CONNECT-4 SOFTWARE QUALITY TESTING AND FINAL REPORT

Matthew Alain, 100349282
INFO 3235 S50, Professor Frank Zhang

Contents

Executive Summary

The program I tested for this project was a game of Connect 4 coded in Python 3 using the tk-interface library to create an interactive GUI for the user to play the game with. The intended functionality of the game is to have the player take turns against a computer placing a token into a seven-by-seven grid, with the goal of having four of one player's pieces in a row, either horizontally, vertically, or diagonally, at which point that player wins and the game ends. This report includes:

- The test plan, which identifies the objectives and scope of the testing.
- The 40 conducted test cases, which explain in detail exactly what functionalities were tested and how to replicate them in the future.
- The test execution summary, explaining that of the 40 tests, 27 passed for a 67.5% pass rate, and the related conclusion that identifies which key components should be targeted for revision in future releases (primarily the scoreboard).
- The operational profile, which explains the probabilities of different results based on the type of user playing the game, as determined by their approximate skill level.
- The conducted boundary testing, showcasing the one-dimensional boundary used to find a defect in the grid index.
- The finite state machine for the program, presented as both a table, graph, and matrix, and how it helped to understand the flow control of the program to identify each possible user input and what the expected result of them was to identify successful and failed tests.

Test Plan

Software Quality Assurance Test Plan

Project Name: **Connect-4 Program**

Version: **1.0** Test Plan Date: **2025-04-02**

Prepared By: **Matthew Alain** Team: **Alain/Yao**

Objectives

Establish quantifiable resource and functional requirements for the project completion

Ensure the project adheres to these requirements

Identify defects to be displayed during live presentation April 2nd.

Test Scope

Functional testing

Usability testing

Performance testing

Test Cases

(Note: See attached Excel file for clearer formatting)

Initial Setup Cases

Case ID	Type	Description	Inputs	Expected Outputs	Actual Outputs	Status	Severity
TC01	Black Box	Initial startup - Program runs	1. Run program	1. Connect 4 window opens	Passed	Passed	Critical
TC02	Black Box	Initial startup - No starting game	1. Run program	1. Connect 4 window opens 2. There should be no current player	1. Connect 4 window opens 2. Instructions shows cyan as current player	Failed	Low
TC03	Black Box	Start game - Player First	1. Run program 2. Click start game (Player)	1. Game should be started 2. "Player first count" should increment 3. Player should immediately be able to click on the board to make a move	Passed	Passed	Critical
TC04	Black Box	Start game - Computer First	1. Run program 2. Click start game (Computer)	1. Game should be started 2. Computer should play the first move	Passed	Passed	Critical
TC05	White Box	Start game - Random First	1. Run program 2. Click start game (Random)	1. Game should be started 2. There should be an equal chance of cyan playing first or second	Passed	Passed	Medium
TC06	Black Box	Increment score - Player first count (cyan plays first)	1. Start game with cyan playing first	1. "Player first count" should increment	Passed	Passed	Low
TC07	Black Box	Increment score - Player first count (cyan plays second)	1. Start game with cyan playing second	1. "Player first count" should not increment	Passed	Passed	Low
TC08	Black Box	Increment score - Player first	1. Start game with a random	1. "Player first count" should increment	1. Player first count is not incremented	Failed	Low

		count (cyan chosen randomly)	player player first 2. Have cyan be randomly selected to play first				
TC09	Black Box	Increment score - Player first count (maroon chosen randomly)	1. Start game with a random player player first 2. Have maroon be randomly selected to play first	1. "Player first count" should not increment	Passed	Passed	Low
TC10	Usage-Based	Increment score - Games played	1. Start game 2. Play game to completion	1. Game should start 2. "Total games played" should increment when game starts 3. "Total games played" should not increment when game ends	1. Game starts 2. "Total games played" increments 3. When game is completed, "Total games played" increments	Failed	Low

Playing the Game Cases

Case ID	Type	Description	Inputs	Expected Outputs	Actual Outputs	Status	Severity
TC11	Black Box	Make a move - No game	1. Run program 2. Do not select start game 3. Click anywhere on the board	1. No piece should be placed	Passed	Passed	Medium
TC12	Usage-Based	Make a move - Player turn (Open column)	1. Start game 2. Wait for player turn 3. Click a column on the board where there is room	1. A cyan piece should be placed in the lowest available space of the clicked column 2. It should become the computer's turn	Passed	Passed	High
TC13	Usage-Based	Make a move - Player turn	1. Start game 2. Wait for player turn	1. No piece should be placed 2. It should	Passed	Passed	Medium

		(Full column)	3. Click a column on the board that is filled with pieces	remain the player's turn			
TC14	Usage-Based	Make a move - Player during computer's turn	1. Start game 2. Wait for computer turn 3. Click a column on the board where there is room	1. No piece should be placed 2. It should remain the computer's turn	Passed	Passed	High
TC15	White Box	Make a move - Computer turn	1. Start game 2. Wait for computer turn	1. Maroon piece should be placed in the lowest available row of a valid column 2. It should become the player's turn	Passed	Passed	High
TC16	White Box	Make a move - Computer during player's turn	1. Start game 2. Wait for player turn	1. The computer should be unable to place pieces during the player turn	Passed	Passed	High

Finishing the Game Cases

Case ID	Type	Description	Inputs	Expected Outputs	Actual Outputs	Status	Severity
TC17	Black Box	Finish game - Player wins game (horizontal)	1. Play game 2. Get four consecutive cyan pieces horizontally	1. Game should end 2. No further moves should be possible 3. Cyan should be declared winner	Passed	Passed	High
TC18	Black Box	Finish game - Player wins game (vertical)	1. Play game 2. Get four consecutive cyan pieces vertically	1. Game should end 2. No further moves should be possible 3. Cyan should be declared winner	Passed	Passed	High
TC19	Black Box	Finish game - Player wins game	1. Play game 2. Get four consecutive	1. Game should end 2. No further moves should	Passed	Passed	High

