Comprehensive Exercise Report

Team Alone of Section 040

Ibrahim Tuncer – 240ADB016

[**Requirements/Analysis**](#_uwgqwd5ezv2w) **2**

[Journal](#_lsityg2iq9m6) 2

[Software Requirements](#_2h0vru1u2mla) 3

[**Black-Box Testing**](#_prhaxdxmf8n8) **4**

[Journal](#_18f11w613jft) 4

[Black-box Test Cases](#_2xn4jzot820y) 5

[**Design**](#_24fdizefyocn) **6**

[Journal](#_esp2ocs9j6bk) 6

[Software Design](#_aifbl1x6rddt) 7

[**Implementation**](#_hya8f3jqkba6) **8**

[Journal](#_acupzfhai7gz) 8

[Implementation Details](#_ojhtwkms2z3b) 9

[**Testing**](#_3qvya3vi836q) **10**

[Journal](#_ckfs4xbl5pyr) 10

[Testing Details](#_bzt1547yxzxi) 11

[**Presentation**](#_hdjvrbf45b1p) **12**

[Preparation](#_xbiquwtmf36n) 12

[**Grading Rubric**](#_u0hfnmdgusmf) **13**

# Requirements/Analysis

Week 2

## Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* After reading the client’s brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.
* Project Description: The project is a Tic Tac Toe game developed in C# using Visual Studio 2022, allowing two players to play the game on a 3x3 grid, with the system determining the winner or if the game ends in a draw.
* Known Requirements from Client Description*:*
* The game must support two players.
* The game board is a 3x3 grid.
* Players alternate turns placing X or O on the grid.
* The game must detect and announce a winner when three of the same symbols align horizontally, vertically, or diagonally.
* The game must detect and announce if the game ends in a draw.
* The game must reset for a new game after a win or draw.
* After reading the client’s brief (possibly incomplete description), what questions do you have for the client? Are there any pieces that are unclear? After you have a list of questions, raise your hand and ask the client (your instructor) the questions; make sure to document his/her answers.
* Should the game have a graphical user interface or be console-based? *Answer: It should have a graphical user interface.*
* Should there be any time limit for each player's move? *Answer: No, there should be no time limit.*
* Are there any specific design preferences or themes for the GUI? *Answer: No specific preferences, keep it simple and user-friendly.*
* Should the game have a scorekeeping feature to track wins, losses, and draws over multiple games? *Answer: Yes, a scorekeeping feature would be a nice addition.*
* Is there a preference for how to display the winner or draw announcement? *Answer: A message box or on-screen text is acceptable.*
* Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
* The project involves creating a GUI in C#, which I am kind of very familiar with.
* Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
* The users are typically casual gamers or anyone interested in playing a simple two-player game of Tic Tac Toe, ranging from children to adults.
* Describe how each user would interact with the software

User Interaction*:*

* Users start the game and see a 3x3 grid.
* Player 1 clicks on a grid cell to place an "X".
* Player 2 clicks on a different cell to place an "O".
* This alternation continues until a player wins or the game ends in a draw.
* Users can reset the game to play again.
* What features must the software have? What should the users be able to do?

Features:

* Two-player functionality.
* 3x3 game grid.
* Turn-based symbol placement.
* Win detection (three in a row).
* Draw detection (no more moves possible).
* Game reset functionality.
* Scorekeeping for wins, losses, and draws.
* Other notes:

## Software Requirements

Overview*:* The Tic Tac Toe game is a two-player game implemented in C# using Visual Studio 2022. Players take turns to place their respective symbols (X or O) on a 3x3 grid. The game checks for a win condition when three symbols align horizontally, vertically, or diagonally, or for a draw when the grid is full. The game also includes a scorekeeping feature and allows players to reset the game after each round.

Requirements:

1. The game must support two players.
2. The game must display a 3x3 grid.
3. Players must alternate turns placing X or O on the grid.
4. The game must detect a win when three symbols align horizontally, vertically, or diagonally.
5. The game must detect a draw when all cells are filled without a winner.
6. The game must provide an option to reset the game after a win or draw.
7. The game must keep track of the score for wins, losses, and draws.
8. The game must have a simple and user-friendly graphical interface.
9. The game must display win or draw notifications.
10. The game must be developed using C# and Visual Studio 2022.

