

Initial Setup

The project will be developed on a Windows desktop system, using Visual Studio Code as the Integrated Development Environment (IDE). Unlike using C++, this project will be implemented using Python.

Project Overview

The Connect4 game is a classic board game where players aim to form a line of four tokens horizontally, vertically, or diagonally. This Python implementation allows two players to play against each other or against the computer. Players take turns dropping tokens into a grid, aiming to outmaneuver their opponent and achieve victory. The game is played on a 6x7 grid, and the first player to connect four tokens in a row wins.

For the Connect4 project, the requirements analysis outlines the key features and functionalities of the game:

Gameplay Mechanics:

Players take turns dropping tokens into a 6x7 grid, aiming to connect four tokens horizontally, vertically, or diagonally.

Two players can play against each other or against the computer.

Players input their moves by specifying the row and column where they want to place their token.

Game Flow:

The game starts with a welcome message and prompts players to enter their names and choose their tokens (X or O).

Players alternate turns until one player wins or the game ends in a draw.

Input Validation:

Input validation ensures that players enter valid moves within the boundaries of the game board and into empty cells.

Winning Condition:

The game checks for the winning condition after each move to determine if a player has won.

End Game Conditions:

The game ends when one player achieves four tokens in a row or when the game board is full (resulting in a draw).

Player Interaction:

Players receive feedback after each move, informing them of the success or failure of their move.

CPU Player (Optional):

The game includes an option for a single-player mode where the player competes against the computer.

The CPU generates random moves as its strategy.

Visual Representation:

The game displays the current state of the board after each move, allowing players to visualize the game progress.

ASCII art can be used to enhance the visual representation of the game board.

Behavior Driven Development (Gherkin Specifications)

Feature: Playing Connect4 Game
As a CLI USER/PLAYER, I want to enter playe game
Scenario: Player starts the game
Given the game has started When the game initializes Then the game displays the current board
Scenario: Player makes a valid move
Given it is player's turn When the player selects a valid column to drop the token Then the token is placed in the chosen column
Scenario: Player makes an invalid move (column full)
Given it is player's turn When the player selects a column that is already full Then the game displays a message indicating the move is invalid

Scenario: Player wins horizontally

Given the player has placed four tokens in a row horizontally
Then the game declares the player as the winner

Scenario: Player wins vertically

Given the player has placed four tokens in a column vertically
Then the game declares the player as the winner

Scenario: Player wins diagonally (positive slope)

Given the player has placed four tokens diagonally in a positive slope
Then the game declares the player as the winner

Scenario: Game ends in a draw

Given the game board is full and no player has won
Then the game declares a draw

Scenario: CPU makes a valid move

Given it is CPU's turn
When the CPU selects a valid column to drop the token
Then the token is placed in the chosen column

Scenario: CPU makes an invalid move (column full)

Given it is CPU's turn
When the CPU selects a column that is already full
Then the game displays a message indicating the move is invalid

Scenario: CPU wins horizontally

Given the CPU has placed four tokens in a row horizontally
Then the game declares the CPU as the winner

Scenario: CPU wins vertically

Given the CPU has placed four tokens in a column vertically

Then the game declares the CPU as the winner
Scenario: CPU wins diagonally (positive slope)
Given the CPU has placed four tokens diagonally in a positive slope Then the game declares the CPU as the winner
Scenario: CPU wins diagonally (negative slope)
Given the CPU has placed four tokens diagonally in a negative slope Then the game declares the CPU as the winner

Data Model

Input:

- User Input (Standard Input)

Output Messages:

- Welcome to the Connect 4 game!
- Current Player: [Player Name]
- [Connect4 Board Display]
- Invalid move! Please choose an empty cell.
- Invalid input! Please enter numbers.
- [Player Name]'s token is [Player Token]
- [Player Name]'s turn ([Player Token]):
- [Player Name] wins!
- It's a draw!

Error Handling:

- Invalid input! Please enter a number.

- Invalid token! Please enter X or O.

