Group Project 03 –

Maintenance Manual

Author:	A. Sadkowski [ols21]
	L. Purnell [lup35]
Config Ref:	SE.GP03.MAN
Date:	11 May 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

1.2 Scope

1.3 Objectives

2. Program Description

2.1 Initialise

The game begins by opening the Main Menu with three options; Start Game, Continue and Exit. Start Game takes the player to the player name input screen, Continue will load the last saved version of the game (if it exists) and Exit will close the program.

2.2 Player name screen

The screen has 4 drop down menus for players to choose their colour, alongside this is a textbox for them to input their names. There are 3 buttons at the bottom of the screen, “Submit Names” will begin the game with the information entered, this will only happen if the names and colour choices are acceptable as per the Test Specification. “Back” will take back to the main menu and “help” will take you to the help screen.

2.3 Help Screen

The help screen has some text in the middle of the screen to provide the player with information, there is a “next” button which will scroll through the different pieces of information that the player might need. The other button is “close” which returns the player to the previous screen they were on when they accessed the help page.

2.4 Game Screen

The game screen will show the current state of the game (start state if new game or saved state if “continue” button pressed)
When the player clicks on a location on the board (Island, Bay or Port) the game will display the relevant information about that location such as the name.
The player can see the statistics panel which shows them information about the treasures stored in their ship, their name, location and bearing. It also has buttons for the player to view their chance cards and their crew cards

2.5 Crew Card Screen

The Crew Card screen displays the current player’s crew cards, it has 6 icons for the different types of crew cards and then a number next to it for the number of that card type there is.

2.6 Chance Card Screen

The Chance Card screen displays the current player’s chance cards, it shows the description of the chance card so that the player can understand what they need to do to get the rewards of the card.

2.7 Battle Screen

If a battle occurs during the game the players will be taken to a battle screen which displays who is attacking and who won the battle, it also says what items have been looted from the loser.

2.8 Trade Screen

When a player enters another player's port they will be met with a trade screen where they can exchange their items for items stored in that port, the game checks if the items have equivalent value and the player can facilitate the trade before completing the transaction.
When a player enters their home port they are met with a similar version of the trade screen which allows them to deposit their treasure into the port, but not take items out.

3. PROGRAM STRUCTURE

List of methods and their information:

Statup

- public static void main(String[] args)
  - Initialises the JavaFX for the game to start.
- public void start(Stage stage) throws Exception
  - This initialises the window of the game by giving it its name and displaying the main menu screen.

SceneControl

- public static void initialise() throws IOException
  - This initialise the stage and the different sceens that can be shown in the game.

ObjectStoreUtil

- public static void writeObject(String path, Object o) throws IOException
  - Writes the Object o to a JSON file at path.
- public static void writeObjectWithBuilder(String path, Object o, GsonBuilder builder) throws IOException
  - Same as writeObject, except takes a GsonBuilder to help in coordination with TypeAdapters provided by Gson
- public static Object readObject(String path, Type taken) throws IOException
  - Reads and returns an object read from the JSON file stored at path
- public static Object readObjectWithBuilder(String path, GsonBuilder builder, Type type) throws IOException
  - Reads and returns an object from the JSON file at path using the provided GsonBuilder
- private static Document loadCardFile()
  - returns the xml file containing the chance cards
- public static NodeList getNodes()
  - Uses the loadCardFile to get the chance card file and then reads the elements by the tag name and returns that list.

MoveAssistant

- private static Tile getTileFromBearing(BearingType bearing, Tile shipPos, int dist)
  - Gets a Tile from a bearing given a movement distance and a ship position
- public static Set<Tile> getValidMoves(Ship ship)
  - Produces a set of the possible valid moves a player can make based on their crew card value and their bearing
- public static Island detectIsland(Ship ship)
  - Gets the surrounding tiles around a player and checks if they are an Island tile, and returns the found Island or null if no Island was found
- public static Ship getOccupiedTile(Tile position, Ship currentShip)
  - returns a ship if the Tile position is occupied by a ship other than currentShip
- public static Set<Tile> getSurroundingTiles(Ship ship)
  - 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.
- public static Set<Tile> getSurroundingSailable(Ship ship)
  - Returns a set of the surrounding tiles which are Islands, this is used when highlighting the surrounding tiles as we only need to highlight the island tiles.
- private static boolean isValidCoordinate(int x, int y)
  - Takes a coordinate and checks that it fits within the board.

