Faculty of Engineering

Department of Electronics and Electrical Communications

ELC2080 Project

Tic Tac Toe game

Testing Document

Under the supervision of:

Dr. Omar Ahmed Nasr

Contents

1.Introduction

1.1 Purpose

This document outlines the test plan, test cases, and expected results for verifying the functional and non-functional requirements of the Advanced Tic Tac Toe Game. It ensures that the system performs as specified in the Software Requirements Specification (SRS) and meets all quality expectations.

1.2 Background

Traditionally, Tic Tac Toe has been played on paper, but the increasing demand for digital and easily accessible entertainment highlights the need for a modernized version. This project addresses that demand by delivering a contemporary, feature-rich application that transforms the classic game into an engaging digital experience.

1.3 Target audience

The Tic Tac Toe desktop application is designed to appeal to a wide and diverse audience, including users of different age groups and technical proficiencies:

- **Families:** the game's simplicity and intuitive interface make it ideal for family-friendly usage, including children.
- **Friends**: it creates an environment full of cheer and happiness while playing. So, it is good to be a fun game that is played with friends.
- **Strategic thinkers**: players interested in challenging the AI to test and improve their tactical decision-making skills.

By catering to this diverse user base, the game aims to deliver an enjoyable experience that balances accessibility with strategic depth.

1.4 Scope

The Tic Tac Toe Game is a C++ application with a graphical user interface (GUI) that allows users to play either against another human or an AI opponent. It includes features such as user authentication, session management, personalized game history, and replay capability. The AI has different levels of difficulty. The system will be tested using Google Test and development will follow version control and CI/CD practices using GitHub and GitHub Actions.

1.5 Definitions, Acronyms, and Abbreviations

- **AI:** Artificial Intelligence.
- **GUI:** Graphical User Interface.
- **CI/CD:** Continuous Integration and Continuous Deployment.
- **SRS:** Software Requirements Specification.
- **Qt:** Cross-platform application development framework for GUI.
- **SQL:** Database engine.
- **UX:** User Experience.
- **UI:** User Interface.
- **OS:** Operating System.
- **Ocell:** the cell on grid has "O".
- **Xcell:** the cell on grid has "X".

1.6 Overview

This project aims to make the user experiences the enduring strategy and challenge of Tic Tac Toe in a modern and user-friendly format. This application offers a captivating and convenient way to test your skills, challenge others or AI opponent and experience the enduring fun of Tic Tac Toe.

The rest of this document details a structured and comprehensive approach to verifying the correctness, quality, and reliability of the Tic Tac Toe game. It includes the test plan and test case specifications. These elements collectively ensure that all functional and non-functional requirements defined in the Software Requirements Specification (SRS) are systematically validated. The test is validated when the output is the same as the expected.

2.Testing plan

2.1 Objective

The objective of a test plan is to define a structured approach for verifying that a software system meets its specified requirements and performs as expected. This involves many aspects:

- **Verify Functional Correctness:**
  Ensure that all features described in the SRS such as gameplay mechanics, AI behavior, user authentication and game history work as intended.
- **Validate Non-Functional Requirements:**
  Test performance, security, usability and reliability under expected conditions.

- **Detect and Document Defects**
  Identify software bugs, errors or deviations from requirements and log them for correction before the final release.
- **Confirm Fixes and Prevent Regression**
  Re-test previously failed scenarios to confirm that defects have been fixed and ensure that changes haven't introduced new issues.
- **Robustness and User Experience**
  We'll test how the game handles unexpected situations (edge cases) gracefully, without crashing or producing unintended results.

2.2 Approach

- **Unit tests**

  Unit tests developed using the Google test framework. Each module tested independently. The focus will be on validating input-output correctness, edge cases, and error handling. So, the test units:

  - Test individual components in isolation.
  - Do specific tests for each function used in the game logic.
  - Focus on core game functionalities:
    - The game initialization like setting up an empty grid for the game.
    - The move validation.
    - The winning and draw conditions.
    - The grid management and if it is updated with the move.
    - The AI performance at each level.

- **GUI and database test**

  We tested them physically by running the game and show each case is working properly or not like

  - Buttons and menus function
  - Sign-in and sign-up functions
  - Test the graphical user interface (UI) interactions
  - Showing the game history
  - Replaying past games
  - The validation of signing with same username used before

- **Integration testing**
  Integration testing plays a pivotal role in the overall testing strategy by evaluating how individual components of the Tic Tac Toe system work together as a cohesive unit. Unlike unit testing, which focuses on isolated modules, integration testing

verifies the seamless coordination of these modules to ensure the system operates as intended. Key areas covered in this testing include:

- **Game Initialization**
  Verifies that a new game session is correctly launched with the proper setup of the game board and active players.
- **Gameplay Flow**
  Assesses the interaction between the user interface and the underlying game logic. Ensures user actions are accurately captured and reflected in the visual state of the board.
- **AI Interaction**
  Confirms the AI component performs strategic decision-making in accordance with the game rules. Validates that the AI responds appropriately to human actions and contributes to a challenging gameplay experience.
- **Game Termination**
  Tests game-ending scenarios, including victories, draws, and invalid states. Confirms that the system correctly identifies and communicates the game outcome to users.

