A Dots and Boxes AI Implementation Using Minimax and Alpha-Beta Pruning

Travis Bubb

CSC447 010
Dr. Schwesinger
December 15, 2021

Table of Contents

Project Overview

My final project for CSC447 is an artificial intelligence agent that plays the game Dots and Boxes. This is a game that consists of an m x n grid of dots. There are two players who take turns drawing a single line that connects two adjacent dots. Note that diagonal lines are not permitted in this game but are instead drawn vertically or horizontally. The main goal of the game is to complete more squares than the opponent. To complete a square, a player must draw a line that completes the last edge of a 1x1 square. There are some cases where a single line can complete two squares. In this case, the player will be rewarded one point for each square. During their turn, a player will draw a single line. If this line completes a square, they will continue to draw one line at a time until they make a move that is not a completion. At this point their turn will end and the other player will be able to make a line. The player with the most completed squares after all lines have been drawn is the winner. For more details about the game, visit https://gametable.org/games/dots-and-boxes/ for a list of rules as well as a simple implementation of the game that can be played.

As a broad overview, my program initiates a game between an AI player and a real user and allows them to play against each other. The program takes input from the human player, during their turn, in the form of two integer values separated by a space. These values represent an x and a y value respectively at which a line is drawn on the board. When it is the AI's turn to make a move, it looks ahead to see possible branches of the game tree and evaluates the best possible move given the current state of the game. The agent evaluates the moves based on which sequence will give it a high score while also trying to give the opponent a low score. This project falls under the *search* category of classical artificial intelligence. The goal of the AI program is to be able to easily beat a random player and consistently beat an amateur human player.

Game Infrastructure

In order to achieve my goals for this project, I first had to develop the infrastructure for the game itself. In order to be able to focus on the artificial intelligence aspect of the project, not too much time was spent on the actual gameplay other than ensuring that it is functional. I was originally planning on using the Curses module but decided that it would be better to not proceed with that route. As a result, the game itself is based in the command line and does not have a fancy GUI other than printing out a simple grid of dots and lines. The main game engine is found in "dab-engine.py" and it takes in two command-line arguments that tell the program the player number the human will be and the size of the board, respectively. The player number can either be 1 or 2, and the board size can be any integer between 3 and 6 inclusive. The board number that is entered, *m*, will create a board that is *m* x *m* big. The game logic itself is fairly easy to understand. There is a main game loop that runs until the entire board has filled up and there are no possible moves left. It will take turns receiving input from the AI program and the player and process their moves accordingly. If a completion is made, the player receives a point and can make another move. Once the game is complete, the final scores will be displayed, and the program will terminate.

For any individual with the code who would also like to experiment with using the random player as well, they are able to do so by simply uncommenting one line of code and commenting another line. For details, refer to the README file that is supplied with the project folder.

AI Implementation

For the artificial intelligence aspect of this game, I noticed that the overall design of the agent was similar to another project I recently completed which was classified as an adversarial

searching problem. Thus, I decided to move forward in a similar manner to try and implement the AI program. I decided to utilize the minimax algorithm with alpha-beta pruning as the core of the program and used Python 3.7 as the language of choice. I chose to use minimax, because I felt it made sense seeing as Dots and Boxes is a two-player, turn-based game. I thought it would allow me to maximize the performance of my AI agent while also minimizing the performance of the opposing player.

The goal of my minimax function is to ultimately maximize the AI agent's score compared to the opponent. It does this by taking in a game state and desired depth, and it loops through all open board positions and their successors to see which path will yield the most desirable outcome. To evaluate the different possible moves, the outcomes are weighed simply by how many points the AI player will have after a given sequence minus the number of points the opposing player will have.

AI Results

One of the biggest issues that was found with the implementation is that it is not very efficient and doesn't allow a very large depth as a result of that. Another factor that may have had an effect is that Python was used. Python is known as a slower language and it is likely that if another more efficient language was used, like perhaps C++ or Rust, that the depth may have been able to be increased. For the 3x3 board the AI worked great with a depth of 3 and 4, finding a move almost instantaneously. For the 4x4 board, 3 worked well and 4 was playable but it was not the fastest gameplay. For the 5x5 and 6x6 boards, a depth of 3 or under had to be used due to the large amount of move possibilities. Anything above a depth of 3 was simply just too slow to be able to use in a game environment. It should be noted that once the game started to get going and more moves were already on the board, the speed of the algorithm significantly increased,

and the moves were computed nearly instantaneously even with the larger board sizes. Another issue that was discovered with using the minimax approach is that it works well during the mid- to late game but not so well during the early game. Due to the nature of minimax, it is more difficult for the agent to determine what moves are good in the early game when there are little to no moves present on the board already.

Putting these problems aside, the AI performed well against a random opponent during testing. The AI wins every time on all the board sizes from 3-6 with a significant margin. If the random player is allowed to score at all, they usually only end up with a score of 1 or 2. After numerous tests, I have yet to see the random player score more than 2 completed boxes in a single game. Once satisfied with the performance against the random player, I then conducted numerous games across different board sizes where I personally played against the AI. Note that I do not claim to be any sort of expert in the game of Dots and Boxes and the results could very well be due to my lack of skill. However, I found that the agent does a decent job playing against me. I discovered that once it gets into the mid- to late game, it attempts to trap its opponent into setting it up for a long run of completions. This was fairly successful, and I lost numerous times even while knowing what the agent was attempting to do. While I was happy with this, the games were much closer than they were with the random player. When the agent did win, it was usually only by a couple of points unless I made a significant blunder during the game. I was able to defeat the agent several times by a slim margin, and sometimes by a larger margin. I found that the games that I did end up winning were largely due to the agent making non-optimal moves during the early turns of the game. Without much information to go off of, the agent struggled to choose optimal moves.

Related Work

In their master's thesis paper, Gerben Blom (1) discusses their journey with Dots and Boxes and how they went about constructing an artificial intelligence agent. They describe in detail some more complicated strategies for the middle and later portions of the game. They used a neural network to train their agent and incorporated the more advanced rules and strategies they define in the paper. Similar to what I found, they realized that the opening stages of the game were not easy to be incorporated into the agent. They implemented their agent to simply pick a random move out of the valid moves if none of the rules that they added were applicable. To aid in choosing a move, they also implemented a critical value heuristic. Briefly, it is a heuristic function that analyzes some common board structures that usually lead to a somewhat predictable result. In doing so, a prediction of different outcomes farther in the game's future can be made and this can help the agent to make a better move. Overall, our approaches were similar at a high-level in that both considered the players' scores after investigating different possible sequences of moves. However, when it comes down to it, Blom's approach was much more complex than mine since they added more layers to their program. Blom implemented reinforcement learning at some stages, ran machine learning simulations to pre-determine the best moves in some situations, and incorporated several different rules that aided their agent to become intelligent. My program on the other hand is much simpler, solely relying on score as an evaluation of the moves.

Conclusion

Looking back at my minimax and alpha-beta pruning agent, I am pleased that it is somewhat intelligent, but I would like to continue to improve upon it in the future. I plan on trying to implement it in a more speed-friendly language so that I can hopefully delve deeper into the game tree. I also would like to implement a more advanced heuristic function, on top of the

score evaluation, that will help predict different scenarios based on the structures present on the board like Blom did for their agent. Another big change I would like to make is to come up with a dedicated early game strategy. I was unable to come up with anything worthwhile during this iteration of the project, and I think that in doing so, the agent would become much better off since it would be starting with optimal moves rather than not. Finally, I would also consider implementing some sort of reinforcement learning that can create an agent based on training simulations and compare this to my minimax algorithm to see if it would perform better or worse in the game.

Sources

1. Blom, Gerben M. *An Artificial Intelligence Approach to Dots-and--Boxes*. 2007, https://fse.studenttheses.ub.rug.nl/9014/1/AI_Ma_2007_GBBlom.CV.pdf.