		(diagonal up)	cyan pieces in an upwards diagonal	be possible 3. Cyan should be declared winner			
TC20	Black Box	Finish game - Player wins game (diagonal down)	1. Play game 2. Get four consecutive cyan pieces in a downwards diagonal	1. Game should end 2. No further moves should be possible 3. Cyan should be declared winner	Passed	Passed	High
TC21	Black Box	Finish game - Computer wins game (horizontal)	1. Play game 2. Get four consecutive maroon pieces horizontally	1. Game should end 2. No further moves should be possible 3. Maroon should be declared winner	Passed	Passed	High
TC22	Black Box	Finish game - Computer wins game (vertical)	1. Play game 2. Get four consecutive maroon pieces vertically	1. Game should end 2. No further moves should be possible 3. Maroon should be declared winner	Passed	Passed	High
TC23	Black Box	Finish game - Computer wins game (diagonal up)	1. Play game 2. Get four consecutive maroon pieces in an upwards diagonal	1. Game should end 2. No further moves should be possible 3. Maroon should be declared winner	Passed	Passed	High
TC24	Black Box	Finish game - Computer wins game (diagonal down)	1. Play game 2. Get four consecutive maroon pieces in a downwards diagonal	1. Game should end 2. No further moves should be possible 3. Maroon should be declared winner	Passed	Passed	High
TC25	White Box	Finish game - Tie game	1. Play game with either player playing first 2. Fill board such that neither	1. Game should end 2. Player should be unable to play 3. Game outcome	1. Game ends 2. Player is unable to play 3. Game outcome is	Failed	Medium

			player has four consecutive pieces	should be declared a tie	not declared as a tie		

Incrementing the Game Result Cases

Case ID	Type	Description	Inputs	Expected Outputs	Actual Outputs	Status	Severity
TC26	White Box	Increment score - Player wins (cyan played first)	1. Start game with cyan playing first 2. Get four consecutive cyan pieces in any direction	1. Game should end 2. No further moves should be possible 3. "Player first wins" should increment	1. Game ends 2. No further moves can be played 3. "Player second wins" increments	Failed	Medium
TC27	White Box	Increment score - Player wins (cyan played second)	1. Start game with cyan playing second 2. Get four consecutive cyan pieces in any direction	1. Game should end 2. No further moves should be possible 3. "Player second wins" should increment	Passed	Passed	Medium
TC28	White Box	Increment score - Computer wins (maroon played first)	1. Start game with cyan playing first 2. Get four consecutive maroon pieces in any direction	1. Game should end 2. Player should be unable to play 3. Neither "Player First Wins" nor "Player Second Wins" should increment	1. Game ends 2. No further moves can be played 3. "Player first wins" increments	Failed	Low
TC29	White Box	Increment score - Computer wins (maroon played second)	1. Start game with cyan playing second 2. Get four consecutive maroon pieces in any direction	1. Game should end 2. Player should be unable to play 3. Neither "Player First Wins" nor "Player Second Wins" should increment	1. Game ends 2. No further moves can be played 3. "Player first wins" increments	Failed	Low

TC30	White Box	Increment score - Tie game (cyan played first)	1. Play game with cyan playing first 2. Fill board such that neither player has four consecutive pieces	1. Neither win count should be incremented 2. Respective tied game counter should be incremented	1. Neither win count is incremented 2. Neither tie count is incremented	Failed	Medium
TC31	White Box	Increment score - Tie game (cyan played second)	1. Play game with maroon playing first 2. Fill board such that neither player has four consecutive pieces	1. Neither win count should be incremented 2. Respective tied game counter should be incremented	1. Neither win count is incremented 2. Neither tie count is incremented	Failed	Medium

Edge Cases and Other Functionalities

Case ID	Type	Description	Inputs	Expected Outputs	Actual Outputs	Status	Severity
TC32	Black Box	End current game - Player turn	1. Start game 2. End current game when it is the player's turn	1. Game should end 2. No further moves should be possible until a new game is started	Passed	Passed	Medium
TC33	Black Box	End current game - Computer turn	1. Start game 2. End current game when it is the computer's turn	1. Game should end 2. No further moves should be possible until a new game is started	1. Game ends 2. Computer places one piece that appears over the ended game screen 3. "Game ended" status is replaced with "Current player: cyan"	Failed	Low
TC34	Black Box	End current game - Idle	1. Run program 2. Click end current game button in menu	1. Because no game is started, no game should end	1. Program moves to the "game ended" screen	Failed	Low

TC35	White Box	Exit button	1. Run program 2. Click exit button in menu	1. Program should close	Passed	Passed	Medium
TC36	Performance	Program responsiveness	1. Run program 2. Start game 3. Play game	1. Program should not crash 2. Program response time should not exceed one second	Passed	Passed	Medium
TC37	Black Box	Start game - Multiple consecutive games	1. Start game with the computer going first 2. During the 0.5 seconds that it takes for the computer to place its first piece, start another game with the computer going first	1. The input from the first game should be discarded 2. The computer should take its turn as normal 3. It should become the player's turn	1. The computer waits 0.5 seconds to take its first turn 2. The previous game's input goes through as the computer's first move 3. It becomes the player's turn 4. The current game's input completes, and a cyan piece is placed 5. It remains the player's turn, and they may place a second piece	Failed	Medium
TC38	White Box	Increment score - Extremely large numbers	1. Start game 2. Repeatedly start new games to overflow the score counter	1. The score should never overflow or reset	Passed	Passed	Low
TC39	Usability	Boundary testing - selecting far right border	1. Start game 2. Wait for player turn 3. Attempt to click on the far right edge of the board	1. Cyan piece should be placed	1. “List index out of range” error message displays 2. No piece is placed	Failed	Low
TC40	Usability	Boundary testing - selecting far left border	1. Start game 2. Wait for player turn	1. Cyan piece should be placed	Passed	Passed	Low

			3. Attempt to click on the far left edge of the board				

Test Execution Summary

Total test cases: 40

Passed: 27

Failed: 13

Pending: 0

Pass rate: 67.5%

All critical and high severity tests passed, all failed tests were either medium or low severity.

One week time frame allocated to testing (March 19th – March 26th)

Testing Conclusion

- No critical tests failed; project is not at risk of crashing during presentation.
- Most failed cases (**TC08, TC10, TC26**, and **TC28-TC31**) relate to scoreboard, final version of this component should be majorly revamped to be considered fully successful.
- Allocated time resource has expired, no further updates to the project can be made prior to project submission, future changes will not be tested.
- **TC07**: Seems unintuitive that there are trackers for count, wins, and ties for going first, but only wins and ties for going second. Not a failed case, but perhaps a design oversight.
- **TC08**: "start_game" function gets passed with "random" as the FIRST player, which then randomly sets the CURRENT player. The player first count is only incremented when “cyan” is the first player, not “random”.
- **TC11**: Unintuitive that game board initially appears playable before game is started. Could cause confusion for players. Not a failed case, but perhaps a design oversight.
- **TC26-TC29**: The “player_went_first” variable is initialized as false and is never changed during the program. To aid in the understanding of where test conditions fail, the following two matrices were created for the expected and actual results of the test cases.

Expected

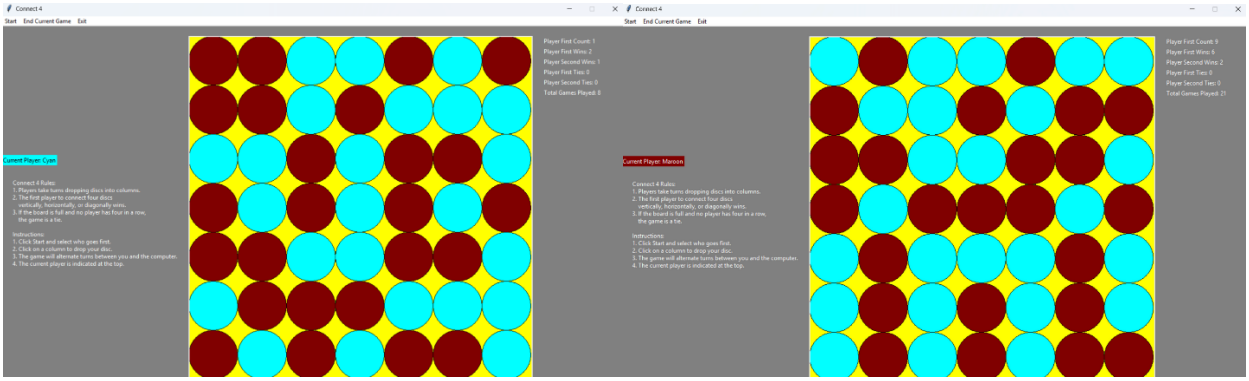
	Cyan wins	Cyan loses
Cyan plays first	Increment "Player first wins"	Nothing
Cyan plays second	Increment "Player second wins"	Nothing

Actual

	Cyan wins	Cyan loses
Cyan plays first	Increment "Player second wins"	Increment "Player first wins"
Cyan plays second	Increment "Player second wins"	Increment "Player first wins"

- **TC30** and **TC31**: The "game_tied" method is never called in the program, so it is never possible to increment the tie counter. This also results in tied games leading to endless loops waiting for either the player or computer to make a valid move, which is never possible. The program does not crash, but it does prevent the game from progressing and requires a new game to be started, as shown in the two images below. This may be an

oversight, as there is a clause within the code that prevents the computer from infinitely looping and crashing the program, but it only stops the computer from making any input.



(Note that in both cases, the window still indicates that it is either the player's turn or computer's turn, but does not allow any moves to be made.)

- **TC37**: Potential exploit to automatically win every game but requires external software to do (See finite state machine section for details). Should be patched before final version.
- **TC39**: Only test case that resulted in an error being thrown but does not crash the program (See boundary testing section for details). Should be either refactored to fix defect or have error message suppressed.

Operational Profile

User Analysis

Player Skill Level	Usage Frequency	Primary Operations
Beginner	10%	Game loss
Novice	35%	Game win (horizontal)
Intermediate	35%	Game win (vertical)
Advanced	10%	Game win (diagonal up)
Expert	10%	Game win (diagonal down)

This user analysis assumes the distribution of the skill level of most players, and what the most likely outcomes of their games are. Because the computer places its moves randomly, it is unlikely that most players will lose against it, so it is assumed that all but the most inexperienced players will win their games. Then, it is assumed that the users who have more skill will be able to use the less common ways of winning with the diagonals, while those with more basic skills will use the horizontal and vertical lines to win.

Test Cases

Starting the game

ID	Scenario	Steps	Expected	Probability	OP Impact

TC03	Start game (player)	1. Run program 2. Click start game (Player)	1. Game should be started 2. "Player first count" should increment 3. Player should immediately be able to click on the board to make a move	45%	Affects all users
TC04	Start game (computer)	1. Run program 2. Click start game (Computer)	1. Game should be started 2. Computer should play the first move	35%	Affects all users
TC05	Start game (random)	1. Run program 2. Click start game (Random)	1. Game should be started 2. There should be an equal chance of cyan playing first or second	20%	Affects all users

Finishing the game

ID	Scenario	Steps	Expected	Probability	OP Impact
TC17	Game win (horizontal)	1. Play game 2. Get four consecutive cyan pieces horizontally	1. Game should end 2. No further moves should be possible 3. Cyan should be declared winner	31%	Affects novice and intermediate players more frequently
TC18	Game win (vertical)	1. Play game 2. Get four consecutive cyan pieces vertically	1. Game should end 2. No further moves should be possible 3. Cyan should be declared winner	26%	Affects novice and intermediate players more frequently
TC19	Game win (diagonal up)	1. Play game 2. Get four consecutive cyan pieces in an upwards diagonal	1. Game should end 2. No further moves should be possible 3. Cyan should be declared winner	15%	Affects advanced and expert players more frequently
TC20	Game win (diagonal down)	1. Play game 2. Get four consecutive cyan pieces in	1. Game should end 2. No further moves should be possible	15%	Affects expert users more frequently

		a downwards diagonal	3. Cyan should be declared winner		
TC21	Game loss (horizontal)	1. Play game 2. Get four consecutive maroon pieces horizontally	1. Game should end 2. No further moves should be possible 3. Maroon should be declared winner	3%	Affects beginner players more frequently
TC22	Game loss (vertical)	1. Play game 2. Get four consecutive maroon pieces vertically	1. Game should end 2. No further moves should be possible 3. Maroon should be declared winner	3%	Affects beginner players more frequently
TC23	Game loss (diagonal up)	1. Play game 2. Get four consecutive maroon pieces in an upwards diagonal	1. Game should end 2. No further moves should be possible 3. Maroon should be declared winner	3%	Affects beginner and novice players more frequently
TC24	Game loss (diagonal down)	1. Play game 2. Get four consecutive maroon pieces in a downwards diagonal	1. Game should end 2. No further moves should be possible 3. Maroon should be declared winner	3%	Affects beginner and novice players more frequently
TC25	Tie game	1. Play game with either player playing first 2. Fill board such that neither player has four consecutive pieces	1. Game should end 2. Player should be unable to play 3. Game outcome should be declared a tie	1%	Affects beginner players more frequently

These test cases indicate the two primary courses that a game follows, with the first test cases representing the frequency of which participant plays first, which is weighted in favor of the player who decides who plays first, then the game outcomes in each possible result. As previously discussed in this section, a computer randomly placing pieces is unlikely to win against the majority of players, so it is assumed that most (about 87%) of games will result in some kind of game win for the player, again with the highest probability being the more straightforward ways to win. It is in far fewer games (about 12%) where the computer will win, a number that closely relates to the estimated number of beginner players. Finally, it is assumed that only one percent of games will end in a tie, as during testing, it took a significant number of

attempts while purposefully trying to end the game in a tie for it to happen, so it is incredibly unlikely that the same would happen to a typical user.

User Profile

ID	Description	User Type	Probability
TC03	Start game (player)	All users	45%
TC04	Start game (computer)	All users	35%
TC05	Start game (random)	All users	20%
TC17	Game win (horizontal)	Novice and intermediate users	31%
TC18	Game win (vertical)	Novice and intermediate users	26%
TC19	Game win (diagonal up)	Advanced and expert users	15%
TC20	Game win (diagonal down)	Expert users	15%
TC21	Game loss (horizontal)	Beginner users	3%
TC22	Game loss (vertical)	Beginner users	3%
TC23	Game loss (diagonal up)	Beginner and novice users	3%
TC24	Game loss (diagonal down)	Beginner and novice users	3%
TC25	Tie game	Beginner users	1%

Weight Contribution

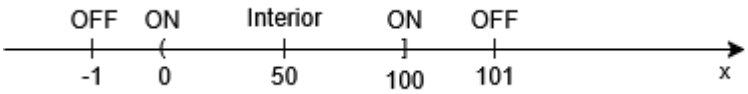
ID	Description	Probability
TC03	Start game (player)	(10% x 45%) + (35% x 45%) + (35% x 45%) + (10% x 45%) + (10% x 45%) = 45%
TC04	Start game (computer)	(10% x 35%) + (35% x 35%) + (35% x 35%) + (10% x 35%) + (10% x 35%) = 35%
TC05	Start game (random)	(10% x 20%) + (35% x 20%) + (35% x 20%) + (10% x 20%) + (10% x 20%) = 20%
TC17	Game win (horizontal)	(35% x 31%) + (35% x 31%) = 21.7%
TC18	Game win (vertical)	(35% x 26%) + (35% x 26%) = 18.2%
TC19	Game win (diagonal up)	(10% x 15%) + (10% x 15%) = 3%
TC20	Game win (diagonal down)	(10% x 15%) = 1.5%
TC21	Game loss (horizontal)	(10% x 3%) = 0.3%
TC22	Game loss (vertical)	(10% x 3%) = 0.3%
TC23	Game loss (diagonal up)	(10% x 3%) + (35% x 3%) = 1.35%
TC24	Game loss (diagonal down)	(10% x 3%) + (35% x 3%) = 1.35%
TC25	Tie game	(10% x 1%) = 0.1%

Interpretation

- As expected, the total sum of the start game probabilities (TC03, TC04, TC05) adds up to 100%, as all users must start the game to play it
- The most likely methods of winning the game are through the more common horizontal (TC17) and vertical (TC18) orientations, rather than the comparatively much smaller diagonal orientations (TC19 and TC20)
- As expected, game losses are incredibly uncommon, but the less common orientations (TC23 and TC24) are more likely due to beginner and novice players being less likely to notice if the computer has the opportunity to win with them

Boundary Testing

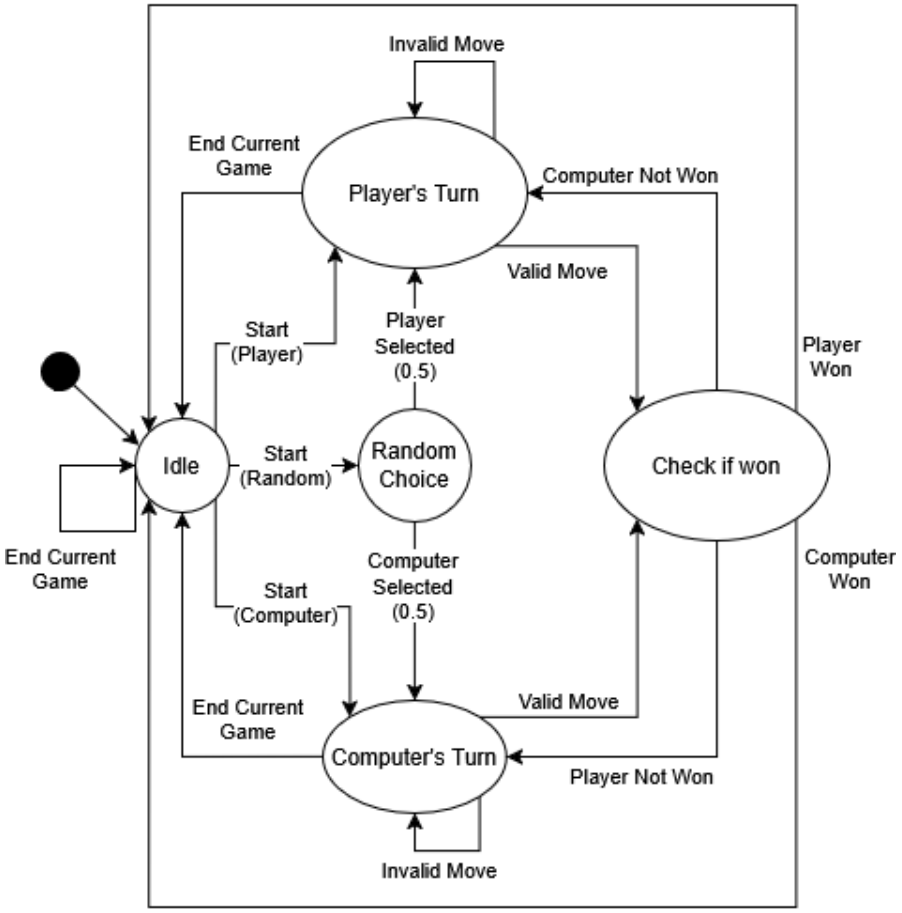
It is difficult to provider boundary testing for this project as many of the utilized values are either Boolean in nature, such as the static button clicks, or are incrementing numeric values that do not take user input. The closest to boundary testing that can be done are tests regarding whether the inputs are valid or invalid, and if they are or are not performed at the correct screen position.



This is a boundary test for the X position of the cursor as it attempts to click the first grid column in order to place a piece. The Y position is unbound as the program only tests for the cursor's X position to determine placement, making this boundary one-dimensional. The left boundary is inclusive, as clicking on the left edge of the column still calls the function, but the right boundary is exclusive, as during testing it was discovered that the right edge is allocated to the next column. This has no visible impact for most of the columns as the borders are only three pixels wide, but clicking on the edge of the rightmost column causes an index error to be thrown, as shown in the image below (with a marker indicating the screen position that was clicked). Similar tests were conducted for other buttons and user inputs, but similar issues were not found during these cases.

Finite State Machine

Diagram



Table

Current State	Input	Next State	Action
Idle	Start game (Player)	Player's turn	Display it is cyan's turn
Idle	Start game (Computer)	Computer's turn	Display it is maroon's turn
Idle	Start game (Random)	Player's turn (0.5) Computer's Turn (0.5)	Display it is cyan's turn Display it is maroon's turn
Player's turn	Select valid column	Check if won	Place cyan piece
Player's turn	Select invalid column	Player's turn	Remain player's turn
Check if won	Four consecutive cyan pieces (yes)	Idle	Announce cyan victory
Check if won	Four consecutive cyan pieces (no)	Computer's turn	Display it is computer's turn
Computer's turn	Computer selects valid column	Check if won	Place maroon piece
Computer's turn	Computer selects invalid column	Computer's turn	Remain computer's turn
Check if won	Four consecutive maroon pieces (yes)	Idle	Announce maroon victory
Check if won	Four consecutive maroon pieces (no)	Player's turn	Display it is player's turn
Player's turn	End current game	Idle	Announce game is ended
Computer's turn	End current game	Idle	Announce game is ended
Idle	End current game	Idle	Announce game is ended

Matrix

	Idle	Player's turn	Check if won	Computer's turn	Random Choice
Idle	End	Start (P)	na	Start (C)	Start (R)
Player's turn	End	Invalid	Valid	na	na
Check if won	-/-	-/-	na	-/-	na
Computer's turn	End	na	-/-	-/-	na
Random Choice	na	-/-	na	-/-	na

