Test Plan

for

Force Connector

Version 1.0 approved

Prepared by

Payton Long, Casey Sanchez, Bethany Roberts, Brandon Shelton & Cole Dalfonso

Skywalker LLC.

03.22.2021

1.1 Contents

1.2 Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Draft Type and Number	Full Name	Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded.	00/00/00

1.3 Introduction

Force Connector is a program developed by Team Skywalker with the objective of running a game of connect four with a graphical user interface (GUI) that the user can interact with. The program follows the basic rules of a connect four game with alternating turns between the user and an AI. Force connector has also implemented additional features including a difficulty selection setting, an animated message upon winning or losing and a counter that records the users winning streak.

1.3.1 Test Plan Objectives

The purpose of this test plan is to validate that the program has upheld the specifications provided in the Software Requirement Specification (SRS) developed by Team Skywalker and approved by client Jason Diaz. The test plan shall also verify the code has met these specifications by laying out the test strategies for testing the code as well as the functions that will be tested by providing expected behavior for key functions along with their results.

2.1 Test Strategy

2.1.1 System Test

System Testing will verify that all integrated components and the program as a whole conforms to the behavior expected and outlined in the SRS document.

2.1.2 User Acceptance Test

User Acceptance Testing will be used to ensure all specified user requirements have been met and that the user can successfully run the software, change difficulty options, begin a game, place pieces, complete a game, and replay subsequent games.

2.1.3 Negative Test

Negative testing verifies that the program will perform as expected when invalid input is entered.

Example: **Full Column Selection**

Condition: The user has selected a full column to place a piece in on their turn.

Input: The user will click the arrow above a column with 6 other pieces already filling it.

Expected Behavior: The system will not allow the piece to be placed in this column but will allow the user to select another column in which to place their piece.

3.1 Environment Requirements

3.1.1 Environment 1

The software/game will need to be tested and passed for compatibility with PC's running on the current version of Windows 10 dating back to the legacy versions from release.

3.1.2 Environment 2

The software/game will need to be tested and passed for compatibility with current and some legacy versions of systems running on Linux based operating systems. (Ubuntu, Linux Mint, Debian etc..)

3.1.3 Environment 3

The software/game will need to be tested and passed for compatibility with Apple systems running on the current versioning of Mac OS Big Sur. (This covers the Macbook lineup, Mac Pro, and iMac systems)

4.1 Functions To Be Tested

4.1.1 Functions in the GameBoard Class

4.1.1.1 Check for Winning Move

Condition: The player is presented with 7 buttons to choose from to make a winning move. The AI also has 7 places it can put a piece down.

Input: The player or AI makes a winning move.

Expected Behavior:

- When a player piece is added to the board, the game board class will update the game board.
- After the game board is updated, the updated board will be checked by the isWinningMove function to see if the move that was last played was a winning move.
- If it is a winning move, the board will show a graphic saying who won and present options to return to the main menu or play again.

4.1.1.2 Initialize an Empty Board

Condition: The player is presented with an option to play the game again.

Input: Nothing

Expected Behavior:

- This function sets all the values of the board to '0'
- If the selected difficulty was hard, then the ai will place a random move on the board.

4.1.1.3 Add a Player Piece to the Board

Condition: The player is presented with 7 buttons to put their piece on the board.

Input: Column number

Expected Behavior:

- When the player chooses a column number, this function will see which row to place the piece based on the available places.
- The player piece should be put on the largest possible row that isn't occupied.

4.1.1.4 Assign the AI's move

Condition: The AI shall put down a piece after or before the players turn, depending on difficulty.

Input: N/A

Expected Behavior:

- The AI will place a piece on a random column and on the next largest row of that column.
- If it is a winning move (meaning there are 4 pieces in a row), the game will end.

4.1.2 Functions in the GameManager Class

4.1.2.1 Start the Game

Condition: The player shall start the application.

Input: N/A

Expected Behavior:

- A main menu will pop up in a JFrame.

4.1.3 Functions in the GameInterface Class

4.1.3.1 loadImages()

Condition: A gameInterface object is constructed.

Input: Local vars background, empireSymbol, rebellionSymbol, and blankSpace.

Expected Behavior:

- background, empireSymbol, rebellionSymbol, and blankSpace should have their respective files loaded and have new ImageIcon assigned to them.
- If any image loading fails, likely because the image is not in the correct directory, then the system should throw an IOException and print that it was not able to open the file.

