Software Engineering Group02 Project Design Specification

Author:	Adrian Debski [add32], Alessandro Lewis [all49], Alexander Gardemann [alg68], Alvaro Fernandez de la Fuente [alf56], David Hernandez Rodriguez [dah73], Khalid Ameen Aslam [kha9], Mukhriz Bin Mustafa [mub11], Vincent Azmi [via8], Xander Davies [xad1]

Config Ref:	SE.G02.DesignSpec
Date:	11th May 2022
Version:	1.5
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 show our thoughts and decisions about how we are designing our game. It shows the relationships between different components/classes in our program and how they work together to achieve our goal.

1.2 Scope

This document specifies all the classes in our program, the relationships between them and how they work with each other.

This document should be read by all project members. It is recommended that the reader is familiar with the UI Specification [1]

1.3 Objectives

The objective of this document is to show how the program works by describing the workings of the classes in terms of their main functionalities and the relationships between them.

2. DECOMPOSITION DESCRIPTION

2.1 Program in the system

We have designed our system to only contain a single program. The program contains classes that handle the logic of the game and the graphics. We have Java Objects to handle the logic, controllers to handle the graphic parts, and also javaFX files to handle the layout.

2.2 Significant classes in the program

There are many classes that make up our program however, the three biggest and most significant classes are the following:

- Board:
    - This class is for the game's board. It handles the graphical part of the game such as initialising the elements of the board and game such as tiles, ports, bays and islands.
- Game:
    - This class is mainly for handling the logic of the game. For example, it sets up the game when it starts. This includes assigning players to their home ports and dealing the cards. It also deals with events that happen during the game for example whether players can move/rotate.
- GameController:
    - This class is for handling the graphical/UI part of the game, including some of the elements of the game such as the displaying treasures held at the players' home ports, crew cards held, the current player's ship, treasures in the ship and treasure/crew on the many islands, ports and bays.
- Player
    - This class stores the players' information such as their position on the board, name, home port, treasures and crew cards. It also has methods to implement the features from the functional requirements such as calculating the steps for moving and allowing players to hold crew cards.

- Trade Controller:
  - This is a class used to manipulate and display the UI elements on the trading screen as well as the logical part of trading. It uses methods for example to add the remove cards between players and ports.

- Battle:
  - This class is to implement the battle between the players (FR12). It handles the logic of the battle for example initiating battle and calculating the combat values of ships to determine the winner.

- BattleController:
  - This class handles the graphical/UI part of the battle screen. For example, it displays the winner of the battle, their ship and the treasures won.

- ChanceCardManager:
  - One of the biggest classes we have as it includes all of the chance cards we implemented and the common algorithms such as getting lowest value crew cards or lowest value treasure for example.
- JsonManager
  - This class handles the saving and loading of the game. We have successfully implemented saving and loading so the user can return to the game at any point.

2.3 Mapping from requirements to classes

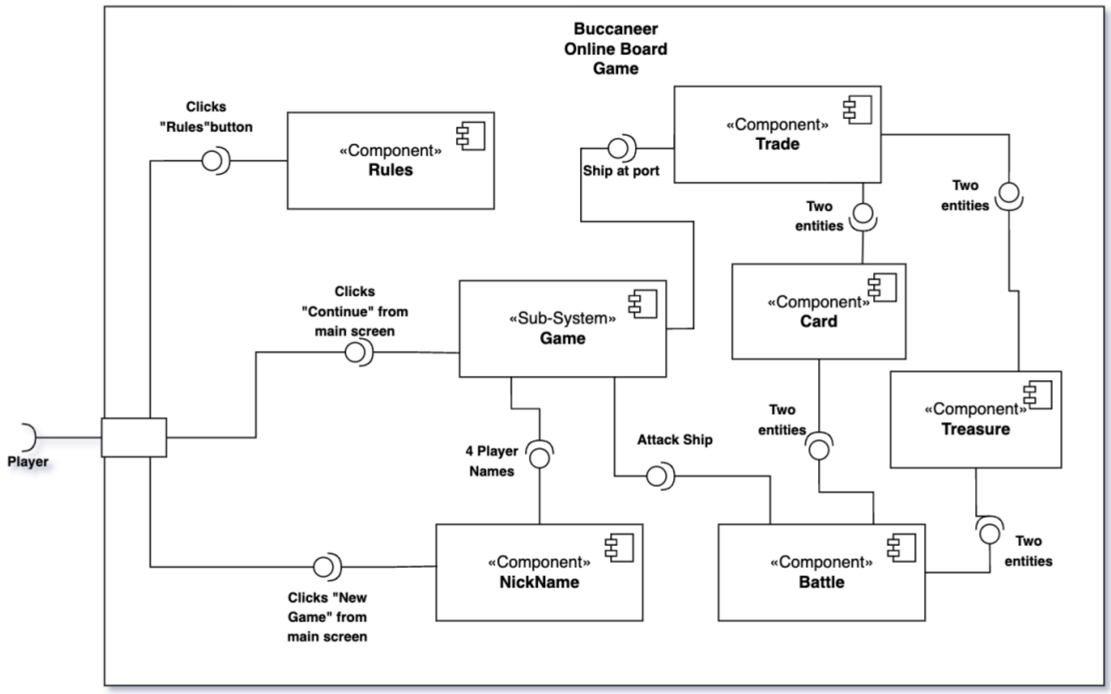
Requirement	Classes providing requirement
FR1	NickNameController, Game, save,load, JsonManager, MainMenuController, NicknamesController
FR2	Game, Player, Port
FR3	Game, Deck, CrewCard,Player
FR4	Game, Deck, ChanceCard, Player, ChanceCardManager, PopupController
FR5	Treasure, TreasureType, FlatIsland, TreasureIsland, Player, Port, SafeZoneController
FR6	Game, Board, Player, Port, Direction, CrewCard, ChanceCard, Treasure,
FR7	Port, Player, CrewCard, Treasure
FR8	FlatIsland, CrewCard, ChanceCard, ChanceCardManager, Treasure, TreasureType, PopupController
FR9	Board, Game, GameController, Tile, BoardElement, Port, Island, Bay, Player, Direction, Treasure, ChanceCard, CrewCard, State, FlatIsland, TreasureIsland, PopupController
FR10	Game, Board, Player, Deck, Port, CrewCard, Treasure