➢ **User Acceptance testing**

Acceptance testing in the Tic Tac Toe game plays a crucial role in verifying that the game functions as intended from a user's perspective, fostering a seamless and enjoyable gameplay experience. This phase simulates real-world user interactions and scenarios. This happens by running and playing the game in the app. We can divide it into many parts to make sure all cases needed to be covered is tested:

- **Gameplay Functionality:**
  - Starting a new game: Verify that users can effortlessly initiate a new game session that includes selecting game modes, adjusting difficulty levels for AI opponents and successfully starting the game with a clear and empty grid.
  - Move validation: Test that the game enforces valid moves by checking that user can’t play outside the grid boundaries or in already occupied cells.
  - Turn management: Ensure turns alternate between the first player and the second player correctly even if one of them is AI or human.
  - Win conditions: Test that the game accurately detects all winning scenarios which are horizontal, vertical or diagonal rows with three matching symbols.
  - Draw condition: Verify that the game recognizes a draw when all cells are filled without a winner emerging.

- **UI and UX:**
  - Clear interface: Test if the game interface is clear and easy to understand for users of all experience levels. The board layout, buttons, and menus should be visually appealing and logically arranged.
  - Response: Test that the game interface responds promptly to user actions. Clicking buttons, selecting moves, and navigating menus should be smooth and immediate.
- **Human player vs AI:**
  - Functionality and algorithm: Test the AI's behavior at different difficulty levels is different and makes strategic moves that become more challenging as the difficulty increases.
  - Response: Ensure the AI responds promptly to user moves and doesn't suffer from excessive delays in making its own choices.
- **Game history:**
  Showing that the matches played is saved successfully and showed with the result of the match and the time of playing. Also, showing that the match is replayed successfully.
- **Handling of errors:**
  Check the system is handling the unexpected action from user like the playing in wrong place successfully.

By performing these tests, we can ensure the Tic Tac Toe game delivers a polished and engaging experience for users.

3.Test cases and scenarios

In the test, we expect some results of variables and compare them with actual values come from the game if the same then the test is passed. We divided it into three main sections for testing:

- **Player type section**
  To test the mode of games player vs player and player vs AI and see each mode if working properly or not.
- **Game model section**
  To test the functions that is used by the controller to control games like the winner function to see if there is a winner also checks the game updates on the grid and player turns.
- **GUI and database section**
  To test that the database is storing the data successfully and the GUI performs properly and integrates with the game logic correctly.

The player type and model sections are tested using test units while the GUI and database as they are on Qt library, so we tested them physically by running the game and tried each scenario and in some cases we tried some scenarios and saw the database file.

Before showing the cases and the scenario of testing we should know how the grid cells is numbered

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)

3.1 Player type section

3.1.1 Player vs player mode

- **Check if "X" played in open cell**
  As the first player to play is always "X" so we chose random row and column (0,1) and played on it in the grid. So, the expected grid now is all empty except the (0,1) cell is an Xcell.
- **Check if "O" can play in open cell**
  As the first player is always "X" so we played twice on the grid to generate Ocell in the grid. We chose 2 random rows and columns (0,1) and (2,2) respectively played on them in order. So, the expected grid now is all empty except the (0,1) is an Xcell and (2,2) is an Ocell.

After these cases now we tested that the player vs player works probably as the it plots "X" or "O" on the selected cell.

3.1.2 Player vs AI mode

We checked that the human player is working properly by the cases we cheeked above and now we try to test that the AI is working properly with its different difficulty levels.

- **Easy level**
  Made the AI for an easy level to play once since it play in random place the expected grid now must have only one occupied cell while other cells are empty.

