# **Tic-Tac-Toe with NumPy**

**Author:** Ian Patricio  
**Date:** August 16, 2025  
**Language:** Python  
**Purpose:** This program implements a 2-player Tic-Tac-Toe game using NumPy. It demonstrates the use of classes, inheritance, and operator overloading in Python. The program follows best documentation practices, including descriptive docstrings for modules, classes, and methods.

## **Python Pseudocode:**

# Module: Tic-Tac-Toe

# Purpose: Allow two players to play a 2D tic-tac-toe game.

# Import numpy for 2D board representation

# Define abstract base class Player

# Attributes:

# symbol -> str ("X" or "O")

#

# Constructor (\_\_init\_\_):

# Store symbol

#

# Method make\_move(board):

# Abstract method to be implemented by subclasses

# Define HumanPlayer subclass inheriting from Player

# Method make\_move(board):

# Ask user for row and column input

# Validate move (must be in empty cell)

# Place symbol on board

# Define Board class

# Attributes:

# grid -> 3x3 numpy array initialized with empty spaces

#

# Method display():

# Print the current board

#

# Method \_\_setitem\_\_(pos, symbol):

# Overload [] operator to allow: board[row, col] = symbol

#

# Method is\_full():

# Return True if no empty spaces remain

#

# Method check\_winner(symbol):

# Return True if given symbol has a winning line

# Define Game class

# Attributes:

# board -> Board object

# players -> list of Player objects (X and O)

#

# Method play():

# Loop alternating turns

# Display board

# Current player makes a move

# Check winner

# If winner found -> announce winner and break

# If board full -> announce draw and break

# Main program:

# Initialize two HumanPlayer objects (X and O)

# Create Game object with players

# Call play()

## **Python Source Code:**

*"""*

*Module: Tic-Tac-Toe with NumPy (OOP Version)*

*Description: This module implements a 2-player Tic-Tac-Toe game using NumPy,*

*object-oriented programming, inheritance, polymorphism, and*

*operator overloading. It supports both human and AI players.*

*Author: Ian Patricio*

*Version: 1.0.0*

*"""*

import numpy as np

import random

class Player:

def \_\_init\_\_(self, symbol):

self.symbol = symbol # "X" or "O"

def make\_move(self, board):

*"""Abstract method: implemented by subclasses"""*

raise NotImplementedError("Subclasses must override this method")

class HumanPlayer(Player):

def make\_move(self, board):

while True:

try:

row = int(input(f"Player {self.symbol}, enter row (0-2): "))

col = int(input(f"Player {self.symbol}, enter col (0-2): "))

if board[row, col] == " ":

board[row, col] = self.symbol

break

else:

print("That spot is already taken. Try again.")

except (ValueError, IndexError):

print("Invalid input! Enter numbers between 0 and 2.")

class AIPlayer(Player):

def make\_move(self, board):

print(f"AI {self.symbol} is making a move...")

empty\_cells = [(r, c) for r in range(3) for c in range(3) if board[r, c] == " "]

row, col = random.choice(empty\_cells)

board[row, col] = self.symbol

class TicTacToeBoard:

def \_\_init\_\_(self):

self.grid = np.full((3, 3), " ")

# Operator overloading for indexing

def \_\_getitem\_\_(self, index):

return self.grid[index]

def \_\_setitem\_\_(self, index, value):

self.grid[index] = value

def \_\_str\_\_(self):

rows = []

for row in self.grid:

rows.append(" | ".join(row))

rows.append("-" \* 9)

return "\n".join(rows)

def is\_full(self):

return np.all(self.grid != " ")

def check\_winner(self, symbol):

# Check rows & columns

for i in range(3):

if np.all(self.grid[i, :] == symbol) or np.all(self.grid[:, i] == symbol):

return True

# Check diagonals

if np.all(np.diag(self.grid) == symbol) or np.all(np.diag(np.fliplr(self.grid)) == symbol):

return True

return False

class TicTacToeGame:

def \_\_init\_\_(self, player1, player2):

self.board = TicTacToeBoard()

self.players = [player1, player2]

def play(self):

turn = 0

while True:

print(self.board)

current\_player = self.players[turn % 2]

current\_player.make\_move(self.board)

if self.board.check\_winner(current\_player.symbol):

print(self.board)

print(f"Player {current\_player.symbol} wins!")

break

if self.board.is\_full():

print(self.board)

print("It's a draw!")

break

turn += 1

# Run the game

if \_\_name\_\_ == "\_\_main\_\_":

player1 = HumanPlayer("X")

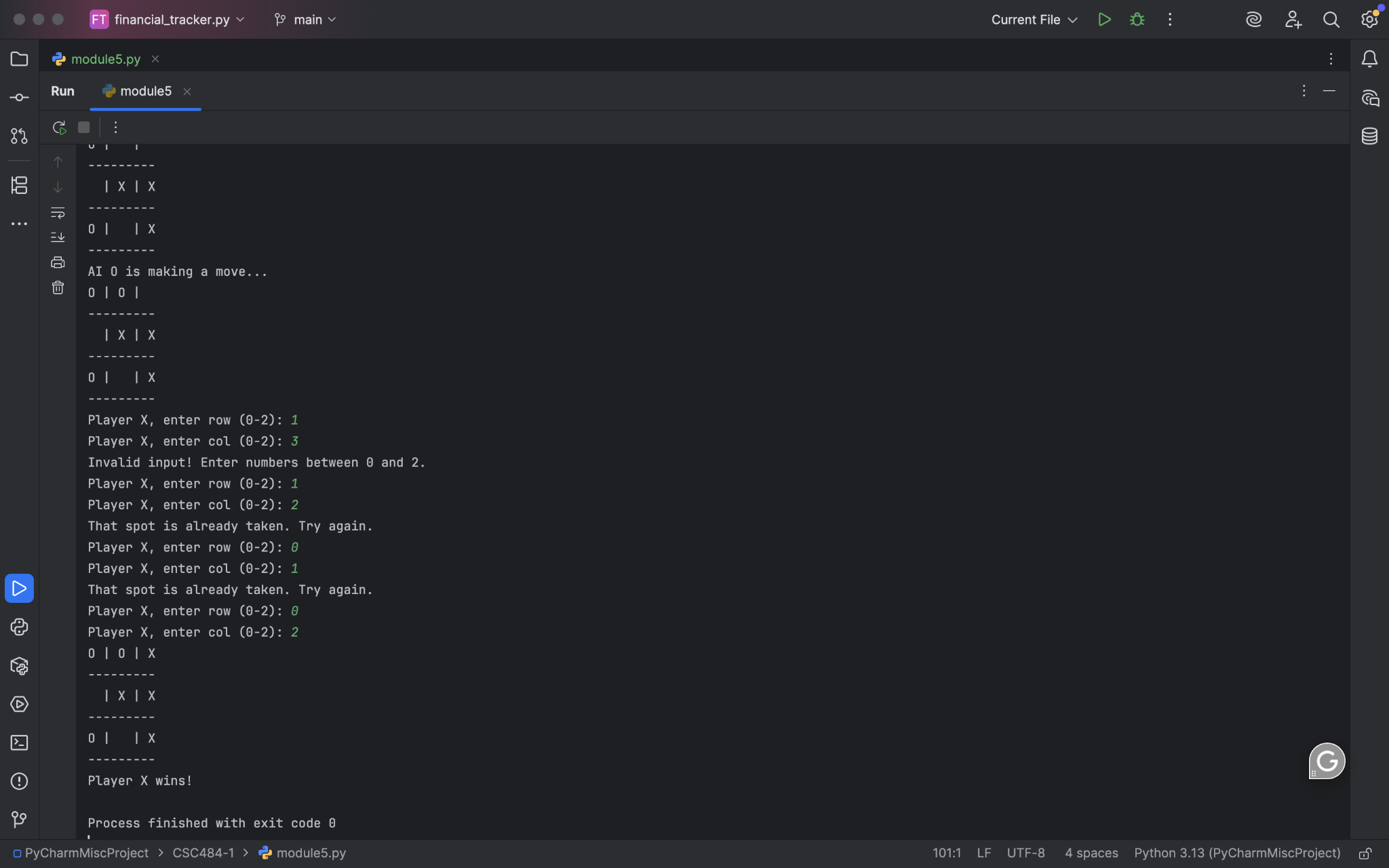
# You can choose HumanPlayer("O") or AIPlayer("O") for polymorphism demo

player2 = AIPlayer("O")

game = TicTacToeGame(player1, player2)

game.play()

## **Screenshot:**



## **Git Repository:**

<https://github.com/ianpatricio-csuglobal/CSC484-1>

## **Discussion of Results:**

Running this Python program allowed two players to play Tic-Tac-Toe interactively in the terminal. The board updated after each move, preventing invalid inputs, and correctly identified wins and draws.

* **Object-oriented design:** Implemented with Player (abstract), HumanPlayer (subclass), Board, and Game.
* **Polymorphism:** Used via the make\_move method in Player and HumanPlayer.
* **Operator overloading:** Board.\_\_setitem\_\_ allows intuitive assignment like board[row, col] = "X".
* **Result:** The game logic works correctly, is modular, and easy to extend (e.g., adding an AIPlayer subclass).