Software Engineering Group02 Project

Design Specification

| Author: | Mukhriz Bin Mustafa [mub11] |
| --- | --- |
| Config Ref: | SE.G02.DesignSpec |
| Date: | 20th March 2022 |
| Version: | 0.1 |
| Status: | Draft |



# CONTENTS

[**CONTENTS**](#_gjdgxs) **2**

[**1. INTRODUCTION**](#_30j0zll) **3**

[1.1 Purpose of this Document](#_1fob9te) 3

[1.2 Scope](#_3znysh7) 3

[1.3 Objectives](#_2et92p0) 3

[**2. DECOMPOSITION DESCRIPTION**](#_wa5r4ldfwm1c) **3**

[2.1 Program in the system](#_m3enqooryo0e) 3

[2.2 Significant classes in the program](#_rn5utppq5k0d) 3

[2.4 Mapping from requirements to classes](#_yhh0iecue051) 3

[**3. DEPENDENCY DESCRIPTION**](#_6jzu0n3g8sqy) **4**

[3.1 Component Diagrams](#_wf11oa8u891w) 4

[**4. INTERFACE DESCRIPTION**](#_jno92xorbk5e) **8**

[4.1 BoardElement interface specification](#_f5wd3d6833s9) 8

[4.2 Bay interface specification](#_e8u4uxvcbkbi) 8

[4.3 Island interface specification](#_aqraoy9x8ue5) 9

[4.4 Port interface specification](#_bcgblsy16nas) 9

[4.5 Tile interface specification](#_ec6magbd4n9y) 9

[4.6 Card interface specification](#_fijfiuvw1bj6) 9

[4.7 ChanceCard interface specification](#_q3ltwtog4hac) 9

[4.8 CrewCard interface specification](#_1j447gf89w5) 10

[4.9 Deck interface specification](#_tx3qka4nwyca) 10

[4.10 Board interface specification](#_h86txtpcttvd) 10

[4.11 Game interface specification](#_ncjhfd7o92wx) 10

[4.12 Player interface specification](#_tiy6s03mbuvw) 11

[4.13 Item interface specification](#_4jnd0odfzcrl) 11

[4.14 Treasure interface specification](#_70svs9kpo3t8) 11

[4.15 Trade Interface specification](#_ux71a099wpyw) 11

[4.16 BuccaneerApplication interface specification](#_wuozpsabhn66) 12

[**5. DETAILED DESIGN**](#_ibt8pzxb8dxg) **12**

[5.1 Sequence diagram](#_ohwhpgxxzlp9) 12

[5.2 Significant algorithms](#_p32gka8h6yup) 12

[5.3 Significant data structures](#_uzi90sfdwrpv) 13

# 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.

## 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.
* GameController
  + This class is also for handling the graphical part of the game, including some of the UI elements like what happens in the event of a mouse click for example.

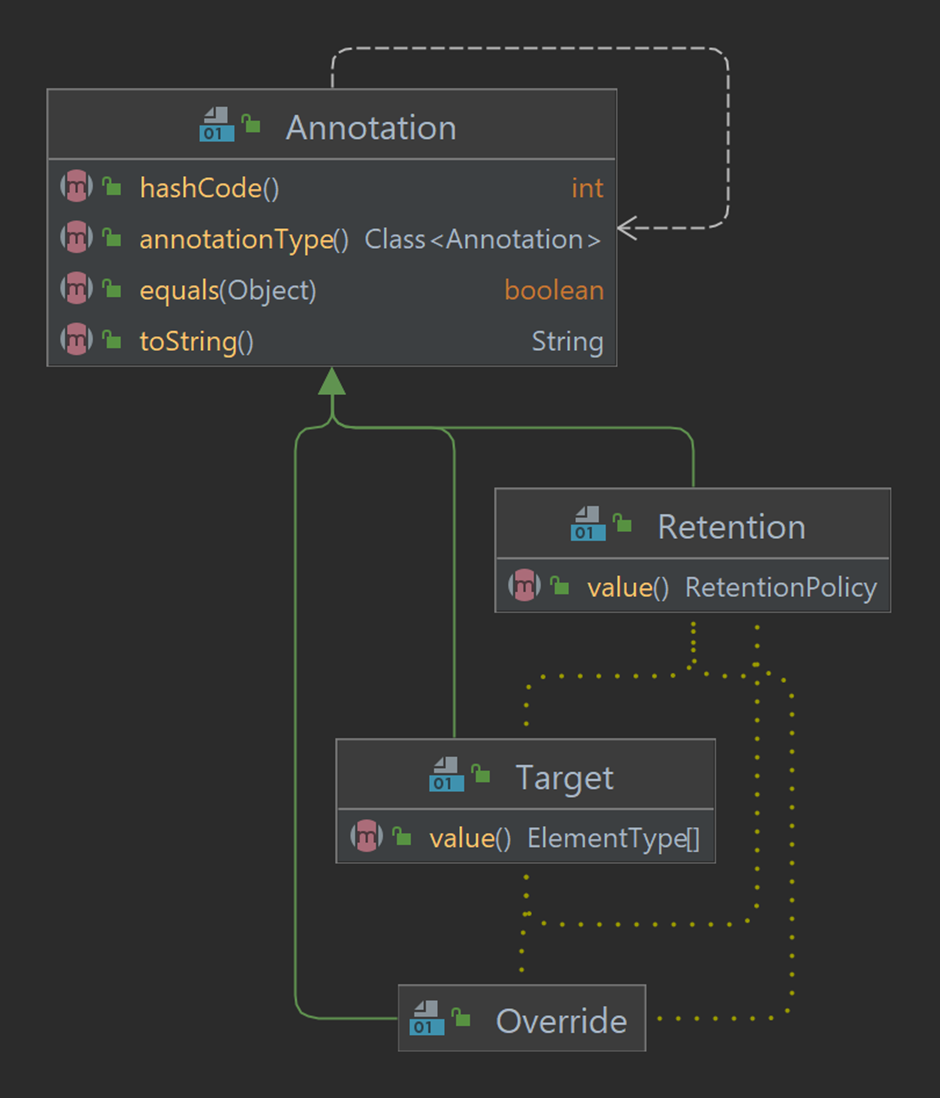
## 2.4 Mapping from requirements to classes