Name Model: In the Connect 4 game, the name model represents the names chosen by the players. Each player selects a unique name as their identifier for the game. The name type contains the set of names chosen by the players, ensuring that each player is identified uniquely.

Let Name be the set of player names:

$$\text{Name} = \{\text{Name1}, \text{Name2} \dots\}$$

Here, each Name represents a player's chosen name, making the name effectively a subset of all possible player names. This approach simplifies the validation of inputs, ensuring that only valid player names are used during gameplay.

For example:

$$\text{Name} = \{\text{"Player1"}, \text{"Player2"}\}$$

In this case, the Name set has a cardinality of n , representing the number of players in the game. Each element of the set is a unique player name, ensuring that the name remains valid and constrained to the selected player names.

Turn Model: In the Connect 4 game, turns are organized as a sequence of players, where each player takes their turn in order. The active player, who is currently taking their turn, is the player at the head of the sequence.

The player sequence can be represented as follows:

Let Players be a sequence of players:

$$\text{Players} = \text{seq} < \text{Player1}, \text{Player2} >$$

Here, each Player represents an instance of a player in the game. The sequence allows for flexibility in the number of players, accommodating changes such as adding more players to the game.

For example:

$$\text{Players} = \text{seq} < \text{PlayerA}, \text{PlayerB} >$$

Axiomatic Definitions and Functions

Axiomatic Definitions:

Player Model: Each player in the Connect 4 game represents one of the two opponents. Attributes:

Name: Identifies the player. The name is chosen by the user and serves as the unique identifier for the player.

Inventory: Not applicable as Connect 4 doesn't have items.

Game Model: The Connect 4 game manages the flow of the game and interactions between the players and the game environment. Attributes:

Current room: Not applicable as Connect 4 doesn't have distinct rooms.

Collection of rooms: Not applicable as Connect 4 doesn't have distinct rooms.

Functions:

start_game(Game): Initializes the Connect 4 game by setting up the board and starting the gameplay loop.

move(Game): Handles player moves in Connect 4 by adding tokens to the board.

play(Game): Main loop of the Connect 4 game where players take turns placing tokens on the board until there is a winner or a draw. Handles player input, moves, and game state changes.

T2 Implementation

Now that we're ready to implement the program, we'll distinguish between two types of functions: pure and impure. Pure functions only operate within their scope and don't modify any external program state, while impure functions do. Additionally, functions can be categorized as total or non-total. Total functions cover all possible input-value pairs, while non-total functions don't. We'll aim to create pure, total functions wherever possible.

Connect4 Class:

```
class Connect4:
    def __init__(self):
        self.board = [[' ' for _ in range(7)] for _ in range(6)]
        self.current_player = None
        self.winning_condition = 4
```

Purity: Pure

Totalization: Totalized

print_board(): Prints the game board to the console.

```
def print_board(self):
    for row in self.board:
        print("| " + " | ".join(row) + " |")
        print("+-+--+--+--+--+--+--+--+")
```

Purity: Impure

Totalization: Not totalized

is_valid_move(): Checks if a move is valid.

```
def is_valid_move(self, row, column):
    return row >= 0 and row < 6 and column >= 0 and column < 7 and
self.board[row][column] == ' '
```

Purity: Pure

Totalization: Totalized

make_move(): Updates the game board with the player's move.

```
def make_move(self, row, column, token):
    self.board[row][column] = token
```

Purity: Impure

Totalization: Totalized

check_winner(): Checks if the current player has won.

```
def check_winner(self, token):
```

```
        # Check horizontal
        for row in range(6):
            for col in range(4):
                if all(self.board[row][col + i] == token for i in
range(self.winning_condition)):
                    return True

        # Check vertical
        for row in range(3):
            for col in range(7):
                if all(self.board[row + i][col] == token for i in
range(self.winning_condition)):
                    return True
```

```
        # Check diagonal (positive slope)
        for row in range(3):
            for col in range(4):
                if all(self.board[row + i][col + i] == token for i in
range(self.winning_condition)):
                    return True
```

```
        # Check diagonal (negative slope)
        for row in range(3):
            for col in range(3, 7):
                if all(self.board[row + i][col - i] == token for i in
range(self.winning_condition)):
                    return True
```

```
    return False
```


Purity: Pure

Totalization: Not totalized

Player Class:

```
class Player:
    def __init__(self, name, token):
        self.name = name
        self.token = token
```

`__init__()`: Initializes player attributes.