# Black-Box Testing

Instructions: Week 4

## Journal

***Remember:*** Black box tests should only be based on your requirements and should work independent of design.

The following prompts are meant to aid your thought process as you complete the black box testing portion of this exercise. Please review your list of requirements and respond to each of the prompts below. Feel free to add additional notes.

* What does input for the software look like (e.g., what type of data, how many pieces of data)?
  + Player clicks on one of the 9 grid cells.
* What does output for the software look like (e.g., what type of data, how many pieces of data)?
* Updated game grid displaying X or O in the clicked cell.
* Notifications for win or draw.
* Updated score.
* What equivalence classes can the input be broken into?
* Valid cell clicks (empty cells).
* Invalid cell clicks (already occupied cells).
* Game reset action.
* What boundary values exist for the input?
* The first move.
* The ninth move, which could result in a draw.
* Any move resulting in a win condition.
* Are there other cases that must be tested to test all requirements?
* Testing the win condition for all possible winning combinations.
* Testing the draw condition with no remaining moves.
* Ensuring the game resets correctly.
* Other notes:

## Black-box Test Cases

Use your notes from above to complete the black-box test plan section of the formal documentation by writing black box test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| 1 | Click on an empty cell | X or O is placed in the cell | X and O placed |
| 2 | Click on a cell that is already taken | No change to the cell | No change |
| 3 | Win by horizontal alignment | Win notification for correct player | Win notification |
| 4 | Win by vertical alignment | Win notification for correct player | Win notification |
| 5 | Win by diagonal alignment | Win notification for correct player | Win notification |
| 6 | Draw condition | Draw notification | Draw notification |
| 7 | Reset game after win or draw | Grid is cleared, score is updated | Grid is cleared |

# Design

Instructions: Week 6

## Journal

***Remember:*** You still will not be writing code at this point in the process.

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* List the nouns from your requirements/analysis documentation.
* Game
* Player
* Grid
* Cell
* Symbol
* Turn
* Win
* Draw
* Score
* GUI
* Which nouns potentially may represent a class in your design?
* Game
* Player
* Grid
* Cell
* Score
* Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
* Player: Symbol (X or O), Name
* Cell: Position, Status (Empty, X, O)
* Grid: Cells (array of Cell)
* Score: Wins, Losses, Draws
* Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.

*Design Option 1:*

* Game: Manages overall game flow, win/draw detection.
* Player: Stores player information.
* Grid: Represents the game grid with a collection of Cell.
* Cell: Represents each cell on the grid.
* Score: Tracks wins, losses, draws.

*Pros:*

* Clear separation of responsibilities.
* Easy to manage and extend.

*Cons:*

* More classes may add complexity.

*Design Option 2:*

* Game: Combines grid and player management, includes score tracking.
* Player: Stores player information.
* Cell: Represents each cell on the grid.

*Pros:*

* Fewer classes, simpler design.
* Easier to implement initially.

*Cons:*

* Less modular, harder to manage changes.
* Which design do you plan to use? Explain why you have chosen this design.
* Chosen Design: Design Option 1. This design is chosen because it provides a clear separation of responsibilities, making the code easier to maintain and extend in the future.
* List the verbs from your requirements/analysis documentation.
* Play
* Place
* Detect
* Announce
* Reset
* Track
* Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
* Game: StartGame, CheckWin, CheckDraw, ResetGame
* Player: MakeMove
* Grid: UpdateCell
* Score: UpdateScore
* Other notes:

## Software Design

* Game:
  + Attributes: Player1, Player2, Grid, Score
  + Methods: StartGame(), CheckWin(), CheckDraw(), ResetGame()
* Player:
  + Attributes: Name, Symbol
  + Methods: MakeMove(Grid grid, int position)
* Grid:
  + Attributes: Cells (array of Cell)
  + Methods: UpdateCell(int position, Symbol)
* Cell:
  + Attributes: Position, Status
* Score:
  + Attributes: Wins, Losses, Draws
  + Methods: UpdateScore(string result)

# Implementation

Instructions: Week 8

## Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompt below and feel free to add additional notes.

* What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation.

