Group Project 11
Design Specification

Author:	Bhagya Wanni Arachchige [bhw], Maciek Traczyk [mat78], Ash Bagnall [asb20], Dean Plank [dep22], James Green [jag77]
Config Ref:	DesignSpecGroup11
Date:	29th March 2022
Version:	1.0
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 clear definition of what the outcomes of the game should be [1].

1.2 Scope

Describes the important features in the design of the program

1.3 Objectives

The objective of this document is to [1]:

- Identify the significant programs in the system
- Identify what classes are relevant to each requirement
- Specify the relationship and dependencies between modules
- Outline a specification of each class of the program

2. DECOMPOSITION DESCRIPTION

2.1 Programs in the system

The system is only composed of with a singular part:

- Desktop Application

2.1.1 Desktop Application

The desktop application is where the board game can be played through a JVM process. Users can select a player with a username and boat of their choosing. Users can interact with the Buccaneer game board's functions using this boat, for instance, they are allowed to move the boat to various Islands and ports located on the game board. Users can finish the game by collecting 20 points of treasure and successfully bringing it back to their home port.

2.1.2 Significant classes

Desktop application

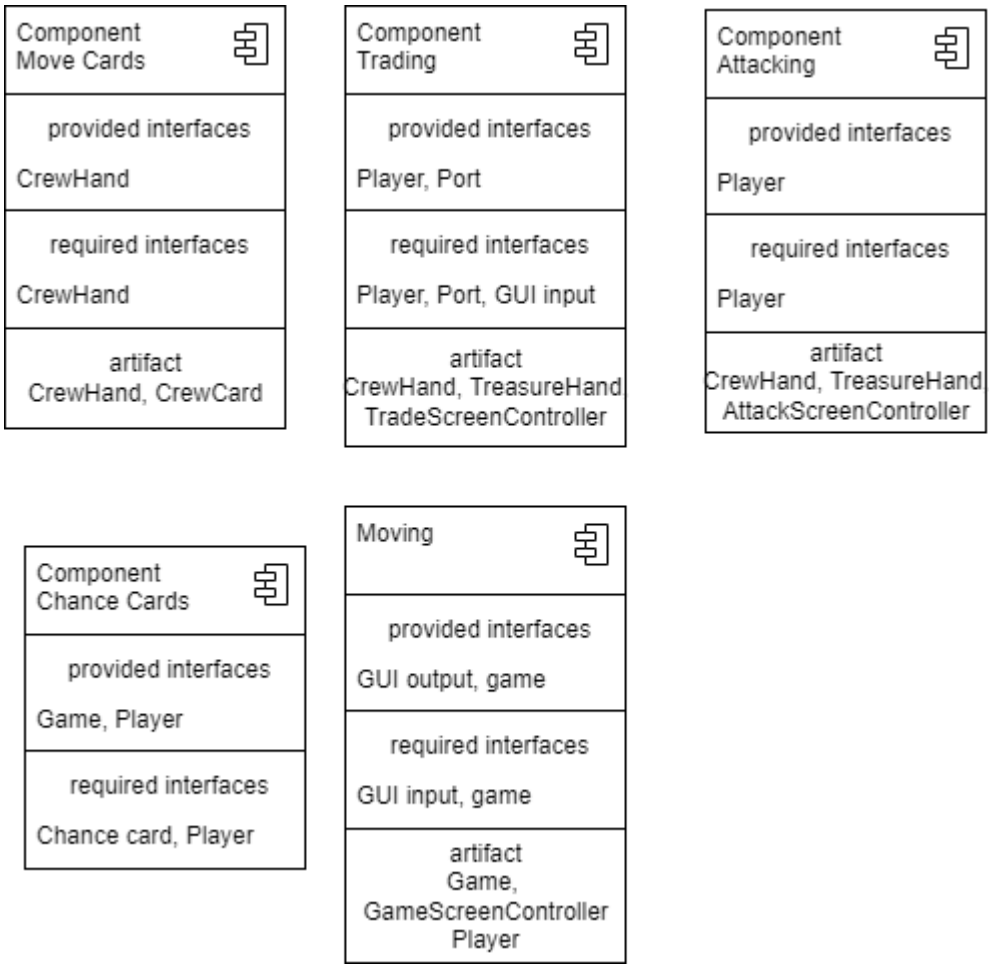
- Game: The class acts as the main body of the game and contains the game board and player objects.
- App: Oversees the shifting between screens/ different stages of the game and the initialisation of these FXML related objects
- Player: The player class contains all details related to the player ship including their name, number, location, and the direction they are facing. This class also has the methods that allow for player movement, attacking etc.
- CrewHand: The CrewHand class is used to handle the transactions between to entities' crewcards, simultaneously storing the crew cards and being able to move then, delete them and add new cards.
- TreasureHand: The TreasureHand class works like the CrewHand class but with the Treasure objects.
- GameScreenController: The GameScreenController class acts a medium between the Game object and the GameScreen FXML that is displayed, communicating changes from one way to the other (Game->GUI & GUI->Game)
- AttackScreenController: The AttackScreenController class deals with a players attack against another player, calculating the winner and displaying it on screen, then dealing with the loser's movement.

- TradeScreenController: The TradeScreenController deals with a player trading with a port, making sure that the player is giving up treasure or crewcard(s) with at least a value equal to the crewcard(s) or treasure (treasure traded for crewcards, crewcards for treasure) they desire from the port.

2.2 Table mapping requirement onto classes

Function Requirement	Classes that meet the requirement
FR1	CharacterScreenController
FR2	Game
FR3	CrewPack, CrewCard, CrewPack
FR4	ChanceCard, ChancePack
FR5	TreasureHand, Treasure
FR6	Game, Player
FR7	PortTile, Game
FR8	FlatIsland
FR9	GameScreenController
FR10	Game
FR11	NextPlayerScreen, Game, GameScreenController
FR12	AttackScreenController, GameScreenController, Game
FR13	IslandTile, TreasureIsland
FR14	IslandTile, FlatIsland
FR15	PortTile, Game, Port
FR16	N/A
FR17	Game

3. DEPENDENCY DESCRIPTION



4. INTERFACE DESCRIPTION

```
The App class contains the main methods that initially launch the program. It
contains the methods that enable
FXML to work.

public class App extends Application {
    private static Scene startScreen;
    private static Scene characterScreen;
    private static Scene gameScreen;
    private static Scene attackScreen;
    private static Scene tradeScreen;
    private static Stage stage;

    // Starts the main game by taking a 'stage' that is a screen
    public void start(Stage stage) throws IOException() { ; }

    // Loads all the images for the game at the start
    private void loadImages() { ; }
```

```
    // The main method
    public static void main(String[] args) { ; }
}


This class contains the methods that handle the actual screen that shows which
player won
in combat.

public class AttackScreenController {
    private Player playerOne; // Player one in the attacking phase
    private Player playerTwo; // Player two in the attacking phase
    private TreasureIsland treasureIsland; // Contains the TreasureIsland
object
    public Player winnerPlayer; // Contains the winner
    public Player loserPlayer; // Contains the loser
    private Game bucGame; // Contains the main game object

    // Starts up the attack
    public void attackStartup() { ; }

    // The main attack method, calculates who won and calls another method
shortly after
    public void attack() throws InterruptedException { ; }

    // Called if players draw
    public void outcomeDraw() { ; }

    // Called if the player who started the attack wins
    public void dealWithMovement() { ; }

    // Called if the player who started the attack doesn't win
    public void dealWithMovementLost() { ; }
}


This class handles the screen at the start, allowing players to change
their names and ship colours.

public class CharacterScreenController {
    ArrayList<Player> players; // A List of the players
    String[] shipColoursReserved;
    String[] shipColoursUnreserved;
    int[][] coords;

    public void initialize() { ; }

    // Sets data to the character screen
```

```
    public void setData() { ; }

    // Updates images on the character screen
    public void updateImage(int num) { ; }

    // Switches to the main game screen (board)
    private void switchToGame() throws IOException { ; }
}


This is the main controller that handles all events on the actual
board of the game.

public class GameScreenController {
    Game bucGame; // The main game object
    private int selectedRow, selectedCol;
    public List<int[]> oldPath = null;
    private ImageView[][] imageGrid = new ImageView[20][20]; // The Tile setup
for the board

    // Sets up the main screen (board)
    public void initialize() { ; }

    // Stars a new game by creating the players etc
    public void newGame(ArrayList<Player> players) { ; }

    // Updates all the visual aspects of the board at once (player positions
etc)
    public void updateVisuals() { ; }

    // Updates just the board visuals
    private void updateBoardVisuals() { ; }

    // Called at the end of a turn
    private void endTurn() throws IOException, InterruptedException { ; }

    // Updates the treasure that is displayed on the board (what the player
has on the ship)
    private void updateVisualTreasureHand() { ; }

    // Updates the player's position they're facing on the board
    public void updatePlayerDirection(Player p) { ; }

    // Rotates the player
    private void rotatePlayerMaster(String direction) { ; }

    private void rotatePlayerNorth() { ; }

    private void rotatePlayerNorthEast() { ; }
```

```
    private void rotatePlayerEast() { ; }

    private void rotatePlayerSouthEast() { ; }

    private void rotatePlayerSouth() { ; }

    private void rotatePlayerSouthWest() { ; }

    private void rotatePlayerWest() { ; }

    private void rotatePlayerNorthWest() { ; }

    // Contains the UI code needed for the path finding algorithm
    public void clickGrid(javafx.scene.input.MouseEvent event) {

    // Returns a list of the coordinates in a path tfrom a point to the player
    public List<int[]> getPathToPointFromCurrentPlayer(int x, int y)

    // Creates the panes in the board itself (used for highlighting squares)
    public void createPanes() { ; }

    // Clears all highlighted cells
    public void clearHighlightedCells() { ; }

    // Clears all highlighted cells but a list of coordinates
    public void clearCellsBut(List<int[]> coordinates) { ; }

    // Clears an individual cell
    public void clearCell(int x, int y) { ; }

    // Highlights an individual cell
    public void highlightCell(int x, int y) { ; }

    // Highlights an individual cell green
    public void highlightCellGreen(int x, int y) { ; }

    // Unhighlights many cells at once
    public void unhighlightMultipleCells(List<int[]> coordinates) { ; }

    // Allows player to view their own crew cards
    public void viewOwnCrewCards() { ; }

    // Controls all the movement a player does (moves them)
    private void movePlayer() { ; }

    private void switchToStart() throws IOException { ; }
}
```

```
This handles the screen that shows that it's a new turn.

public class NextPlayerScreenController {

    private void switchToGame() { ; }