- **Normal level**
  - Check if it blocks the winning move for the opponent
    Made AI defined as an O_player and made a grid where “X” is having a potential winning move. So, we made the grid to have Xcell in (0,0) and (0,1) while Ocell in (1,0). The AI is expected to play in (0,2) to block “X” from winning. So, the expected grid is to have empty cells except cell (0,0) and (0,1) have Xcell and (1,0) and (0,2) have Ocell.
  - Check if it makes the winning move
    Made AI defined as an O_player and made a grid where “O” is having a potential winning move. So, we made the grid to have Xcell in (0,0), (2,2) and (2,1) while Ocell in (1,1) and (0,2). The AI is expected to play in (0,2) to win the game. So, the expected winner is “O”.
  - Check if it prefers win over block
    Made AI defined as an O_player and made a grid where “O” is having a potential winning move and “X” is having a potential winning move. So, we made the grid to have Xcell in (0,0), (2,2) and (0,1) while Ocell in (1,1) and (1,0). The AI is expected to play in (1,2) to win the game. So, the expected winner is “O”.
- **Hard level**
  We tested the same conditions in normal level to make sure it achieves them also from running it we expect when playing same moves it plays the same move every time as minimax calculation won’t be changed for same play.

3.2 Game model section

It is divided into 2 main parts first the errors or wrong action can be happened, so we check if it is handled correctly and the other part is the controller functionality is working correctly.

3.2.1 Exception tests

- **Check if player can't play in out of cell boundaries is handled**
  Make the system to play in coordinates that doesn’t exist like (3,5). We expect that the error handler of that case “IllegalCellException” would be activated and handle that error.
- **Check if player can't play in already played cell is handled**
  Make an empty grid and play in any cell like (0,0) and then try to play again in the same cell. We expect that the error handler of that case “IllegalCellException” would be activated and handle that error.
- **Check if required cell from getCell is out of boundaries is handled**
  Make a call of the function getCell with parameters of cell that is not exist like (4,3). We expect that the error handler of that case “IllegalCellException” would be activated and handle that error.

- **Check if asking for who is next to play when game is over is handled**
  Make an empty grid and play until the game is over by making a winner then call the function who is next. We expect that the error handler of that case "IllegalStateException" would be activated and handle that error.
- **Check if getting winner function is called and the game still in progress is handled**
  Make an empty grid and play one move in any cell like (0,0) then update the status to make it playing then ask for get winner. We expect that the error handler of that case "NoWinnerException" would be activated and handle that error.
- **Check if getting winner function is called and the game ended as draw is handled**
  Make an empty grid and play moves in order to end as draw and update the status after each move then ask for get winner. We expect that the error handler of that case "NoWinnerException" would be activated and handle that error.

3.2.2 Control functionality

- **Check if the game starts with an empty grid**
  Start a new game then the expected grid is grid full of empty cells.
- **Check if the game plays in the correct cell**
  Make a grid full of empty cells and play on a random cell from it like (1,2). The expected grid is grid full of empty cell except cell (1,2) is Xcell.
- **Check if check draw catches the draw correctly**
  Make an empty grid and a bool variable called isDraw with initial value false. Play a game that ends with draw then make isDraw equal the return from checkDraw function. The expected variable value is true.
- **Check if check draw only seeing draw when the grid is complete**
  Make an empty grid and a bool variable called isDraw with initial value false. Play some moves not making the grid full completely. Then make isDraw equal the return from checkDraw function. The expected variable value is false.
- **Check if can be a winner if the grid is completed**
  Make an empty grid and a bool variable called isWin with initial value false. Play a game that ends with a win and the final move makes the grid full completely. Then make isWin equal the return from checkWin function. The expected variable value is true.
- **Check winning condition**
  - **Row win condition**
    Make an empty grid and play a game making the X_player wins by doing three "X" in one row (row 0). The expected winner is X.
  - **Column win condition**
    Make an empty grid and play a game making the O_player wins by doing three "O" in one column (column 1). The expected winner is O.
  - **Diagonal win condition**

Make an empty grid and play a game making the X_player wins by doing three "X" in a diagonal. The expected winner is X.

- **Check if the game starts with playing status**
  Start a new game before playing any move check the status. The expected status is playing.
- **Check if the game in progress has playing status**
  Start a new game and play any move in random place like (1,2) then update the status. The expected status is playing.
- **Check if the game status updated after win**
  Start a new game and play a game ends with a winning state for any player and after each move in the game update the status. The expected status is win.
- **Check if the game status updated after draw**
  Start a new game and play a game ends with a draw state for and after each move in the game update the status. The expected status is draw.
- **Check if get cell returns the cell correctly**
  Make an empty grid by starting a new game and play one move in any cell like (1,1) and update the status as the first player is always an X_player. The cell value is expected to be Xcell.
- **Check if it's correctly identifying who's next**
  Make an empty grid by starting a new game and play one move in any cell like (1,1) and update the status as the first player is always an X_player. The expected value of who is next is "O".
- **Check the functionality of isTheGameOver function**
  - Checks it return false when start a game without any move.
  - Check it returns true when the game is over either by a win or a draw state.
- **Check undo functionality**
  - Check if it removes the cell chose:
    Start a game and play a move in a random place like (0,0) then do undo on this cell the expected final value of the cell will be empty.
  - Check if it changes status after calling it:
    Play a game with making a win state for a player with each move update the status at the end the status will be win. When call the undo function the expected status will be changed and return to be playing.

3.3 GUI and database section

In this part, we check the following cases by direct running the game system and do the actions to test these cases and see the result.

3.3.1 Database

- **Register success and data stored correctly**
  We signed-up and create a new username and password (e.g. sameh & 2118). The expected is to see sameh and hashed password in the database file.
- **Refuse to enter the username again**
  Try to sign-up with a username that already exists. The expected behavior is to refuse to sign-up and show a warning message.
- **Return if the password is wrong**
  Try to sign-in with wrong password. The expected behavior is to refuse to sign-in and show a warning message.
- **Ty to sign-in for registered user**
  Tried to sign-in with a signed-up username and password. The expected behavior is to go to the main menu successfully.
- **In the game save for the two players if logged in**
  Sign-in for player one and choose player vs player mode and player 2 is a user not a guest so we signed-in to user 2. After the registration and choosing the mode, we played a game and the game history for each user. The expected behavior is to see the game save to the history for both players showing the correct outcome "win/draw/lose" according to player (e.g. if player 1 won then player 2 lost).
- **It can convert the vector of moves to text**
  Play a game with any mode with at least one player is signed-in like player vs AI. The expected is to see the moves of the game saved in the database as a text in format of one move "row,column;" and all moves appended at the end without space until final move has no semi-colon.
- **It can convert the text of moves to vector of moves**
  Replay any game from the history of any registered player. The expected behavior is to show the full game again.
- **Game history**
  Sign-in with an account and play a game then show the history of the account. The expected behavior is to show a table of games with the result of the game.

3.3.2 GUI

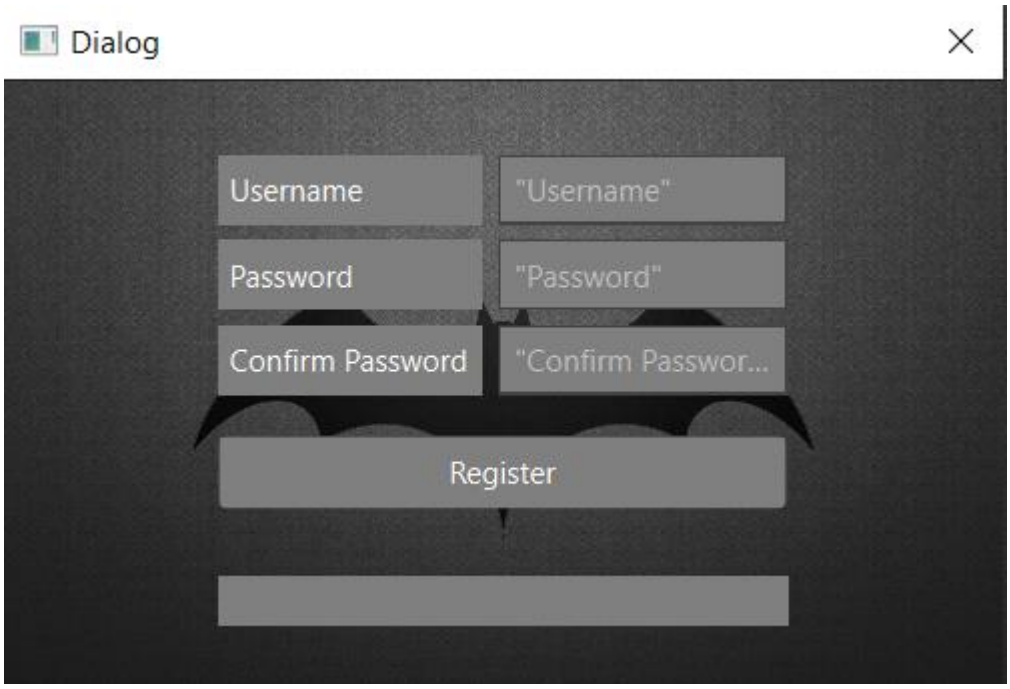
Test each button that works properly and goes to the expected page or do the expected behavior like showing a message or a sub-window.

3.3.2.1 Login page “Main window”

- **Play as guest button**
  When clicking on it, the expected behavior is always to go to the game mode selection page.
- **Sign-in button with wrong password and correct username**
  Enter an existing username with a wrong password. The expected behavior is to show a message indicates that the password is wrong.

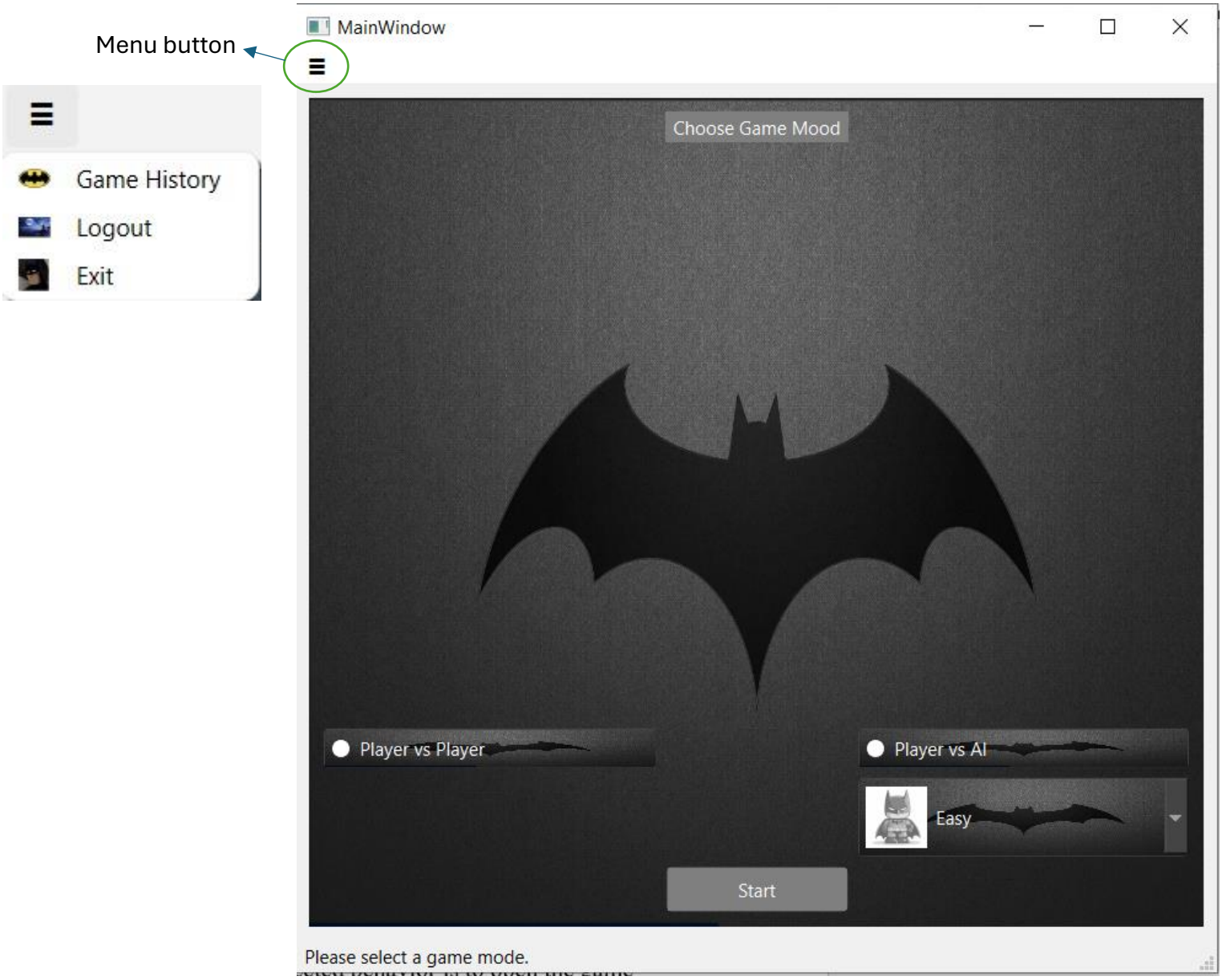
- **Sign-in button with unregistered username**
  Enter a username that haven't signed up before. The expected behavior is to show a message indicates that user has no account on the app.
- **Sign-in with correct username and password**
  Enter an existing username and password that are signed up before. The expected behavior is to go to game mode selection page and showing welcome message.
- **Sign-up button**
  Show a sub-window requires a new registration information "username / password / confirm password".

3.3.2.2 Sign-up sub-window



- **Unmatched password and confirm password input**
  Try to sign-up with a username and enter password then write a different one in the confirm password. The expected behavior is to refuse to sign-up and show a message that the 2 inputs are not matched.
- **Enter an existing username**
  Try to sign up with already signed up username. The expected behavior is to refuse to sign up and show a message that the username is already exist.
- **Enter a new username with matched password and confirm password**
  Enter the data correctly with a new user name and enter a confirm password that is the same of password input. The expected behavior is to show a message that indicates a successful sign up and close the sub-window and return to login page.

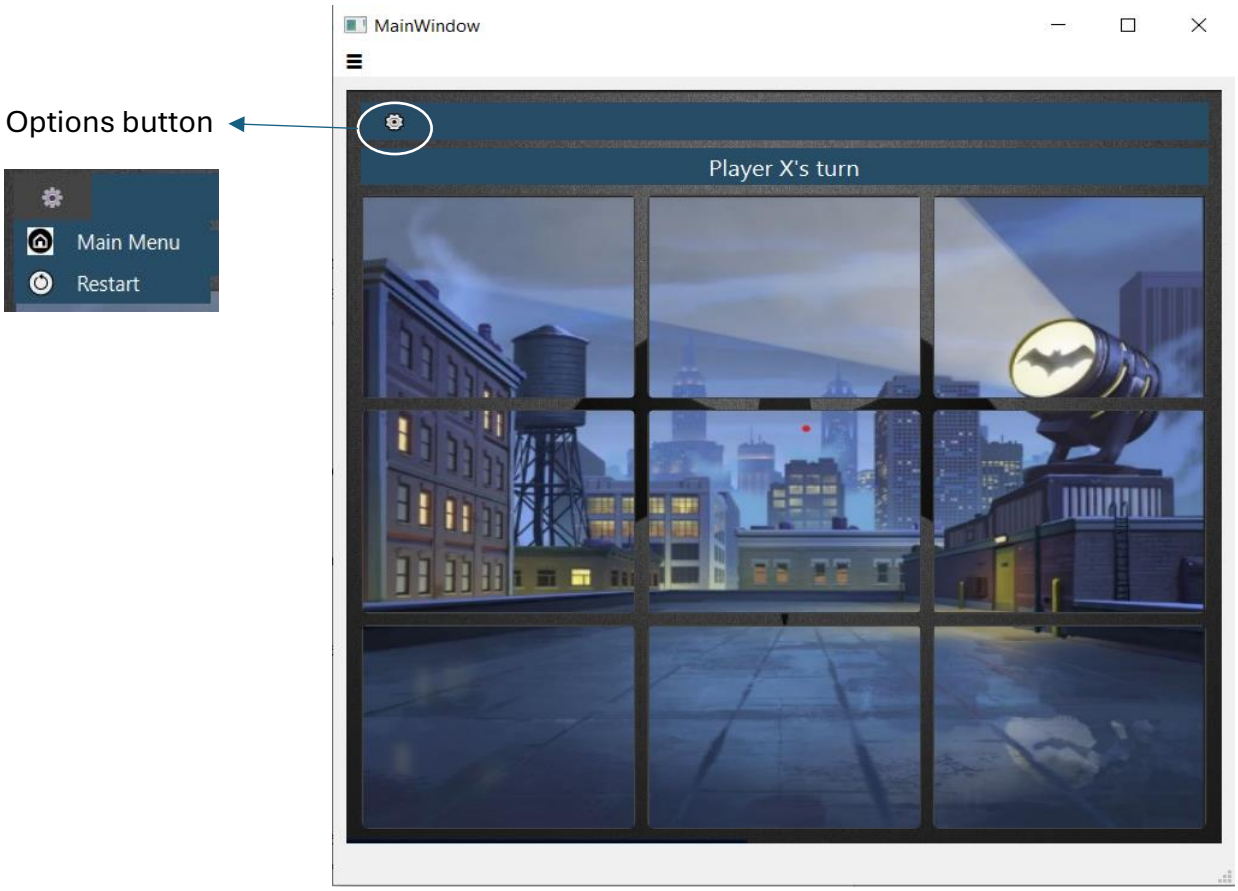
3.3.2.3 Mode selection page



- **One mode selection test**
  Choose one mode like player vs player then try to choose player vs AI. The expected behavior is to disable the player vs player mode as they are radio buttons.
- **Player vs AI mode selection**
  Choose player vs AI mode then choose the difficulty "by default Easy" then click on start button. The expected behavior is to go to the game grid automatically.

