Software Engineering Group 17
The Project Maintenance Manual

Author: deo4, adl12, aaw13, wgf

Config Ref: SE.G17.MM
Date: 02/05/2017
Version: 1.0
Status: Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

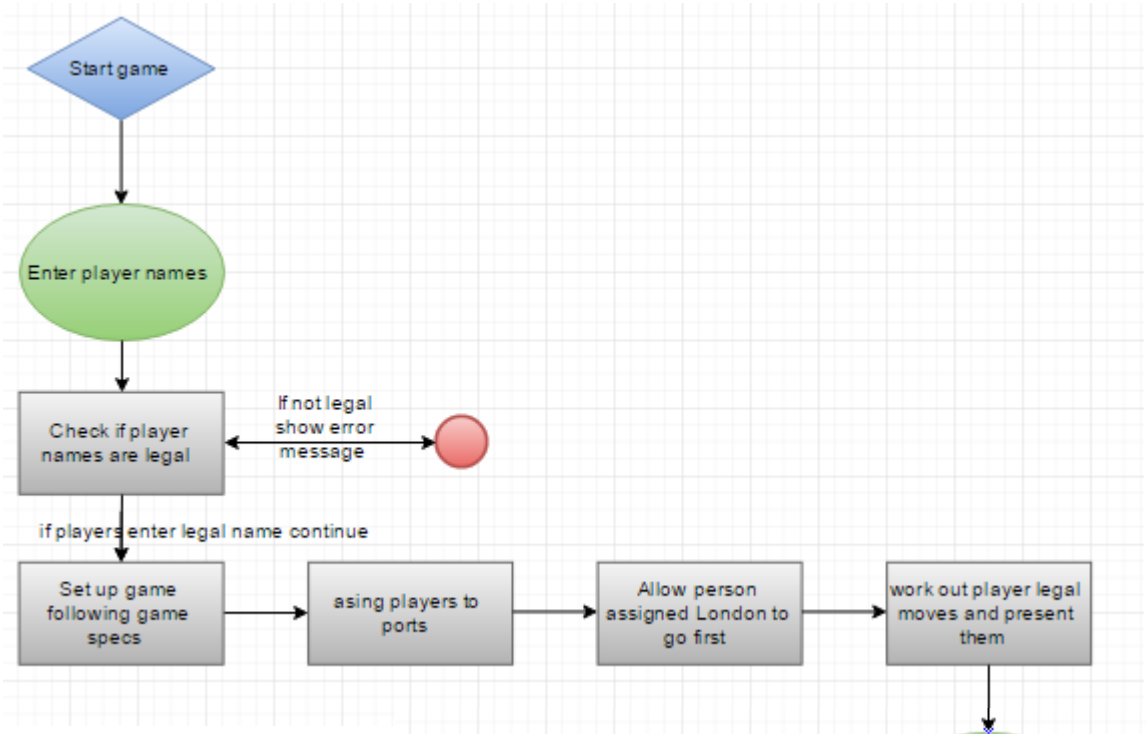
CONTENTS

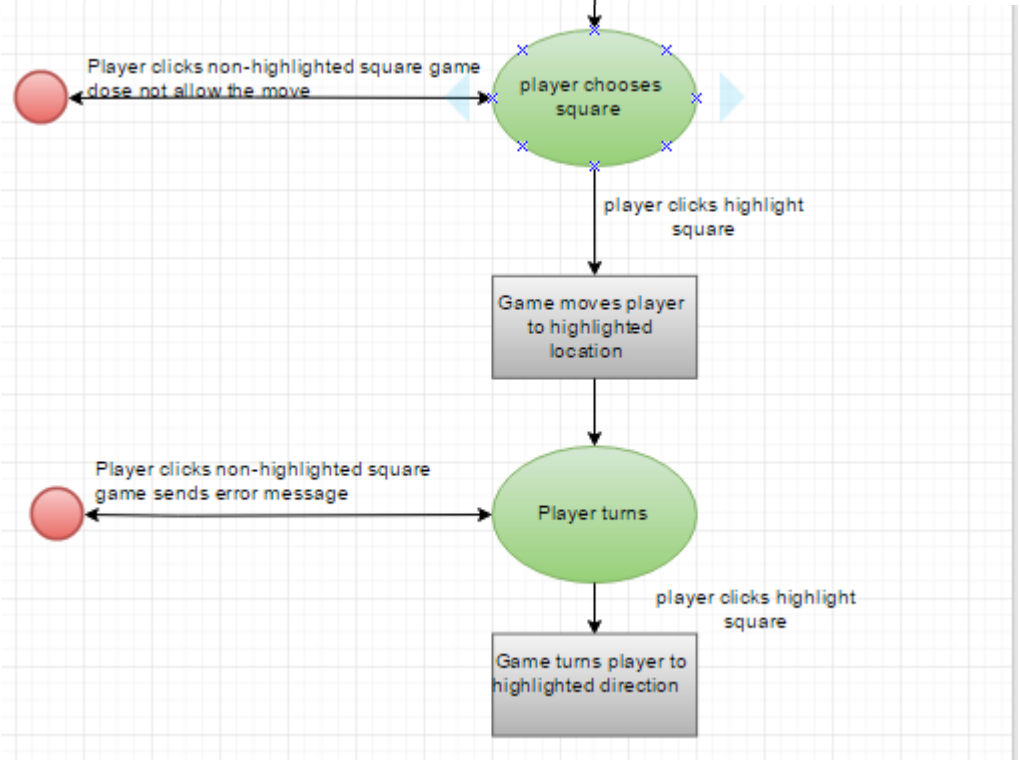
1. PROGRAM DESCRIPTION

This program is adapted from the board game Buccaneer, the program needed to allow to have four people to play the game the same time on the same computer with the computer prompting each player to make moves when appropriate this is handled in the main game function by using an array to store the four players and this allows them to enter their names on the start screen which is a GUI .The program will need to keep track of what the player is holding in terms of cards, treasure and what position on the board position the player is currently located this is done in the player method. The game needs to also only allow a player to have two pieces of treasure on the ship at a time while displaying the state of the game to the players on screen this is done using arrays. The program needs to set up the start position of the game, assigning home ports to players randomly and dealing out crew cards out to players randomly along with applying crew cards and treasure to ports as appropriate. The program then needs to detect a player reaching Treasure Island, Flat Island or the Anchor Bay, then take any appropriate action this is done using a player tracker. When a player choses to take a chance card at Treasure Island, the program will implement the consequences of the chance card selected this is done using a document that stores the chance cards and what they do. The program should be able to manage and exchange of treasure at the ports according to the rules of the game, the game does this by checking where a ship is after its move if it's in the player's home port it deposits the treasure, otherwise it shows a trading screen. The game Should be able to indicate legal moves to a player that is about to move from the rules of the game this is done by using the position helper. The game will implement the attacking rule when to ships cross this is done by checking if the squares it's moving through has a player if yes asks them if they want to attack, if the player says yes then it changes the position there moving to then the other player's position, it then moves the player and checks the position they've moved to if that has another player if there is then it does an attack if not then the move is done. Finally, the game should detect when a player has won once it has 20 or more points worth of treasure at their home port this is done with the get score class in the player method

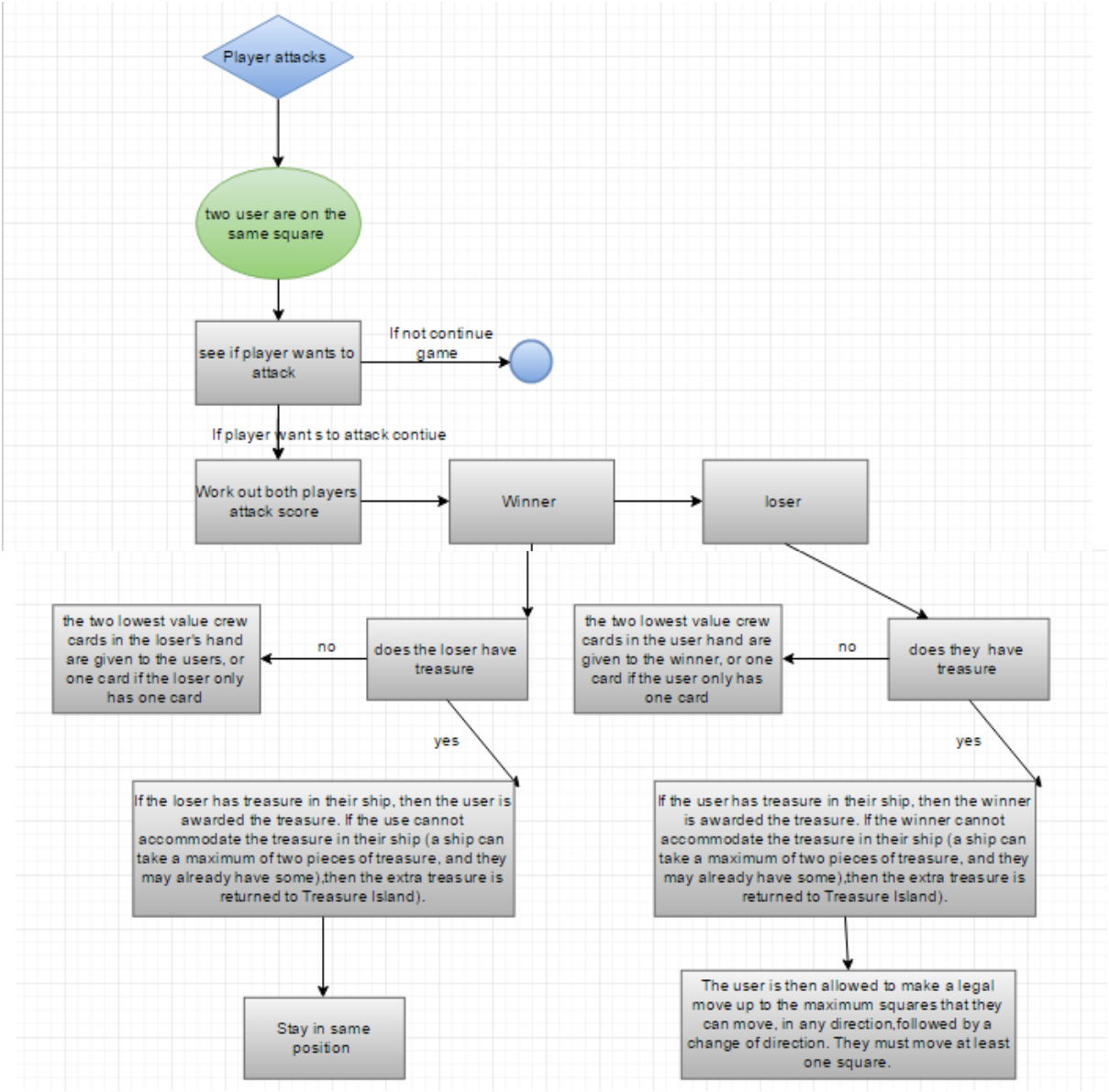
2. PROGRAM STRUCTURE

2.1 **flow charts**



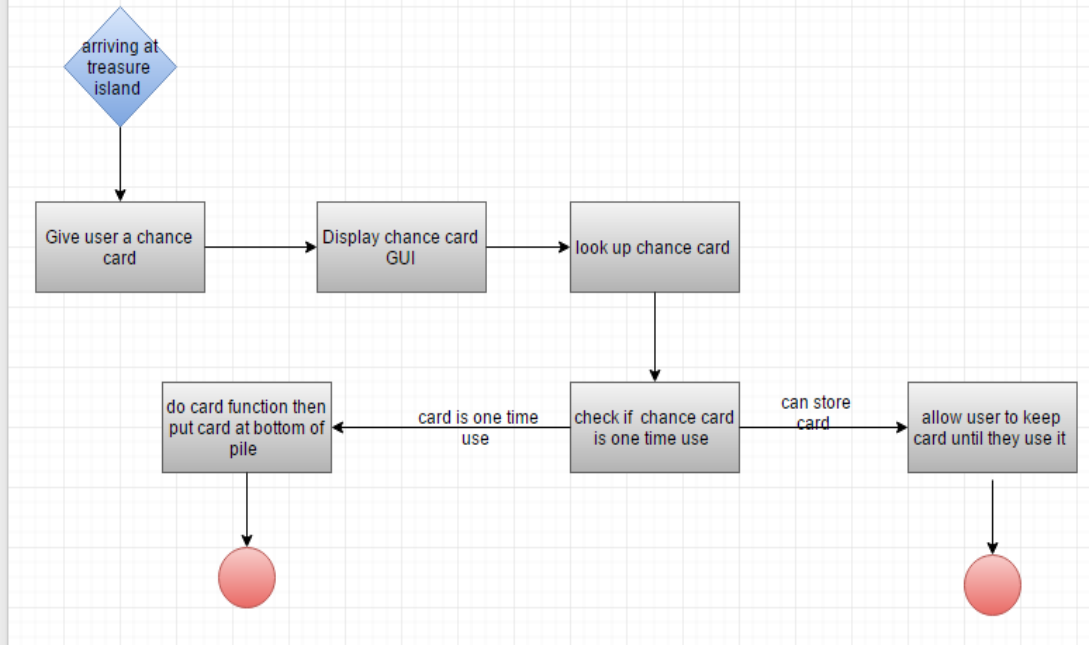


The program structure mainly using flow charts, below is a flowchart on moving and turning that shows the start of the game where the users have to enter their names then the game checks if the names entered by the user are legal if there are not a pop up will come up saying one of the names are not legal, then you enter the names again after legal names are entered the game will start to set up following the specs given it will first assign the players to ports after this it will check what user has been assigned the London port as they are first to move the game will present available moves with showing highlighted squares so if a player clicks a non-highlighted square the game does nothing, but when the user does lick a highlights square it will move to that location on the board. Then the player gets the option to turn and will have the available turns highlighted if they chose a non-highlighted the game presents an error message but when the play clicks a highlighted square they turn in that direction



This is a flowchart shows attacking in the game. So, first two of the users in the game are on the same square the game then prompts the attractor seeing if they want to attack if they don't the game will continue but if they do the game will then work out both of the players attack score, then it will check if the attacker wins or loses if the attacker wins the game then checks if the loser has any treasure if they don't the two lowest value crew cards will be given to the winner, if the loser has only one crew card then only that will be given to the winner, but then if the loser has treasure on the ship then the winner is awarded the treasure provided they have another room on their ship if they cannot the extra treasure is returned to treasure island after this is done the attacker stays in the same position, the loser also goes through this sequence but doing the opposite but after all that is done the user is then allowed to make a legal move up to the maximum squares that are available they must at least move at least one square.

This is a flow chart showing what happens when you go to treasure island you have a prompt that shows you a chance card it then looks up what the chance card does it then checks if the card is a onetime use or if the user can store it and use it later.



2.2 **pseudo code**

Inputting user name
Start-up game
Enter names
Check if names are legal
Load up game board

Attacking
Two player on the same game square
Attacking screen comes up
compare two players attack score
pick winner
Winner gets reward

Movement
work out all player's available moves
highlight all available moves
allow player to pick movement square

Chace card
user arrives at treasure island
draws a card
chance card executed or stored

2.3 **reference to design specification**

there is more information on the program structure in the design specification in sections:

- 2.2 – 2.4
- 3.1
- 4.1 – 4.33
- 5.1 – 5-2

3. ALGORITHMS

The algorithms used in this project can be found in the design specification in sections 5.3 and 5.4.

4. THE MAIN DATA AREAS

4.1 Card Decks

Cards within the game that have not currently been assigned (they are still in a deck, rather than in a player's hand etc.) are stored within a CardDeck. The card deck conforms to FIFO, so that cards that have recently been added to the deck will not be added until the rest of the deck has been cycled. The card deck has two different ways of importing cards into its deck. For chanceCards, a CSV File is used, which stores the ID and the text of the card. For crewCards, the program iterates through all the different colours and values for chance cards, and then ensures that the correct amount of cards for each are present. The Card deck also has a shuffle function that can sort the cards in a random manner

4.2 GameBoard and GameSquares

The game board keeps track of all of the elements on the game board, including ships, ports, bays/creeks and islands. This way, we can query the location of an element (e.g. a ship) and see if it is adjacent or on another element. The gameboard is a 20 by 20 array of game squares. A GameSquare is an ArrayList of GameObjects (such as ships) that also has a respective location and a board assigned to it.

4.3 Enum Data

Some classes require some types of data that are not present in standard java libraries. As a result, we can use enums in the enumData Package to accommodate for this. CardColor stores the colour of a crewCard, direction stores 8 cardinal points of a compass, in order to be used to keep track of what direction a ship is facing. Game state is used to keep track of what the state the current player is in at that moment of time. TreasureType is used to store a type of treasure, and also has functions to get the value for the type of treasure, as well as getting the name.

4.4 Position

The position of a ship is stored as a Position object - all objects within a game square should have one

5. FILES

5.1 Images

The Program uses various image file types to display the crew cards, chance cards, player's ships and the full game board which are located in the **Buccaneer/**res/images.

5.2 CSVs

The Program uses CSVs to store the chance cards' information and the locations of the ship images. They are located at **Buccaneer/**res/data.

5.3 Sound

The Program uses a sound file as music for the Game. This is located at **Buccaneer/**res/sound.

5.4 Fonts

The Program uses a custom, pirate themed font for all the text. This is located at **Buccaneer/**res/font.

6. INTERFACES

Our program does not have interfaces; the entire program just requires just a PC, qwerty keyboard and mouse.

7. SUGGESTIONS FOR IMPROVEMENTS

- JUnit Tests - out JUnit tests do not provide full coverage of all of the functionality within the application. This could be expanded further.
- The program access files using the Java IO file system, rather than getting them as an input stream. This would enable us to store the program in a JAR in the future
- Implement 3D Graphics - currently the whole program is based on images - in the future, this could be improved to use 3D graphics, although a whole rewrite of the program may need to be done in order to do so.
- Currently dialog boxes display in different windows. In the future, a stack pane could be used to display messages over the current game, therefore reducing the amount of windows on screen to 1.
- Currently the game has to be larger than a certain resolution, due to the size of the images used. In the future, the game could scale the game to match other resolutions
- The only sound in the game is the BGM, which in itself is not much.
- The attacking system could be rewritten
- The Trading system could be rewritten.

8. THINGS TO WATCH FOR WHEN MAKING CHANGES

- The locations of Ports, Islands and Bays are hard coded. Care should be taken when changing these.
- Island in Position Helper does not use the position of the islands, but rather the theoretical location of them.
- The size of the board is hardcoded, and care should be taken when changing it - this includes images, and the functions to translate a position
- The size of the images and each grid slot are hard coded - if the images change, the appropriate GUI methods should be changed to
- Adding another player is ill advised, as a large amount of code relies on the player count being 4
- The board is inverted - the bottom left is position 0,0 where normally the top left is. They are helper functions in position Helper to deal with this

9. PHYSICAL LIMITATIONS OF THE PROGRAM

The application should use no more than 1GB of memory at a time, although during normal operation the amount of memory used normally fluctuates around half of this.
The resulting files, after compilation take around 30Mb
Although the application uses very little CPU Power, less powerful computers may take longer to load the program and to perform certain operations. During testing, turns always took considerably less than 1 second to complete, but this may change depending on the environment performed in.
The screen resolution is recommended to be 1920 by 1080 pixels. The application may support some smaller screens, but below a certain resolution, elements of the game may go off the edges of the screen.

10. REBUILDING AND TESTING

Building can be carried out in one of two ways:

- Ant Build File
- Standard Java Application Build

Building with ANT

Note that the program does not officially support building Jar File artifacts due to the way that resources are accessed - Whilst some resources are access via input streams (which can be accessed from within a JAR file), others are not, and are access as external files.

To build the project with the Ant Build File, select build.xml as the ant build file. Normally you can change to the project root directory (not the source directory) and run ant. To build the project, use ant compile.
Ant supports automated running of JUnit tests - to test the program, run ant test.
Ant also supports auto generation of JavaDoc Documentation. To generate this, run ant doc, and open the resulting index.html file within the docs folder.

If you wish to compile, test and generate docs, you can use the main build target to do so. This will also run the dist target, which generates a .jar artifact. However, this is not likely to work

The Ant build system can be integrated with CI systems, and it is recommended to do so while under active development. Configure your CI system to run ant test. Many CI systems (such as travis ci) already do this anyway, so they should not be too much to configure.

Building as a Java Application

Building as a java application is slightly more involved, but may be quicker; The following steps are for the IntelliJ IDEA IDE, but steps should be similar for other IDE's.

Download the project, or clone it from github. If the project is in an archive file, then unzip it, and store it in an appropriate directory. Open IDEA, and select Import Project - find the appropriate directory and select it. On the import project screen, select create project from existing sources, and press next. Accept the default Project name and Location, and press next. On the next screen, untick the test directory that is marked "test", and then click next. Ensure that the lib folder is selected as a Library. Click next, select Buccaneer as a module, and then click on next. If asked to use Buccaneer. iml as a module file, click Reuse. Select java 1.8 as a project SDK (ensure that Java 1.8 JDK is installed and configured with idea), and then click on next, and then finish. Open the Project window, open the src folder and right click on src/buccaneer/main/GameApp.java, and select Run. The application should build, and then run.

To run JUnit test, open the src/test folder, and right click on the test you want to run, and click on test. The test will run, and the status of the test should be displayed.

11. REFERENCES

[1] Software Engineering Group Projects: General Documentation Standards. C. J. Price, N. W. Hardy, B.P. Tiddeman. SE.QA.03. 1.8 Release

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
0.1	N/A	02/05/2017	Layout of document and start of program description needs how it does	Deo4
0.2	N/A	04/05/2017	Program structure and algorithms almost done	Deo4
0.3	N/A	05/05/2017	Explained flow charts in the program structure section and explained where to find information on algorithms started main data areas section	Deo4
0.4	N/A	08/05/2017	Suggested improvements filled out the rebuilding and testing section	Aaw13 Adl12
0.4	N/A	08/05/2017	Completed how the program does mentioned things in the program description and completed physical limitations	Deo4
1.0	N/A	09/05/2017	Completed document. Filling out all remaining parts	Adl12 Aaw13