4.1.3.2 addPieceToBoard(int col, int row, char color)

Condition: Called from Gameboard after a move is determined to be valid.

Input: int col (1-7), int row (1-6), char color (r, b)

Expected Behavior:

- An image representing the game piece should be displayed.
- The image should be at the correct row and col with (1,1) corresponding with the top left corner of the board.
- The image should either be empireSymbol if b is color, or rebellionSymbol if r is color.
- Invalid parameters are not checked for because it is assumed that the checking happens in Gameboard before the function is called.

4.1.3.1 initGameInterfaceBoard()

Condition: A gameInterface object is constructed.

Input: N/A

Expected Behavior:

- addPieceToBoard(int col, int row, char color) is called for every valid (col,row) combination with color = '0'. This should make every location for a piece appear blank.
- All of the column buttons for the user placing pieces should become enabled.

4.1.3.1 displayPlayerWins()

Condition: The user plays a winning move

Input: N/A

Expected Behavior:

- winFrame is set to visible. The win message is displayed
- The column placement buttons are disabled.
- Both the user and the AI should not be able to place a piece

4.1.3.1 displayPlayerLoses()

Condition: The AI plays a winning move

Input: N/A

Expected Behavior:

- lossFrame is set to visible. The loss message is displayed
- The column placement buttons are disabled.
- Both the user and the AI should not be able to place a piece.