BuccaneerImage

- public BuccaneerImage(String s)
  - This constructs a new BucaneerImage, which can store the Url of an image stored in the resource folder
- public String getInputUrl()

    - This returns the Url of the image stored

Battle

- public static Ship startAttack(Ship attacker, Ship defender)
    - This method begins the attack sequence for a ship attacking another ship, it will return the loser of the battle.
- private static int calculateStrength(Ship ship)
    - This calculates the fighting strength of ship and returns that value as an integer
- private static ArrayList<String> awardLoot(Ship winner, Ship loser)
    - this takes loot from the loser and gives it to the winner's ship
- private static CrewCard getLowestValue(ArrayList<CrewCard> crewCards)
    - Takes the Crew Card with the lowest value from the loser's hand and returns it

BattleController

- public void initialize()
    - This creates the battle screen when it is called
- public void populateText(String winnerName, String loserName, ArrayList<String> lootText)
    - This fills in the correct information for the battle screen, such as the player names, who won and the treasure that has been looted
- private void back()
    - This is the method for the back button which returns to the game screen.

BearingType (enum)

- public int getDegrees()
    - Gets the degrees (0 to 360) of that bearing (North = 0, South = 180).
- public static BearingType bearingFromDegrees(int degrees)
    - gets a bearingType from an integer of the degrees of rotation
- BearingType(int degrees)
    - Creates a new bearing with the degrees value attatched to it.
- public static BearingType calculateBearing(Tile from, Tile to)
    - takes in the starting tile and the tile the ship should be facing and then calculates the correct bearing based on the difference in coordinates between the two tiles, it then returns a BearingType

ColorType

- public String getName()
    - returns the name associated with the ColorType
- public String getShipImage()
    - gets the Url connecting the ColorType to it's ship image in the resources folder
- public String getPortImage()
    - gets the Url connecting the ColorType to it's port image in the resources folder
- ColorType(String name, String imageLocation, String portImage)
    - creates a new type of ColorType with a name and file location for the assocated port and ship image

GameController

- public void initialise()
    - initialises the board and game graphics
- private void backButton()
    - sets the back button on the game page to take the player to the player menu screen
- public boolean equals(Object o)
    - Checks if two objects have the same class
- private void trade()
    - Opens the trade screen page for the user to trade with a port
- private void openHelp()
    - Opens the help screen which displays information the user might need to play the game
- private void addTreasureButton()
    - Provides a button which is hidden, to allow the user to add treasure to their port, this is useful for testing the win condition of the game

- private void openCrewCards() throws IOException
    - Opens the crew cards panel to show the crew cards the current player has

GameLogic

- public GameLogic()
    - initialises a new GameLogic object with all of the relevant

4. ALGORITHMS

4.1 MoveAssistant

getValidMoves(Ship ship)

Creates a Set of tiles that a player can move to, it does this by first checking if the player has crew cards, if not, the player can only move one tile. Otherwise, the player can move up to the value of their crew cards.

It then checks if the player is in a port if they are all possible bearings from their bearing – 90 degrees and + 90 degrees are calculated.

From there, it gets each tile from the player up to their maximum movement distance in every possible bearing.

getSurroundingTiles(Ship ship)

Creates a Set of tiles that a player can rotate to by running a nested for loops, both loops run from -1 to +1 and each coordinate is checked to see if it is a sailable tile. Coordinate 0,0 is omitted.

4 THE MAIN DATA AREAS

5 FILES

6 INTERFACES

7 SUGGESTIONS FOR IMPROVEMENTS

7.1 Movement

BUG: Board highlights an erroneous set of tiles

The game has mechanics to highlight possible movements and rotations. Occasionally, the possible movements and rotations are overlapped atop each other, creating a confusing situation for the user. Generally, this bug erratically occurs after specific chance cards that involve movement are activated. Historically, the bug has also happened after battle situations; the bug has not been seen in the post-battle scenario since an earlier stage of development and seems to have been incidentally fixed by reworks of GameLogic. It cannot be considered a major bug, as the highlighting is fixed after the user finishes their turn. To remedy this issue, it would be advisable to investigate when and where the highlighting functions are called in GameLogic to establish whether they are being invoked appropriately.

