Marianopolis College

Connect Four

Programming Techniques & Applications

Jericho Adalin and Quang Loc Tran

420-LCW-MS-01

Robert Vincent

May 28, 2020

Table of Contents

Mini-manual 3

Design guide. 4

GUI 4

Board creation 5

Game Logic 5

Mini-manual

This program is a simple rendition of the classic two-player game Connect-Four. Represented by the red pieces, the player user must place four markers in a row — either horizontally, vertically, or diagonally — in order to win. Opposing the user is a simply coded AI with three difficulties, ranging from one to three, with one being the easiest.

To use the program, the user must download all of the files in the .zip folder, notably ui.py, game.py, and board.py. The difficulty of the AI will be at the top of the game.py file. To change difficulties, a user must edit the file and change the value of "DIFFICULTY". Again, the "one" difficulty is the easiest, as the computer will randomly choose a column to place its tile. Then, the user must launch the game by running the ui.py.

This will launch the Connect Four game and the user can place their tiles by clicking on any one of the seven columns on the board. Once either player has achieved a Connect Four, or the board gets filled without anyone winning, a message box will appear outlining the result of the game. After this box is closed, the program will end, and the game will close. The user can reopen the game by once again launching the file ui.py.

Design guide.

The implementation of a “Connect Four” game in python comes in 3 distinct sections: Game Logic, Graphical User Interface, and Board creation, with each part carrying its own weight.

GUI

This part enables users to interact with the game. The module of choice for the visual and more tangible part of the board was Tkinter. At its simplest, the board is an evenly spaced out matrix of circles set upon a rectangular backdrop and in its standard format is composed of 6 rows and 7 columns. In order to recreate the game’s distinct look, the width of window and or canvas may of any appropriate size as long as it is divisible by 7. To calculate the desired width, the diameters of circles and the identical vertical and horizontal gaps need to be accounted for. As for the height of the window, use the same coefficient as the width, but instead multiply it by 6. This coefficient would equate to 1 full gap and a circle. Due to relative inexperience, the associated code was pulled from previous assignments and labs and adapted to fit the requirements set by the game’s visual design.

When it comes to interpreting a user’s click on the window, the game’s mechanic greatly simplify the conditions that need to be met. Unlike most games of this genre which allows users to directly place their piece at a desired location according to the x and y coordinates, it instead ignores this convention and has a placement system that imitates gravity, whereby the pieces fall into the chosen column until they meet another piece, in which case the piece is positioned directly above it. Considering such, a click only needs to be defined relative to one of the 7 rectangles, each associated to a specific column. Hence the

division of the board into 7 rectangles. Therefore to position one's click, simply divide by the coefficient used to determine the width of the window.

Board creation

As it pertains to the representation of the board in python, it follows the same structure as the Othello game code from the previous semester, which means that to implement it, we simply used a matrix where each position in the matrix can be easily associated to an x and y coordinate. Furthermore, it can also have a value at said position which can be modified according to the type of piece present. It facilitates the determination of the current state of a given position which can either be empty or occupied by piece, since we can give a unique value for each state. In all, it allows each circle/piece to have, its own designated area on the board.

Game Logic

For the game itself, we once again used the Othello game as a base in order to understand what was required for a game to properly function we also consulted the portion on four in a row on inventwithpython.com (http://inventwithpython.com/pygame/chapter10.html). Respectively, from Othello, we took and modified many of the important and complex board methods, while from the website we took a glance at and based our detection code of a 3 piece and 2 piece sequence on theirs for the 4 piece.

The game features a simplified version of the minimax algorithm to determine the computer player's move at the highest difficulty. The remainder of the game relies heavily on Boolean logic to repeatedly verify if the conditions for a victory are met. To check these conditions, we have implemented methods to verify if a legal four piece sequence is on the

board. The turn base style of the game also relies on Booleans as the function which performs the moves leans upon the current state of the game, only placing pieces if the game is not over.

The computer ai uses these Boolean checks to determine how much it is gaining and or losing on a single instance of play. To get quantifiable scores for every move, we utilized the sequence checking methods to return a score for a given a sequence: 2 for 2, 3 for 3 and 4 for 4. The most complex algorithm for a computer move simply tests all of its possible moves on a given board and then test all possible countermoves (player moves) and gets a list of the moves which allow for the weakest countermoves. The strength of a countermove is determined by the possible scores that the HUMAN player can get, if the human player can get a 4 from the move that computer makes it will not prioritize said move, it would opt instead for a 2 or 3. Due to a lack of game knowledge the parameters to determine a score are extremely simplified and in some cases worse than it playing randomly. This is also due to the sequence detecting methods which does not account for a bad connect3, which is a 3 piece sequence that cannot result in a win.

The final important point of the game is the move method, which as mentioned previously due to the games mechanics becomes very easy. All it requires is the destination column since we cannot pick where the piece lands relative to the y axis. This also play into determining what a legal move is since the only condition which restricts a move from being played is if a column is full. Performing a move just involves checking what is currently in a column.