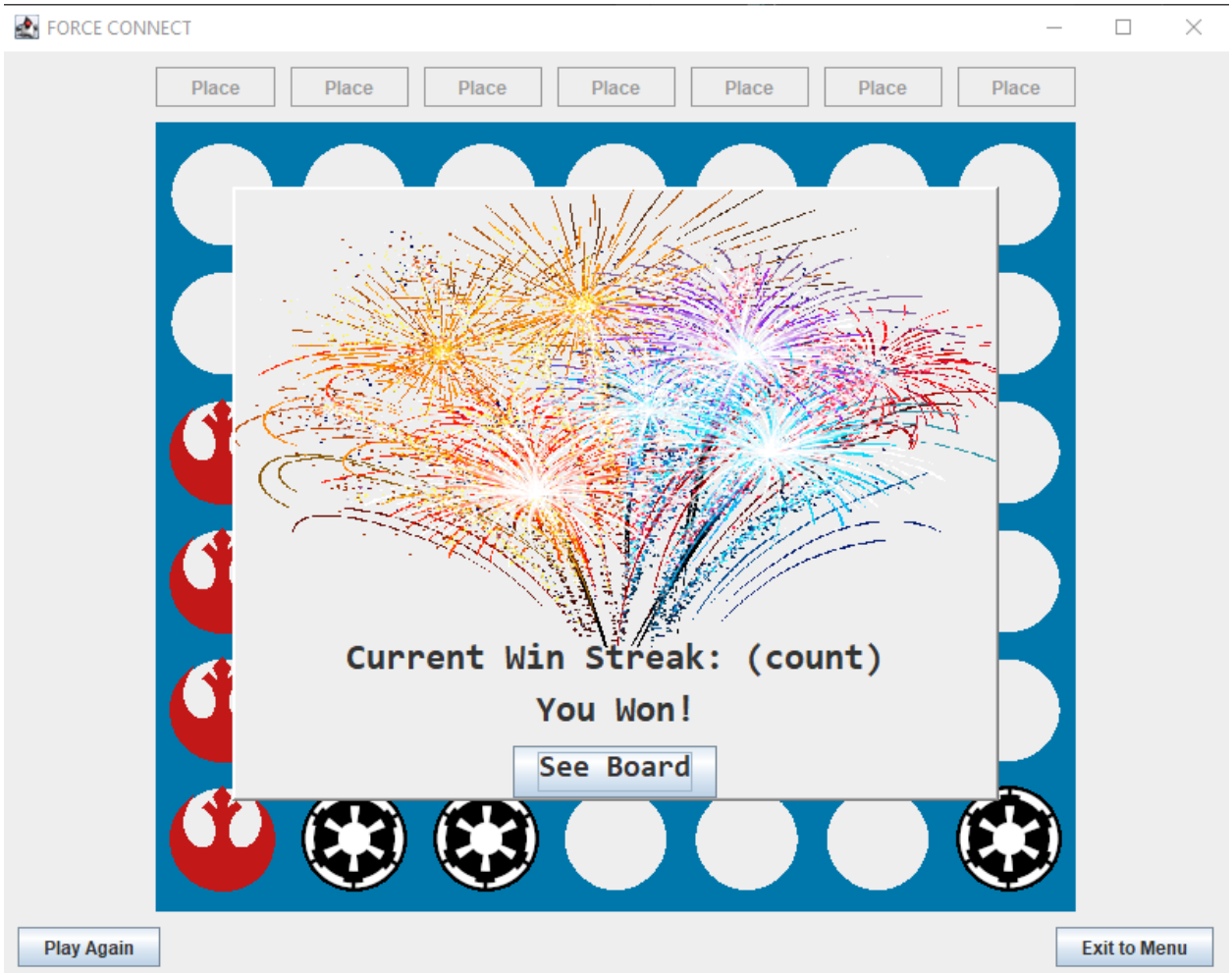

Figure 1. According to the SRS, there should be a trophy in the upper corner that indicates the amount of wins the player obtained.

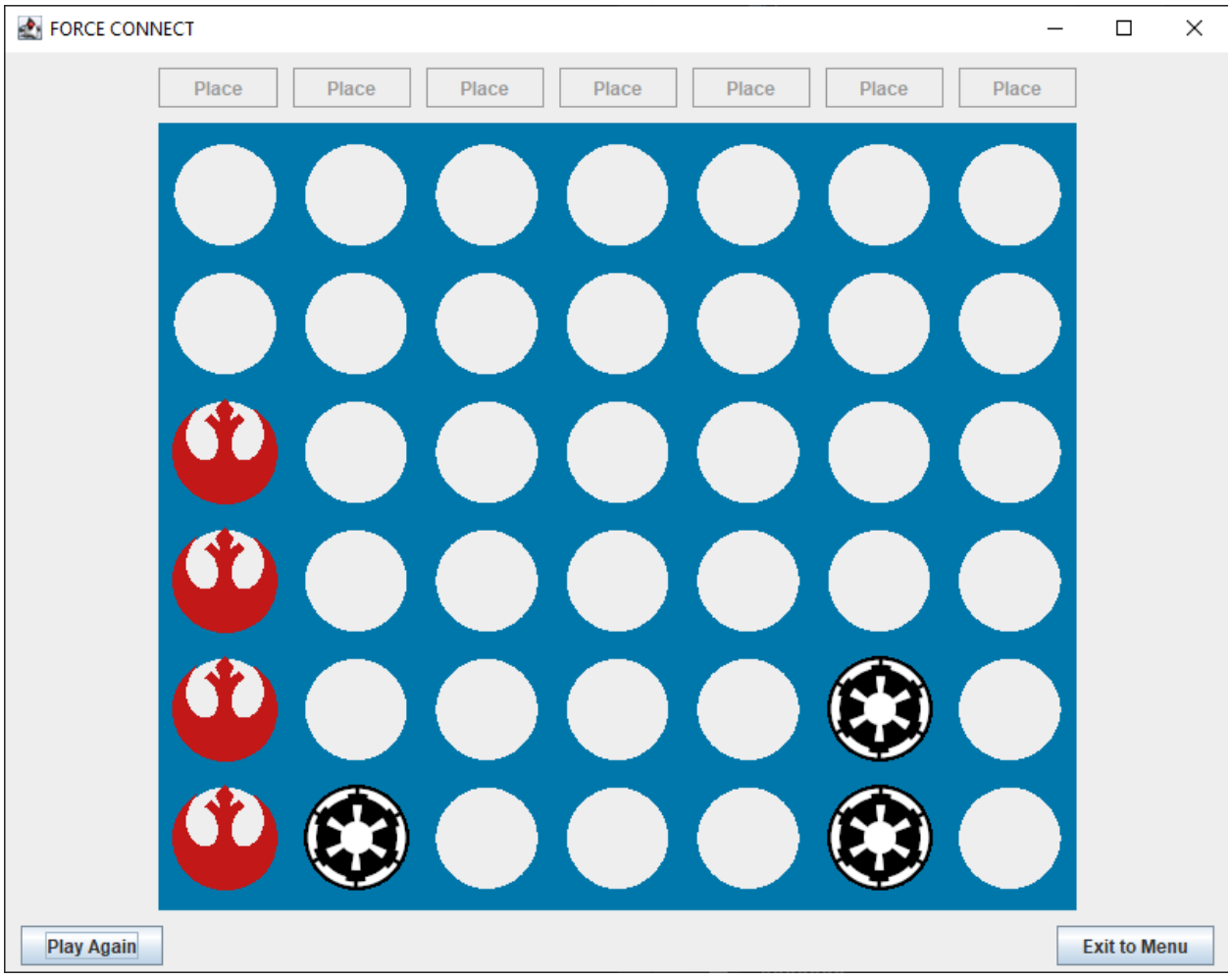

Figure 2. Our SRS states that black should be the first player to place a piece. The piece colors are instead tied to if it is the AI (black) or User (red) who places the piece.