7.2 Battle

BUG: Entering a possible battle situation causes a crash

This is a critical bug which, unfortunately, first manifested itself during the acceptance testing. After attempting to initiate a battle - either by clicking on a ship directly or attempting to 'jump over' it – the game crashes, citing a null pointer exception. This was extremely frustrating as nobody in the group had ever encountered this issue despite extensive testing. The bug was eventually reproduced after a long session of testing and using the debugging tool. Due to the method to add treasure to a ship sometimes returning a null value, it is possible that Chance Cards 5 and 6 may add null values to the ArrayList of treasure. Consequentially, when a player is awarded loot from a player who has previously picked up Chance Card 5 or 6, it is possible the program will attempt to remove a null value from the loser's treasure ArrayList, causing the exception. There are two possible fixes to this problem. Either add a null check in Battle so these null values are ignored or change the Chance Card methods so that the null values are never inserted into the ArrayList in the first place.

BUG: Player gets prompted to initiate a battle if they move to a tile ahead of the enemy ship

At present, the battle can be initiated by clicking on a ship directly or by 'jumping over' it (i.e., selecting to move to a tile after the one the opponent is occupying). Both the player doing the movement and the player being 'jumped over' are offered the chance to initiate a battle. Unfortunately, at present, these prompts are triggered no matter what movement the user selects – so long as the opponent ship is in range and in the proper direction. If both players refuse the erroneous prompts to battle, the game behaves as expected. However, if one player accepts, the player is taken directly to the other ship, after which the movement highlighting will work erratically. It is likely this issue can be resolved by reworking the order of execution after the game changes state to AfterBattleMove.

7.3 Trading

FEATURE: Players are unable to trade treasure for crew cards at a port they do not own

At present, a player will be unable to trade treasure for crew cards while situated in foreign ports. This feature needs to be implemented via the Trading and Port classes,

7.4 Chance Cards

FEATURE: Several chance cards involving the trading mechanic are not implemented

As a group, we decided to replace a small number of chance cards to reduce complexity. These cards were all linked to the trading system, and by simplifying them we gained more time to work on more critical game systems. We still implemented the Kidd's Chart card, to demonstrate that a player is capable of 'holding onto' cards with different effects.

7.5 Ships

BUG: Ships can hold more than two treasures.

On extremely rare occasions, the player has been observed as possessing more than two treasures. The root cause of this issue is currently unknown, as the bug is very hard to reproduce. It is speculated that the root cause is certain methods accessing the treasure ArrayList directly, rather than through the wrapper functions (which have checks to prevent this). The Battle class and some chance cards access the ships' treasure – so it is quite possible the problem may be in one of these. It is also speculated that one or more of the chance cards may be bugged so that they erroneously add treasure to a ship rather than a port.

7.6 Saving and Loading

BUG: The game can be continued even after a player has won

After a game has been won, the game returns the player to the main menu. From here, they are expected to start a new game, which would start afresh, with a new save file overwriting the previous one. However, if the user instead opts to 'Continue' on the main menu, the completed game is loaded into and somewhat playable still. This issue could be rectified by deleting the save file after a game is completed or adding suitable checks on the 'Continue' button to prevent completed games from being loaded into.

FEATURE: Saving for any additional classes

When new classes are implemented into the program, there is a chance that any properties on that class may be undesirable or impossible to save properly. In these cases, the saving system must be altered to handle these special cases or ignore them outright if they are not needed. In addition, the loading system may need to be altered to handle any additional properties existing in the main game driver class.

8 THINGS TO WATCH FOR WHEN MAKING CHANGES

8.1 **External Libraries and JDK**

The program is currently based on JDK 17. The game has also been verified to work with JDK 18, although support is untested for versions prior to JDK 17.