}


public class StartScreenController {

    private void newGame() throws IOException { ; }

    public void loadGame() throws IOException { ; }
}


This class handles controlling the screen that allows players to
trade with ports.

public class TradeScreenController {
    Port port; // Contains a port to trade at
    Player player; // Contains the player trading
    ArrayList<CheckBox> playerTreasureCheckboxes = new ArrayList<>();
    ArrayList<CheckBox> playerCardCheckboxes = new ArrayList<>();
    ArrayList<CheckBox> portTreasureCheckboxes = new ArrayList<>();
    ArrayList<CheckBox> portCardCheckboxes = new ArrayList<>();

    // Begins a trade with the port and the player
    public void tradeStartup(Player playerIn, Port portIn) { ; }

    // Starts the trade sequence (dynamically creates the treasure and cards
etc on the popup)
    public void beginTradeSequence() { ; }

    // Called when the player wants to trade treasure for cards
    private void tradeTreasureForCards() { ; }

    // Called when the player wants to trade cards for treasure
    private void tradeCardsForTreasure() { ; }
}


This class contains the information specific to a certain chance card.

public class ChanceCard {
    private int num; // The card number as in the specification
    private String desc; // Description of the card
```

```
    // Returns the card number
    public int getNumber() { ; }

    // Returns the card description
    public String getDescription() { ; }

    // A switch statement including all references to methods that correlate
with the card number
    public void useChanceCard(Game game) { ; }

    // A sub-class that contains all the methods for each chance card
    public static class ChanceActions {
        private static double calcDistanceToPoint() { ; }
    }
}


This holds multiple ChanceCards and treats them as if this was a deck of
cards.

public class ChancePack {
    private ArrayList<ChanceCard> cards; // A list that has all the chance
cards in

    // Executes a chance card from the top of the pack
    public ChanceCard getChanceCard() { ; }

    // Creates the pack from their given descriptions and places them into an
ArrayList 'cards'
    private void createPack() { ; }
}


The CrewCard class holds all the data necessary regarding a single Crew card
in the game. It holds both a value
and a color.

public class CrewCard implements Displayable {
    private int value; // Value of the crew card
    private String colour; // The colour of the crew card

    // Creates a crew card
    public CrewCard(int val, String col) { ; }

    // Gets the value of the crew card
    public int getValue() { ; }

    // Gets the colour of the card
```

```
    public String getColour() { ; }
}



The CrewHand class holds an array of object CrewHand that acts as the player's
current hand of crew hands
within the game. Contains functionality to calculate the necessary values for
combat and movement.

public class CrewHand {
    private ArrayList<CrewCard> cards; // A list of crew cards in the hand

    public CrewHand() { ; }

    // Adds a crew card to the hand
    public void addCard(CrewCard card) { ; }

    // Moves a crew card from one hand to another
    public void moveFromHandToHand(CrewHand hnd, CrewCard card) { ; }

    // Gives a crew card from the top
    public boolean giveCardFromTop(CrewHand hnd) { ; }

    // Gives a crewcard from a given index in the list
    public boolean giveCardFromIndex(CrewHand hnd, int index) { ; }

    // Gets total cards
    public int getTotalCards() { ; }

    // Gets the combat value, returns positive difference between red and
black cards
    public int getCombatValue() { ; }

    // Gets the total of black cards
    public int getBlackValue() { ; }

    // Gets the total of red cards
    public int getRedValue() { ; }

    // Gets the total value of all cards
    public int getMoveAbility() { ; }

    // Returns the lowest value crew card
    public CrewCard lowestValue() { ; }

    // Returns the highest value crew card
    public CrewCard highestValue() { ; }
```

```
    // Gets all the cards (list)
    public ArrayList<CrewCard> getCards() { ; }
}


The CrewPack class holds and keeps track of all CrewCards in the game and
handles handing out cards to
players and keeping them all in one place within the Game object. Acts as a
card pack, hands out cards on the
top of the stack.


public class CrewPack {
    public ArrayList<CrewCard> cards;

    public CrewPack() { ; }

    // Adds a card to a player
    public void addCardToPlayer(Player ply) { ; }

    // Adds a card to a hand
    public void addCardToHand(CrewHand hand) { ; }

    // Adds a card to this hand
    public void addCard(CrewCard card) { ; }

    // Gets a card from the hand
    public CrewCard getCard(int index) { ; }

    // Gets all cards
    public ArrayList<CrewCard> getCards() { ; }
}


The FlatIsland class holds all functionality required for this island.

public class FlatIsland {
    public CrewHand crewHand;
    public TreasureHand treasureHand;

    public FlatIsland() { ; }

    // Gives loot to a player from the island
    public void giveLoot(Player p) { ; }
}


The PirateIsland class holds all functionality required for this island.
```

```
public class PirateIsland {
    public CrewHand crewHand;

    public PirateIsland() { ; }

    // Deals a card from the top of the hand in pirate island to a player
    public void dealFromTop(CrewHand hnd, int numCards) { ; }
}


The TreasureIsland class holds all functionality required for this island.

public class TreasureIsland {
    private TreasureHand treasures;
    private ChancePack chanceCards;

    public TreasureIsland() { ; }

    // Returns a chance card from the island
    public ChanceCard getChanceCard() { ; }