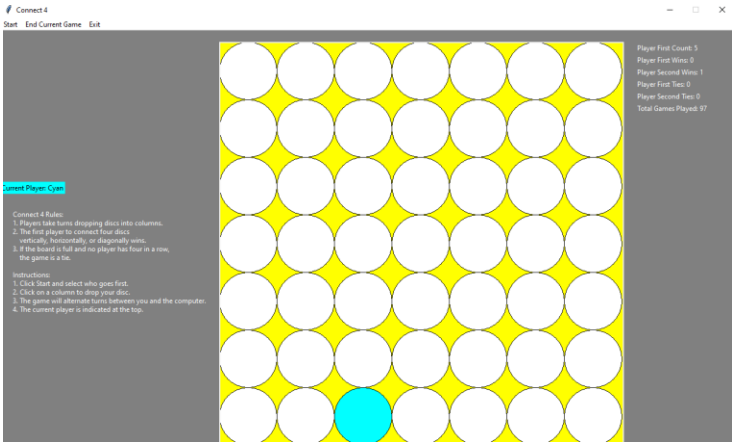
The finite state machine diagram that I created for this project contains five states, with the more important states being weighted and shown as larger, and much more transitions, indicating that this is like a classical mealy machine. This is verified because there are more states than transitions, the program output changes only when certain inputs are provided in certain states, and the changes happen instantly. It is a classical mealy machine because it lacks the reliable control systems of a modified mealy system, and the changes to the state are not delayed until they are synchronized, they simply take place immediately. Because this finite state machine contains a state that determines an outcome randomly, there is an indication that the choice is evenly distributed as if the diagram were a Markov Model, though it lacks the other indications of probability necessary to be a proper Markov Model.

The list of states includes Idle, Player's Turn, Check if Won, Computer's Turn, and Random Choice, with most inputs having two versions, one for the player and one for the computer. These inputs include Start Game, Selected Randomly, Valid Move, Invalid Move, Won, Not Won, and End Current Game, with the only input that is not a parallel between the two players being the Start Game (Random). The random choice state, as well as many of the transitions within the state machine are implicit and cannot be consciously taken by the player. A list of only the explicit transitions, as well as the states they travel between, are provided in the finite state machine matrix. Of note, "Computer's Turn", "Check if Won", and "Random Choice" have no explicit transitions, as they are all processed automatically, leaving only two states where the player has implicit transitions: "Idle" and "Player's Turn".

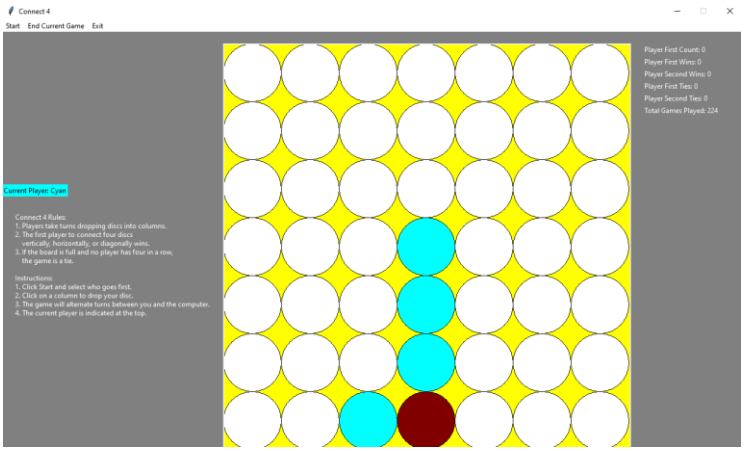
Creating these representations of the program's control flow allowed for a clear listing of each possible user input and the corresponding expected output. This allowed for a more comprehensive identification of cases to test for, and what the expected output of those cases are to determine whether they passed or failed. The matrix specifically also identified the implicit inputs that could not be handled through black box testing and required looking into the source code to determine whether the cases were successful or not. The primary test case this helped identify was a sizeable error regarding asynchronous operations when leaving one state while inputs are still being processed. This defect revealed that with fast enough actions from the user, or with the assistance of software to provide specified inputs in rapid succession, the computer's moves from a previous game could be carried forward into a new game, leading to the player starting with additional pieces on the game board. This led to a variety of new test cases where a queued computer input was followed by each other possible user input to change states, identifying even more defects, as pictured below.

Computer turn -> End game:

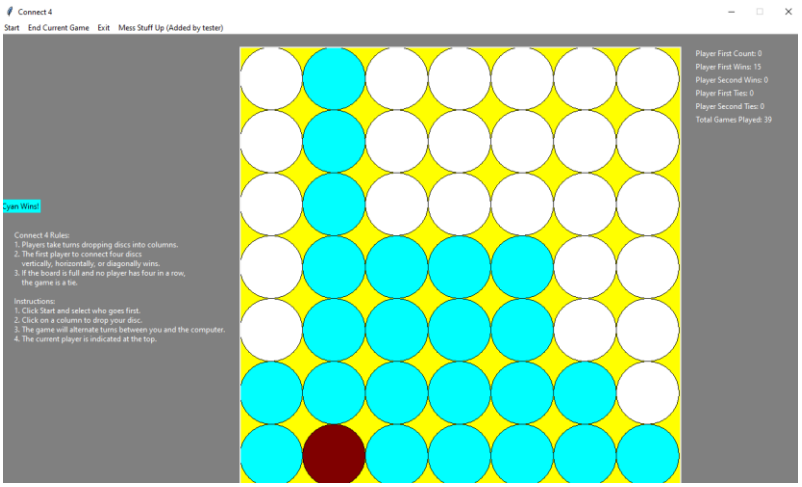
Computer turn -> New game (player's turn first):



Computer turn -> New game (computer's turn first) repeated five times:



New game (computer's turn first) repeated 25 times:



(Note that this was a theoretical test case and involved modifying the code to simplify the testing process on my current hardware. While I was unable to perform this test in the current version of the program, it could theoretically be done by a user with more specialized software than I possess)

Conclusion

This report presents the efforts that went into identifying various defects and errors within David Yao's connect-4 program. Ultimately, while the game is functionally playable, the scope of the project was too large to deliver at a level that would be acceptable by common software quality standards today. Using the testing models above, it was shown that there existed too many errors in the scoring, tie game, and start new game components, and that if the project is to be iterated on in the future, those are the sections of the program that should be given the most attention when refactoring. The report also demonstrates the effective of each of the above testing models on a conceptual level, as the use of each one revealed at least one new defect that resulted in a failed test case. These testing models can be applied to my own programs in the future to ensure higher-quality end products that should meet common quality standards.

Appendix A: Program Code

```
import tkinter as tk
import random
import Statistics as stats # type: ignore
import checkwin as cw  # type: ignore
from Computer import ComputerPlayer  # type: ignore

class Connect4:
    def __init__(self, root):
        self.root = root
        self.root.title("Connect 4")
        self.root.geometry("1280x720")  # Set the window size
        self.root.configure(bg="grey")  # Set the background color to black
        self.root.resizable(False, False)  # Disable window resizing
        self.current_player = "cyan" # Initialize the current player
        self.board = [[None for _ in range(7)] for _ in range(7)] # Initialize
the board
        self.computer_player = ComputerPlayer("maroon")  # Initialize the
computer player
        self.stats = stats.Statistics()  # Initialize the statistics class
        self.player_went_first = False # Initialize the player_went_first
variable

        self.create_widgets() # Create the widgets
        self.create_menu() # Create the menu
        self.create_statistics_frame() # Create the statistics frame

    def create_widgets(self):
        self.player_indicator = tk.Label(self.root, text="Current Player: Cyan",
bg="cyan", fg="black") # Create the player indicator
        self.player_indicator.grid(row=1, column=0, columnspan=3, sticky="w") #
Place the player indicator in the top left
        self.rules_frame = tk.Frame(self.root, bg="grey") # Create the rules
frame
        self.rules_frame.grid(row=0, column=0, rowspan=4, sticky="w", padx=(20,
20), pady=(100, 20)) # Move rules to the left

        rules_text = (
            "Connect 4 Rules:\n"
            "1. Players take turns dropping discs into columns.\n"
            "2. The first player to connect four discs\n    vertically,
horizontally, or diagonally wins.\n"
            "3. If the board is full and no player has four in a row,\n    the
game is a tie.\n\n"
```

```
            "Instructions:\n"
            "1. Click Start and select who goes first.\n"
            "2. Click on a column to drop your disc.\n"
            "3. The game will alternate turns between you and the computer.\n"
            "4. The current player is indicated at the top."
        )

        self.rules_label = tk.Label(self.rules_frame, text=rules_text, bg="grey",
fg="white", justify="left")
        self.rules_label.pack()

        self.canvas = tk.Canvas(self.root, width=700, height=700, bg="yellow")
        self.canvas.grid(row=0, column=2, rowspan=4, sticky="e", padx=(0, 0),
pady=(20, 0))  # Move board to the right
        for row in range(7):  # Create the board
            for col in range(7):
                x0 = col * 100
                y0 = row * 100
                x1 = x0 + 100
                y1 = y0 + 100
                self.canvas.create_oval(x0, y0, x1, y1, fill="white",
outline="black")  # Create the circles


    def create_menu(self):
        menubar = tk.Menu(self.root)
        self.root.config(menu=menubar)

        game_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Start", menu=game_menu)
        game_menu.add_command(label="Player First", command=lambda:
self.start_game("cyan"))
        game_menu.add_command(label="Random First", command=lambda:
self.start_game("random"))
        game_menu.add_command(label="Computer First", command=lambda:
self.start_game("maroon"))

        menubar.add_command(label="End Current Game", command=self.end_game)
        menubar.add_command(label="Exit", command=self.root.quit)

    def create_statistics_frame(self):
        self.stats_frame = tk.Frame(self.root, bg="grey")
        self.stats_frame.grid(row=0, column=3, rowspan=4, sticky="n", padx=(20,
20), pady=(20, 20))
```

```
        self.stats_labels = {
            "player_first_count": tk.Label(self.stats_frame, text="Player First
Count: 0", bg="grey", fg="white"),
            "player_first_wins": tk.Label(self.stats_frame, text="Player First
Wins: 0", bg="grey", fg="white"),
            "player_second_wins": tk.Label(self.stats_frame, text="Player Second
Wins: 0", bg="grey", fg="white"),
            "player_first_ties": tk.Label(self.stats_frame, text="Player First
Ties: 0", bg="grey", fg="white"),
            "player_second_ties": tk.Label(self.stats_frame, text="Player Second
Ties: 0", bg="grey", fg="white"),
            "total_games": tk.Label(self.stats_frame, text="Total Games Played:
0", bg="grey", fg="white"),
        }

        for i, label in enumerate(self.stats_labels.values()):
            label.grid(row=i, column=0, sticky="w")

    def update_statistics_frame(self):
        stats = self.stats.get_statistics()
        self.stats_labels["player_first_count"].config(text=f"Player First Count:
{stats['player_first_count']}")
        self.stats_labels["player_first_wins"].config(text=f"Player First Wins:
{stats['player_first_wins']}")
        self.stats_labels["player_second_wins"].config(text=f"Player Second Wins:
{stats['player_second_wins']}")
        self.stats_labels["player_first_ties"].config(text=f"Player First Ties:
{stats['player_first_ties']}")
        self.stats_labels["player_second_ties"].config(text=f"Player Second Ties:
{stats['player_second_ties']}")
        self.stats_labels["total_games"].config(text=f"Total Games Played:
{stats['total_games']}")

    def end_game(self): # End the current game
        self.canvas.delete("all") # Clear the board
        self.board = [[None for _ in range(7)] for _ in range(7)]
        self.player_indicator.config(text="Game Ended", bg="black", fg="white")
        self.canvas.unbind("<Button-1>") # Unbind the click event
        self.update_statistics_frame()

    def start_game(self, first_player):
        self.board = [[None for _ in range(7)] for _ in range(7)]
        self.canvas.delete("all")
        for row in range(7):
            for col in range(7):
```

```
                x0 = col * 100
                y0 = row * 100
                x1 = x0 + 100
                y1 = y0 + 100
                self.canvas.create_oval(x0, y0, x1, y1, fill="white",
outline="black")

