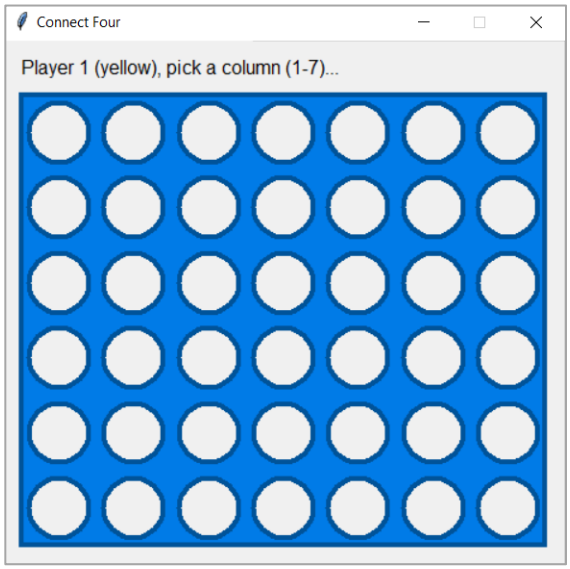
CSC 3510
Introduction to Artificial Intelligence
Florida Southern College

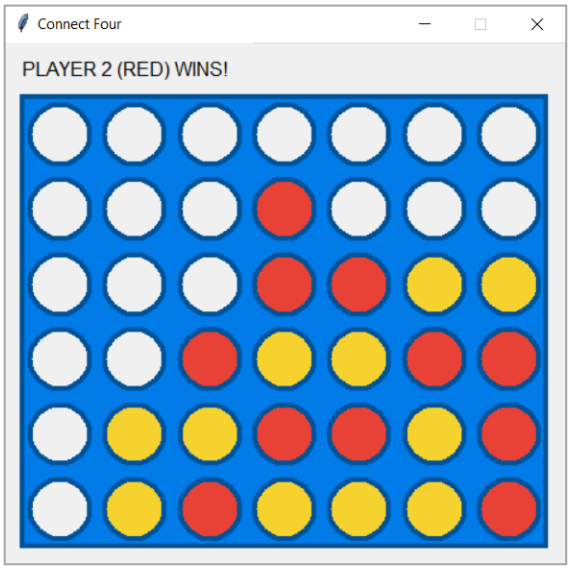
HW4: Adversarial Search
Due: Thursday, December 7, 2023

The purpose of this assignment is to ensure that you are able to:
- clone or fork GitHub repositories for AI software development
- program adversarial search algorithms in Python
- understand the effect of advanced techniques such as alpha-beta pruning on runtime
- creatively design custom heuristics to improve AI performance at specific tasks
- work collaboratively with a peer to efficiently solve problems

For this assignment, you will work in pairs to develop a Python-based AI player for Connect Four, a competitive two-player board game. Here's how the game works: players take turns dropping colored discs into a vertically suspended grid as depicted below.



(a) The empty board at the start of a game.



(b) The red player won this game by connecting four discs diagonally.

The goal is to connect four of one's own discs in a row, either horizontally, vertically, or diagonally, before the opponent does. Players strategically place their discs to block their opponent's moves and create opportunities for their own winning combinations. The game is known for its simplicity yet offers engaging gameplay, requiring both tactical planning and adaptability. It is an excellent framework for testing adversarial search algorithms.

To get started, clone/fork this repository from GitHub: https://github.com/meicholtz/connect4

Follow the instructions for creating a virtual environment, installing the required Python library, and running the main program (`connect4.py`). It is highly recommended that you study the codebase to understand how it works before starting your own code. Then, complete the tasks listed below.

1. Adversarial search focuses on algorithms designed to address competitive multi-agent environments with conflicting goals, which is why games are a suitable platform for this type of search. For your first task, make an AI player that uses the minimax algorithm. You will submit your code as a Python file. The name of your file does not matter, but the contents do matter. At a minimum, you must include a function with the following signature:

```
def get_computer_move(board, which_player):
    """Search for the best move based on the current game state.

    Parameters
    ----------
    board : np.array of ints
        2D array for the current state of the board
        (0=empty, 1=player1, 2=player2).
    which_player : int
        The AI player may want to know which player [1, 2] they are!

    Returns
    -------
    choice : int
        The column (using 1-indexing!) that the player wants to drop a disc into.
    """
```

   You can add other code (e.g. helper functions, library imports, global constants) as necessary, but only the `get_computer_move` function is called in the `connect4.py` program.

   IMPORTANT NOTES:
   - You may assume that the input `board` to the `get_computer_move` function is a copy of the actual board rather than the board itself, so you do not need to worry about messing up the actual game in your simulations.
   - The standard board size is 6×7, but your code should be written to work for other board sizes.
   - The leftmost column of any board is at index 1, not 0! Be careful of off-by-one errors.
   - You are encouraged, but not required to use α-β pruning to speed up the search.
   - **The time limit for making any given move is 5 seconds**, so you must limit the depth of your search and include a custom heuristic to estimate the utility of the simulated game state. Be creative in your design!

2. Based on the requirements outline above, your AI player must include a custom heuristic to estimate utility at a simulated depth in the game tree. For your second task, write a brief description of your heuristic and submit it as a pdf.

Submissions (code + pdf) will be graded on the following criteria:
- **Length:** must be at least 50 lines of code
- **Clarity:** could someone easily understand your AI strategy by reading your code and supplemental pdf?
- **Comments:** does your code contain relevant comments?
- **Content:** does your code contain appropriate Python programming elements such as functions, loops, conditional statements, and various data types?
- **Correctness:** does your program run without error and does it actually implement the proposed AI strategy? Test your code thoroughly!
- **Performance:** can your AI player finish games at all, can it win games, and can it play efficiently?

BONUS: There is the potential for extra credit at the discretion of the instructor for many features including, but not limited to, creative heuristics for utility estimation, performance relative to your peers in a head-to-head tournament, and performance relative to Dr. Eicholtz.