Purity: Pure

Totalization: Totalized

player_move() Function:

```
def player_move(game, player):
    while True:
        try:
            row = int(input(f"{player.name}'s turn ({player.token}):  
Enter row (1-6): ")) - 1
            column = int(input(f"{player.name}'s turn ({player.token}):  
Enter column (1-7): ")) - 1
            if game.is_valid_move(row, column):
                game.make_move(row, column, player.token)
                break
            else:
                print("Invalid move! Please choose an empty cell.")
        except ValueError:
            print("Invalid input! Please enter numbers.")
```

player_move(): Handles the player's move input.

Purity: Impure

Totalization: Not totalized

cpu_move() Function:

```
def cpu_move(game, player):  
    print(f"{player.name}'s turn ({player.token}):")  
    while True:  
        row = random.randint(0, 5)  
        column = random.randint(0, 6)  
        if game.is_valid_move(row, column):  
            game.make_move(row, column, player.token)  
            break
```

cpu_move(): Generates a random move for the CPU.

Purity: Impure

Totalization: Not totalized

start_game() Function:

```
def start_game():  
    print("Welcome to Connect 4!")  
    while True:  
        try:  
            num_players = int(input("Enter number of players (1 or 2):  
"))  
            if num_players == 1 or num_players == 2:  
                break
```

```
        else:
            print("Invalid number of players! Please enter 1 or 2.")
    except ValueError:
        print("Invalid input! Please enter a number.")
```

```
game = Connect4()
```

```
player1_name = input("Enter Player 1's name: ")
player1_token = input("Enter Player 1's token (X or O): ").upper()
while player1_token not in ['X', 'O']:
    player1_token = input("Invalid token! Please enter X or O: ").upper()
player1 = Player(player1_name, player1_token)
```

```
player2_token = 'X' if player1_token == 'O' else 'O'
if num_players == 1:
    player2_name = "CPU"
else:
    player2_name = input("Enter Player 2's name: ")
player2 = Player(player2_name, player2_token)
```

```
print(f"{player2.name}'s token is {player2.token}")
```

```
game.current_player = player1
```

```
while True:
    print("\n" + " ".join(str(i) for i in range(1, 8)))
    print("+---+---+---+---+---+---+---+---+")
```

```

        game.print_board()
        if game.current_player == player1:
            player_move(game, player1)
        else:
            cpu_move(game, player2) if num_players == 1 else
player_move(game, player2)

```

```

        if game.check_winner(game.current_player.token):
            print("\n" + " ".join(str(i) for i in range(1, 8)))
            print("+---+---+---+---+---+---+---+")
            game.print_board()
            print(f"{game.current_player.name} wins!")
            break

```

```

        if all(token != ' ' for row in game.board for token in row):
            print("\n" + " ".join(str(i) for i in range(1, 8)))
            print("+---+---+---+---+---+---+---+")
            game.print_board()
            print("It's a draw!")
            break

```

```

        game.current_player = player2 if game.current_player ==
player1 else player1

```

```

start_game()

```

start_game(): Initializes the game and controls its flow.

Purity: Impure

Totalization: Not totalized

Explanation:

Pure functions only depend on their input parameters and do not modify any external state. In this implementation, `__init__()`, `is_valid_move()`, `check_winner()`, and `Player.__init__()` are pure functions.

Impure functions either modify external state or have side effects. Functions like `print_board()`, `make_move()`, and `player_move()` are impure because they either print to the console or modify the game state.