The solution utilises the JavaFX library in order to draw graphics on the screen to form a graphical user interface (GUI). Gluon provides different versions of the JavaFX library for different operating systems: Windows, Linux and MacOS. The Windows and Linux versions of JavaFX 18.0.1 are currently packed into the project via the /libs directory. Two IntelliJ configurations, one for Windows and one for Linux, are provided in the /.idea folder, which have the appropriate VIM options to launch the GUI. The configuration files and inclusion of OS-appropriate libraries allows for IntelliJ to be set up with ease on Windows and Linux machines, which our coding team used for development. It would be easy enough to get working on MacOS as well though, simply by also including the MacOS version of JavaFX and making a new configuration with corresponding VIM options.

The external library GSon 2.9.0 is included and facilitates the parsing of Java objects into JSON text. This is used heavily in the saving mechanic, which ensures data persistence in the event of crashes or manual program termination.

External library JUnit 4.13.1 is included to facilitate the unit testing, which was a core part of our development and testing phases of the project. Although superseded by JUnit 5, JUnit 4 co-operates with JavaFX differently to it's newer part, and it is only through this older version the unit testing can work properly, without errors. The hamcrest-core-1.3 library is also included and plays a role in enabling unit testing for the solution.

8.2 **JavaFX User Interface**

Streamlining the implementation of JavaFX was a key concern, especially in the early stages of development. The system was designed in such a way so that new screens could be implemented easily, avoiding excessive duplication of code. The SceneControl class is the principal class involved with storing JavaFX scenes and the primary stage. SceneControl also holds static references to all the FXML controllers. By providing universal access to scenes and controllers, it becomes much easier to interface with the UI thread from classes that are not controllers. This allows for better abstraction of code and drastically reduces bloating of the controller classes.

When developing any new screens, the FXML URL should be inputted into an FXML Loader in order to generate a reference to the scene, which can then be stored statically. In the controller's initialize function, access SceneControl and store a reference to the controller instance; this allows other classed to interface with the user interface, using the controller as an intermediary. The methodology to change screens is to change the scene of the Primary Stage in SceneControl to the desired scene, also stored statically in SceneControl.

8.3 **Game Persistence**

The game parses the entire game situation into JSON, which is then stored as a text file. By reading in this text file, the game is able to restore back to its prior state in the event of crashes or the user exiting the game of their own volition

The bulk of the loading/saving mechanism is conducted by the ObjectStoreUtil class. Whenever new classes are introduced that need to be saved, ObjectStoreUtil should be amended as necessary to support the loading and saving of these.

Due to the complex nature of the loading/saving process and the variety of different object types requiring storage, several helper classes exist to assist in parsing data. The savegame package contains a number of ‘adapters’ to facilitate the parsing of certain elements which are incompatible with the standard method in ObjectStoreUtil. Be aware that saving is a complex process and not all objects can be parsed easily with the Gson library. Hence, when expanding upon this solution, implement adapter classes where necessary, akin to those already present, in order to process any new troublesome data types.

The loading of a saved game is a relatively simple process, where each property of the main game driver class ‘GameLogic’ is set to the value loaded from the stored JSON. Some specific values are not saved, therefore the values set for those rely on other processes to set them, such as images, rectangles and buttons which are loaded from FXML when JavaFX is initialised instead.

The saved game state does account for some redundant objects, such as the game board, which will never change over the course of gameplay. The reason this is stored is twofold: first, in case any changes are made to the program structure that would facilitate it being saved, and two, because ignoring it would increase the complexity of both the loading and saving methods because a new board would need to be constructed on load instead of relying on the standard method of setting methods on GameLogic.

9 PHYSICAL LIMITATIONS OF THE PROGRAM

10 REBUILDING AND TESTING

REFERENCES

[1] QA Document SE.QA.10 – Producing the Final Report.

DOCUMENT HISTORY

Version	Issue No.	Date	Changes made to document	Changed by
0.1	N/A	09/05/22	Created the document	ols21
0.2	N/A	10/05/22	Wrote section about future improvements and bug fixes that could be made by a future development team	lup35
0.3	N/A	11/05/22	Wrote section about things future developers on this solution should keep in mind	lup35
0.4	N/A	11/05/22	Improved section about saving and anything to watch out for with the persistence system	jac132
0.5	N/A	11/05/22	Added to section 4	ols21
1.0	N/A	11/05/22	Released document	ols21