Mullumbimby High School
Assessment Task

Student Name:	

Task Name:	Tic Tac Toe		
Year:	11	**Faculty:**	TAS
Subject:	Software Engineering	**Teacher:**	Steedman
Date Assigned:	02/04/2026	**Date Due:**	02/05/2026
Weighting:	35%	**Total Mark:**	___ / 100

Outcomes to be Assessed:

SE-11-01 - describes methods used to plan, develop and engineer software solutions
SE-11-02 - explains how structural elements are used to develop programming code
SE-11-04 - applies safe and secure practices to collect, use and store data
SE-11-06 - applies tools and resources to design, develop, manage and evaluate software
SE-11-08 - applies language structures to refine code
SE-11-09 - manages and documents the development of a software project

Task Description:

As a software engineer, you are tasked to design and develop a Tic Tac Toe game in Python. You will work through each stage of the software development cycle while demonstrating your skills with concepts such as data types and structures, input validation, functions, loops, and lists.

Submission Instructions:

Hand in:
1. This completed workbook
2. Your project via GitHub Classroom

Absence / Misadventure Instructions:	**Academic Integrity:**	**Acknowledgement:**
All Stage 6 students have been issued with an Assessment Policy Handbook, which outlines the procedures relating to absence/illness/misadventure. This handbook can also be found on the school website. Students must submit an illness/misadventure form to the Head Teacher Senior Studies (Mrs Elliott) for foreseeable absences/misadventures before the task due date and immediately on return to school for unforeseeable absences / misadventures. The following penalties for late submission without an acceptable reason will apply: one day late - 20% deducted, two days late - 40% deducted. Three or more days late will be awarded zero.	Any instances of academic dischonesty may be interpreted as a non-attempt (failing grade). Please refer to the assessment policy booklet for more information. To ensure academic integrity, teachers may require students to submit their work using Google Docs or other platforms with version history tracking (or some other means of evidence of authorship). This enables teachers to verify that the content has been created by the student and not generated by an AI tool.	CLICK HERE OR SCAN

Tic Tac Toe

Requirements:

Tic Tac Toe is a game for two players, X and O, who take turns marking spaces on a 3×3 grid. The objective of the game is to win by placing three of their marks in a horizontal, vertical, or diagonal row. If neither player succeeds in doing this and all spaces are taken, then the result is a tie. In this assignment, you are tasked with coding this game.

Your program will prompt a specific player (X or O) to enter the row and column of the space that they want to mark. The program will evaluate whether the move is valid. If it is valid, the space on the board will be updated with a mark corresponding to the player's symbol. If it is invalid, the player will be asked to enter the row and column again. The program should run until Player X wins, Player O wins, or when there's a tie. A tie occurs when the board is full and neither player has won. The program ends with the announcement of the game's outcome.

Specifications (15):

Requirements / Specifications Table (10)

From the above requirements and your understanding of the game, break these down into table format, pairing each requirement with a potential specification (how you might implement it).

Priority	Requirements	Specifications
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

IPO Diagram (5)

Create an Input-Process-Output (IPO) diagram for the Tic Tac Toe game, based on the given requirements.

Inputs	**Processing**	**Outputs**

Design (10)

Screen Design (5)

Create a screen design for the Tic Tac Toe game in the form of a neat, detailed sketch. This should visually represent how the game will appear to the players as they play. A reminder that it is a console program.

Structure Chart (5)

Create a structure chart for your Tic Tac Toe game. This chart should visually represent the top-down, modular structure of your program, showing how different functions and components are organised and interact. Remember to break your program down into its key functions. The chart should flow from left to right demonstrating the flow of data between functions as well as the control structures used.

.

Development (60)

Your Tic Tac Toe game should reflect a solid understanding of basic programming concepts and demonstrate good coding practices. Focus on writing clean, efficient, and well-documented code. This project is an opportunity to showcase your skills in Python programming and problem-solving.

Well-Structured Code (10):

- Organise code into functions for better readability and maintainability.
- Use meaningful variable and function names.
- Follow Python's PEP 8 style guide for coding conventions.

Game Logic (10):

- Accurately check for win conditions (three in a row horizontally, vertically, or diagonally).
- Determine and announce a tie if the board is full without a winner.
- Alternate turns between players X and O.

Effective Use of Data Structures (8):

- Implement the game board as a 2D list.
- Use appropriate data types for storing player marks, game status, etc.

Control Structures (8):

- Use loops (e.g., while, for) for game rounds and iterating through the board.
- Implement conditional statements (if-else) for decision-making like checking win conditions.

Input Validation (5):

- Ensure that the player's input (row and column) is within the valid range and the chosen cell is empty.
- Handle unexpected or erroneous inputs gracefully without crashing.

User Interface (5):

- Clear and user-friendly console output showing the game board and instructions.
- Display appropriate messages for player turns, invalid moves, and game outcomes.

Error Handling (5):

- Implement basic error handling to manage unexpected issues during gameplay.
- Clearly explain to the user what the error is and how to resolve it.

Documentation and Comments (5):

- Include comments explaining the purpose of functions and key sections of code.
- Write a brief program header that includes authorship, purpose of the program, and date.

Version Control (4):

- Use GitHub to manage and document the development process.
- Commit changes regularly with clear, descriptive commit messages.

Testing and Debugging (10)

Testing Table (10)

Create a comprehensive testing table for your Tic Tac Toe game. This table should document various test cases, including normal, boundary, and erroneous conditions, to ensure the game functions as intended.

ID	Test Case Description	Expected Output	Actual Output	Observations
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Evaluation (5)

Discuss how closely your final game aligns with the initial requirements and specifications. Highlight any areas where the game meets or exceeds expectations and any where it falls short. Reflect on any challenges faced during development and how you overcame them. Discuss what you learned from these experiences and how they have contributed to your understanding of software development.