FR11	Game, Board, Player, Port, GameController, State, Battle, tile,
FR12	Game, GameController, Board, Player, CrewCard, Treasure, BattleController, State, Battle, TreasureType
FR13	Game, Board, Player, TreasureIsland, ChanceCard, ChanceCardManager, Deck, PopupController
FR14	Game, Board, Player, FlatIsland, CrewCards, Treasure, TreasureType
FR15	Game, GameController, Board, Player, Port, CrewCards, Treasure, TradeController, TreasureType, SafeZoneController, ChanceCard, ChanceCardManager, PopupController
FR16	Game, Board, AnchorBay, Treasure, ChanceCard, PopupController
FR17	Game, Board,Port, Player, Treasure, EndGameController, TreasureType, MainMenuController

3. DEPENDENCY DESCRIPTION

3.1 Component Diagrams

This is the Component Diagram for our Game:



4. INTERFACE DESCRIPTION

4.1 BoardElement interface specification

- Type: public
- Extends: nothing
- Public methods:
    - BoardElement(String name): The constructor.
    - String toString(): Returns the name of the board element.
    - String getName(): Also returns the name of the board element.

4.2 Bay interface specification

- Type: public
- Extends: BoardElement
    - This is so that bays, islands and ports can all be treated as board elements.
- Public methods:
    - Bay(String name): The constructor.

4.3 Island interface specification

- Type: public
- Extends: BoardElement
    - This is so that bays, islands and ports can all be treated as board elements.
- Public methods:
    - Island(String name): The constructor.

4.4 Port interface specification

- Type: public
- Extends: BoardElement
    - This is so that bays, islands and ports can all be treated as board elements.
- Public methods:
    - Port(String name, int value): The constructor.
    - void addCard(CrewCard card): Adds a crew card to the port.
    - void addTreasure(Treasure treasure): Adds treasure to the port.
    - ArrayList<CrewCard> getCards(): Returns a list with all the crew cards.
    - String getName(): Returns the port's name.
    - String toString(): Returns a string with information about the port.

4.5 Tile interface specification

- Type: public
- Extends: nothing
- Public methods:
    - Tile(boolean light, int x, int y): The constructor.
    - boolean isLight(): Returns whether the tile is light or dark.
    - boolean isPath(): Returns whether the tile is in the path of the current player.
    - void setPath(): Sets the tile to be considered in the path of the current player.
    - int getXCor(): Returns the x coordinate.
    - int getYCor(): Returns the y coordinate.
    - BoardElement getElement(): returns the BoardElement the tile belongs to.
    - void setElement(BoardElement element): Sets the BoardElement the tile belongs to.
    - String toString(): Returns a string with the x and y coordinates.

4.6 Card interface specification

- Type: public

- Extends: nothing
- Public methods: nothing. This is just a superclass for ChanceCard and CrewCard

4.7 ChanceCard interface specification

- Type: public
- Extends: Card
    - The class deck uses a list of Card to define methods for CrewCard and ChacneCard
- Public methods:
    - ChanceCard(int id): The constructor for unknown description.
    - ChanceCard(int id, String description): The constructor for a specified description.
    - String getDescription(): Returns the description of the card.
    - void setDescription(String description): Sets the description of the card.
    - String toString(): Returns a string with the id and description of the card.

4.8 CrewCard interface specification

- Type: public
- Extends: Card
    - The class deck uses a list of Card to define methods for CrewCard and ChacneCard
- Public methods:
    - CrewCard(): Constructor for default color (black) and value (1)
    - CrewCard(int color, int value): The constructor. for the color parameter 0 is black and 1 is red.
    - int getColor(): Returns the color.
    - void setColor(int color): Sets the color of the card. 0 for black, 1 for red.
    - int getValue(): Returns the value of the card.
    - void setValue(int value): Sets the value of the card.
    - String toString(): Returns a string with the color and value of the card.
    - int compareTo(CrewCard o): Compares two cards by value so they can be sorted.
    - boolean equals(Object o): Compares the value and color of the cards for equality.

4.9 Deck interface specification

- Type: public
- Extends: nothing
- Public methods:
    - void shuffleDeck(): shuffles the deck of cards.
    - List<Card> getCards(): Returns the deck of cards.
    - void addCard(Card card): Adds a card to the deck.
    - Card getCard(): Gets the first card from the deck and removes it from the deck.
    - String toString(): Returns a string with information for all the cards.

4.10 Board interface specification

- Type: public
- Extends: nothing
- Public methods:
    - Board(): The constructor. Makes a 20x20 tiled board.
    - void draw(): Refreshes the rendering of the board.
    - Group getBoardGroup(): (Something about javaFX, not sure)
    - Tile getTile(int x, int y): Returns the tile on the coordinate (x, y)
    - void cleanTiles(): (Not sure, maybe setPath is related to the different colors in tiles)
    - void markReachable(): (Seems to mark reachable tiles)
    - Port[] getPorts(): Returns an array with all the ports