| Requirement | Classes providing requirement |
| --- | --- |
| FR1 | Player |
| FR2 | Port,Player |
| FR3 | Deck, CrewCard |
| FR4 | Deck, ChanceCard, Player |
| FR5 | Treasure, TreasueType, Island |
| FR6 | Player, Port, Direction, CrewCard, ChanceCard, |
| FR7 | Port, ChanceCard, Player |
| FR8 | Island, CrewCard, Treasure, TReasureType |
| FR9 | Board, BoardElement, Port, Island, Player, State, Tile, Direction, Treasure, ChanceCard, CrewCard, TreasureType |
| FR10 | BuccaneerAplication, Game, Board, Player, Deck, Port, CrewCard, TreasureType, Treasure |
| FR11 | BuccaneerAplication, Game, Board, Player, Port, Direction, Game controller, Tile, State, |
| FR12 | BuccaneerAplication, Game, Board, Player, CrewCard, Treasure, TreasureType |
| FR13 | BuccaneerAplication, Game, Board, ChanceCard |
| FR14 | BuccaneerAplication, Game, Board, Island, Cards, Treasure |
| FR15 | BuccaneerAplication, Game, Board, Port, ChanceCard |
| FR16 | BuccaneerAplication, Game, Board, ChanceCard, Bay, Treasure |
| FR17 | BuccaneerAplication, Game, Board,Port, Player, Treasure |

