Group 14 - Maintenance Manual

Authors: Thomas Wilkinson [tjw21], Jac Lingard [jal74], Jay Staddon [jas130], Hassan Farooq [haf22], Dylan Murphy [dym27], Hancheng Zuo [haz15], Sebastian Sogan [ses35], Kacper Prywata [kap48]
Config Ref: SE.GP14.Maintenance
Date: 05 May 2022
Version: 1
Status: Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. PROGRAM DESCRIPTION

The purpose of the project is to create a fully functioning, computerised version of an old board game called buccaneer using java. The program follows a set of requirements which were designed to make sure that the project would in as much detail as necessary follow the same rules and mechanics as the game it was designed after.

2. PROGRAM STRUCTURE

The program has many different classes and packages in order to be able to have all the functionality needed for the game to work like it does in the board game version. These classes are represented in a design diagram in the design spec [1] in section 5.3 as well as some level as pseudo-code in section 5.2 which describes parts of the program we found problematic. The methods for the program are also all shown under section 4, giving a brief description of what each one does.

3. ALGORITHMS

The main key algorithms were utilised in game loop and the chance cards. An example of this would be the “calculateMove” function within the game loop. Using a switch statement to determine the direction a ship is facing, differing sets of two binary operators along with the crew card count could be passed to the function to calculate the moves on the map grid a ship could make. This was possible as the binary operator would affect the x or y coordinates depending on the direction, to determine possible moves.

Chance Card 6 utilises an algorithm that will find the nearest port based on the direction you are heading. It does this once again with a direction switch which will locate 7 squares forward in the direction you were heading. I picked 7 as all it enables the direction to be estimated without exceeding the board coordinates from treasure island. It then finds the nearest port by iterating through all the islands and using pythagoras’ theorem to locate the nearest dock.

Likewise Chance Card 7 uses a similar pythagoras algorithm to locate the nearest ship. These are the most advanced algorithms used, all others should be pretty trivial to understand. The use of stream functions also made some complex tasks easier to understand, such as sorting items by highest/lowest or filtering by an item with a certain name.

4. MAIN DATA AREAS

The main data areas are all listed in the Design Specification[1].

5. FILES

Everytime the end turn button is pressed the game will write to a file called “save.dat”. This contains GZip compressed json data as the game saves all its key data so its current state can be restored upon restart of the program. If this file is present the game will ask if the save wants to be loaded. This file will be deleted by the program if the player refuses to load the save, the save fails to load or if a player wins the game.

6. INTERFACES

The program we produced doesn’t require any interfaces that a standard desktop computer does not already have. A keyboard is required to input the names at the start of the game and a mouse is required in order to navigate the GUI. The program does not interface with any external components other than a mouse, keyboard and monitor and so the computer running the program only needs ports to deal with those peripherals.

7. POSSIBLE IMPROVEMENTS

When it came to making our program, we were very well organised and managed to produce something that both met and exceeded the requirements given to us by the customer. However, if given the opportunity to go back and add more features or to change how the code was written, there would be a few things that we would consider changing.

The first thing that we would have liked to add is the zoom button we originally planned to have in our game. This was to give users the option to enlarge a certain section of the board and see it more clearly. When it came to coding this we ran into the problem that resizing the map grid was going to be quite tricky and would require us to redesign it. We decided that it would be more simple to remove the feature and not remake the game. With more time however, we would be able to rework the map grid and make the zoom button function as planned.

The next thing we had ambitions to achieve was online play between players. Our code was made using Maven, which we planned to use to allow cross OS between players. Unfortunately, we didn’t have enough time to add this feature due to time needing to be spent on things such as documentation. If we had the opportunity to go back then we would likely add this feature.

The final thing we would have liked to add is the option for players to play with bots. This would allow a user who doesn't have 3 people to play with, to fill the missing gap with an AI that would play the game as any other player would. The reason we didn't include this was due to its complexity, meaning it would require a lot of time which was better spent elsewhere in the project.

8. THINGS TO WATCH FOR WHEN MAKING CHANGES

- Coordinates used in some chance cards are hard coded

Because the position the ship is moved to when executing some of the Chace Cards are hard coded, make sure to change the position if changing the layout of the board. This can be solved by dynamically accessing the coordinates of the island where you want the ship to land.

- CommodityImageView class

In case when more Cards or Treasures are to be added, make sure to get familiar with the CommodityImageView class and name the new images in appropriate convention and add code to implement dependency between the image and the new class.

- Adding new Crew Cards

When adding more crew cards make sure that the new class extends the ChanceCardBase class and that it is created in the following location:
"src/main/java/com/buccaneer/backend/cards/chance".
If the card can be kept by the players, it must implement the ICommodity interface.

- Location of the save file

The location of the file saving the state of the game is currently hard-coded in the root directory of the project. When making changes to its location make sure that all the paths in the SaveData class are updated.

9. PHYSICAL LIMITATIONS

	Windows	Mac
Processors	Intel Pentium 4, Intel Centrino, Intel Xeon, or Intel Core Duo (or compatible) 1.8 GHz minimum	Dual-Core Intel
Operating Systems	Windows XP with Service Pack 3 or Windows Vista Home Premium, Business, Ultimate, or Enterprise (certified for 32-bit editions)	Mac OS X 10.4.10 minimum
Memory	1GB of RAM (2GB recommended)	1 GB of RAM (2 GB recommended)
Disk Space	808 MB of free disk space	808 MB of free disk space
Screen Resolution	1024x768 pixels (1280x800 recommended) with 16-bit video card	1024x768 pixels (1280x800 pixels recommended) with 16-bit video card
Java SE Development Kit (JDK) *	JDK 6 Update 13 minimum (JDK 6 Update 14 recommended).	JDK 5 Update 16 (version 1.5.0_16) minimum (Java for Mac OS X 10.4, Release 7 or Java for Mac OS X 10.5 Update 2 or later)

10. REBUILDING AND TESTING

When rebuilding the system familiarise yourself with the project structure and file/pathway locations, the main source code's pathway is as follows; project structure: buccaneer > src > Buccaneer > src >. In this location you will find 2 further structures: main and test, main includes the project files and further pathways to packages and files that are responsible for creating and running the system, and test includes the JUnit software tests used to test some aspects of the software. main has the following structure: main > java > com > buccaneer > backend, buccaneer, controllers, custom, models.

Backend package includes the chance cards class under the package cards and also includes GameLoop, MapSquare, ObjectLoader and SaveData.java which bring the game together and allow the system to function as a whole, these files are responsible for creating, retrieving, loading and saving all the data which is used when running the game/loading the game via Buccaneer.java in backend > buccaneer.

Controllers package includes: board, input, instructions, menu and the win condition, Custom package includes: ComodityImageView which handles the images for commodities, Models package: includes Commodities package which has classes for holding treasure and crew card data, this package also includes the ship and island class which both extend the TradeBase class responsible for handling data in array lists.

Buccaneer > src > main > resources: includes an image package which consists of the visual states a ship can be in, a custom package which holds all the UI elements for commodities and also some fxml files for UI views.

The test package is laid out as: test> java > buccaneer > backend > BackendTest.java this file contains all the programmatic tests and also comments for tests that are UI based (tests to be done by running the system), all tests follow a notation style of: SE-F-XXX where XXX is replaced by the current test number (filled with 0 if less than 3 digits, E.G SE-F-001 and not SE-F-1) followed by a newline and a comment outlining what the test is testing. Naming conventions for functions use camel case and also should describe as discreetly what the test does, all names start with: "test" followed by test function, E.G. testPortAssignment.

Rebuilding the system should not be difficult unless the main classes and data structures used are updated, the system has been split into multiple packages so it is easy to navigate and pinpoint where new classes or new code can go, if existing functions are rebuild to add or remove parameters then either a new constructor should be created or rest of the uses cases for that constructor needs to be updated and correct parameters need

to be passed. Any code to be added or code issues raised should have comments with the #TODO tag in order to allow easier navigation and access to prototype/draft code. No classes should be removed unless appropriately implemented elsewhere to prevent build errors or loss of functionality.

Tests refer to two other documents: Test Specification (found in buccaneer > docs > Test Specification[3]) and Test Reporting (found in buccaneer > docs > Test Reporting[2]). Test specification includes the earlier notation style in a table with the columns: Test Ref which has the test number (in the form of: SE-F-XXX), Requirement being tested, Test Content, Input, Output and Pass Criteria. When a test is to be added, add a new row to the end of the document, increment the Test Reference and fill all sections to the fullest with clear goals and pass criterias. Once new tests are written you can start drafting or prototyping these tests in BackendTest.java, when new problem is discovered, tests should be added at the bottom of the file with reference number incremented in the previously discussed format, tests should pass individually then also as a whole to expose any threading issues or shared data related issues, it is only considered a pass if they all pass together and not only individually. Atlast the results should be reported in Project Test Report where Test Number, Test Functionality, Test Result and in case of a test fail then an Explanation of Failure should be completed. If you wish to add proof for UI tests there is a table below where you can input the Test reference, screenshots and an explanation of what you tested or what is happening in the screen shots, but this is not necessary. To run tests use the testing documents and code comments to refer to Test Reference in order to run the test desired.

Documents are in standard form and use Microsoft Word so there is no need for Latex or form converting, unless it is a personal preference then you are able to run the Word documents through a Latex converter, but for ease of use and familiarisation it is better kept to a default of Word.

11. REFERENCES

[1] SE.GP14.DesignSpec - Design Specification
[2] SE.GP14.TestReport - Project Test Report
[3] SE.GP14.TestSpect - Testing Specification

12. DOCUMENT CHANGE HISTORY

Version	Issue No.	Date	Changes made to document	Changed by
0.1	N/A	05/05/2022	New Document	TJW21
0.2	N/A	10/05/2022	Completed section 7	JAS130
0.3	N/A	10/05/2022	Completed sections 3 and 5	SES35
0.4	N/A	10/05/2022	Completed section 9	HAZ15
0.5	N/A	11/05/2022	Completed section 6	DYM27
0.6	N/A	11/05/2022	Completed section 10	HAF22
0.7	N/A	11/05/2022	Completed section 1	JAL74
0.8	N/A	11/05/2022	Completed section 4	JAS130
0.9	N/A	11/05/2022	Completed section 8	KAP48
1	N/A	11/05/2022	Completed section 2	JAL74