    // Gets the treasure hand of the island
    public TreasureHand getIslandTreasureHand() { ; }

    // Gets the total treasure the island has
    public int getNumberOfTreasures() { ; }
}


The Player class contains all information and methods to manipulate a player's
data.
This varies from setting their name, to actually moving them on the board.

public class Player {
    public static final String[] DIRECTIONS =
{"N","NE","E","SE","S","SW","W","NW"};
    private HashMap<String, int[]> directionalMovement;
    private int playerNumber; // Player number
    private String playerName;
    private String shipImageName;
    private int col; // Current column index
    private int row; // Current row index
    private String direction; // Direction player is facing
    public CrewHand crewHand = new CrewHand();
    public TreasureHand treasureHand = new TreasureHand();
    public boolean canMoveInAnyDirection = false;
    public String playerHomePort;

    public Player() { ; }
```

```
    public Player(String playerName,int playerNumber) { ; }

    // Gets move total
    public int getMoves() { ; }

    // Checks if player can move to a coordinate
    public boolean canMoveTo(int col, int row, Tile[][] gameBoard)  { ; }

    // Moves player to a coordinate
    public boolean moveTo(int desCol, int desRow, Tile[][] gameBoard) { ; }

    // Moves the player forward X times
    public boolean moveForward(int spaces, Tile[][] gameBoard) { ; }

    // Returns if player can move in a straight line to a position
    public boolean canMoveInStraightLine(int desCol, int desRow, Tile[][]
gameBoard) { ; }

    public boolean canMoveInStraightLine(int desCol, int desRow, Tile[][]
gameBoard, boolean limitedByMovement) { ; }

    // Returns the closest player to this player
    public Player getClosestPlayer(ArrayList<Player> players)   { ; }

    // Allows player to move in all directions after loosing attack
    public void setAllowMoveInAnyDirection(boolean a)  { ; }

    // Returns if a player can move in any direction
    public boolean canMoveInAnyDirection() { ; }

    // Rotates player
    public void rotate(String turnDir) { ; }

    // Sets the player's number
    public void setPlayerNumber(int num) { ; }

    // Returns player's direction
    public String getDirection() { ; }

    // Sets a players direction
    public void setDirection(String dir) { ; }

    // Sets a players coordinate
    public void setCoordinate(int col, int row) { ; }

    // Sets the player's home port
    public void setHomePort(String homePortName) { ; }
```

```
    // Checks if a coordinate is inline with a player (valid move)
    public boolean inlineWithPlayer(int toCol, int toRow) { ; }

    // Checks if a coordinate is in moving distance of a player's turn
    public boolean withinMovingDistance(int toCol, int toRow) { ; }

    // Checks if a path up to a coordinate is free from the player
    public boolean pathUpToTileFree(int toCol, int toRow, Tile[][] gameBoard)
{ ; }

    // Gets column coordinate
    public int getCol() { ; }

    // Gets row coordintae
    public int getRow() { ; }

    public void setColCoordinate(int col) { ; }

    public void setRowCoordinate(int row) { ; }

    // Sets the player's icon
    public void setIconName(String shipImageName) { ; }

    public String getIconName() { ; }

    public int getPlayerNumber() { ; }

    public void setPlayerName(String name) { ; }

    public String getPlayerName() { ; }

    public String getHomePort() { ; }
}



The Popups class handles calling and creating all popups that are used in the
game.

public class Popups {
    private int playerNum; // Player's number (used to get their data)
    private int choice; // Choice of a multiple choice popup
    private String choice1; // Choice of a multiple choice popup (string
version)

    // Popup allowing player to take treasure or cards
    public int chooseTreasureOrCards(String title, int treasureVal, int
cardVal,int targetTVal, Game game) { ; }
```

```
    // Popup displaying treasure at a port
    public void displayTreasure(String title,Game game) { ; }

    // Popup displaying crew cards for a player
    public void displayCrewCard(String title,Game game) { ; }

    // Popup for chance cards, allowing a player to pick another player to use
a card on
    public int PickPlayer(String title, String message, ArrayList<Player>
players) { ; }

    // Asks if a player wants to attack another player
    public String askToAttackPlayer(String title, String message, Game game) {
; }

    // Displays a message
    public void displayMessage(String title, String message) { ; }

    // Asks if a player wants to complete an action
    public int yesOrNo(String title, String message) { ; }
}


The Port class holds all cards and information that is required by the
specification.
It also contains methods to allow a player to trade with a port.

public class Port {
    private String portName;
    private int col;
    private int row;
    private CrewHand crewHand;

    public Port(String name, int col, int row) { ; }

    public void tradeCardsForTreasure(Player player,int totalCrewCards, int
totalTreasure, int[] tradeTreasure, int[] tradeCards)

    // Puts all treasure from the player into the port
    public void putAllTreasure(Player ply) { ; }

    // Checks if trade is valid and trades treasure and cards with a player
    public void tradeTreasureForCards(Player player,int totalCrewCards, int
totalTreasure, int[] tradeTreasure, int[] tradeCards)

    // Returns if it's a player's home port
    public boolean isHomePort() { ; }
```

```
    public int getCol() { ; }

    public int getRow() { ; }

    public CrewHand getPortCrewHand() { ; }

    public String getPortName() { ; }

    public TreasureHand getPortTreasureHand() { ; }
}
```

The HomePort class extends Port and contains further methods to help player's store
cards in their safezone.

```
public class HomePort extends Port{
    private Integer playerNumber;
    private TreasureHand safeZone = new TreasureHand();

    public HomePort(String name, int x, int y, int playerNum)

    public TreasureHand getSafeZoneHand()

    public void addToPlayerHand(Player player)

    public void addToSafeZone()

    public Integer getPlayerNumber()

    @Override
    public boolean isHomePort() {
        return true;
    }
}
```

The Displayable class contains information regarding an image in the game.

```
public interface Displayable {

    // Returns the icon name of an image (displayable)
    public String getIconName();
}
```

The Tile class contains the information about a specific tile on the board (20x20 board)

```
public interface Tile extends Displayable {
    public void setIconName(String icon);

    public String getTileName();

    public String getIconName();

    public boolean isAttackAble();

    public boolean isTraversable();

    public boolean isIsland();
}


The Treasure class contains information about a specific piece of treasure.

public class Treasure implements Displayable {
    private String name;
    private int value;

    public Treasure(String name, int value)

    // Gets the name of a treasure item
    public String getName() { ; }

    // Returns the value of a treasure object
    public int getValue() { ; }

    @Override
    public boolean equals(Object o) { ; }

    @Override
    public String getIconName() { ; }
}



The TreasureHand class contains a list of multiple treasure items and contains
the
functionality that allows treasure to be traded amongst players and ports.

public class TreasureHand {
    private ArrayList<Treasure> treasures;
    private boolean playerHand;

    public TreasureHand() { ; }

    // Adds treasure to the hand
```

```
    public boolean addTreasure(Treasure treasure) { ; }

    // Gives treasure from the top of one hand to this hand
    public boolean giveTreasureFromTopOfHand(TreasureHand hnd) { ; }

    // Gives a treasure from a given index in the list
    public boolean giveTreasureFromIndex(TreasureHand hnd, int index) { ; }

    // Gets the total value of all treasures in the hand
    public int getTotValOfTreasure() { ; }

    // Returns the lowest value treasure
    public Treasure lowestValue() { ; }

    // Returns the treasure by name
    public int getTreasureIndexByName(String name) { ; }

    // Moves treasure from one hand to another
    public void moveFromHandToHand(TreasureHand hnd, Treasure obj) { ; }

    // Gets a treasure index by it's value
    public ArrayList<Treasure> getTreasureIndexByValue(int tValue) { ; }

    public int getTotalTreasure() { ; }

    // Returns the highest value treasure
    public Treasure highestValue() { ; }

    public ArrayList<Treasure> getTreasures() { ; }
}



The Game class contains all the main methods required for the game to run.
It contains variables such as the 20x20 board and the players in the game.

public class Game {
    private ArrayList<Player> players; // All the players in the game
    public  Tile[][] gameBoard; // The 20x20 board
    public HashMap<String,Port> ports; // All the game ports
    public  static  final String[] turnOrderByPortName =
{"London","Genoa","Marseilles","Cadiz"}; // Ports by name
    public boolean needReplace = false;

    public Game(ArrayList<Player> players) { ; }

    // Returns all ports
    public List<Port> getPorts() { ; }
```

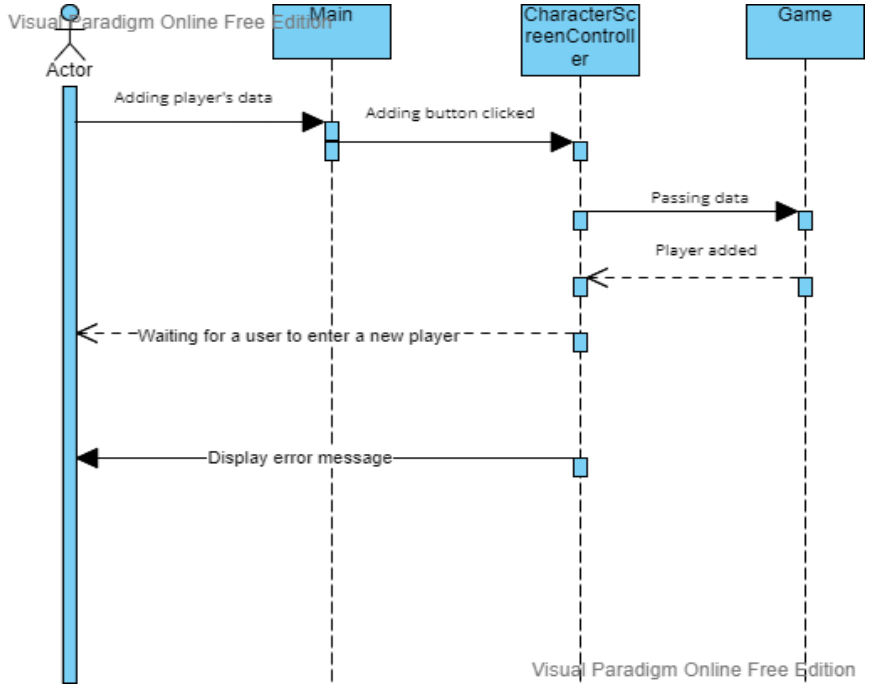
```
    // Gets the players in the game
    public ArrayList<Player> getPlayers() { ; }

    // Starts the game by setting up the board etc
    public void startGame() { ; }

    // Detects if the game has ended (20 value treasure)
    public Player detectEndState() { ; }

    // Checks if an island is around a given point
    public Object checkIfIslandAround(int x, int y) { ; }

    // Distributes treasure to ports etc
    public void distributeTreasure() { ; }

    // Distributes cards to the players and pirate island
    public void cardDistribution() { ; }

    // Setups all the ports, assigns them to players
    private void initialisePorts() { ; }

    // Returns the current turn number
    public int getTurn() { ; }

    // Sets the turn number
    public void setTurn(int newTurn) { ; }

    // Starts the next turn
    public void nextTurn() { ; }

    public int getMovesLeft() { ; }

    public Player getCurrentPlayer() { ; }

    public Player getCurrentPlayer_() { ; }

    // Gets player by index
    public Player getPlayer(int playerNum) { ; }

    // Sets up all the treasure for the game
    private void initTreasure() { ; }

    // Populates the 20x20 board with Tiles
    public void populateTiles() { ; }

    // Following methods return a given island
    public PirateIsland getPirateIsland() { ; }
```

```
    public TreasureIsland getTreasureIsland() { ; }

    public FlatIsland getFlatIsland() { ; }


    // Handles a player's movement (helps move the player to a coordinate)
    public boolean handlePlayerMovement(int toCol, int toRow) { ; }

    // After an attack has occured, the player's need to be moved. This
handles that
    public void dealWithAfterAttack(Player winner, Player loser) { ; }

    public void playerEndTurnSequence(boolean should, Tile pl, int[] coor) { ;
}

    // Allows an interaction with an island from a player
    public void interactWithIsland(String nameOfIsland) { ; }


    // The following methods handle when you interact with an island
    private void treasureIslandHandler() { ; }

    private void flatIslandHandler() { ; }

    private void pirateIslandHandler() { ; }

    // Creates and returns a new ocean tile
    private OceanTile makeOceanTile() { ; }

    // Rotates the player
    public void rotate(String turnDir) { ; }
}
```

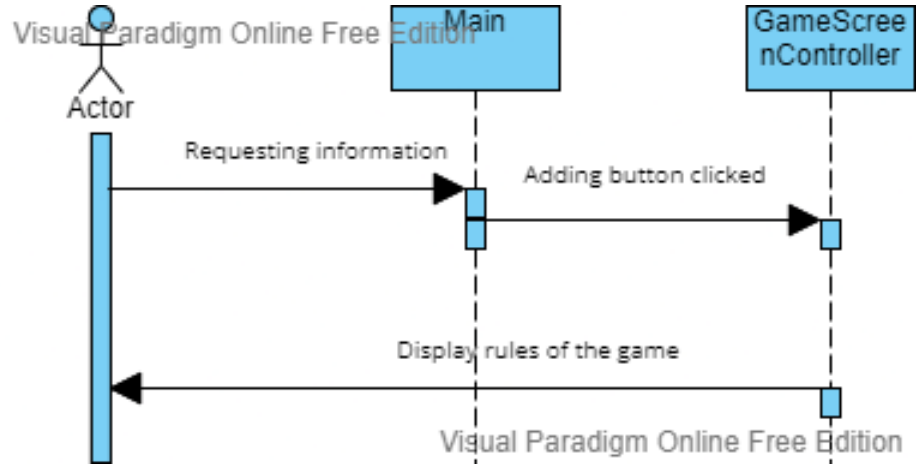
5. DETAILED DESIGN

5.1 Sequence Diagrams

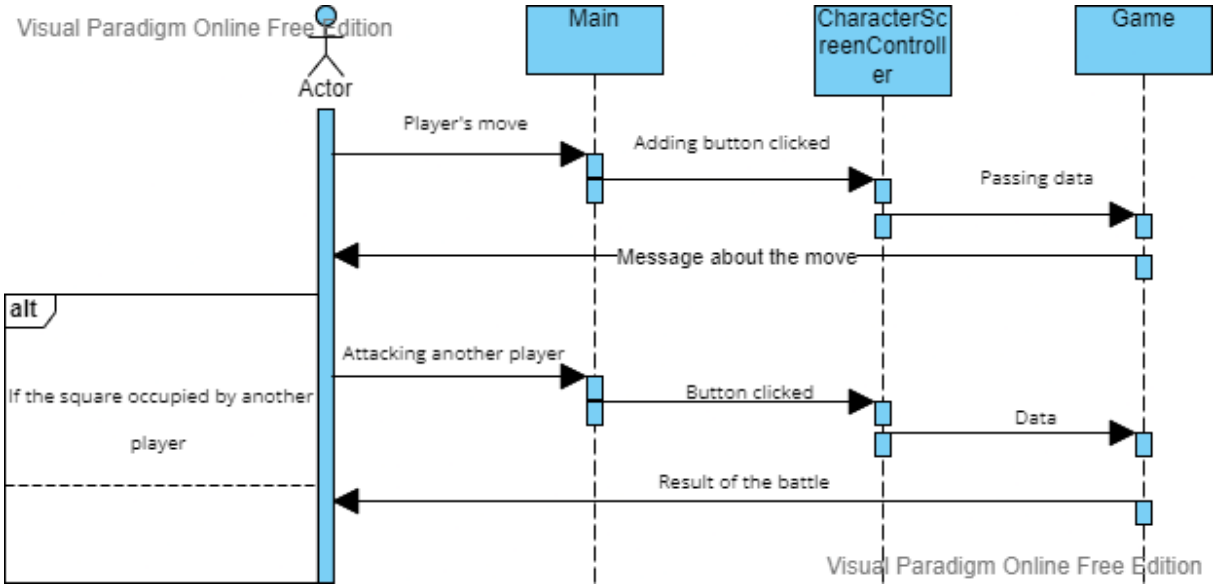
5.1.1 Use Case 1.1:



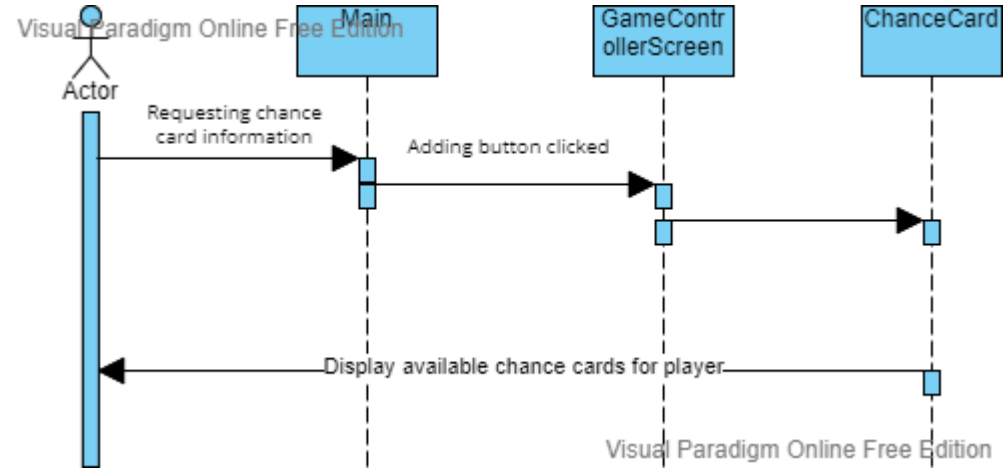
5.1.2 Use Case 1.2:



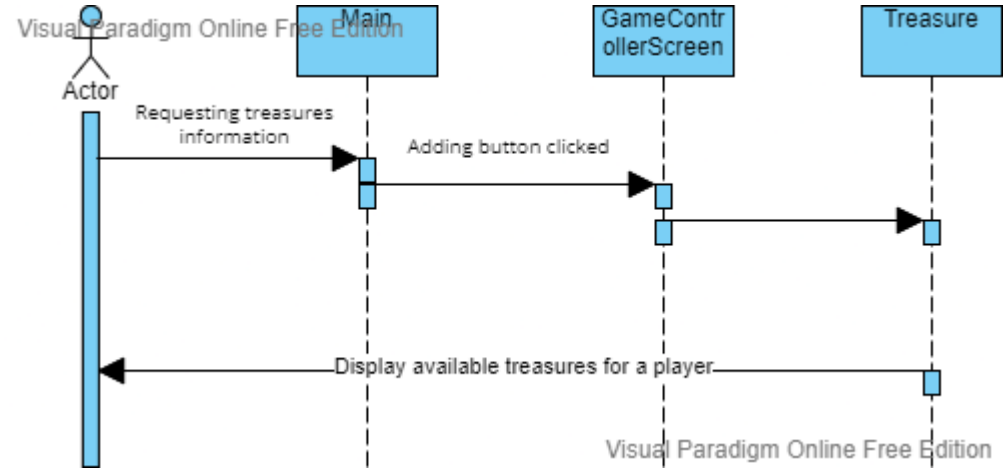
5.1.3 Use Case 1.3:



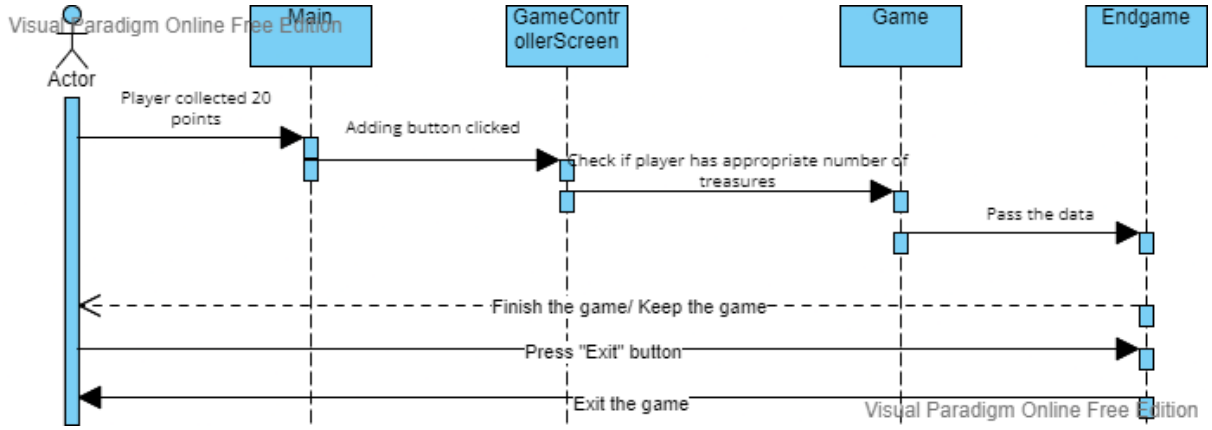
5.1.4 Use Case 1.5:



5.1.5 Use Case 1.6:



5.1.6 Use Case 1.8:



5.2 Significant algorithms

During our meetings with the group, we discussed the difficult parts of the code. We concluded that one of the more difficult tasks will be the implementation of chance cards. Since each card will contain an event, this will cause the game to react differently each time. This will make the game flow different every time. This will require separate 'event handlers' for each card.

Pseudocode:
If player arrives at Treasure Island, then
Pick chance card from pile
If card (14) selected
Use appropriate 'game handler' to proceed
(Example: Receive treasures for a total of six points)
Implement the consequences of the chance card
(add six points to player's account)
Move card to bottom of the pile
Next player's move

A significant algorithm that came up was dealing out chance cards to a player equalling to a certain value, here is an algorithm that we came up with to solve this: (next page)

```java
//transfers treasure from treasure island to a player using a combined val.
private static void giveTreasureClosestToValue(int valueDesired, TreasureHand toHnd, TreasureHand fromHnd){
    int treasureSlotsAvailable =2 - toHnd.getTreasures().size();
    Treasure[] doubleTreasure = new Treasure[2];
    Treasure singleTreasure = null;
    int doubleTreasureValue = 0;
    ArrayList<Treasure> doubleTreasures = new ArrayList<>();
    // check if there are any treasures available to collecct
    if (treasureSlotsAvailable == 0){
        return;
    }
    ArrayList<Treasure> lookedUpTreasures = new ArrayList<>();
    for (int i=valueDesired; i>= 2; i--){
        lookedUpTreasures = fromHnd.getTreasureIndexByValue(i);
        if (lookedUpTreasures.size()>0 ){
            singleTreasure = lookedUpTreasures.get(0);
            break;
        }
    }
```

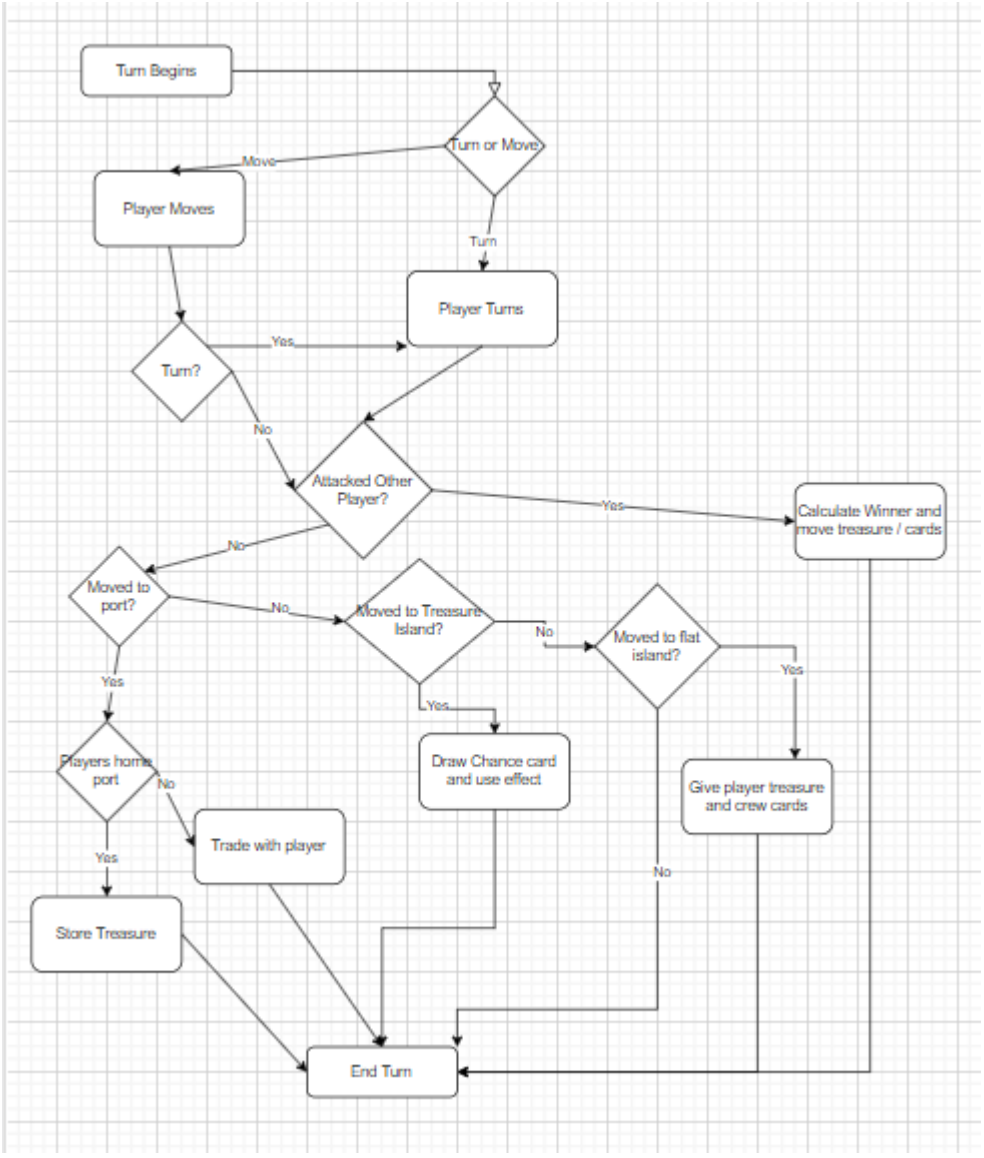
```java
    if (singleTreasure != null){
        if (singleTreasure.getValue()<valueDesired && treasureSlotsAvailable == 2){
            for (int i = valueDesired; i >=2; i--){
                Treasure iTreasure = null;
                Treasure jTreasure = null;
                if (fromHnd.getTreasureIndexByValue(i).size()>0){
                    iTreasure = fromHnd.getTreasureIndexByValue(i).get(0);
                    for (int j=i; j>=2; j--){
                        if (fromHnd.getTreasureIndexByValue(i).size()>0){
                            if (fromHnd.getTreasureIndexByValue(i).size()>1 && i==j){
                                jTreasure = fromHnd.getTreasureIndexByValue(i).get(1);
                            }
                            else{
                                jTreasure = fromHnd.getTreasureIndexByValue(j).get(0);
                            }
                            int sum = iTreasure.getValue() + jTreasure.getValue();
                            if (doubleTreasureValue < sum && sum <= valueDesired){
                                doubleTreasureValue = iTreasure.getValue() + jTreasure.getValue();
                                doubleTreasure[0] = iTreasure; doubleTreasure[1] = jTreasure;
                            }
                        }
                    }
                }
            }

        }
```

```java
        if(singleTreasure.getValue() > 0){
            if (singleTreasure.getValue() > doubleTreasureValue ){
                fromHnd.giveTreasureFromIndex(toHnd, fromHnd.getTreasureIndexByName(singleTreasure.getName()));
            }
            else{
                fromHnd.giveTreasureFromIndex(toHnd, fromHnd.getTreasureIndexByName(doubleTreasure[0].getName()));
                fromHnd.giveTreasureFromIndex(toHnd, fromHnd.getTreasureIndexByName(doubleTreasure[1].getName()));
            }
        }

    }
    System.out.println("the end");
}
```

5.3 Significant data structures

Two very important data structures used throughout the program are the CrewHand and the TreasureHand, which, as mentioned in 2.1.2, deal with the moving, removing, adding and storage of crew cards and treasure for an entity. There is also the “ChanceCard” class, which holds some simple information about the number of the chance card and it’s description. Inside this class is a private inner class called ChanceActions which basically is used to act out all of the chance cards’ different functionalities. It was done in this way due to a lack of time being available to put thought into successfully implementing a more Object Orientated approach.

5.4 State Flow Diagram



REFERENCES

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

DOCUMENT HISTORY

Version	*Issue No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	29/03/22	N/A - original version	BHW