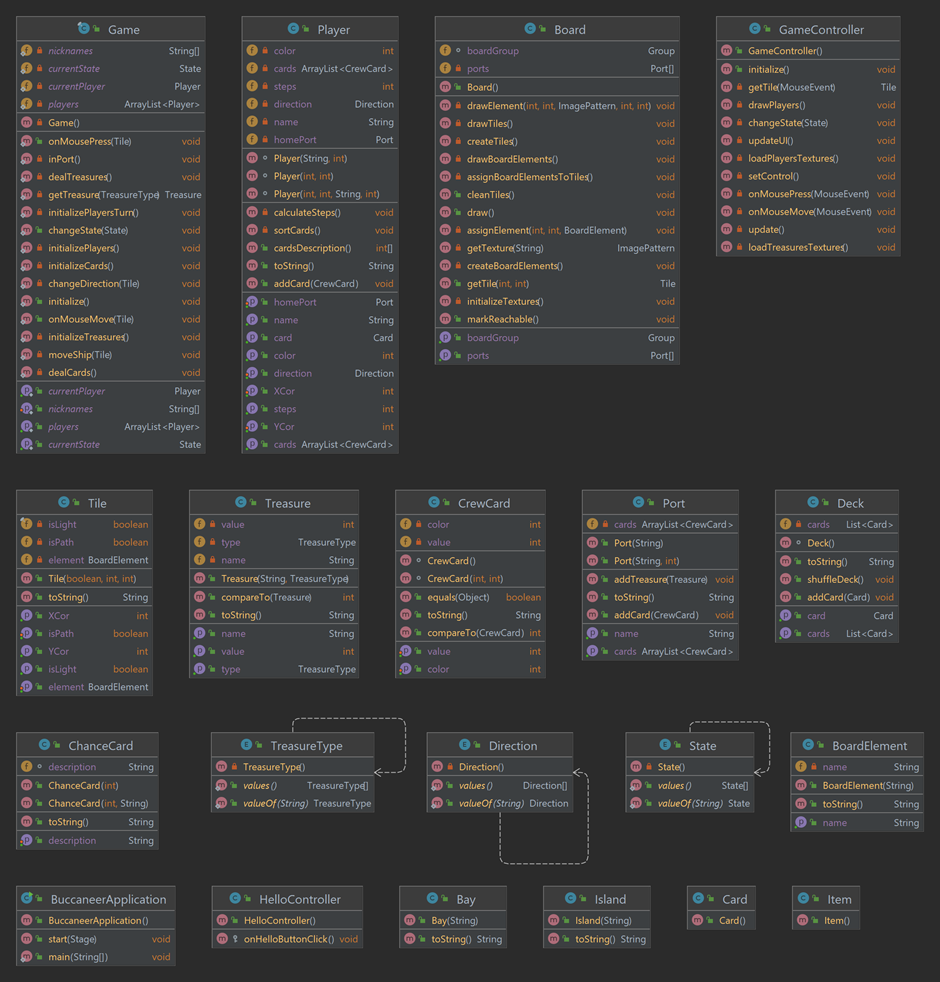
# 3. DEPENDENCY DESCRIPTION

## 3.1 Component Diagrams

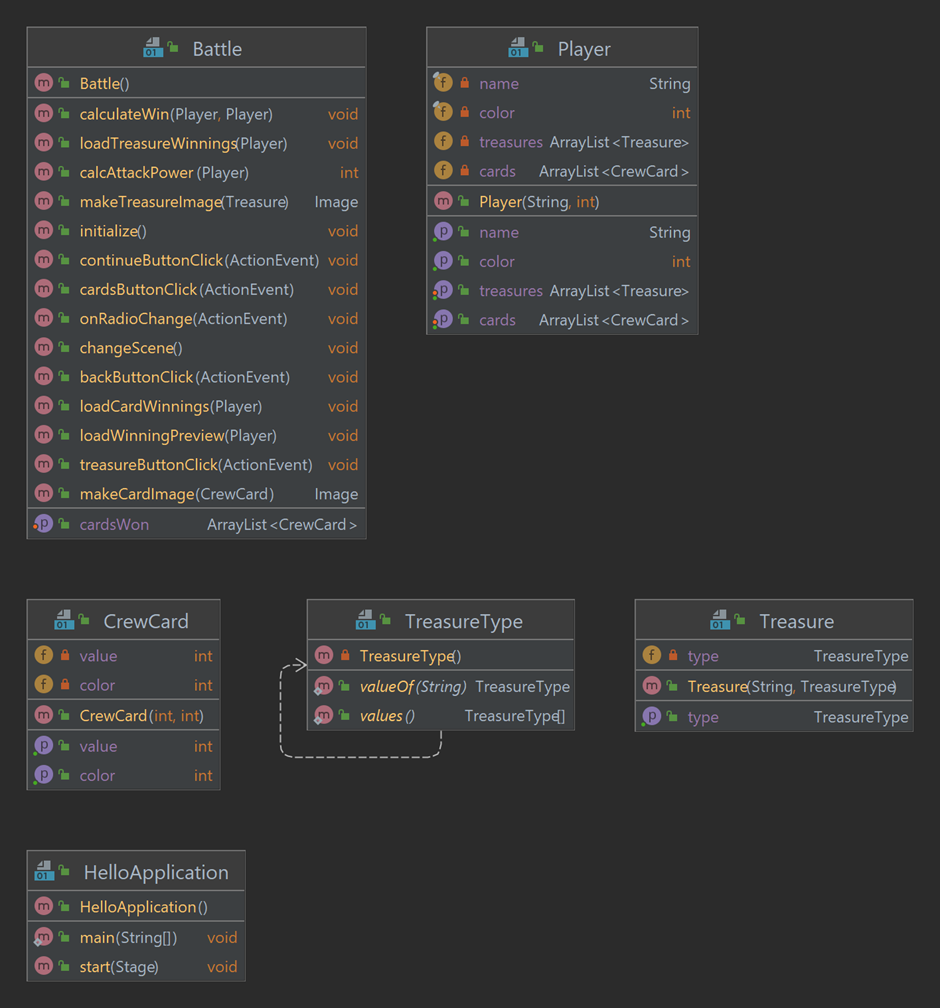
This is the Component Diagram for the Menu:



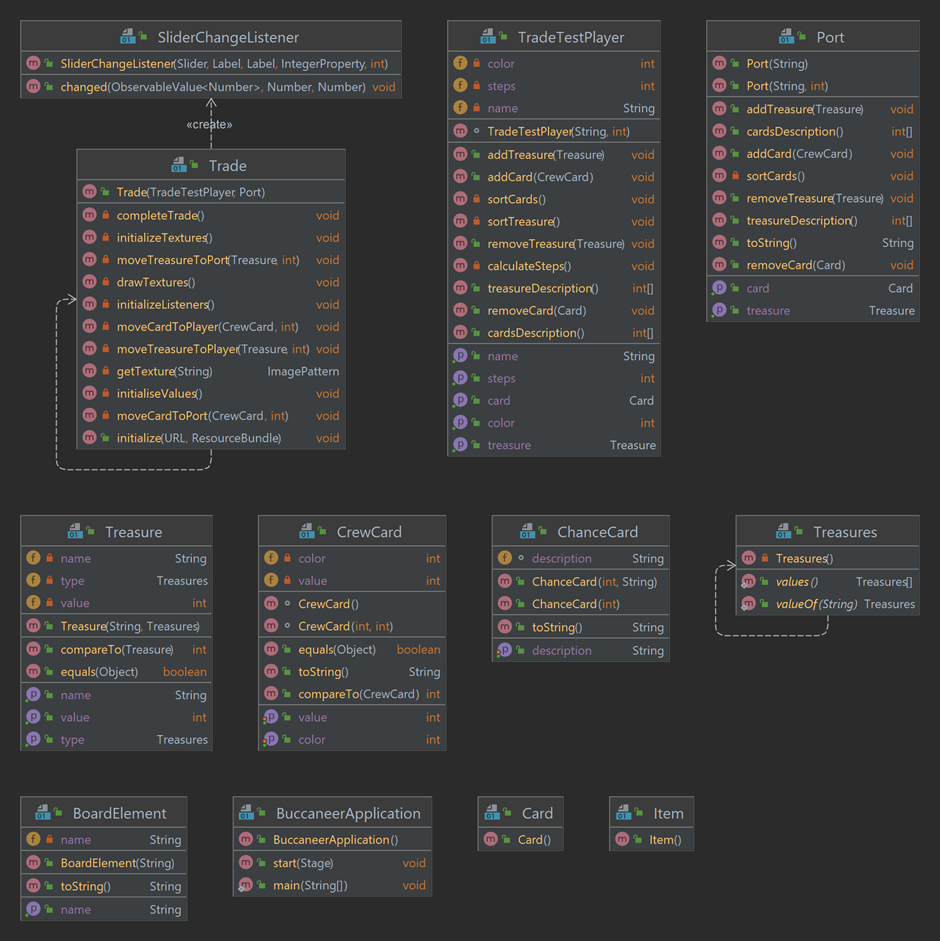
This is the Buccaneer Prototype component diagram:



The battle scene prototype component diagram:



The trading scene component diagram:



# 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.

==========

## 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 Item interface specification

- Type: public

- Extends: nothing

- Public methods: nothing (why)

## 4.14 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.

==========

## 4.15 Trade Interface specification

- Type: public

- Extends: None

- (Why)

- Public methods:

- Trade(Player player, Port port): The constructor for the fx-controller.

==========

## 4.16 BuccaneerApplication interface specification

- Type: public

- Extends: Application

- This is required by JavaFX

- Public methods:

- void start(Stage stage): Starts the JavaFX stuff

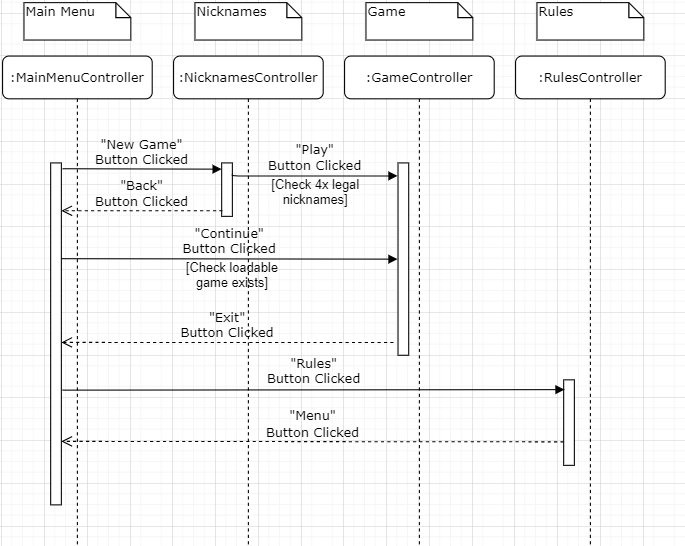
- static void main(String[] args) Main method that starts the entire application.

# 

# 5. DETAILED DESIGN

## 5.1 Sequence diagram

Menus Navigation Sequence Diagram



## 5.2 Significant algorithms

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

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

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.

## 5.3 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.

**REFERENCES**

[1] Software Engineering Group Project: The Main Use Case of the System. SE.G02.UseCaseDoc. 1.0 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 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |