Connect N Design Document

Group #20: Timothy Goon, Jadon Laforest & Matthew Worzala

Idea Sources

From the rules of connect 4 it was clearly evident that lining up many of your tokens while minimizing the amount of tokens the opponent can line up is the major goal of the game. Some of us started watching videos of winning strategies of Connect 4, but we quickly realized most of the strategies used took advantage of inexperienced human players and were not meant for/against artificial intelligence. One important point throughout all these videos that did become a major part of all our iterations was the idea of controlling and playing toward the middle of the board. Further, from this website and many others supported the idea of playing toward the center. That allowed us to come to the conclusion that placing your tokens in a space that allows for more "freedom of movement" in terms of moves and potential lines is beneficial for the early game at the very least.

The two major concepts described led us to come up with a general heuristic for a non-terminal connect N board. It evaluates each of the tokens based on the potential for that token to contribute to a win. For all of your tokens on the board, the heuristic looks in all eight directions around each token (up, diagonals up, right, diagonals down, etc.) for N-1 spaces. If it encounters a free space it adds 1 to the token's score. If it encounters another one of your tokens it adds 2. If it ever encounters an opposing token or the edge of the board the score for that row is discarded. The score of the board is the sum from looking at all of your tokens and subtracting the score from doing the same for the opposing tokens. A diagram of how a score for a single token is calculated is shown in Figure 1 below.

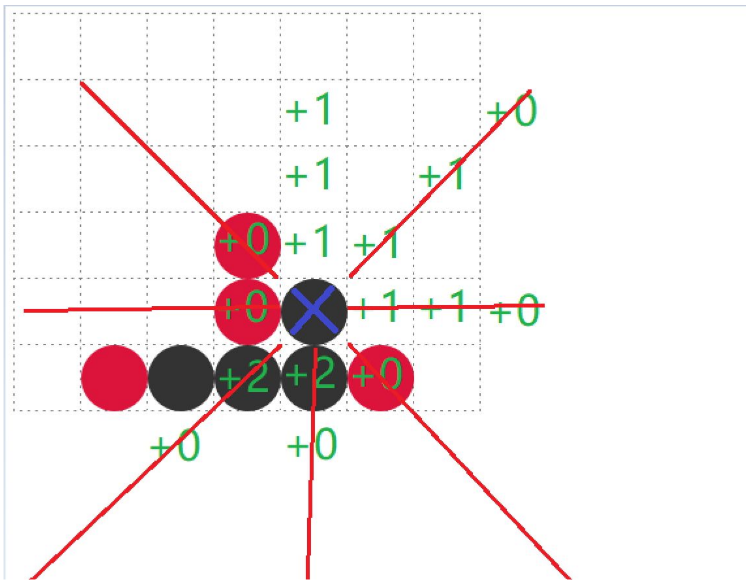

Figure 1

Diagram of how our initial heuristic for non-terminal boards calculates a score for a single token. The score for this token is 3.

Initial Testing

Methods

The testing methods to check for the heuristic's correctness was done independent of the game with example inputs we thought up and performed like the testing for any other method. We tested our AI versus the random agent and ourselves (human input). Eventually, improving the AI through human testing became inefficient since it was hard to find specific cases where the heuristic failed. We then began to primarily test our AI versus the random agent, both as player one and player two.

The initial heuristic's effectiveness was tested by implementing it into an alpha beta pruning minimax algorithm and having it play 100 games vs a random agent at depth 4 as player 1 and another 100 as player 2. The best result we found only using this heuristic is 98/100 games won as player 1 and 96/100 games won as player 2. As we tried to improve this heuristic using this testing method, there was never a need to find an opponent better than the random agent as it never beat the random agent 100/100 games as both player 1 and 2 executed as the only heuristic for the alpha beta agent.

Successes & Failures

Using our own inputs versus our heuristics let us see their behavior and which problems they were running into. It made it clear where we needed to focus our improvements. For example, when first testing, sometimes our initial heuristics would just let us stack tokens in the first column, so that was one of the first things we focused on fixing. Another was fixing a bug that did not see/block a win condition when the opponent left a gap in a line of 3 tokens. These tests led to many improvements and iterations of these baseline heuristics.

A few of our initial iterations also ran into issues with running at odd vs even depths or had extremely varying performances as player 1 vs player 2. Although we were not able to pinpoint the causes for all of these problems, they did help us think more simply as we approached our final heuristic.

Improvements

At first, we tried to do various minor edits to the scoring mechanics and the numbers for the static values but nothing we tried ended up better than the initial heuristic. The following is a list of the ideas that we tried:

- Changing the weights between the addition of points from your tokens vs the subtraction of points from opposing tokens to try to have the agent play more offensively or defensively
- Allowing the agent to keep the score of rows that encounter the edge of the board or an opposing token
- Changing the ratio between the points added when lining up your token vs lining up an empty space

- Adding points when your token is lined up with many opposing tokens (blocking)
- Using the max score of a direction rather than the sum of all eight directions
- Adding an additional score multiplier based on the amount of one player's tokens were in a row of the n slots checked
- Starting 8-direction searches from empty slots as opposed to player tokens
- Some combination of the previous ideas

Eventually, some improvements started getting better performance than the initial heuristic. One combines the ideas of two group members, 8-direction scoring from player tokens with a token counter that will boost the score if it sees that the player has a winning move or is 1 turn from a winning move. This heuristic (non-terminal 1) improved the win rates versus the random bot- 99/100 for player 1 and 97/100 for player 2. From there, we continued to tweak the point values and scores by testing the heuristics versus each other. This led to another improvement upon the combination heuristic (non-terminal 2) where player 2 was able to win 99/100, but player 1's performance was reduced to 98/100.