4.11 Game interface specification

- Type: public

- Extends: nothing
- Public methods:
    - static void initialize(): Initializes the game.
    - static void changeState(State newState): Controls the state of the game.
    - static void onMousePress(Tile tile): Manages the clicks on the board.
    - static Player getCurrentPlayer(): Returns the current player.
    - static State getCurrentState(): Returns the current state.
    - static ArrayList<Player> getPlayers(): Returns a list with the players.
    - static void setNicknames(String[] nicknames): Sets the nicknames for the players.
    - static boolean isInPort(): checks if player is in port
    - static boolean isInHomePort(): checks if player is in their home port
    - static boolean canTrade(): checks if a player can trade depending on what port they're in
    - static Player nearestShip(): checks with player is closest to you
    - static void syncCrewCards(): moves crew cards deposited in a players port to their hand
    - static void createPopup(String title, String message, int numButtons, ArrayList<String> labels)
    - static void startBattle(Player one, Player two): starts battle between two players
    - static void load(): loads the json file to return state of game
    - static boolean checkLegalDirection(): checks if the player can actually move in that direction
    - static void setTurn(int turn): lets the player move/rotate when it is their turn

4.12 Player interface specification

- Type: public
- Extends: nothing
- Public methods:
    - Player(int x, int y, String name, int color): The constructor.
    - int getXCor(): Returns the x coordinate
    - void setXCor(int xCor): Sets the x coordinate
    - int getYCor(): Returns the y coordinate
    - void setYCor(int yCor): Sets the y coordinate
    - String getName(): Returns the name of the player
    - int getColor(): Returns the color of the ship
    - int getSteps(): Returns the number of steps the player can move.
    - Direction getDirection(): Returns the direction of the ship.
    - void setDirection(DIrection dir): Sets the direction of the ship.
    - void setHomePort(Port port): Sets the player's home port.
    - Port getHomePort(): Returns the home port.
    - int[] cardsDescription(): Returns an array counting how many crew cards the player has for each color and value.
    - void addCard(CrewCard card): Gives the player a crew card.
    - Card getCard(): Gets the first card from the player and removes it.
    - ArrayList<CrewCard> getCards(): Returns a list of all the crew cards.
    - String toString(): Returns a string with the name of the player.

4.13 Treasure interface specification

- Type: public
- Extends: Item
    - (Why)
- Public methods:
    - Treasure(String name, TreasureType type): The constructor.
    - TreasureType getType(): Returns the type of the treasure.
    - String getName(): Returns the name of the treasure.
    - int getValue(): Returns the value of the treasure.
    - int compareTo(Treasure o): Compares two treasures so they can be sorted.
    - String toString(): Returns a string with info about the treasure.

4.14 TradeController Interface specification

- Type: public
    - Extends: None
        - (Why)
    - Public methods:
        - Trade(Player player, Port port): The constructor for the fx-controller.

4.15 Main interface specification

- Type: public
- Extends: Application
    - This is required by JavaFX
- Public methods:
    - void start(Stage stage): Starts the JavaFX stuff
    - static void main(String[] args) Main method that starts the entire application.

4.16 JsonManager interface specification

-Type: public.
-Extends: Nothing j.
-Public methods:
    -void save: saves game state into Json file.
    -void load: loads game state from Json file.

4.17 ChanceCardManager interface specification

-Type: public.
-Extends: Nothing.
-Public methods:
    -void applyEffects: applies the effects of the chance cards to the game.

4.18 BackgroundController interface specification

-Type: public
-Extends: Nothing
-Public methods:
    -void initialize(URL url, ResourceBundle resourceBundle): initialises background for screen

4.19 Battle interface specification

-Type: public
-Extends: Nothing
-Public methods:
    -static void start(Player p1, Player p2): starts a battle between two players
    -static void finish(): runs when battle is finished so winnings can be transferred
    -static void calculateWin(): sets the winner and loser of the battle based on attack power
    -static void setCardsWon(ArrayList<CrewCard> c): takes the loser's deck of cards and works out the two lowest
    -static int calcAttackPower(Player p): calculates the attack power of a player
    -static Player getWinner(): returns the winner
    -static Player getLoser(): returns the loser
    -static ArrayList<CrewCard> getWonCards(): returns the cards won

4.20 BattleController interface specification

-Type: public
-Extends: Nothing
-Public methods:
-void initialize(): starts the battle and checks whether the loser has treasures
-void updatePlayers(Player winner, Player loser): updates the players on the screen
-void continueButtonClick(ActionEvent actionEvent): finishes the battle and switches the scene back to the game screen
-image makeTreasureImage(Treasure t): make an image for a treasure depending on its type
-image makeCardImage(CrewCard c): make an image for a card depending on its type
-void loadTreasureWinnings(ArrayList<Treasure> t): loads the treasures the winner has won
-void loadCardWinnings(ArrayList<CrewCarde> wonCardst): loads the cards the winner has won

4.21 Coast interface specification

-Type: public
-Extends: BoardElement
-Public methods:
-Coast(String name): constructor for the superclass
-String toString(): toString method

4.22 EndScreenController interface specification

-Type: public
-Extends: nothing
-Public methods:
-EndScreenController(): constructor method
-void initialize(): initialises the end screen
-image makeTreasureImage(Treasure t): creates the treasure image
-void loadTreasureWinnings(ArrayList<Treasure> t, ArrayList<Treasure> sz):loads treasures the winner has won
-void quitButtonPress(ActionEvent actionEvent): closes the game
-void newButtonPress(ActionEvent actionEvent): Takes the user to the nickname screen to start a new game

4.23 FlatIsland interface specification

-Type: public
-Extends: Island
-Public methods:
-FlatIsland(String name): constructor method
-void setCrewCards(): gets the crew cards
-void addCard(CrewCard card): adds the card to the deck
-Card getCard(): removes card from the deck
-int[] cardsDescription(): returns summary of crew cards
-void addTreasure(Treasure treasure): it adds treasure to the list
-void removeTreasure(Treasure treasure): removes treasure from the list
-ArrayList<Treasure> getTreasures(): returns list of the treasures
-List<Card> getCrewCards(): returns list of the cards
- int[] treasureDescription(): returns summary of the treasures

-void setTreasures(ArrayList<Treasure> treasures): sets list of treasures to the new one
-String toString(): toString method