Totalized functions cover all possible input-value pairs. Pure functions, such as `is_valid_move()`, `check_winner()`, and `Player.__init__()`, are totalized since they cover all possible valid inputs. Impure functions like `print_board()`, `make_move()`, `player_move()`, and `cpu_move()` are not totalized since they don't cover all possible inputs (for example, `print_board()` only covers printing the current state of the board).

Output

Welcome to Connect 4!

Enter number of players (1 or 2): 2

Enter Player 1's name: Kim

Enter Player 1's token (X or O): x

Enter Player 2's name: Stephan

Stephan's token is O

1 2 3 4 5 6 7

```
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
```

Kim's turn (X): Enter row (1-6):

Kim's turn (X): Enter row (1-6): 1

Kim's turn (X): Enter column (1-7): 2

1 2 3 4 5 6 7

```
+---+---+---+---+---+---+
|   | X |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
```

Stephan's turn (O): Enter row (1-6):

```
Kim's turn (X): Enter row (1-6): 4
Kim's turn (X): Enter column (1-7): 1
```

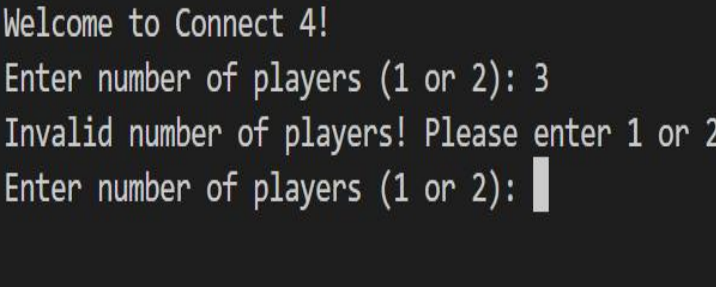
```
1  2  3  4  5  6  7
+--+--+--+--+--+--+--+
| X | X | O | X |   | O |   |
+--+--+--+--+--+--+--+
| X |   | O | O |   |   |   |
+--+--+--+--+--+--+--+
| X |   | O |   |   | O |   |
+--+--+--+--+--+--+--+
| X |   | X |   |   |   |   |
+--+--+--+--+--+--+--+
|   |   |   |   |   |   |   |
+--+--+--+--+--+--+--+
|   |   |   |   |   |   |   |
+--+--+--+--+--+--+--+
Kim wins!
```

Testing

Both manual and automated tests are crucial for verifying that the software behaves as intended, following the specifications outlined in Gherkin and other planning documents. We'll begin by conducting manual tests, which involve using the software to validate whether the expected outputs are produced for specific user inputs.

Manual Testing

Test Case ID		1	Passed/Failed
Feature	Start Game		
Steps to Do	Enter number of players: 2	"Welcome to Connect 4!" printed	
Expected Output	"Enter Players name:"	Welcome message printed	
Actual Result	<pre>Welcome to Connect 4! Enter number of players (1 or 2): 2 Enter Player 1's name: Kim Enter Player 1's token (X or O): O Enter Player 2's name: Stephan Stephan's token is X 1 2 3 4 5 6 7 +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ Kim's turn (O): Enter row (1-6): █</pre>		

Test Case ID		2	Passed/Failed
Feature	Start Game		
Steps to Do	Enter number of players: 3	"Invalid number of players! Please enter 1 or 2." printed	
Expected Output	-	Error message printed	
Actual Result			

Test Case ID		3	Passed/Failed
Feature	Player Move		
Steps to Do	Enter row: 4, Enter column: 2	Token 'X' placed at (3, 1) on the board	
Expected Output	Board with 'X' at (3, 1)	Updated board displayed	

Feature	Player Move	
Actual Result	<pre> kim's turn (X): Enter row (1-6): 4 kim's turn (X): Enter column (1-7): 2 1 2 3 4 5 6 7 +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ X +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ stephan's turn (O): Enter row (1-6): </pre>	

Test Case ID	4	Passed/Failed
--------------	---	---------------

Feature	CPU Move	
---------	----------	--

Steps to Do	CPU's turn	Random token placed on the board
-------------	------------	----------------------------------