*Programming Concepts:*

* Object-Oriented Programming (OOP): To create and manage different classes such as Game, Player, Grid, Cell, and Score.
* Event Handling: To handle user interactions like clicking on the grid cells.
* Conditional Statements: To check for win and draw conditions.
* Loops: To iterate through the grid cells for updates and checks.
* Data Structures: Arrays to manage the grid cells.
* Other notes:

## Implementation Details

* **How to Play:**
  1. Start the application to see the game grid.
  2. Player 1 (O) makes the first move by clicking on any empty cell.
  3. Player 2 (X) then clicks on another empty cell.
  4. Continue alternating turns until a player wins or the game ends in a draw.
  5. A message will appear announcing the result.
  6. Click the "New Game" button to start a new game.
* **Scorekeeping:**
  1. The score is updated automatically after each game.
  2. Wins, losses, and draws are displayed on the screen.
* **Requirements:**
  1. .NET Framework installed.
  2. Visual Studio 2022 for development.
  3. Microsoft Form apllication used.

# Testing

Instructions: Week 10

## Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
* No changes to the requirements.
* List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.

Game:

* + Equivalence Classes: Valid moves, win detection, draw detection.
  + Boundary Values: First move, ninth move.
  + Paths: Start game, reset game, check win/draw.

Player:

* + Equivalence Classes: Valid/invalid moves.
  + Boundary Values: Edge cells.
  + Paths: Make move.

Grid:

* + Equivalence Classes: Empty/filled cells.
  + Boundary Values: All cells filled.
  + Paths: Update cell status.

Score:

* + Equivalence Classes: Increment wins/losses/draws.
  + Boundary Values: Initial score.
  + Paths: Update score.
* Other notes:

## 

## 

## Testing Details

1. **TestGameInitialization.cs**
   * Tests if the game initializes correctly with an empty grid and zero scores.
2. **TestPlayerMove.cs**
   * Tests player move functionality and ensures only valid moves are registered.
3. **TestWinCondition.cs**
   * Tests all possible win conditions (horizontal, vertical, diagonal).
4. **TestDrawCondition.cs**
   * Tests for the draw condition when the grid is full without a winner.
5. **TestScoreUpdate.cs**
   * Tests the scorekeeping functionality after wins, losses, and draws.

*Black-Box Test Plan:*

* Execute each test case and record the actual results.
* Compare actual results with expected results to validate functionality.

# Presentation

Instructions:Week 12

## Preparation

The following prompts are meant to aid your thought process as you complete the presentation portion of this exercise. It is recommended that you examine the previous sections of the journal and your reflections as you work on the presentation as it is likely that you have already answered some of the following prompts elsewhere. Please respond to each of the prompts below and feel free to add additional notes.

* Give a brief description of your final project
* Final Project Description: The project is a Tic Tac Toe game developed in C# using Visual Studio 2022. It allows two players to play on a 3x3 grid, determines the winner or if the game ends in a draw, and includes scorekeeping.
* Describe your requirement assumptions/additions.

*Requirement Assumptions/Additions:*

* Assumed a graphical user interface was needed.
* Added scorekeeping for better user experience.
* Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?

*Design Options:*

* Option 1: Separate classes for Game, Player, Grid, Cell, and Score for better modularity.
* Option 2: Combined Game management with score tracking and fewer classes for simplicity.

*Decision:* We choosed Option 1 because, better separation of responsibilities and easier maintenance.

* How did the extension affect your design?
* Adding the scorekeeping feature required an additional Score class and methods to update and display scores.
* Describe your tests (e.g., what you tested, equivalence classes).
* Tested player moves, win conditions, draw conditions, and score updates.
* Equivalence classes included valid/invalid moves and win/draw scenarios.
* What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
* Importance of clear requirements and modular design.
* Effective use of OOP principles for manageability.
* Thorough testing to ensure robust software.
* What functionalities are you going to demo?
* Game play with two players.
* Win and draw detection.
* Scorekeeping and game reset functionality.
* Who is going to speak about each portion of your presentation? (Recall: Each group will have ten minutes to present their work; minimum length of group presentation is seven minutes. Each student must present for at least two minutes of the presentation.)
* Introduction and Project Description: Ibrahim Tuncer
* Requirements and Design: Ibrahim Tuncer
* Implementation and Testing: Ibrahim Tuncer
* Demo: Ibrahim Tuncer
* Other notes: