Connect4 with Minimax and Alpha Beta Pruning

Group Name: ArtificialIdiot
Blaise Saunders - s3561722
Boxuan Lu - s3654900

Git repo:
https://github.com/BoxHezi/artificialidiot-ai1901-connectfour.git

Introduction

Initially when brainstorming heuristics we decided that a lot of the things a person looks for when playing the game are symptoms of a larger design pattern. E.g. aiming for the middle columns, grouping pieces. We decided that an AI that could play the game properly would exhibit those traits of its own volition, and that would be our marker for success. We also decided that with neither of us being particularly skilled in AI we would decide to stick with the provided pseudocode, just writing and evaluate board function and worrying about other optimizations e.g. Alpha Beta pruning later. With this in mind we sat about penning an evaluate board function, we decided after some research into existing Mini-max connect 4 implementations that we would need to break the board down into windows for winning, but we differed on what we thought a window should be. Whether it should check with a window size of four, overlapping where need be, or to just check windows however long they be, running another sliding window within that window. We both set off to work on our own implementations helping with each other's progress along the way. We ran into a tonne of issues with off by one errors and the like, and heavily shared code. At the end we both had completed AIs that would consistently win against random, monte carlo and any human. We dueled our AIs and the fixed window AI won when going first and second so we polished, added alpha beta and submitted that.

Strength

Consistently aims to get as many opportunities at connecting 4 as possible, written in a way that it will scale to any board size, efficiently looks 5 moves in to the future. Can win against the random agent and the Monte Carlo agent consistently.

Weakness

When depth is not deep enough, the AI won't block losing moves, then two pieces are connected together and two empty slots around those two pieces. To improve efficiency we could add functions to check for the cases it misses to be able to run it with a lower mini-max depth. Additionally the agent appears to have an error in where when a winning or losing move is present it simply chooses the leftmost move on the board in some edge cases. After many hours of studying this we cannot determine why.

Discussion

Mini-max function from the start up code is modified, for using alpha-beta pruning is needed. To use alpha-beta pruning, two more variables are needed, alpha and beta. Alpha beta pruning is used due to the large amount of time required when searching depth 5 Mini-max tree.

Pseudocode of heuristic function

```
position_list = generate_position_list(board)

fun eval_horiz(position_list)
    window=[position_list[row][col],  position_list[row][col+1],  position_list[row][col+2],
    position_list[row][col+3]]
    score = eval_window(window)
    return score
fun eval_vert(position_list)
    window = [position_list[row][col], position_list[row+1][col],
position_list[row+2][col],
    position_list[row+3][col]]
    score = eval_window(window)
    return score
fun eval_diag(position_list)
    window = [position_list[row][col], position_list[row+1][col+1],
    position_list[row+2][col+2], position_list[row+3][col+3]]
    score = eval_window(window)
    return score

fun eval_window(window)
    score = 0
    if count(window.count(self piece)) == 4:
        score += 100
    else if count(window.count(self piece)) == 3 and count(window.count(0)) == 1:
        score += 50
    else if count(window.count(self piece)) == 2 and count(window.count(0)) == 2:
        score += 10

    if count(window.count(opponent piece)) == 3 and count(window.count(0)) == 1:
        score -= 40

    return score
```

Reference

En.wikipedia.org. (2019). *Alpha–beta pruning*. [online] Available at: https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning [Accessed 2 May 2019].