PROJECT III (PART B)

Jhonatan Parada

ET574

1. New object attributes created (2). These ones will be used to define the current's player chip 2d coordinates.

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

  def switch_player(self):
```

2. New methods created (3). Set_coordinates will be a void method, and get_coordinates will be a non-void method that will return a tuple of the current's player's chip coordinates. The first value will be the row, and the second one will be the column

```
class Connect4:

  def set_coordinates(self, row, column):
    self.current_row = row
    self.current_column = column

  def get_coordinates(self):
    return self.current_row, self.current_column

  def check_win(self, player):
```

3. In the drop_chip method, after placing the chip in the board, I will call the set_coordinates method and pass it 2 arguments, the coordinate (or index) of the row and column of the cell holding the chip.

```
class Connect4:
  def drop_chip(self, column):
    for row in reversed(self.board):
      if row[column] == ' ':
        row[column] = self.current_player

        # sets the coordinates of current's player chip
        self.set_coordinates(self.board.index(row), column)

        break

    return True

  def play_game(self):
```

4. The check_win method uses the first two nested while loops to iterate over the cells surrounding the player's current chip cell. The third and last loop iterates over the chips it encounters that match the same as the current player. The last loop also uses information from the first two loops to calculate the coordinates of a cell in the different 4 alignments that chips can connect.

```
class Connect4:
  def check_win(self, player):
    row, column = self.get_coordinates()
    connect = 4

    i = -1 # i represents the the column
    while i != (1 + 1): # range (-1, 0, 1) for column

      y = -1 # y represents the row
      while y != (1 + 1): # range (-1, 0, 1) for row

        # skips self and upward cell
        if i == 0 and (y == -1 or y == 0): y += 1; continue

        x = 1 # x represents the number of chips that need to be aligned to win the game
        while x != (connect):

          # Defines the coordinates of a cell after a transition from the current chip
          new_row = row + (y * x)
          new_column = column + (i * x)

          # If either coordinate is negative, that means that the coordinate is out of range
          if new_row < 0 or new_column < 0: break

          # checks that the target cell exists and that it is equal to player
          try:
            if self.board[new_row][new_column] != player: break
          except: break

          x += 1
        else:
          # getting to this point means that everything went good during the loop
          # therefore, there a specified number of chips are aligned
          return True

        y += 1
      i += 1
    return False
```

Let’s imagine we have a 5x5 board where the last chip dropped was the one in blue, let’s call this chip the original chip, we will look for a connection from this point.

	O		O	
	X	X	O	
	O	O	O	

Let’s say that the coordinate of the original chip is (A,B)

Cartesian Plane	Python Syntax
(A, B)	Board[B][A]

Let’s say we want to access the cells closer to the left of the original chip (horizontal and diagonal), but how will the code know the coordinate of those cells?

Let’s highlight the cells with different colors to see it clearer.

	O		O	
	X	X	O	
	O	O	O	

If

= board[B][A]

Then,

= board[B – 1][A – 1]

= board[B][A – 1]

= board[B + 1][A – 1]

Let’s express yellow, green and purple in a different way:

= board[B + (– 1)][A – 1]

= board[B + (0)][A – 1]

= board[B + (1)][A – 1]

See the pattern? That’s why the sentinels of the first while loops start as –1 and finish as 1 (though they could start as 1 and finish as –1 but we would have to use the -= operator instead of +=) because it will iterate over 3 rows and 3 columns.

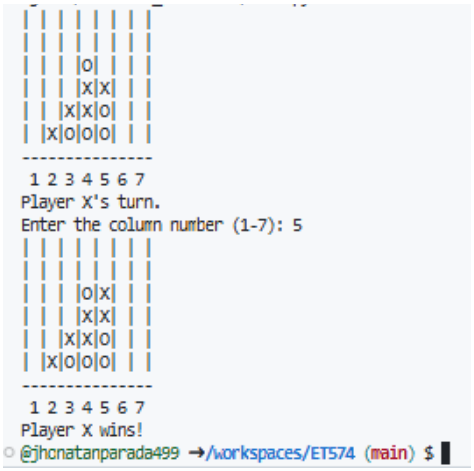
Finally, let’s imagine this scenario where X was the last chip dropped:

			X	
		X	O	
	X	O	X	
X	O	X	O	

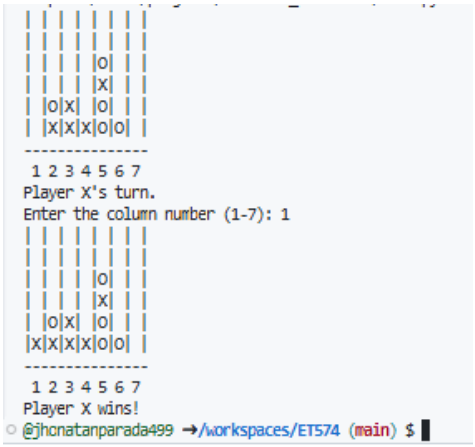
X = board[B][A]
X = board[B + 1][A - 1]
X = board[B + 2][A - 2]
X = board[B + 3][A - 3]

The check_win method makes use of these two principles to detect every possible connection from the current chip dropped. The method then utilizes if statements to for example skip and not check on the cell above the orginal cell or that the new cell coordinate is inside the board.

5. Winning Diagonally



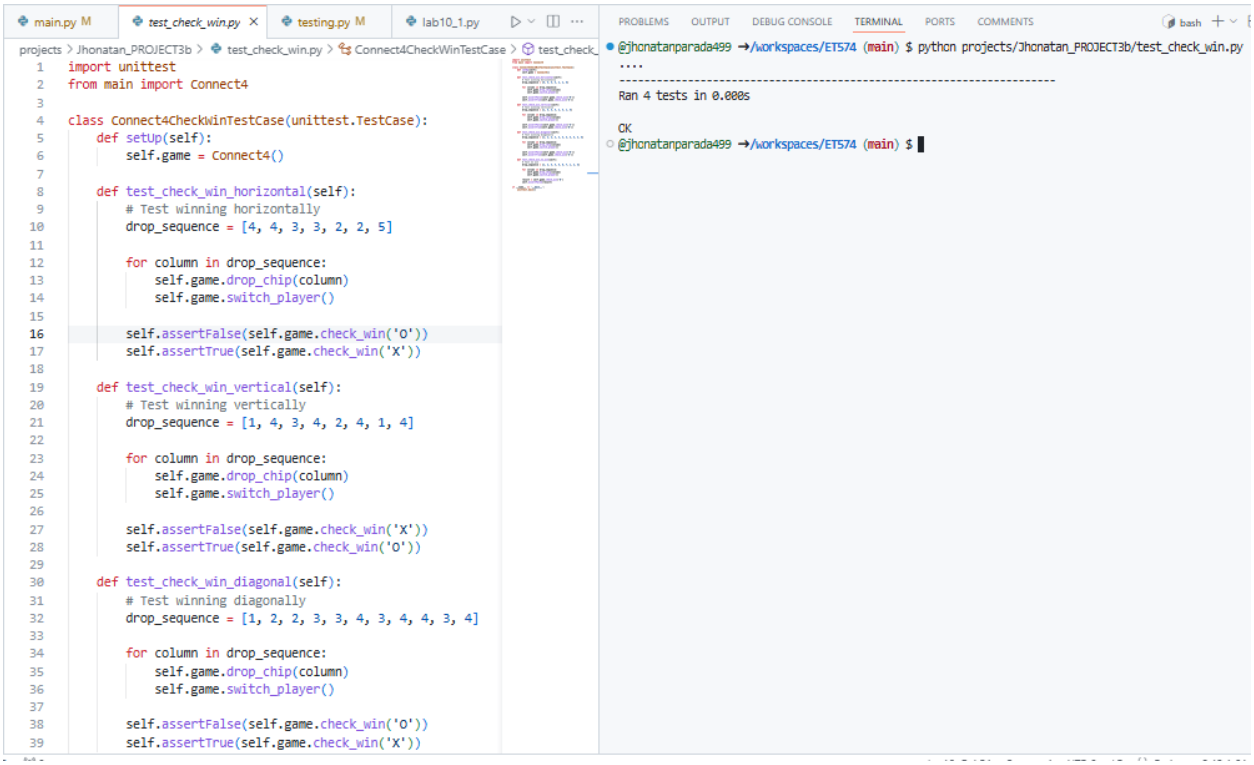
6. Winning horizontally



7. Winning Vertically

```
| | | | | | | |
| | | | | | | |
| |o| | |o| | |
| |o| |x|x| | |
| |o|x|x|o|x| |
| |x|x|x|o|o| |
--------------
 1 2 3 4 5 6 7
Player O's turn.
Enter the column number (1-7): 2
| | | | | | | |
| |o| | | | | |
| |o| | |o| | |
| |o| |x|x| | |
| |o|x|x|o|x| |
| |x|x|x|o|o| |
--------------
 1 2 3 4 5 6 7
Player O wins!
@jhonatanparada499 →/workspaces/ET574 (main) $
```

8. Testing main.py with unittest module.



Knowing this, let's simplify the check_win method:

```
def check_win(self, player):
  row, column = self.get_coordinates()
  indexes = range(-1,1 + 1)
  connect = 4

  for c in indexes:
    for r in indexes:
      if c == 0 and (r == -1 or r == 0): continue
      for i in range(1,connect):
        new_row = row + (r * i)
        new_column = column + (c * i)

        if new_row < 0 or new_column < 0: break
        if new_row > (6 - 1) or new_column > (7 - 1): break
        if self.board[new_row][new_column] != player: break

      else: return True
  return False
```

The if statements behave like filters. C stands for column and r stands for row. We got rid of the try-except statement by checking that the new coordinate is within the limits of the board. The variable indexes can be replaced by a list like [-1,0,1]. The variable connect can be manipulated so instead of playing connect 4 we can be playing connect 3, 5, 6 or even 2. It would not make sense to give the connect variable a number greater than the number of columns or rows of the board.