Taking a different approach to both the algorithm and to our heuristic, we developed a new agent that used negamax and a more simplistic heuristic.

Final Agent & Heuristic Explanation

How it works

After our initial heuristic, we were still losing against random on occasion even with a depth of 5 or 6, so we took a few steps back and tried a slightly different approach. Instead of aiming for a complicated heuristic, we optimized the algorithm for speed. We tried a heuristic which just gave a high weight to winning for each player. With some speed optimizations, we were able to process around 15k boards per second which meant that we were able to look ahead much deeper. This strategy proved effective and was able to beat random consistently and quickly with a depth of 4+. We were also able to reach a depth of 10 while still finding a move within ~15 seconds.

After this, we reintroduced parts of our more complicated heuristic which improved the heuristic performance at the cost of more processing time.

In our final agent, we used the negamax variant of minimax, which is a cleanup more than anything else. Negamax leverages the fact that something good for an opponent is the same amount bad for you. This means that a score of 10 for you is the same as -10 for your opponent and there can be a single recursive call to determine the best move instead of one for each player. There is no noticeable performance boost using negamax, however it is a cleaner solution and easier to work with.

Additionally, we order our moves from the middle of the board to the outside. This is likely not an optimal ordering, however, it is quicker to compute and moves closer to the center are likely to be better than moves near the edge, especially toward the beginning of the game.

Final Performance Testing

At this point the alpha beta/negamax agent always wins vs the random agent so long as it is run with a depth greater than four. Therefore, we had the agent play vs this alpha beta minimax agent at its max depth of eight. The test was done by manually playing two games at the same time and having a human interface between the website and the project code.

Eventually, we found that all of our combinations of non-terminal heuristics with our single terminal heuristic at depth six could all beat the depth eight website agent so we turned to playing our different agents against each other to determine which was better. The tournament resulted in our initial non-terminal and terminal heuristic in a combination as the best at playing vs similar agents.

During this stage, we also tested our agent against group 24's agent which used a more complicated heuristic but was only able to reach depth 5 or 6. Our agent was able to beat theirs in most cases. These tests made it easy to see that a simpler heuristic that could reach a greater depth can be much more effective than a complicated heuristic at lower depths.

Lastly, we had our different agents play vs a perfect agent with the solved version of connect 4 to see how long each would last. The results are tabulated in Table 1 below.

Heuristics	Turns Until Loss
Terminal 0 alone	32
Non-terminal 0, Terminal 0	32
Non-terminal 1, Terminal 0	32
Non-terminal 2, Terminal 0	38

Table 1

Table of how many turns each combination of our heuristics lasted vs the perfect player.

In the end we decided to go with the combination of our Non-terminal heuristic 0 and Terminal heuristic 0. Although, this combination did not perform the best in the test vs the perfect player we decided that that test is unrealistic for our goals and based our decision on the other tests.

Final Depth Testing

Once we were happy with where our heuristic and agent performed in the tests we ran, we started mainly testing the optimization and limits of the code. This meant testing with the random agent at different depths and tracking the time it took per turn. This helped us to determine which configuration we should run with for our final code. We also did testing for the depth of our negamax search in the worst case by removing all pruning. We found that for the parameters of default connect 4, a depth of six put the agent safely within the 15 second turn time limit. For the larger, 10x8 board, only a few turns resulted in going over 15 seconds at a depth of 6. We assumed that in most cases there should be a good chunk of cases pruned- especially because our algorithm starts with moves from the center- so we were confident that in almost all normal games, it would not pass the time limit.

Future Improvements

With more time and effort, our final agent and heuristic could definitely be improved. At the beginning, our testing produced many bugs and issues for us to fix, but by the end, our tests resulted in few apparent problems. We think that the heuristic could most likely be improved on edge cases or in cases where an opponent is setting up lots of different win conditions. Although it is clear that there are other ideas to try that we have not thought of yet, it's hard to pinpoint the exact ways to improve our heuristic since most of our testing resulted in strong performances- other than games versus the "god" AI.

We could also do additional testing to optimize the values of the constants so that the non-terminal heuristic could interface better with the terminal heuristic. We spent a large amount of time tuning values of our final heuristic by changing a few values and then running the agent against older versions. This method of tuning values is very similar to how a genetic algorithm might go about it. Given more time, we would have liked to try out a genetic algorithm for this task, however, we were not confident that it would come up with any useful data within our timespan.

Overall, we are happy with where our agent and heuristic ended up. Being able to make something that we can't beat ourselves was definitely an interesting and enlightening experience.