**TIC TAC TOE (v1.3)**

Use Case, UML, Sequence Diagrams

A screenshot of a computer

Description automatically generated with medium confidence**UML Diagram**

**Graphical user interface

Description automatically generated with medium confidence**

**Use Case Diagram**

A picture containing qr code

Description automatically generated

**Diagram, application

Description automatically generated**

A picture containing graphical user interface

Description automatically generated

Timeline

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

System Test Cases

**Test Case 1**

Purpose: Verify that menu buttons are working

Setup: Obtain TicTacToe v1.3 (latest version). Follow directions below

Test Data:

|  |  |  |
| --- | --- | --- |
| **Action** | **Input** | **Output** |
| Launch Game |  | The game’s main menu should appear with four button options to choose from |
| Check that the ‘Single Player’ button works | Click the ‘Single Player’ button | Verify that the game transitions to the appropriate startup menu scene where a player can choose the difficulty |
| Select back from the startup menu | Click the ‘back’ button | The game transitions back to the main menu with four options |

Notes:

1. Do not check for colors, positions, labels, spelling.
2. The ‘Quit’ button is not checked.
3. The remaining buttons (multiplayer and settings) can be checked by applying the concept of this same system test

**Test Case 2**

Purpose: Verify that game start up elements work and are applied to the game

Setup: Obtain TicTacToe v1.3 (latest version). Follow directions below

Test Data:

|  |  |  |
| --- | --- | --- |
| **Action** | **Input** | **Output** |
| Select Single Player | Click the ‘Single Player’ Button | The game shows the startup menu scene where a player can customize their game |
| Check that selecting the difficulty functions correctly | Click and toggle between the basic and advanced difficulty | Verify that only one option can be checked at a time  When the game starts, the difficulty of the computer player should be applied and function accordingly |
| Check that the board size and win condition functions correctly | Enter three numbers indicating the desired width and height of the board, as well as the number in a row for the win condition | When the game starts, these conditions should be applied and function accordingly |
| Check that selecting either X or O functions correctly | Click and toggle between the X and O selection | Verify that only one option can be checked at a time  When the game starts, either X or O should show on the gird first depending on what was selected |

Notes:

1. The multiplayer startup menu can be checked by applying the concept of this same system test, minus the difficulty.
2. Testing the ‘Start Game’ button is implied when testing to make sure selected conditions are applied to the game.
3. Do not check for colors, positions, labels, spelling.

**Test Case 3**

Purpose: Verify that changing the game themes in settings functions

Setup: Obtain TicTacToe v1.3 (latest version). Follow directions below

Test Data:

|  |  |  |
| --- | --- | --- |
| **Action** | **Input** | **Output** |
| Select Settings | Click the ‘Settings’ Button | The game shows the settings menu where the theme can be changed from three buttons |
| Select Theme1 | Click the ‘Theme1’ Button | By clicking the button, verify that theme is automatically changes and is applied to the game |
| Select Original | Click the ‘Original’ Button | By clicking the button, verify that the theme automatically reverts to the original style of the game |

Notes:

1. Do not check for colors, positions, labels, spelling.

**Test Case 4**

Purpose: Verify that in-game elements function correctly

Setup: Obtain TicTacToe v1.3 (latest version). Follow directions below

Test Data:

|  |  |  |
| --- | --- | --- |
| **Action** | **Input** | **Output** |
| Select start game | Click the ‘Start Game’ button | Transitions to the in-game screen with a grid, timer, and pause button |
| Verify that clicking the grid functions | Click a space on the grid | Clicking a space on the grid should show a player piece that fits inside square.  Clicking anywhere else on the screen should NOT spawn a player piece |
| Verify that clicking the pause button functions | Click the ‘Pause’ button | Clicking the button should pause the timer and disable clicking inside the grid |
| Verify that clicking the resume button functions | Click the ‘Resume’ button | Clicking the button should un-pause the timer and re-enable clicking inside the grid |
| Verify that clicking the quit button functions | Click the ‘Quit’ button | Clicking the button should trigger a pop-up confirming to leave to the game. Yes brings the player back to the main menu. No resumes the game |
| Verify that the game transitions to a “timer ran out” scene when the timer hits 0 | Wait for the timer to hit 0. | After the timer hits 0, the game transitions to a timer ran out scene with an option to play again or to return to the menu |

Notes:

1. Testing the ‘play again’ and ‘return to menu’ buttons will be done in a different system test case
2. Game logic is not tested.
3. Do not check for colors, positions, labels, spelling.

**Test Case 5**

Purpose: Verify that end-game scenes and elements function correctly

Setup: Obtain TicTacToe v1.3 (latest version). Follow directions below

Test Data:

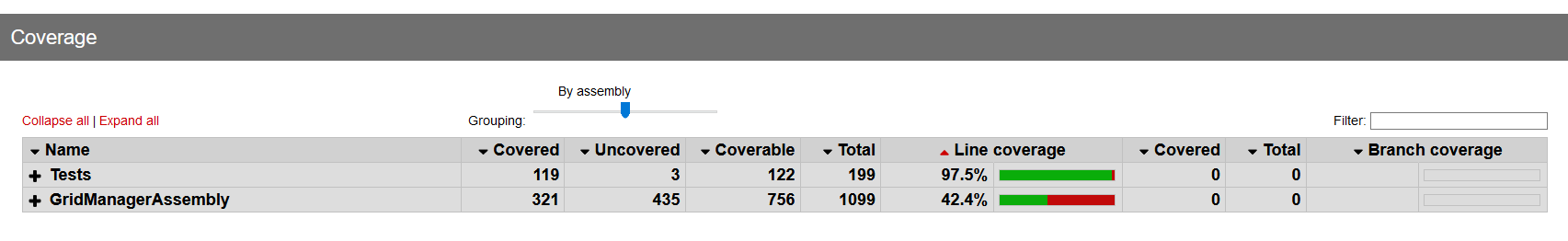
|  |  |  |
| --- | --- | --- |
| **Action** | **Input** | **Output** |
| Allow player piece X to win on purpose | Satisfy win condition for X on grid | Transitions to end-game screen where X is the winner and options to play again or return to menu |
| Test play again | Click ‘play again’ button | Should return to in-game screen this time with O getting the first turn |
| Test return to menu | Click the ‘return to menu’ button | Should return to the game’s main menu |

Notes:

1. The player piece O can be tested with the same concept using this same system test case.
2. Do not check for colors, positions, labels, spelling.

Code Coverage Report

\*note: the image below only shows the overall code coverage for the project. For more specifics, in the project folder, click “Assets” > “Tests” > “CodeCoverage” > “Report”. Inside the “Report” folder, scroll all the way down and find the “index” file. Double click to open an interactive, in-depth report.



In the image above, the line coverage for GridMangerAssembly, only shows 42.4% covered as the other ~60% contains the UI. The UI has already been tested using System Test cases above. Additionally, some of the lines that appear as not tested contain brackets and not any form of logic.

README File Contents

\*note: the README txt file can be also found in the root of the project folder

TIC TAC TOE (v1.3) - Program Description

This program is a game that allows two users to play tic tac toe locally. Running the game requires Unity to be installed.

Downloading Unity to run the Game:

1. Go to https://unity3d.com/get-unity/download and click "download Unity hub"

2. Once downloaded, open Unity. On the leftmost column, click "Installs", click "add". Find and install

2019.4.21f1 (should be the recommended release)

3. Once finished, click on "Projects" on the leftmost column and click "add". Find corresponding project folder

named TicTacToe(v1.3)

4. After added, click to open the project

Running the Game:

1. Once the project has been open, navigate to the Assets folder located at the bottom.

2. Navigate to the "Scenes" folder then the "MenuScenes" folder. Double click "Menu".

3. Once the "Menu" scene appears, click the play button located near the top middle of the screen.

4. This will allow interacting with the game.

Running the Unit Tests:

1. Click "Window" > "General" in the drop down menu > Then click "Test Runner"

2. In the "Test Runner" window, select "PlayMode" in which you should see all the name of the various test cases.

3. (Note: Ensure the game is not running for this step). Click "Run All" in the Test Runner window.

4. A Red X will appear for failed cases and a Green Check will appear for passed cases.

\*NOTE: All code are contained inside the "Tests" and "Scripts" folders, which are inside the "Assets" folder

Project Strengths and Weaknesses

-Looking thoroughly through the project, nothing seems to be missing from the game.

-One of the project's strengths is its level of interaction with the player. The game features easy-to-follow menus, making it a user-friendly experience. Another strength the project has is its game transitions, making it convenient for players to start another game or return to the menu without many clicks in between. Overall, the game's design and functionality make it a solid tic tac toe application.

-One of the project's weaknesses is testing. While the game logic can easily be tested, the UI is particularly difficult with the amount of UI elements within the game, which be easily missed and overlooked when testing the application. Another weakness the project has is its advanced computer player. The advanced bot is sometimes not able to

predict or fully decide where to make a placement. However, this is only noticeable depending on where the player places their piece on the grid.

Contributions and Time Spent

**Jeremiah** – (6 hours) Worked on the abstract factory UI script, attaching it to the game scenes, rigging the button presses, etc. Assisted with the advanced bot script.

**Rural** – ( hours)

**Kymberlee** – (12 hours) Worked on creating the settings menu scene that houses the buttons for the theme changes. Created system test cases for testing the UI of the game. Updated the UML, Sequence, and Use Case diagrams to incorporate abstract factory pattern (UML specific) and the new settings feature. Installed code coverage package and obtained the code coverage results for the game application. Worked on final documentation.

Commit Log