4.24 GameController interface specification

-Type: public
-Extends: nothing
-Public methods:
-void initialize(): initialises the game screen

4.25 MainMenuController interface specification

-Type: public
-Extends: nothing
-Public methods:
-nothing

4.26 NicknamesController interface specification

-Type: public
-Extends: MainMenuController
-Public methods:
-void initialize(): Create nickname array and check names

4.27 PirateIsland interface specification

-Type: public
-Extends: Island
-Public methods:
-PirateIsland(String name): Constructor method
-void setTreasures(ArrayList<Treasure> treasures): sets treasures on pirate island
-void addCard(CrewCard card): adds cards to pirate island
-Card getCard(): get crew card from pirate island
-int[] cardsDescription(): gets card descriptions
-ArrayList<Treasure> getTreasures(): gets treasure from array list
-List<Card> getCrewCards(): gets card from list
-void addTreasure(Treasure treasure): adds treasure
-void removeTreasure(Treasure treasure): removes treasure
-int[] treasureDescription(): gets treasure descriptions

4.28 PopupController interface specification

-Type: public
-Extends: nothing
-Public methods:
-PopupController(String text, int nButtons, ArrayList<String> l): constructor method to set message, number of buttons and button labels
-PopupController(String text, int nButtons, ArrayList<String> l, int id): constructor method with chance card id
-void initialize(): initialises buttons and labels

4.29 RulesScreenController interface specification

-Type: public
-Extends: MainMenuController
-Public Methods:
-nothing

4.30 SafeZoneController interface specification

-Type: public
-Extends: Nothing
-Public Methods:
-SafeZoneController(Player currentPlayer): constructor method
-void initialize(URL location, ResourceBundle resources): initialises a controller after its root element has been completely processed

4.31 SliderChangeListener interface specification

-Type: public
-Extends: Nothing
-Public Methods:
-SliderChangeListener(Slider tradeSlider, Label tradeLabel, IntegerProperty totalChangeValue, int totalMultiplier): constructor method
-void changed(ObservableValue<? extends Number> observable, Number oldValue, Number newValue): changes slider information based on state of the game

4.32 State interface specification

-Type: public
-Extends: Nothing
-Public Methods:
-Nothing

4.33 TreasureIsland interface specification

-Type: public
-Extends: Island
-Public Methods:
-void setTreasures(ArrayList<Treasure> treasures): set the treasure on the island
-void setChanceCards(Deck chanceCards): set the chance cards on treasure island
-TreasureIsland(String name): constructor for treasure island
-void addCard(ChanceCard card): add a card to the deck of cards
-Card getCard(): get a chance card from the top of the deck
-void shuffle(): shuffles the cards in the deck
-void addTreasure(Treasure treasure): add a treasure to the treasures stored on the island
-ArrayList<Treasure> getTreasures(): get the treasures located on treasure island
-Treasure getTreasure(TreasureType type): get a certain treasure from the island if it is present
-List<Card> getChanceCards(): gets a list of chance cards present on the island in a list format
-int[] treasureDescription(): returns a description of all treasure present on the island