Marking Rubric

Requirements & Specifications					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Requirements/ Specifications Table	The table includes only a few requirements and specifications, lacking detail or clarity. 1-3	The table covers most requirements but may lack specific details or clarity in specifications. 4-6	The table covers all requirements with clear, relevant specifications. 7-8	The table is exceptionally detailed, covering all requirements with well-thought-out specifications. 9-10	/10
IPO Diagram	The diagram includes basic elements but lacks detail and may have inaccuracies. 1	The diagram is mostly complete but lacks some detail or clarity in representing the process. 2	The diagram is complete and accurately represents the program's inputs, outputs and processing with clear details. 3-4	The diagram provides a detailed, accurate, and insightful representation of the program's inputs, outputs and processing. 5	/5

Design					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Screen Design	Basic sketch with minimal detail. Lacks clarity in representing gameplay, input, and output. 1	Sketch is somewhat clear but missing key elements like input prompts or output displays. 2	Detailed and clear sketch showing gameplay elements. Includes most input prompts and output displays. 3-4	Thorough and detailed sketch with all elements of gameplay, input, and output clearly represented. 5	/5
Structure Chart	Rudimentary chart with limited understanding of program structure. Few key functions and data flow represented. 1	Chart shows some understanding of program structure. Most key functions are included, but data flow is unclear. 2	Well-structured chart showing all key functions. Clear data flow between functions, but might lack some detail. 3-4	Comprehensive and detailed structure chart. Clearly demonstrates understanding of program structure, key functions, and data flow. 5	/5

Development					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Well Structured Code	Code has minimal organization, with poor naming and inconsistent style. 1-3	Code is somewhat organized, with variable names that occasionally lack clarity. Moderate adherence to PEP 8. 4-6	Code is organized and readable, with clear variable names and good adherence to PEP 8. 7-8	Code is exemplary in organization, with exceptionally clear naming and strict adherence to PEP 8. 9-10	/10
Game Logic	Only basic game logic is implemented; fails to correctly handle win conditions or turn alternation. 1-3	Implements most game logic, but with occasional errors in win condition checks or turn alternation. 4-6	Implements all game logic correctly, including win conditions and turn alternation. 7-8	Game logic is not only correct but also demonstrates creative or optimized solutions. 9-10	/10
Data Structures	Basic or incorrect structures, causing some functionality limitations. 1-2	Uses 2D lists with some inefficiencies; moderate grasp of structure utility in game logic. 3-4	Effectively employs 2D lists and appropriate data types for robust game state management. 5-6	Advanced and optimised use of data structures, enhancing game efficiency. 7-8	/8

Control Structures	Limited or incorrect use of loops and conditionals, leading to errors or inefficiencies. 1-2	Adequate implementation of loops and conditionals, with minor logical errors or inefficiences. 3-4	Effectively utilises loops and conditionals, ensuring efficient and effective game flow. 5-6	Advanced and efficient use of control structures, optimising game logic and performance. 7-8	/8
Input Validation	Input validation is incomplete or frequently fails, leading to crashes or bugs. 1	Basic input validation implemented, handling common errors but missing edge cases. 2	Robust input validation, preventing invalid inputs and handling errors gracefully. 3-4	Comprehensive and foolproof input validation, enhancing user experience and game stability. 5	/5
User Interface	UI is rudimentary, providing minimal information, lacking in clarity and user engagement. 1	UI is functional but lacks refinement; necessary information is displayed but not always clear. 2	UI is clear and user-friendly, effectively conveying all necessary game information. 3-4	UI is exceptional, offering an intuitive and engaging user experience, with attention to detail. 5	/5
Error Handling	Basic or no error handling, often resulting in program crashes or unhandled exceptions. 1	Some error handling is present, but lacks comprehensiveness; errors are not always clearly communicated. 2	Good error handling that effectively catches and communicates common errors. 3-4	Exceptional error handling, proactively managing all potential errors with clear, helpful feedback to the user. 5	/5
Docs / Comments	Sparse or no comments; lacks a program header or authorship information. 1	Some comments that provide basic understanding; simple program header included. 2	Clear and helpful comments throughout; comprehensive program header with full details. 3-4	Exceptional documentation offering deep insights into code functionality; thorough, informative program header. 5	/5
Version Control	Minimal or incorrect use of version control; non-descriptive commit messages. 1	Basic use of version control with somewhat clear commit messages; occasional commits. 2	Regular use of version control with clear, descriptive commit messages. 3	Excellent, consistent use of version control with detailed, informative commit messages for every significant change. 4	/4

Testing and Debugging					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Testing Table	Test cases are incomplete or incorrect. Lack of details in descriptions, expected and actual outputs. 1-3	Test cases cover only some aspects of the game. Descriptions and outputs are present but lack detail. 4-6	Comprehensive test cases covering most game scenarios. Clear descriptions and expected/actual outputs. Minor errors may exist. 7-8	Complete and detailed test cases covering all aspects of the game including normal, boundary, and erroneous conditions. Clear, detailed descriptions and accurate expected/actual outputs. 9-10	/10

Evaluation					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Evaluation	Reflection is minimal or off-topic. Little to no connection with the project requirements and specifications. 1	Basic reflection on the project. Some connection with the project requirements and specifications, but lacks depth. 2	Clear reflection showing understanding of the project requirements. Good insights into challenges and learning experiences. 3-4	In-depth and insightful reflection. Strong connection with project requirements and specifications. Detailed discussion of challenges, solutions, and learning experiences. 5	/5