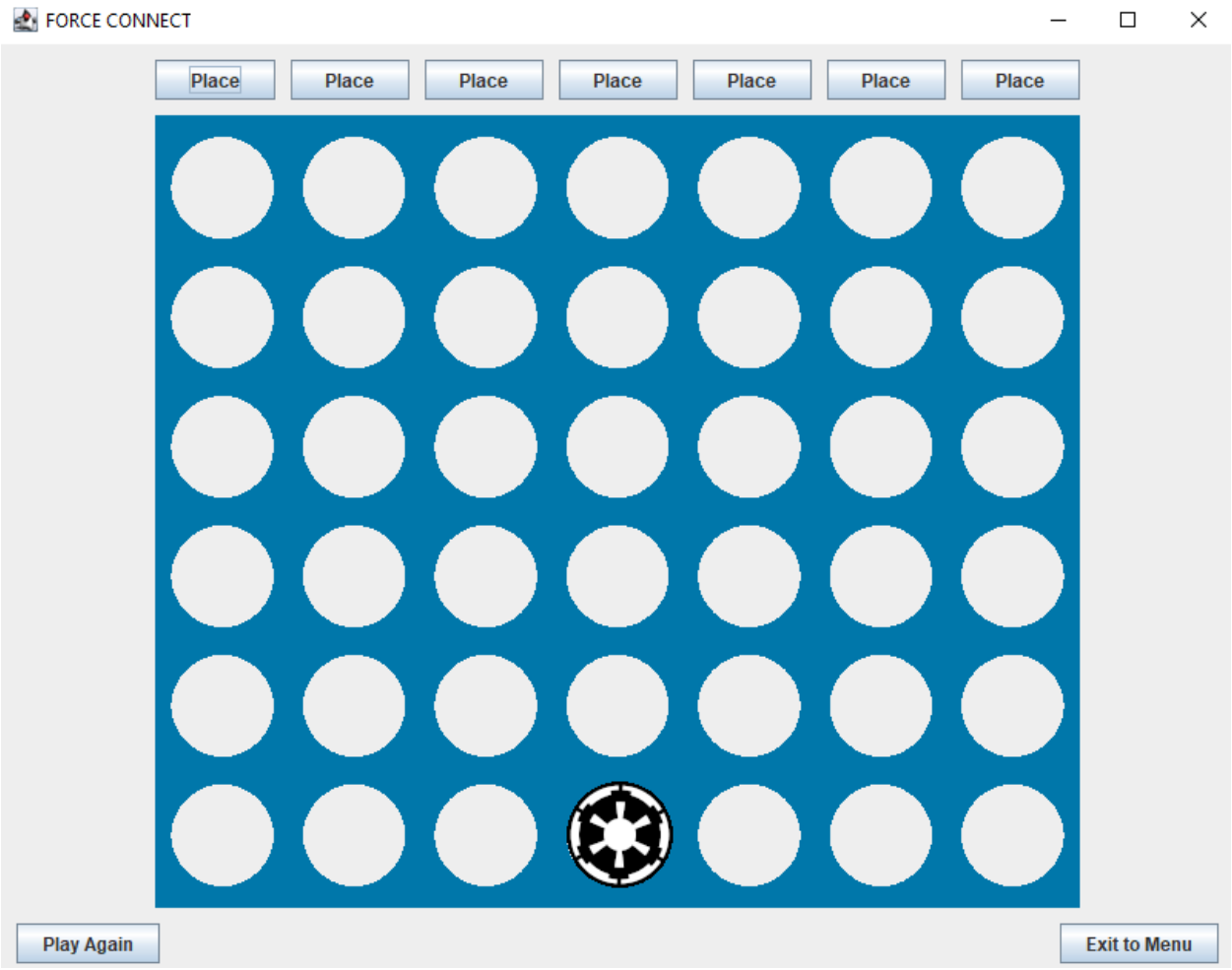

Figure 3. Our implementation of difficulty involves the User going first on “easy” and the AI going first on “hard”. Therefore, sometimes black goes first and sometimes red goes first.