Expected Output	Board with CPU's token placed	Updated board displayed
-----------------	-------------------------------	-------------------------

Actual Result	<pre> CPU's turn (O): 1 2 3 4 5 6 7 +---+---+---+---+---+---+ O +---+---+---+---+---+---+ X +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ Stephan's turn (X): Enter row (1-6): </pre>	
---------------	---	--

Test Case ID	5	Passed/Failed
--------------	---	---------------

Feature	Check Winner	
---------	--------------	--

Steps to Do	Create winning condition	Check winner function returns True
-------------	--------------------------	------------------------------------

Expected Output	"Player X wins!" or "Player	Winner message displayed
-----------------	-----------------------------	--------------------------

Feature	Check Winner	
	0 wins!"	
Actual Result	<pre>Stephan's turn (X): Enter row (1-6): 4 Stephan's turn (X): Enter column (1-7): 3 1 2 3 4 5 6 7 +---+---+---+---+---+---+ X O +---+---+---+---+---+---+ X +---+---+---+---+---+---+ X O +---+---+---+---+---+---+ X +---+---+---+---+---+---+ O +---+---+---+---+---+---+ +---+---+---+---+---+---+ Stephan wins!</pre>	
Test Case ID	6	Passed
Feature	Game continue until winner declare	
Steps to Do	Continue playing	Check for a continue until win
Expected Output	"Game conitune "	Game continue

Feature	Game continue until winner declare	
Actual Result	<pre>1 2 3 4 5 6 7 +--+--+--+--+--+--+ +--+--+--+--+--+--+ X +--+--+--+--+--+--+ +--+--+--+--+--+--+ +--+--+--+--+--+--+ +--+--+--+--+--+--+ +--+--+--+--+--+--+ CPU's turn (O): 1 2 3 4 5 6 7 +--+--+--+--+--+--+ O +--+--+--+--+--+--+ X +--+--+--+--+--+--+ +--+--+--+--+--+--+ +--+--+--+--+--+--+</pre>	
Test Case ID	7	Passed/Failed
Feature	Check Valid Move	
Steps to Do	Place token on an occupied cell	Check if move is valid
Expected Output	"Invalid move! Please choose an empty cell."	Error message displayed

Feature	Check Valid Move	
Actual Result	<pre> 1 2 3 4 5 6 7 +---+---+---+---+---+---+ +---+---+---+---+---+---+ x +---+---+---+---+---+---+ o +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ +---+---+---+---+---+---+ john's turn (X): Enter row (1-6): 2 john's turn (X): Enter column (1-7): 2 Invalid move! Please choose an empty cell. john's turn (X): Enter row (1-6): █ </pre>	

Test Case ID	8	Passed/Failed
--------------	---	---------------

Feature	Check Valid Move	
Steps to Do	Enter row: -1, Enter column: 3	Check if move is valid
Expected Output	"Invalid move! Please choose an empty cell."	Error message displayed
Actual Result	<pre> john's turn (X): Enter row (1-6): -1 john's turn (X): Enter column (1-7): 3 Invalid move! Please choose an empty cell. john's turn (X): Enter row (1-6): █ </pre>	

Test Case ID	9	Passed/Failed
--------------	---	---------------

Feature	Check Valid Move	
Steps to Do	Enter row: 4, Enter column: 8	Check if move is valid
Expected Output	"Invalid move! Please choose an empty cell."	Error message displayed

Feature	Check Valid Move	
Actual Result	<pre>john's turn (X): Enter row (1-6): 4 john's turn (X): Enter column (1-7): 8 Invalid move! Please choose an empty cell. john's turn (X): Enter row (1-6): █</pre>	

Test Case ID	10	Passed/Failed
--------------	----	---------------

Feature	Check Valid Move	
Steps to Do	Enter row: a, Enter column: b	Check if move is valid
Expected Output	"Invalid input! Please enter numbers."	Error message displayed
Actual Result	<pre>john's turn (X): Enter row (1-6): a Invalid input! Please enter numbers. john's turn (X): Enter row (1-6): b Invalid input! Please enter numbers. john's turn (X): Enter row (1-6): █</pre>	

Test Case ID	11	Passed/Failed
--------------	----	---------------

Feature	Check Valid Move	
Steps to Do	Enter row: 3.5, Enter column: 2.5	Check if move is valid
Expected Output	"Invalid input! Please enter numbers."	Error message displayed
Actual Result	<pre>john's turn (X): Enter row (1-6): 3.5 Invalid input! Please enter numbers. john's turn (X): Enter row (1-6): 2.5 Invalid input! Please enter numbers. john's turn (X): Enter row (1-6): █</pre>	

Automated testing:

For automated testing of frakle “unittest” library is best and using it.

The unittest library in Python is a built-in testing framework that allows you to write test cases for your code in a structured and organized manner. It provides a set of tools for constructing and running tests, as well as making assertions about the behavior of your code.

Setup for testing

```
import unittest
from unittest.mock import patch
from io import StringIO
from connect_4 import Connect4, Player, player_move, cpu_move, start_game

class TestConnect4(unittest.TestCase):
    def setUp(self):
        self.game = Connect4()

    def test_is_valid_move(self):
        self.assertTrue(self.game.is_valid_move(0, 0))
        self.assertTrue(self.game.is_valid_move(5, 6))
        self.assertFalse(self.game.is_valid_move(-1, 0))
        self.assertFalse(self.game.is_valid_move(0, 7))

    def test_make_move(self):
        self.game.make_move(0, 0, 'X')
        self.assertEqual(self.game.board[0][0], 'X')
```

```
class TestConnect4(unittest.TestCase):
    def test_check_winner(self):
        # Test horizontal win
        for i in range(3):
            self.game.make_move(0, i, 'X')
            self.assertTrue(self.game.check_winner('X'))

        # Test vertical win
        for i in range(3):
            self.game.make_move(i, 0, 'O')
            self.assertTrue(self.game.check_winner('O'))

    @patch('builtins.input', side_effect=['1', '1', '1', '2', '2', '1', '3', '2', '3', '2', '4', '2', '4', '3', ' '])
    def test_player_move(self, mock_input):
        player = Player("Player", 'X')
        player_move(self.game, player)
        self.assertEqual(self.game.board[0][0], 'X')

    @patch('random.randint', side_effect=[0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5])
    def test_cpu_move(self, mock_randint):
        player = Player("CPU", 'O')
        cpu_move(self.game, player)
        self.assertEqual(self.game.board[0][0], 'O')
```


Running the code and output of tests:

```
Welcome to Connect 4!
Enter number of players (1 or 2): 1
Enter Player 1's name: Stev
Enter Player 1's token (X or O): O
CPU's token is X

 1  2  3  4  5  6  7
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
Stev's turn (O): Enter row (1-6): 2
Stev's turn (O): Enter column (1-7): 2
```


Stev's turn (O): Enter row (1-6): 3
Stev's turn (O): Enter column (1-7): 3

1	2	3	4	5	6	7									
+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+
			X												
+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+
			O												
+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+
					O										
+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+
+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+
+	-	-	+	-	-	+	-	-	+	-	-	+	-	-	+

CPU's turn (X):

```
Stev's turn (0): Enter row (1-6): 3
Stev's turn (0): Enter column (1-7): 1
```

```
 1  2  3  4  5  6  7
+---+---+---+---+---+---+
|   | x |   |   | x |   |
+---+---+---+---+---+---+
|   | o | x |   | x |   |
+---+---+---+---+---+---+
| o | o | o | o |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
|   |   |   |   |   |   |
+---+---+---+---+---+---+
Stev wins!
```

Game is stopped when user quit it.

Appendix A

Full automated testing code:

```

class TestConnect4(unittest.TestCase):

    def test_check_winner(self):
        # Test horizontal win
        for i in range(3):
            self.game.make_move(0, i, 'X')
            self.assertTrue(self.game.check_winner('X'))

        # Test vertical win
        for i in range(3):
            self.game.make_move(i, 0, 'O')
            self.assertTrue(self.game.check_winner('O'))

    @patch('builtins.input', side_effect=['1', '1', '1', '2', '2', '1', '3', '2', '3', '2', '4', '2', '4', '3', ])
    def test_player_move(self, mock_input):
        player = Player("Player", 'X')
        player_move(self.game, player)
        self.assertEqual(self.game.board[0][0], 'X')

    @patch('random.randint', side_effect=[0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, ])
    def test_cpu_move(self, mock_randint):
        player = Player("CPU", 'O')
        cpu_move(self.game, player)
        self.assertEqual(self.game.board[0][0], 'O')

```

```

    @patch('builtins.input', side_effect=['2', 'Player1', 'X', '2', 'Player2', 'O'])
    def test_start_game(self, mock_input):
        with patch('sys.stdout', new=StringIO()) as fake_out:
            start_game()
            output = fake_out.getvalue().strip()
            self.assertIn("Player2's token is O", output)

if __name__ == '__main__':
    unittest.main()

```

T4 Git version control discussion

Git commits serve as checkpoints in a project's history, capturing the state of the codebase at specific moments. These commits are invaluable for the Farkle project, allowing developers to revert back to previous versions if needed and providing a clear record of changes. This functionality enables quick recovery from errors or unintended modifications and helps track changes over time, ensuring the project's stability and progress.

Pushing commits to the Git repository ensures that changes made on a local machine are shared with others, providing team members access to the latest version of the codebase. Similarly, pulling updates the local machine with changes made by others, ensuring everyone works with the most recent version. For team collaboration on the Farkle project, pull requests are particularly useful. They allow developers to propose changes, have them reviewed by peers, and then merge them into the main repository, maintaining code quality and stability through thorough examination before integration.

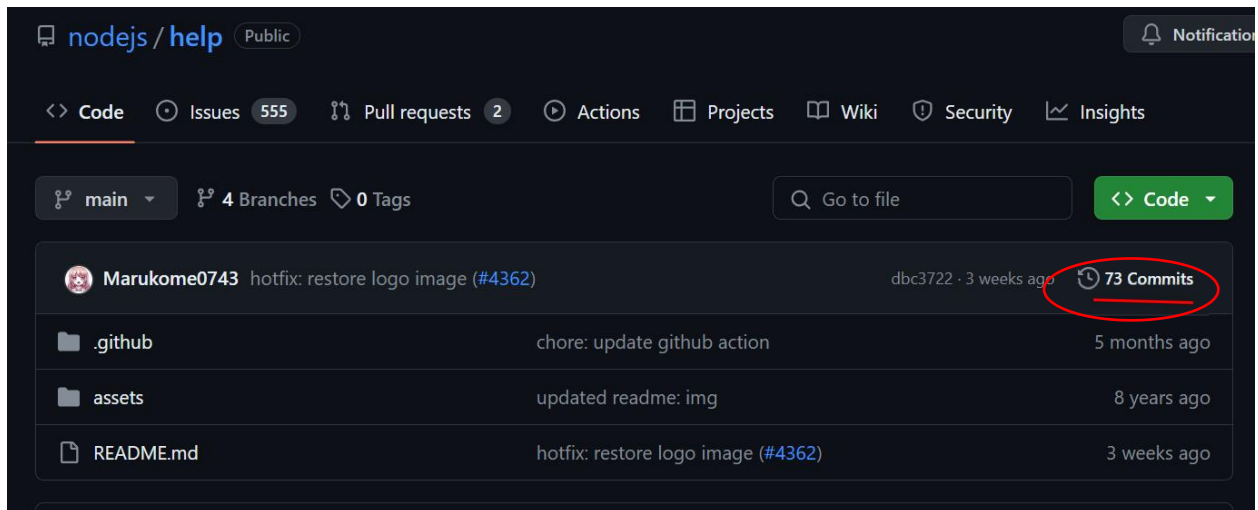
.

Case study

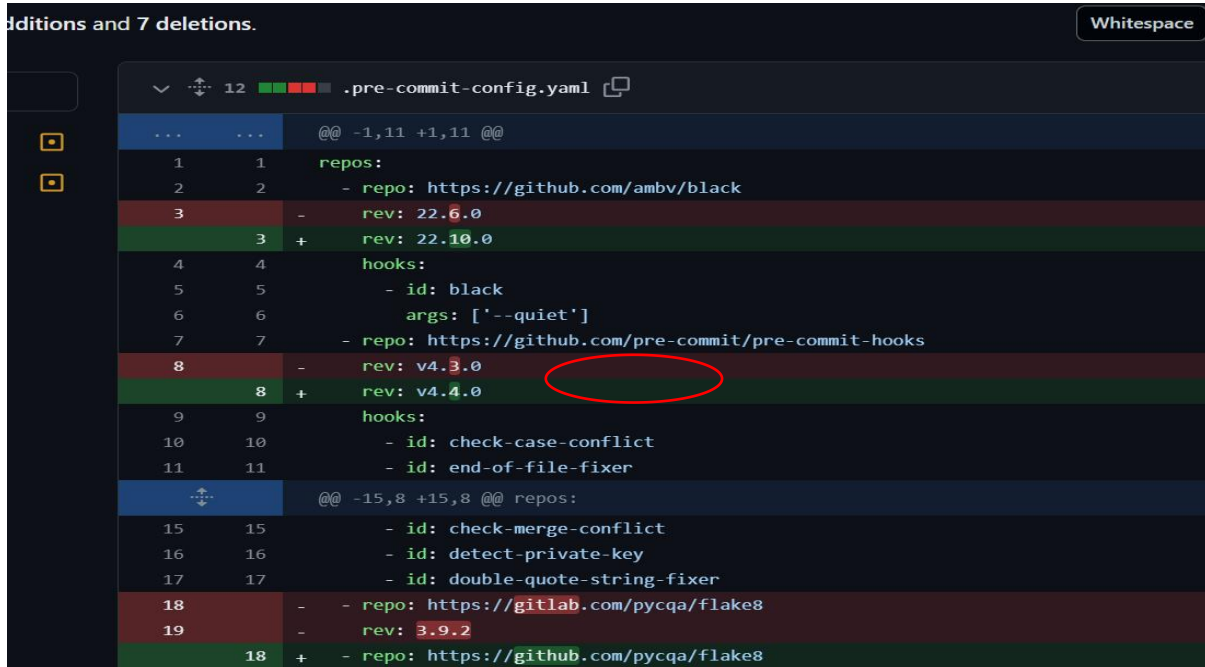
For a case study of git, I study nodejs repository to get idea about git and github.

<https://github.com/nodejs/help>

The commit history for kislyuk can be easily accessed by clicking the commits symbol when navigating GitHub, underlined in red in figure.



Commits are displayed in chronological order, with the latest commit appearing at the top of the list. By scrolling down or selecting "older," you can view earlier commits in the repository. Change in project code is highlighted with red and green highlighters as



Total number of branches/folders in repository are display as:

main

4 Branches

0 Tags

Go to file

Code

Marukome0743

hotfix: restore logo image (#4362)

dbc3722 · 3 weeks ago

73 Commits

.github	chore: update github action	5 months ago
assets	updated readme: img	8 years ago
README.md	hotfix: restore logo image (#4362)	3 weeks ago

References:

The benefits and need of documentation of code:

<https://swiftspeedappcreator.com/appblog/importance-of-documentation-in-software-maintenance/>

Accessed [4/05/2024]

Text connect 4 game guide:

https://en.wikipedia.org/wiki/Connect_Four Accessed [04/05/2024]

Git guid sample:

<https://github.com/git-guides/git-clone> Accessed [29/04/2024]