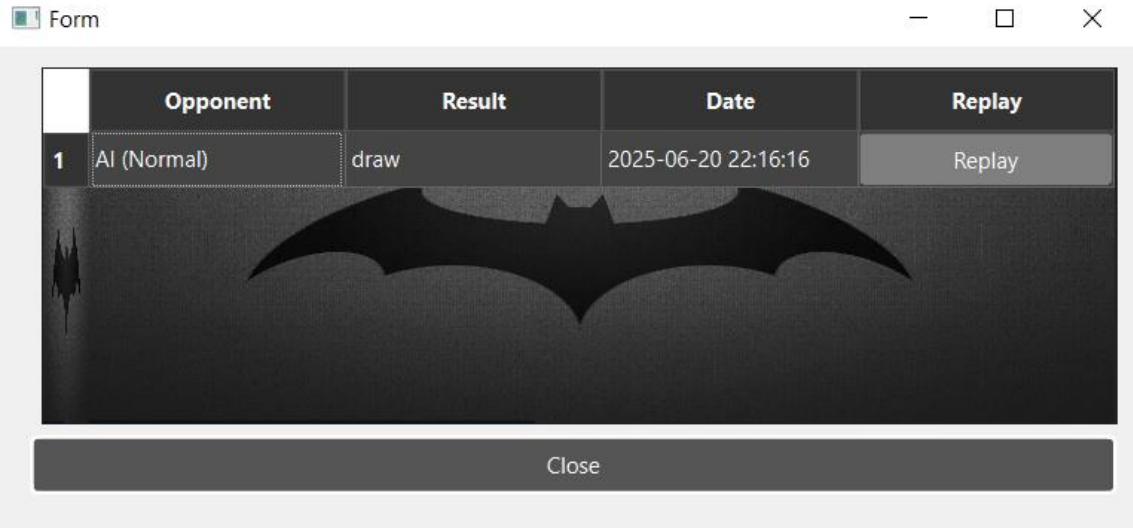
- **Player vs player mode selection**
  Choose player vs player **mode** click on start button. The expected behavior is to show window above the page to choose if player 2 wants to play as a guest or to sign-in or sign-up that have same characteristics of login page.
- **Menu button**
  Click on the menu button. The expected behavior is to chow a list that contains 3 choices "log out / exit / history".
- **Log out selection**
  Choose log out option from the menu list. The expected behavior is to go back to the login page that appeared at the beginning.
- **History selection**
  Choose history option from the menu list. The expected behavior is to open the game history sub-window.
- **Exit selection**
  Choose exit option from the menu list. The expected behavior is to close the app.

3.3.2.4 Game play page



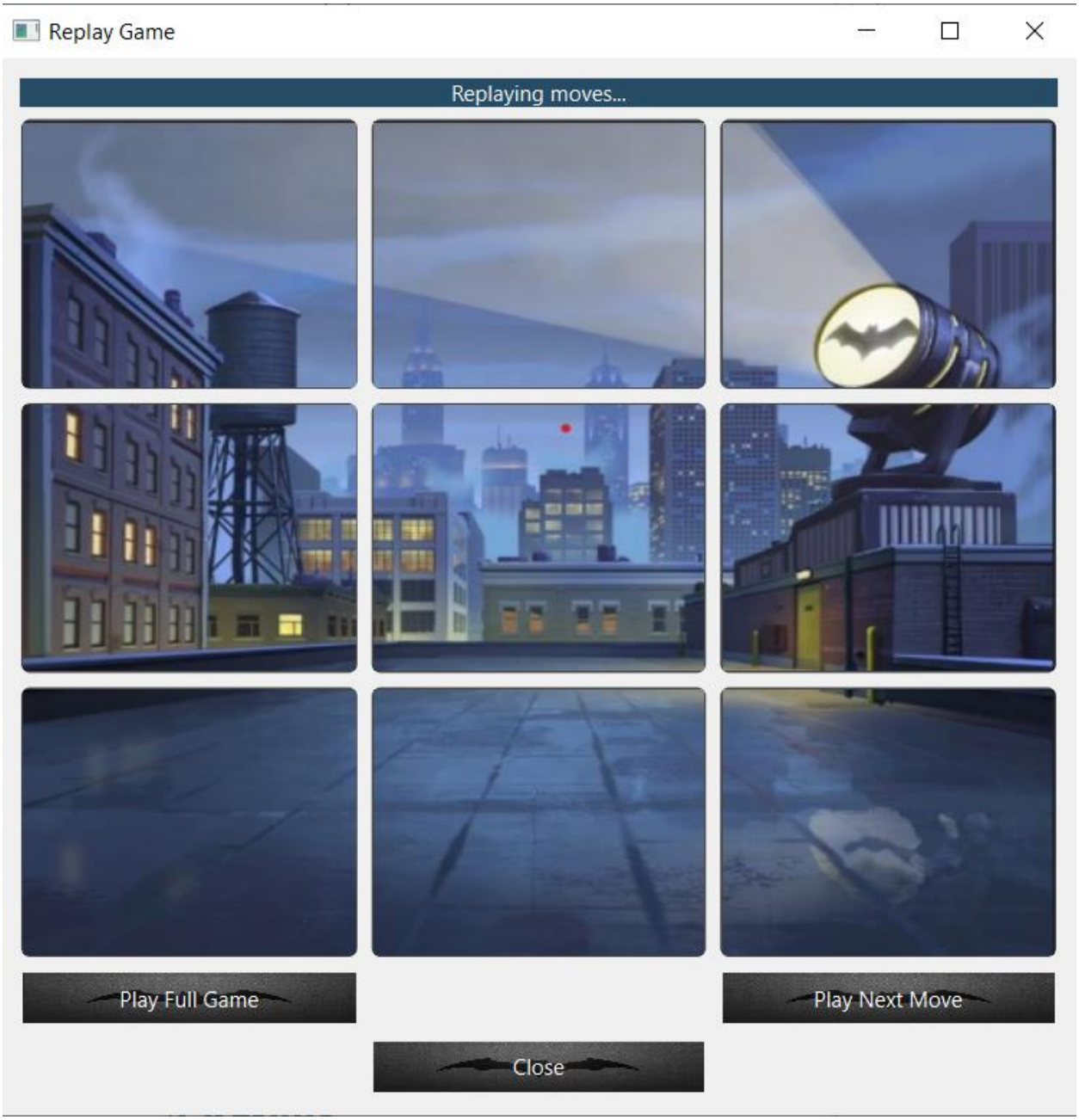
- **Play on the grid**
  When click on a random open cell, the expected behavior is to plot the symbol according to the player turn and change the header to show which player has the turn.
- **Winning the game**
  Play a game with each mode and try to achieve a winning condition for any player. The expected behavior is show a message indicates that the player won.
- **Draw a game**
  Play a game with each mode and try to achieve the draw condition. The expected behavior is show a message indicates that the game is ended as a draw.
- **AI play turn**
  When entering with game mode of player vs AI, the expected behavior when you play is to automatically play according to the algorithm of difficulty level with small delay.
- **Play on not empty cell**
  Play a game and choose to play on played before cell. The expected behavior is to show a message that it is not acceptable to play here with no changing in the grid.
- **Option button**
  When clicking on the option button, the expected behavior is to show a list that has restart and main menu options.
- **Main menu selection**
  Start a game and choose the main menu option from options button. The expected behavior is to return to the main menu “mode selection page” either in the middle of the game or after it is finished.
- **Restart selection**
  Start a game and choose the restart option from options button. The expected behavior is to open an empty grid either in the middle of the game or after it is finished.

