COMP 1405 - Final Project Analysis

Connect 4 AI

By: Erica Li

November 7, 2020

Introduction

This program uses the numpy and pygame modules.

Connect 4 is a classic childhood game. I created a connect 4 game where you play against the computer, and I implemented this using the minimax algorithm.

Abstraction Analysis

I created an internal representation of the board as a 2D array with 6 rows and 7 columns. The array is first filled with 0's, and then later 1's, and 2's: 0 represents an empty spot, 1 represents a player chip (red), and 2 represents an AI chip (yellow). I chose to use a numpy array instead of a 2D list because arrays have better space and time complexity. As well, it was more convenient because I could easily take any column of the array by saying `array[:][col]`. This operation cannot be done with 2D lists.

I partitioned the board into "windows" of 4 consecutive spots. The windows can be horizontal, vertical, sloping upward diagonal, or sloping downward diagonal. The windows are represented by 1D lists of 4 integers. For example, [1, 2, 0, 1] would represent the following window: red, yellow, empty, red.

Algorithmic and Runtime Analysis

There are three principle functions that do most of the algorithmic work:

score_window(window, chip):

We are trying to maximize the AI's score while minizing the player's score.

Inspect the window that was passed as a parameter and assigns a score based on the following rules:

+150 for a four-in-a-row of chip

+50 for three spots taken up by chip and one empty spot

+2 for two spots taken up by chip and two empty spots

-100 for three spots taken by the opponent chip

-5 for two spots taken up by the opponent chip and two empty spots

This function has a runtime of $O(1)$ because it is a few simple if-statements.

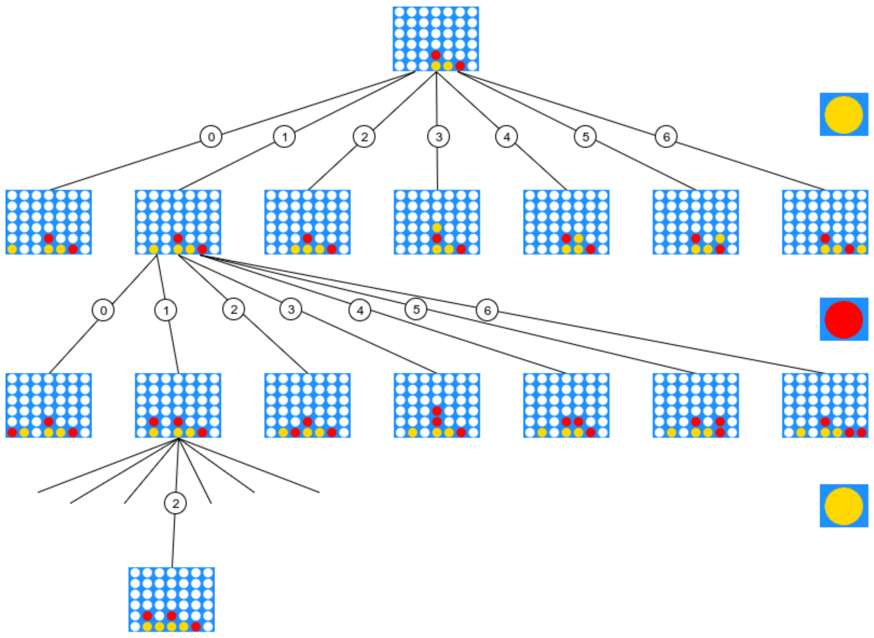
get_score(board, chip):

Iterate through every possible window on the board and add up the total score of the current board, based on calling the score_window(window, chip) function.

I used 4 for loops: horizontal, vertical, sloping upward and sloping downward. In total there are 69 windows on the board. The runtime of this function is $O(n)$, where n is the number of windows on the board.

minimax(node, depth, maximizingPlayer):

My minimax algorithm is a recursive function that searches down the game tree up to a specified depth. Each node of the graph represents the current iteration of the board. The following graph is of depth 3:



It essentially "looks into the future" by predicting the future moves of it's opponent, assuming the opponent is playing optimally. It does this by using the get_score() function to maximize it's own score while trying to minimize the opponent's score in each hypothetical outcome. The total number of comparisons made is:

$$7^n + 7^{n-1} + 7^{n-2} + ...7^1 = \sum_{i=1}^{n} 7^i$$, where n is the depth of the search tree.

Therefore our minimax algorithm is $O(7^n)$, since we only consider the fastest growing term. This is shown in the following table:

Depth	**Number of nodes**	**Comparisons**
1	$7^1 = 7$	7
2	$7^2 = 49$	56
3	$7^3 = 343$	399
4	$7^4 = 2401$	2800

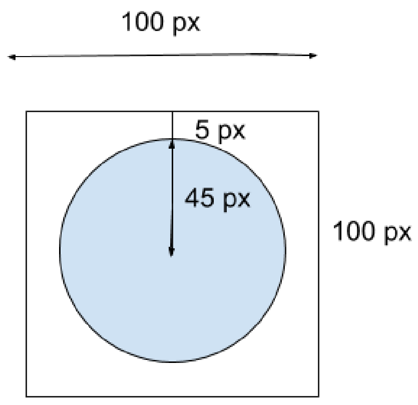
I decided to restrict my search tree to a depth of 3. This was because I didn't want to slow down my program too much and I think 399 comparisons is a good amount.

Next Steps for Improvement

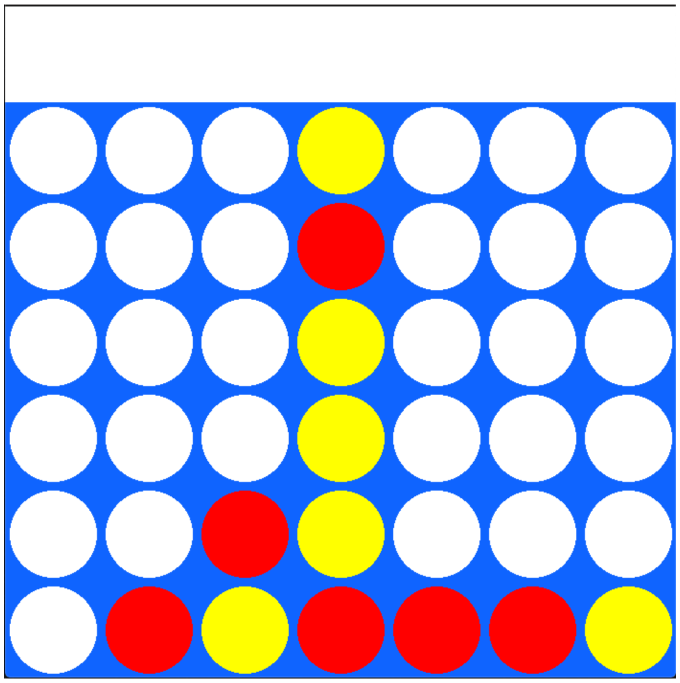
An improvement I can make is to implement alpha-beta pruning to optimize my minimax algorithm. Alpha-beta pruning improves the runtime by removing any branches that are unnessecary to search because there already exists a better move. In it's best case, alpha-beta pruning can square root the runtime of the "naive" minimax. It's best case is $\Omega(\sqrt{b^d})$, where b is the branching factor and d is the depth. It's worst case is $O(b^d)$ still.

Graphics and Sounds

These are the dimensions for the square that each chip or empty space is bounded within:



This is what the game looks like:



I also added a chip dropping sound that plays each time a new chip is placed in the board. This makes the game more enjoyable and realistic.