        if first_player == "random":
            self.current_player = random.choice(["cyan", "maroon"])
        else:
            self.current_player = first_player

        self.player_indicator.config(
            text=f"Current Player: {self.current_player.capitalize()}",
            bg=self.current_player,
            fg=(self.current_player == "cyan" and "black" or "white"))
        self.player_indicator.grid(row=1, column=0, columnspan=3, sticky="w")

        self.canvas.bind("<Button-1>", self.handle_click) # Bind the click event

        if self.current_player == "maroon":
            self.root.after(500, self.computer_move)  # Add delay before computer
move

        self.stats.record_first_player(first_player)  # Record the first player
        self.update_statistics_frame()

    def handle_click(self, event):
        if self.current_player == "maroon": # Prevent player from making a move
while it's the computer's turn
            return

        col = event.x // 100
        for row in range(6, -1, -1):
            if self.board[row][col] is None: # Check if the cell is empty
                self.board[row][col] = self.current_player
                x0 = col * 100
                y0 = row * 100
                x1 = x0 + 100
                y1 = y0 + 100
                self.canvas.create_oval(x0, y0, x1, y1, fill=self.current_player,
outline="black") # Create the disc

                if cw.check_win(self.board, self.current_player):
                    self.player_indicator.config(fg="black")
```

```
                    self.player_indicator.config(text=f"{self.current_player.capi
talize()} Wins!", bg=self.current_player)
                    self.canvas.unbind("<Button-1>")
                    self.stats.player_won(self.player_went_first)
                    self.update_statistics_frame()
                    return

                self.current_player = "maroon" # Switch players
                self.player_indicator.config(text="Current Player: Maroon",
bg="maroon", fg="white")
                self.root.after(500, self.computer_move)  # Add delay before
computer move
                break

    def computer_move(self):
        col = self.computer_player.make_move(self.board) # Get the computer's
move
        if col is None:
            return

        for row in range(6, -1, -1): # Find the first empty cell in the column
            if self.board[row][col] is None:
                self.board[row][col] = self.current_player
                x0 = col * 100
                y0 = row * 100
                x1 = x0 + 100
                y1 = y0 + 100
                self.canvas.create_oval(x0, y0, x1, y1, fill=self.current_player,
outline="black")

                if cw.check_win(self.board, self.current_player):
                    self.player_indicator.config(text=f"{self.current_player.capi
talize()} Wins!", bg=self.current_player) # Display the winner
                    self.canvas.unbind("<Button-1>")
                    self.stats.player_won(not self.player_went_first)
                    self.update_statistics_frame()
                    return

                self.current_player = "cyan" # Switch players
                self.player_indicator.config(text="Current Player: Cyan",
bg="cyan", fg="black")
                break

if __name__ == "__main__":
    root = tk.Tk()
```

```
    game = Connect4(root)
    root.mainloop()
```

```
class Statistics:
    def __init__(self):
        self.player_first_count = 0
        self.player_first_wins = 0
        self.player_second_wins = 0
        self.player_first_ties = 0
        self.player_second_ties = 0
        self.total_games = 0

    def record_first_player(self, first_player):
        if first_player == "cyan":
            self.player_first_count += 1
        self.total_games += 1

    def player_won(self, went_first):
        if went_first:
            self.player_first_wins += 1
        else:
            self.player_second_wins += 1
        self.total_games += 1

    def game_tied(self, went_first):
        if went_first:
            self.player_first_ties += 1
        else:
            self.player_second_ties += 1
        self.total_games += 1

    def get_statistics(self):
        return {
            "player_first_count": self.player_first_count,
            "player_first_wins": self.player_first_wins,
            "player_second_wins": self.player_second_wins,
            "player_first_ties": self.player_first_ties,
            "player_second_ties": self.player_second_ties,
            "total_games": self.total_games,
        }
```

```
def check_win(board, player):
    # Check horizontal locations for win
    for row in range(len(board)):
```

```
        for col in range(len(board[0]) - 3):
            if board[row][col] == board[row][col + 1] == board[row][col + 2] ==
board[row][col + 3] == player:
                return board[row][col]

    # Check vertical locations for win
    for col in range(len(board[0])):
        for row in range(len(board) - 3):
            if board[row][col] == board[row + 1][col] == board[row + 2][col] ==
board[row + 3][col] == player:
                return board[row][col]

    # Check positively sloped diagonals
    for row in range(len(board) - 3):
        for col in range(len(board[0]) - 3):
            if board[row][col] == board[row + 1][col + 1] == board[row + 2][col +
2] == board[row + 3][col + 3] == player:
                return board[row][col]

    # Check negatively sloped diagonals
    for row in range(3, len(board)):
        for col in range(len(board[0]) - 3):
            if board[row][col] == board[row - 1][col + 1] == board[row - 2][col +
2] == board[row - 3][col + 3] == player:
                return board[row][col]

    return 0
```

```
import random

class ComputerPlayer:
    def __init__(self, player_color):
        self.player_color = player_color

    def make_move(self, board):
        valid_columns = [col for col in range(len(board[0])) if board[0][col] is
None]
        if not valid_columns:
            return None
        return random.choice(valid_columns)
```