3.3.2.5 Game history sub-window



- **Arranged game history**
  Play multiple games for one user then goes to the history page. The expected behavior is to show a table that contains the games that are played in descending order “from newest to the oldest”.
- **Replay button**
  Click on the replay button. The expected behavior is to show replay window above them.

3.3.2.6 Replay window



- **Play full game button**
  Click on the play full game button. The expected behavior is to play the full game with the same order it was played in the match.
- **Play next move button**
  Click on the play next move button. The expected behavior is to play the move that has the order in the vector "when click at the first it plays first move only from the match". Also, when click again on it, it shows the next move in the order.

4.Results

All planned test cases were executed successfully. The results of all tests matched the expected outcomes as specified in the test case specifications. No defects, failures, or anomalies were identified during the execution of these test cases. As a result, the system is considered to have passed the test phase with full compliance to the SRS.

	Test cases number	*Passed*	*Failed*
Player type and game mode	33	33	0
GUI and database	35	35	0

5.Conclusion

The testing phase of the Advanced Tic Tac Toe Game has been successfully completed, encompassing an extensive suite of unit, integration, system, and user acceptance tests. The objective was to verify the system's adherence to the functional and non-functional requirements as specified in the Software Requirements Specification (SRS). Through rigorous and structured testing strategies, all components of the system-ranging from core gameplay logic and AI behavior to GUI responsiveness, user authentication, and game history management-were systematically evaluated.

Each gameplay mode, including player vs player and player vs AI (across multiple difficulty levels), was validated to ensure logical correctness, consistent state management, and a smooth user experience. The AI's performance demonstrated proper decision-making under different game conditions, with accurate move prediction and efficient response times. Additionally, the

game's ability to detect win, draw, and invalid conditions was confirmed through exhaustive test scenarios.

Error handling mechanisms were also thoroughly tested, including responses to out-of-bound moves, repeated actions, invalid states, and authentication failures. These edge cases were gracefully managed using custom exceptions and validation layers, indicating a strong focus on robustness and system stability. The undo functionality, game status updates, and replay features were verified to operate as expected under various gameplay outcomes.

The database and GUI components were tested interactively to confirm successful data persistence, user interface behavior, and seamless interaction with the game logic. Features like game history retrieval, replay functionality, and hashed password storage were confirmed to be secure and functional. All recorded test cases produced expected results, and no failures or inconsistencies were detected throughout the evaluation.

These outcomes collectively demonstrate that the system satisfies its design intentions and is ready for real-world usage. The application offers a reliable, secure, and engaging experience to a wide user base and successfully modernizes a classic strategy game in a digital environment.