4.34 TreasureType interface specification

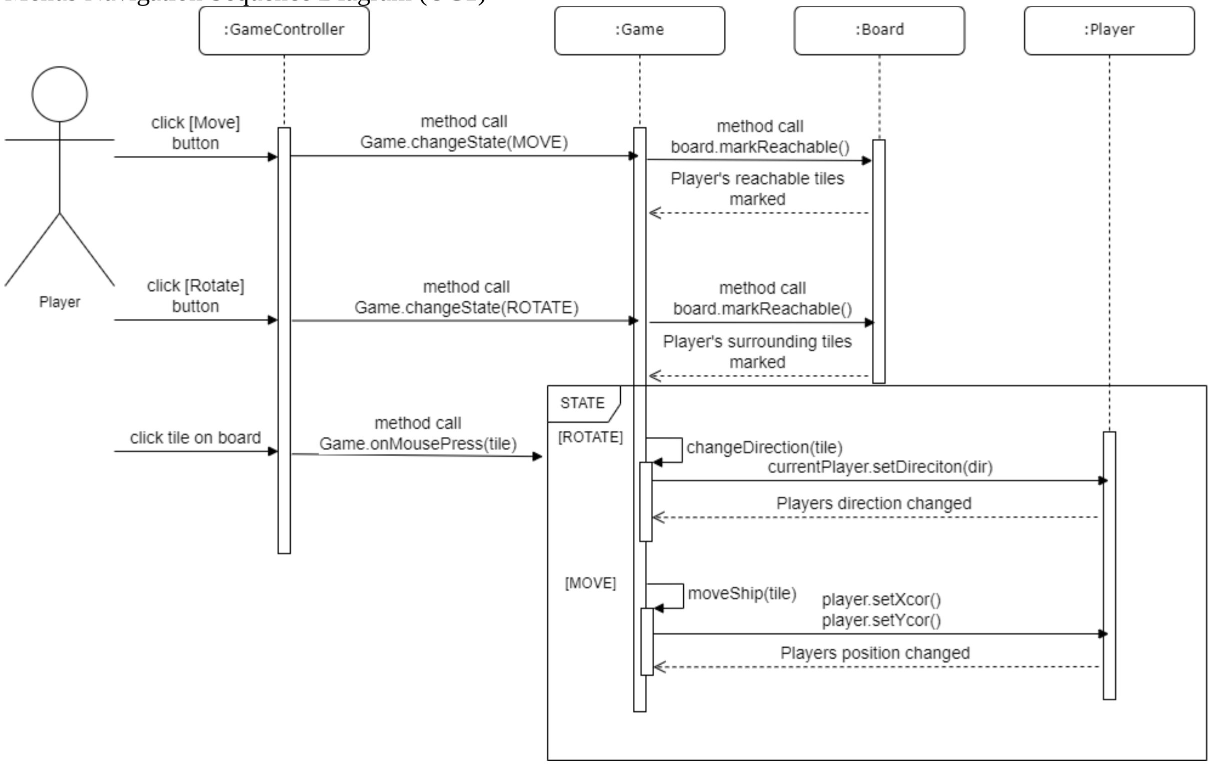
-Type: public

-Extends: Nothing
-Public Methods:
-Nothing

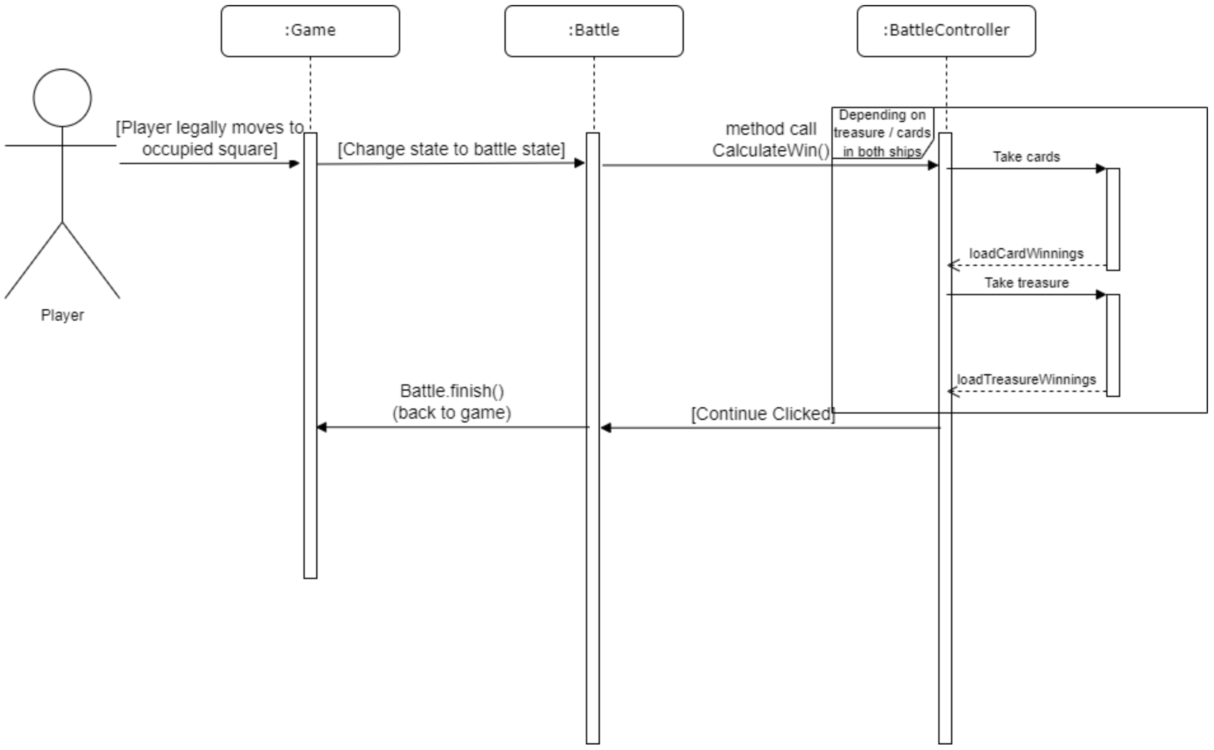
5. DETAILED DESIGN

5.1 Sequence diagram

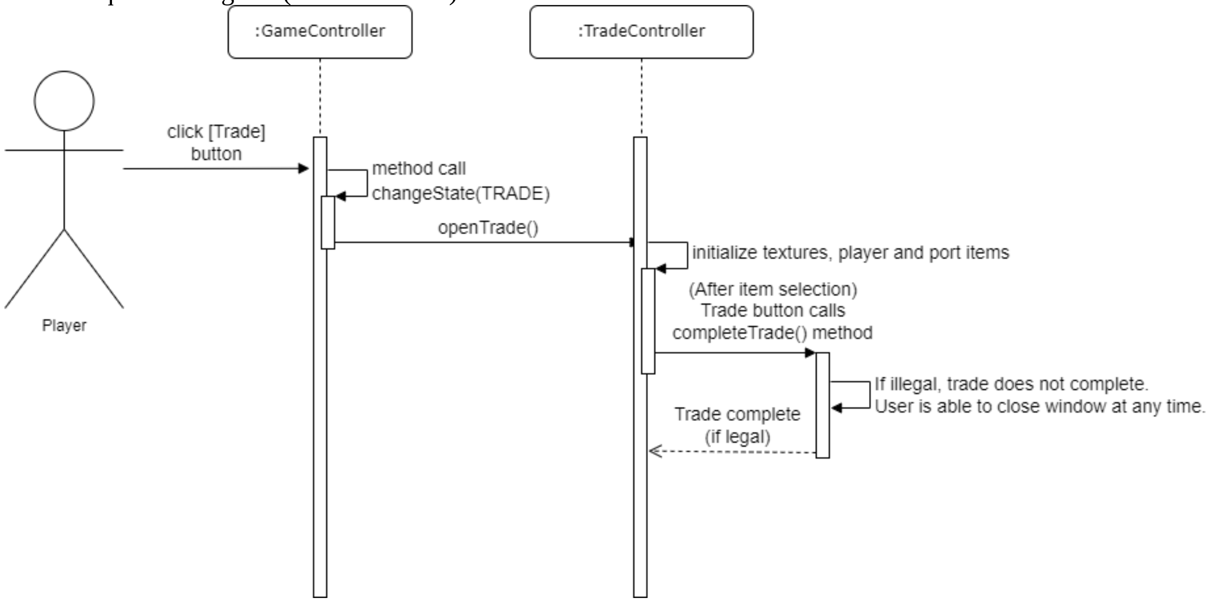
Menus Navigation Sequence Diagram (UC1)


