CSC111 Project Report: How can we build an AI for playing and winning a Five in a Row game?

Qianning Lian, Xueqing Zhai, Zeyang Ni, Aiden Lin

April 3, 2021

Abstract

The main goal of this project is to build an AI for playing a chess game called Five In a Row (Gomoku), which will choose its move to win based on the opponent's move. The implementation method we used are under 4 main classes: ChessGame, GameTree, Player, and Piece. For visualization, we used a new library called tkinter to display the game board and pieces. This report will describe our computation in detail and discuss the limitations of our project.

Introduction

Our research question is: How can we build an AI for playing and winning a Five in a Row game?

The game Five in Row is a popular traditional board game in east Asia. Many Asians have played and enjoyed the game with their friends and family in their childhood. This year, however, because of the COVID-19 pandemic, people are not allowed to gather and play games together with their friends. Therefore, we decided to build a five in a row game that allows humans to play with an AI so that people can enjoy the game even in quarantine. We wish to bring people happiness during difficult times in this way.

We are building an AI for playing a 2-player board game called Five in Row. Two players, black and white, play on a board of size 15 by 15. Players place their pieces on the board by turns. Black takes the first move, and the first piece must be placed in the center of the board. The goal for each player is to link their pieces in a row of 5 and/or to stop the other player from doing so. The player who first links their pieces in a row of 5 wins. The row of five could be horizontal, vertical, or diagonal. The game ends in a draw when there is no space for another piece to be placed.

Learn more about the game with the following link: `https://en.wikipedia.org/wiki/Gomoku`

Our idea of building an AI is inspired by the gametree class and minichess game class in Assignment 2. This game is different and slightly more complicated than the minichess game in Assignment 2, but all possible moves could still be represented by a tree structure. Because of the different rules and winning strategy for this game, our gametree have different attributes from the gametree in A2, but would have a similar structure in general.

Data sets

We did not apply any data set for this final project.

Computational Overview

`Inheritance: The project consists of several steps:`

I Get every possible moves that is worth taking.

II For each potential move, estimate the opponent's next move and evaluate the score of that move.

III Repeat step I and II according to the given depth (which is 2 in this case) and form a tree.

IV Evaluate the score (attack score and defence score) of each leaf and pass it to its ancestor according to MiniMax algorithm.

V Using alpha-beta pruning searching to accelerate the process.

VI Make the move that has the most favorable score to the AI.

VII Display the move in a tkinter window.

For simplicity of the game, the AI will always be the black player, thus will always make the first move (fixed at the centre of the board due to the rules of the game).

- *Find all possible moves:*

First, find all possible moves. The user can take any move on the board, but for simplicity, the AI will only take moves that are in a 5 by 5 horizontal, vertical and diagonal of all pieces currently on the board.

Then, find all worth taking moves. By worth taking we mean that the next move should contribute to your success in some way. And there are several situations that can be evaluated as worth taking. By in a row we mean that it can potentially become five in a row vertically, horizontally and diagonally.

We have built a function called get_valid_moves, which returns a list of valid moves for pieces. There are two cases to make the function return empty list. First case is that the piece has no place to move. Second case is that the piece is the first piece for the chess game. In this case, we eliminate it by setting the first piece to be black and always in the center of the chessboard.

- *Estimate the opponent's next move:*

Our project uses trees to represent possible moves on the chess board. The trees will be generated after each move of the opponent with a certain depth. The depth is set to be 2 for efficiency of the game, which means that the AI will calculate one move for itself, and one move for the user.

By giving the score to every possible step that the players (AI itself and the opponent) can place, the AI selects the branch which is most promising by MiniMax algorithm. The scores are calculated based on the situation of the board - the positions of pieces. After generating all possible moves for one subtree, the scores are calculated for each leaf and would be passed up to their parent after horizontal comparison, until the root is reached. Upon that, we also want to use the alpha-beta pruning algorithm to reduce run time.

- `Alpha-beta algorithm:`

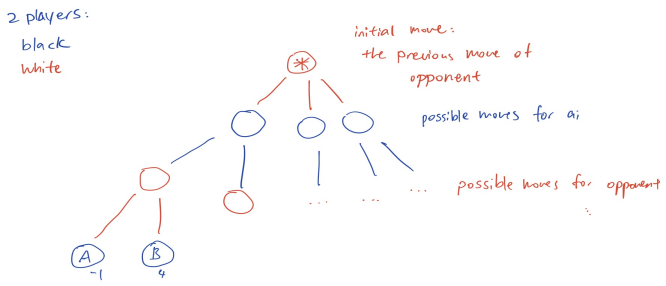


Figure 1: Calculating score for 1st subtree

Looking at the graph above. Suppose the score for leaf A is -1, for leaf B is 4. We assume that the opponent would take the step that results in worst case of our AI's game, so the node above A and B would have value -1 (minimum of all scores of its subtrees).
Until now, the algorithm is the same as in A2.

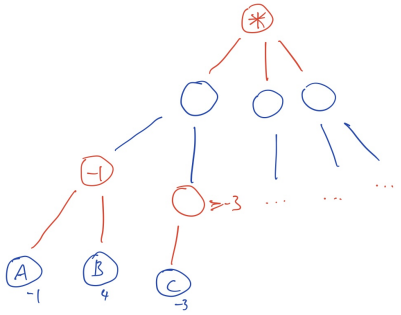


Figure 2: Calculating score for 2st subtree

Suppose that when generating C, C has a score of -3. Then, since our opponent always chooses the worst case for us, the node above it must have score ≤ -3. We notice that this score is already worse than the first subtree (with score -1). So immediately, we can delete this whole branch and move to the next one.

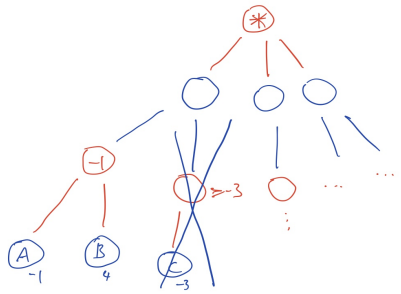


Figure 3: Calculating score

This algorithm would reduce our run time significantly, which is important for our project since the game board and the number of possible moves for a game of Five in Row is much larger than the minichess game in A2.

- `MiniMax algorithm:`

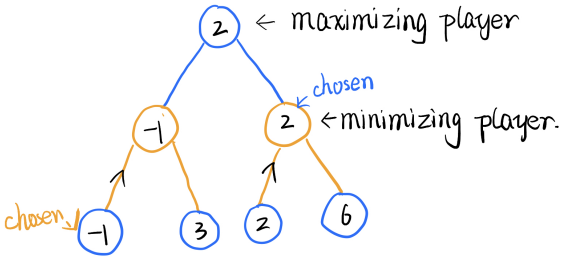


Figure 4: MiniMax algorithm

For this game, we will always take black player (the first player to move) as Maximizing player (always choose the situation with highest score when possible), and white player as Minimizing player (always choose the situation with lowest score when possible). The minimax function loops through all the subtrees of the given tree, and finds the max score or min score depending on the colour of the player.

In the game tree, each subtree will call minimax and reset its score. The score will be used in Alpha-beta pruning later to decide whether to delete the branch or not.

- *Evaluate scores:*

The evaluation part is designed to evaluate the score of current situation based on following criteria. We were inspired by the use of graphs and vertices in Assignment 3, and we used graphs to represent a piece and the pieces around it. Checking surrounding pieces and evaluating score for pieces are all based on the graph properties of the pieces.

If there are no enemy pieces, and there are 1, 2, or 3 in a row of one direction (horizontal, vertical, or 2 diagonals): for each piece and the pieces around it,

1. If the distance between the piece and its neighbour is 1 (i.e. one is right next to the other): 800
2. Distance is 2: 100
3. Distance is 3: 25.

If there is an enemy piece around the piece, than the above score will be reduced in half.

1. If there are 4 in a row, and no enemy piece around: 4800
2. If there are 4 in a row and 1 enemy piece around: 1200

   (If there are more than 2 enemy pieces, it means that these pieces can not possibly win on this direction.)
3. If there are 5 in a row, score will be infinity.

For white pieces, these scores will be negative, so the AI would consider defending to stop its score from decreasing.

The line up of black pieces evaluates to positive score representing the favorable situation to black player and detrimental to white player.

The line up of white pieces evaluates to negative score representing the favorable situation to white player and detrimental to black player.

The final score is gained by adding up the two scores, black player would be inclined to choose the move with higher score and white player would be inclined choose the move that has the lowest score.

By the MiniMax algorithm and Alpha-beta searching, current player then picks the move that is most favorable to itself.

- *Display the game:* `Tkinter` *Library:*

In this project, we will use tkinter, a library that allows us to build a simple GUI. It would help us to display visualizations of the board and the pieces, and react to the mouse activities made by users.

1. We first create a window for the instructions to display by creating a tk variable. The window is titled "Five in a Row Game". We created these using tk.title and tk.geometry functions.
2. The window is filled with text of instructions for the game, as well as a button labeled "I see". These components are created using tk.Label and tk.Button. They are placed onto the window by the pack() function.
3. The actual game board and all the pieces are on a canvas created by tk.canvas. Pieces are drawn by using canvas.create_ovals(). The string of game state is created and changed by tk.StringVar().set().
4. When restarting the game, the board is cleared by canvas.delete("all"), which clears the canvas to the initial state.
5. When the user clicks a region that is considered a valid move for the piece, the move would be made and the piece would be drawn. The AI would start making its move as the user releases the click.

Instructions for Running the Program

1. Download all files from Markus
2. Run main.py
3. Read the instructions. After finish reading, click "I see" to proceed
4. Play the game
5. After finishing one round, click "restart" button to the right of the board to restart another game
6. Quit by directly closing the window

Changes and Elaboration of Computational Plan

- `The major function of each file`

I Chessgame.py This file initializes a game, and is responsible for taking moves and getting the situation score from piece.py

II piece.py This file contains two classes, Piece and Pieces, which can be viewed as vertex and graph respectively. The Pieces graph is responsible for tackling the connection between one piece and another and calculate the score of current situation.

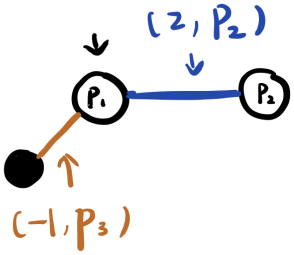
III GameTree.py This file is responsible for generating trees of current game situation.

IV Player.py This file contains an AI player class which takes in the current game situation as an input and outputs the next move of AI player.

V main.py This file is responsible for visualization using tkinter.

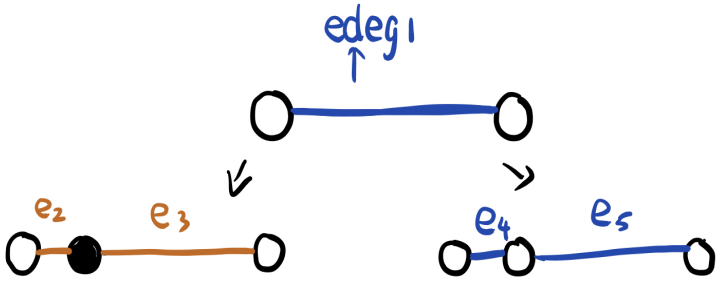
- `Using graphs and vertices to tackling the connections between pieces.`

I We create a Piece class to represent every single piece, for each piece instance, there are three instance attributes: kind('black' or 'white'), neighbours(The adjacent pieces of current piece), and coordinate(the coordinate of this piece on the board.). The neighbour is stored as a dictionary using each direction('vertical', 'horizontal', 'right diagonal', 'left diagonal') as keys and the pieces in that direction and its distance between as values(in forms of tuple[distance: int, piece]). The distance between friendly pieces count as positive number and the distance between hostile pieces count as minus number. These tuples will be used to calculate score of current situation.



The distance between enemy piece count as minus number.

II The Pieces class is responsible for adding new piece to the graph and update its neighbours(The number of every piece's neighbour in one direction should be smaller than 2 since the edge shouldn't step across other piece instance in one direction.). When adding the neighbours, each coordinate near the current piece is visited to determine whether there's a piece there, and for each direction(eight specific directions), the process of recognizing neighbour terminate when first neighbour in that direction is encountered. And every time a new edge if added between two pieces, we need to examine whether there redundant edges that should be deleted after the update.



break edge1 and form new edges.

III To calculate the score of this graph, we need to visit every single piece and calculate the score of that piece using the function '_single_evaluation', this function recursive count the edges that extend from current piece and stop when the overall distance add up to 5. If a piece has only one neighbour on it's right hand with distance 1, the score of this piece is the score of edge with distance 1(800 or -800 depending on the color of the piece in this case). And the score of the whole graph is the sum of every piece's score. So in the previous case the total score

would be 1600 instead of 800. We repeatedly count the edge that has been previously counted to substantially increase the score than it look to be. Since the situation of three or four pieces in a row is way more strategically important to situation where there's only 2 in a row.

- `Running time concern`

1. Originally, we wanted a better performing AI and thus wanted to set the gametree with a depth of 5. However, the actual running time of the project varies greatly on the hardware, even a gametree with depth of 2 would take around 20 seconds at later stage of the game for some computers. But when testing, an AI with depth of 2 is good enough for the most of the time, the AI meets our expectations by beating all group members at least once.
2. We had thoughts about letting player to choose the colour (black or white), but doing that would result in many adjusts to the current gametree and evaluation algorithm. We decided that since the evaluation is not the main focus of this project, it is better to just set the AI to be black all the time.
3. After learning more about the tkinter library, none of the functions we listed in the proposal were used in the actual project. For detailed changes, see the corresponding section in computation overview.

Discussion

Limitations:

1. . The gametree is only of depth of 2 in the actual game. In some cases, due to the property of apha-beta pruning(to shorten the running time),when after 2 steps, both black and white may have score of infinity (negative infinity) (for example, black wins by taking one step, but after that, white also wins by taking one step). In this case, the AI will choose to block the white player instead of winning by placing the fifth piece since we did not count the score of graph at each stage to shorten the running time. If the AI has a gametree with higher depth, then the problem would not exist. After testing the game on several different computers, windows computer with better Discrete graphics would react fast even with the depth 4 game, but Mac gets slow with depth 2 game. Therefore we put depth 2 gametree as the final version.
2. . We once wanted to add feature to choose difficulty, easy being the depth 2 gametree, hard being the depth 4 gametree. However, adding Buttons or RadioButtons would slow down the running time dramatically. For better game experience, we just set the difficulty to be depth 2.
3. . Tkinter is not the most suitable library for games since it is mainly built for GUI. Pygame would be a better choice, we could add more features to the game like background music, choice of difficulty, score count, etc.
4. . This AI does not have the ability to "learn". It will always follow the same implemented strategy. So if the user figures out a way to win, the user could use the exact same strategy to win again and again.

References

"Alpha–Beta Pruning." Wikipedia, Wikimedia Foundation, 6 Mar. 2021, `https://en.wikipedia.org/wiki/Alphabeta\_pruning`.

Dufour, Bruno, and Wen Hsin Chang. "An Introduction to Tkinter." An Introduction to Tkinter, `www.cs.mcgill.ca/~hv/classes/MS/TkinterPres/`.

"Gomoku." Wikipedia, Wikimedia Foundation, 13 Mar. 2021, `https://en.wikipedia.org/wiki/Gomoku`.

Lague, Sebastian. Algorithms Explained – Minimax and Alpha-Beta Pruning. YouTube, YouTube, 20 Apr. 2018, `www.youtube.com/watch?v=l-hh51ncgDI`.

Liu, David and Waller, Isaac. "CSC111 Assignment 2."

Liu, David and Waller, Isaac. "CSC111 Assignment 3."

"Graphical User Interfaces with Tk." Graphical User Interfaces with Tk - Python 3.9.2 documentation, 13 Mar. 2021, https://docs.python.org/3/library/tk.html.