Y coordinates of the ship to the X and Y coordinates of the click. In addition, because the click may happen at a diagonal point to the ship, the differences would also have to be compared to each other. If the absolute difference in X is greater than the absolute difference in Y, the ship would face horizontally; otherwise, it would face vertically. From there, the sign of the difference may be used to determine exactly which way to face. If the difference is -X, face left, +X, face right, +Y, face down, -Y, face up.

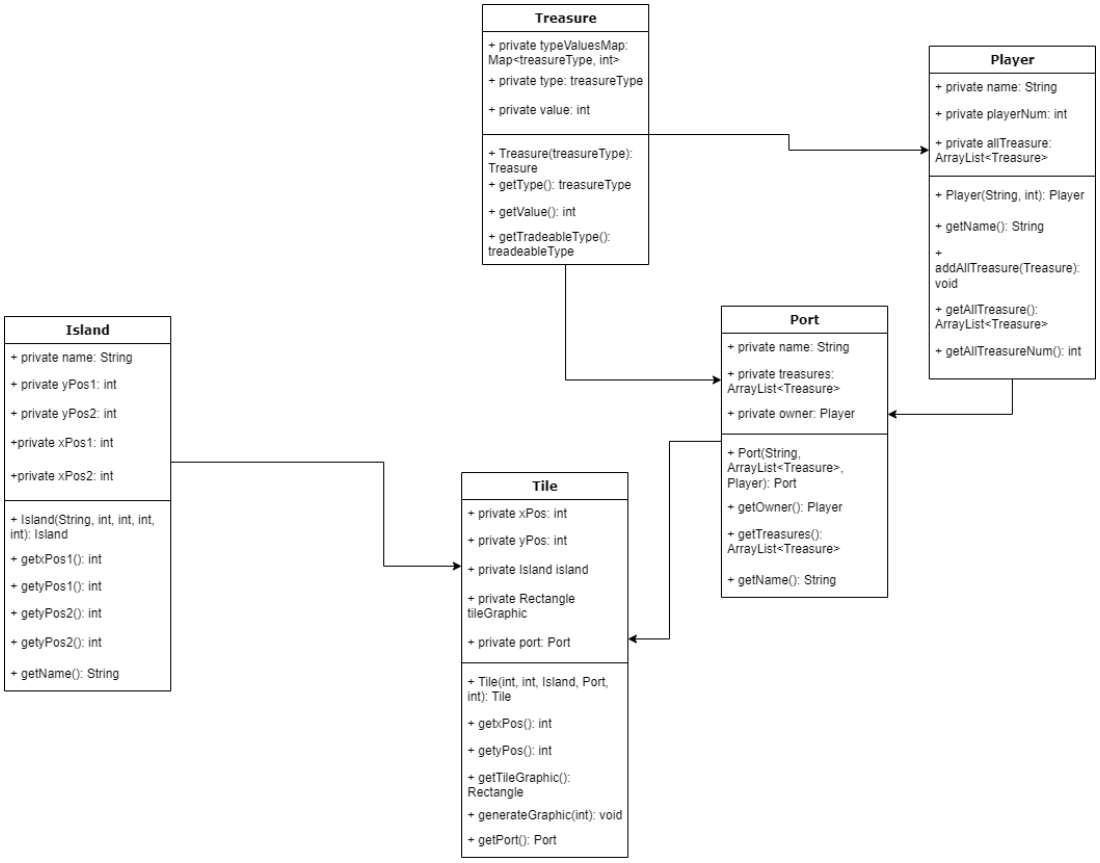
5.3 Significant data structures

Chance cards on the Treasure Island and Crew cards on Flat Island and Pirate Island will be implemented with a Queue.



Treasure on all three Islands will be stored in an Array List.

The Tile objects will be stored in the board as a 2-Dimensional Array.



REFERENCES

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

[2] QA Document SE.QA.02 – General Documentation Standards

DOCUMENT HISTORY

Version	Issue No.	Date	Changes made to document	Changed by
0.0	N/A	08/03/22	Started new document	std36
0.1	N/A	10/03/22	Added titles and table	tob31 std36
0.2	N/A	17/03/22	Fixed formatting issues with table of contents	std36
0.3	N/A	21/03/22	Completed section 2, began work on sections 4 & 5	std36, cjh26, lup35, ols21
0.4	N/A	21/03/22	Amended document formatting to fit document standards	ols21
0.5	N/A	24/03/22	Document formatting fix	ols21
0.6	N/A	25/03/22	Sequence diagrams	jaf43
0.7	N/A	25/03/22	A Component diagram & a interface desc	tob31
0.8	N/A	30/03/22	Merged separate copies of document	nar29
0.9	N/A	30/03/22	Formal review changes	cjh26, ols21, tob31, nar29, lup35, std36
0.10	N/A	1/04/22	Diagram updates	jaf43, mnc6
1.0	N/A	1/04/22	Released	jaf43
1.1	N/A	11/05/22	Added Class diagram	std36, tob31
1.2	N/A	11/05/22	Added component diagram	mnc6
1.3	N/A	11/05/22	Updated section 2 to accurately reflect the current program	ols21