:GameController
:Game
:Board
:Player
Player
click [Move] button
method call Game.changeState(MOVE)
method call board.markReachable()
Player's reachable tiles marked
click [Rotate] button
method call Game.changeState(ROTATE)
method call board.markReachable()
Player's surrounding tiles marked
STATE
[ROTATE]
click tile on board
method call Game.onMousePress(tile)
changeDirection(tile)
currentPlayer.setDireciton(dir)
Players direction changed
[MOVE]
moveShip(tile)
player.setXcor()
player.setYcor()
Players position changed

Battle Sequence Diagram (USE CASE 10)


:Game
:Battle
:BattleController
Player
[Player legally moves to occupied square]
[Change state to battle state]
method call CalculateWin()
Depending on treasure / cards in both ships
Take cards
loadCardWinnings
Take treasure
loadTreasureWinnings
Battle.finish() (back to game)
[Continue Clicked]

Trade Sequence Diagram (USE CASE 11)


:GameController
:TradeController
Player
click [Trade] button
method call changeState(TRADE)
openTrade()
initialize textures, player and port items
(After item selection) Trade button calls completeTrade() method
If illegal, trade does not complete. User is able to close window at any time.
Trade complete (if legal)

5.2 Significant algorithms

5.2.1 Rotating ship

To rotate a Ship, adjacent Tile should be clicked. To determine if the Tile which has been clicked is a legal choice, some calculations must be done using Ship's and Tile's coordinates. In the first version each case was covered with appropriate if statement e.g.
We know the direction should be changed to North West if Tile(x) - Ship(x) = -1 and Tile(y) - Ship(y) = -1
The downside of this solution is the fact that this kind of calculation has to be done before each if statement, so in the worst scenario we would have to carry it out 8 times before we get an output. We started analyzing the 2D grid to find a more optimal solution.
It's been noticed that X value for this grid looks like that:

0	1	2
0	1	2
0	1	2

And the Y value looks like that:

0	0	0
1	1	1
2	2	2

And the goal was to give each of this Tile a unique number:

0	1	2
3	4	5
6	7	8

Therefore the formula for this problem is

$$value = Tile(x) - Ship(x) + 1 + 3 * (Tile(y) - Ship(y) + 1)$$

X is always from 0 to 2, so to get the desired result we have to increase the value of X by multiple of 3 in each row

5.2.2 Moving ship

To move a Ship, Tile which is on its path must be clicked. To generate all the Tiles on the path the algorithm needs to know the direction, coordinates and number of steps of the ship. The loop is run from 1 to the number of step inclusive. Depending on the direction, all Tiles on the path of the Ship are flagged to be on the path, until the end of the loop or in case the island or border of the board is reached.

5.2.3 A* path Finding

Chance number 7 required calculating the distance to the nearest ship without stepping on the lands. I decided to implement the A* path finding algorithm, which can be described in pseudocode as following:

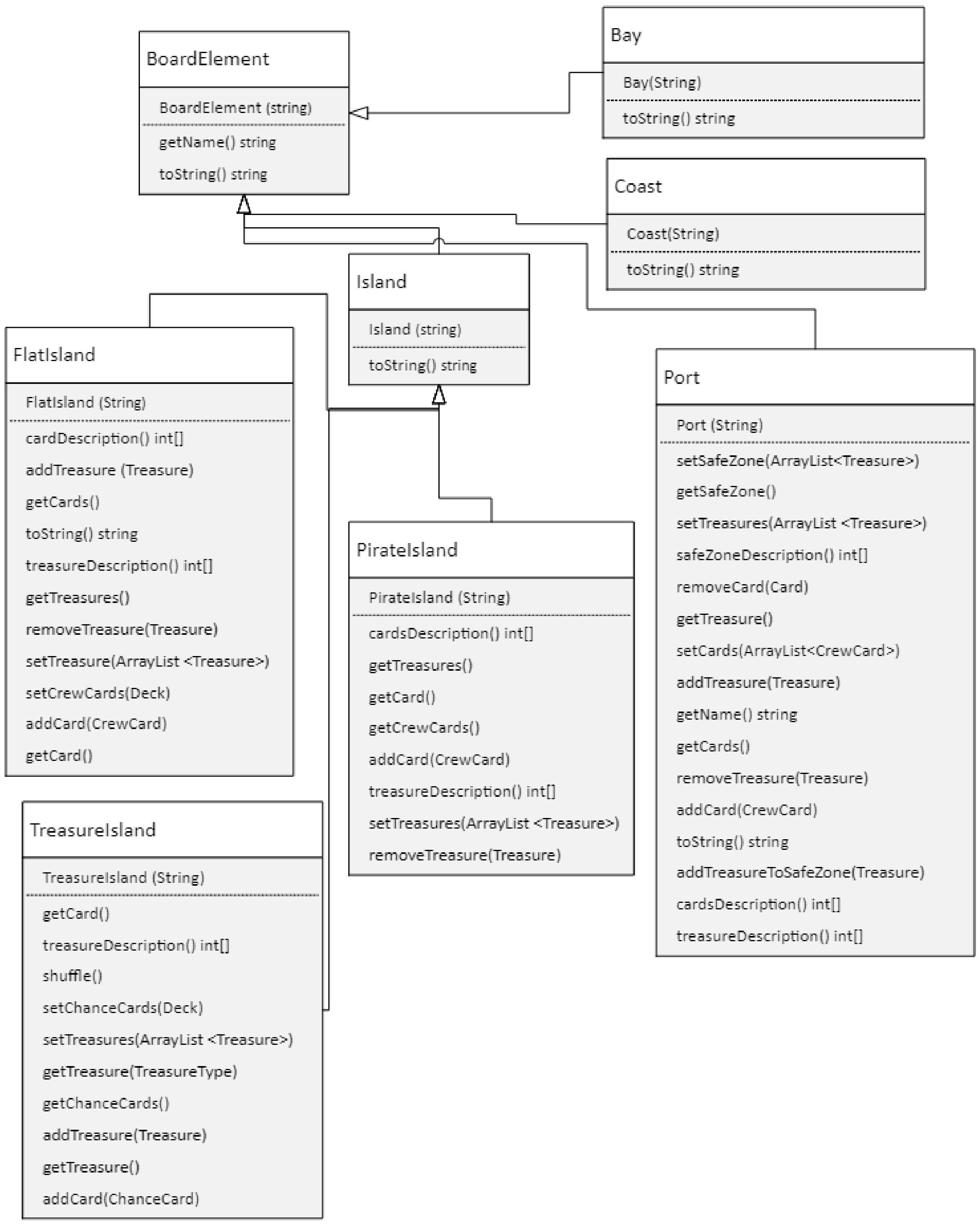
```
take coordinates of start point and end point
currentCords = startCords
nodeCords = 0
numofSteps = 0
while startCords are different from endCords
do
```

```
{
numOfSteps++
tmpCords = 0
distance = 9999
for each tile adjacent to the current one
do
{
assign nodeCords to proper values basing on iterator value
check if the tile to be consider is within the board AND it's not the current Tile AND it's not an Island
calculate the euclidean distance from this tile to the goal tile (without taking the square root)
if this distance is lower than the distance variable
do
{
update distance
assing tempCords to the cords of tile which is considered in this iteration
}
}
currentCords = tmpCords
}
return numOfSteps
```

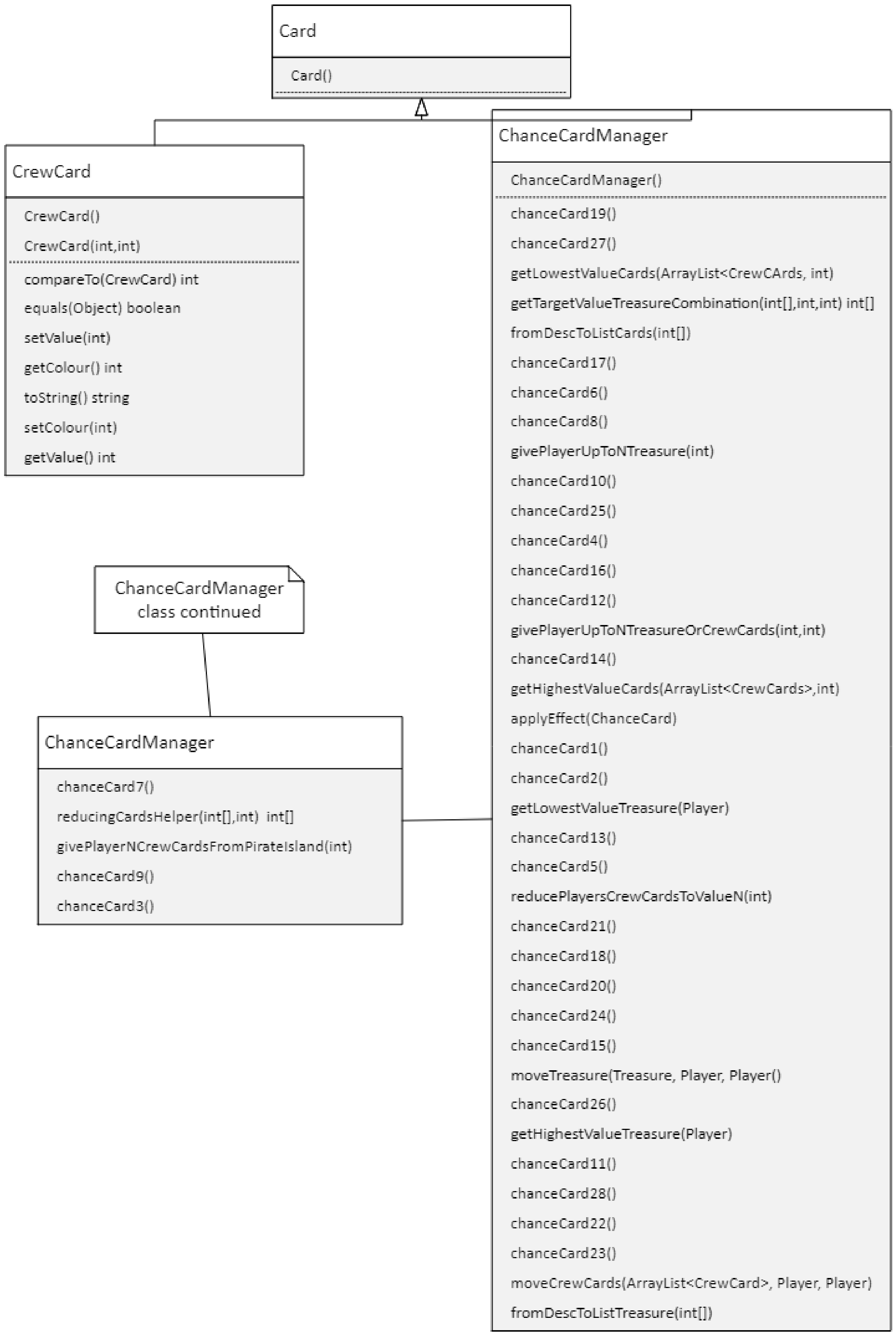
In summary:
Look for the best node(Tile) until the goal is reached.
In each iteration consider all nodes adjacent to the current one.
Pick the one which has the best score (the shortest distance to the goal)
The new one will become the new current node.
Since the path cost for moving between nodes is always one we have to consider only the heuristic function which is the euclidean distance in this case.

5.3 UML Class Diagrams

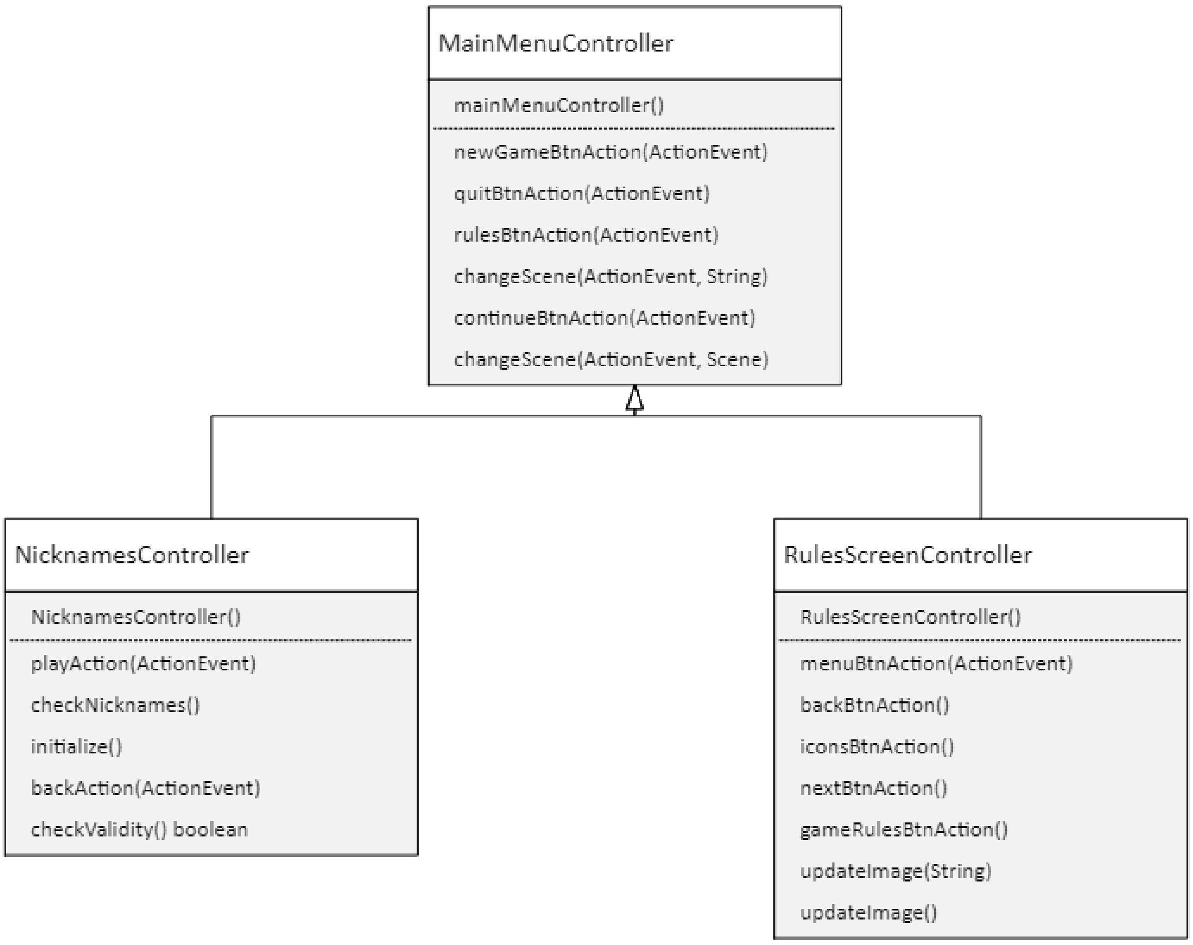
This diagram shows the relationships between the necessary classes in order to form majority of the working game.



This diagram shows the relationships between the necessary classes in order for both crew and player cards to work.



This diagram shows the relationships between each class that's required for the main menu to work as intended.



5.4 Significant data structures

Deck class is to simulate behaviour of the real deck of cards. It’s been said that cards should be dealt from the top and returned to the bottom. We recognized the First In First Out principle here, so we concluded the queue would be the most appropriate data structure. The ArrayList implementation of it is used in the Deck class along with provided methods to get/return cards in the deck.

The save file is a JSONObject data structure which is updated each turn as the game progresses. This is so when the game is read via the continue button on the main menu, the game can resume from the latest possible moment.

REFERENCES

[1] Software Engineering Group Project: The Main Use Case of the System. SE.G02.UseCaseDoc. 1.0 Release
[2] Software Engineering Group Projects: Design Specification Standards. SE.QA.05. 2.3 Release

DOCUMENT HISTORY

Version	*Issue No.*	*Date*	*Changes made to document*	*Changed by*
0.1	N/A	20/03/2022	Added Introduction section + Section 2.1 and 2.2	mub11
0.1	N/A	20/03/2022	Mapping from requirements to classes.	dah73 kha9
0.1	N/A	21/03/2022	Added Menu Navigation Sequence diagram	via8
0.2	#6	27/03/2022	Added author names in alphabetical order.	mub11
0.2	#8	27/03/2022	Corrected Mapping from requirements to classes.	dah73 add32
0.2	#10	27/03/2022	Added Read and Write JSON interface specification	dah73
0.3	#7	29/03/2022	Expanded section 2.1. Added a little more information.	mub11
0.3	N/A	29/03/2022	4.14 TradeController 4.16 Read and Write Json, 4.17 Chance Card Manager	dah73 add32 alg68
1.0	N/A	29/03/2022	Final check before releasing on gitlab.	mub11
1.1	N/A	09/05/2022	Updated mapping from requirements to classes based on new classes in system and completed interface description	kha9
1.1	N/A	09/05/2022	Updated 5.1 Sequence Diagram	via8
1.2	N/A	1005/2022	Added a component diagram	mub11, dah73
1.3	N/A	10/05/2022	Added A* Path finding algorithm	add32
1.4	N/A	10/05/2022	Added final contributions, updated UML	all49
1.5	N/A	11/05/2022	Final Checks	mub11