**CSE-3353 Spring 2019 Semester Project: Connect Four AI**

Eric Miao, Preston Cobb, Sheng Nan

**Introduction:**

We constructed two AIs which can solve the game connect four. One utilizes game trees and minimax while the other uses brute force. The minimax AI is vastly more efficient and difficult to beat than the brute force method, but both are capable of winning the game against a human opponent. Our minimax implementation is written in Python, while our brute force is coded in C++. This allowed us to keep the programming a bit simpler for the more efficient algorithm and give as much of a speed advantage to the less efficient algorithm as possible.

**Usage:**

Minimax/Gametree/DFS AI:

* 1. Navigate to the directory containing all of the python files
  2. Run “python GameLauncher.py” (code was written with python 3 in mind)
  3. Select your mode (player vs computer or player vs player) and input your name.
  4. From here you may select a column by entering its corresponding number.
  5. The computer will continue to play against you until either one of your wins or the board fills

Brute Force:

1. Either compile the project using a standalone c++ compiler or run it within Qt Creator using the .pro file.
2. After running the program, select “Play with AI” (option 2)
3. Input both your name and the computer’s name
4. Select your desired column by inputting its corresponding number
5. Play until one of you wins or you tie.

**Heuristics:**

Minimax:

Minimax follows the simple concept of minimizing your opponent's gain. To accomplish this, each state of the game is assigned a value based on how good or bad it is for the player. The common implementation of this is to assign states that are good for the player a positive value, while states that are good for the opponent are given a negative value. The values of these states can vary as well. If it is that player’s turn and they have 3 in a row with an opportunity to win, then that state will typically have an extremely high value to ensure that it is always picked. A state with 2 in a row may have a much smaller value and 1 in a row will have a value even smaller than that. This allows the player to choose the best move, even if there are many moves that lead to a better state. Minimax also give the programmer freedom to decide how far the tree will go before deciding on a best move. In our case, we set our AI to look 6 turns ahead.

Game Tree:

The Game Tree is a fundamental heuristic on top of which all others are built. The purpose of a game tree is to represent the possible moves and their outcomes in a game. A tree will start at the current or beginning state of the game and continue to branch off based on the possible moves for each player to make. Once these moves are mapped out, it is easy to use another heuristic such as minimax to find the optimal move in the current game state.

Depth First Search:

Depth First Search (DFS) is a search method that searches down a branch until it hits the end and then backtracks and continues to the next node. We used this to find the branch with the highest value, which would indicate that it is the best for the AI and worst for the player.

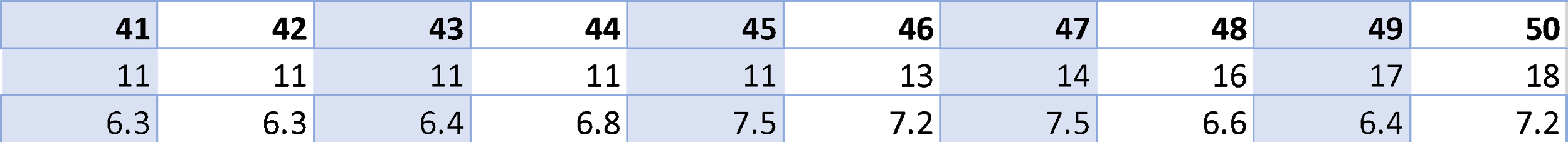
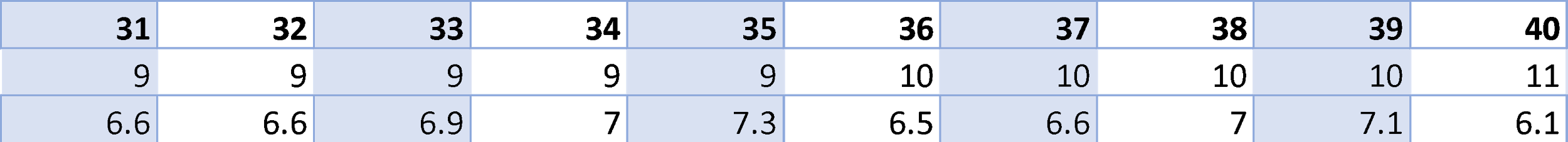
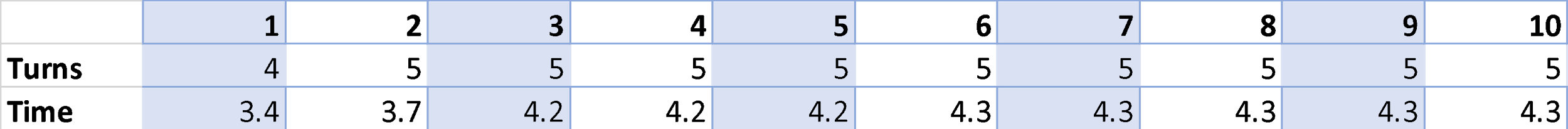
Brute Force:

Brute Force exhausts all possible options until it finds a solution to the problem. In the context of connect four, that means that for each move, all possible moves and all their possible outcomes are calculated until a satisfactory one is reached. This can work in smaller games such as tic tac toe without very many repercussions, but even in a rather small game such as connect four it becomes very apparent that this method can become very computationally expensive. In a standard connect four game, there are 42! / 6^7 possible combinations of moves. While unlikely, it is possible that a brute force algorithm could exhaust many or all of these possibilities.

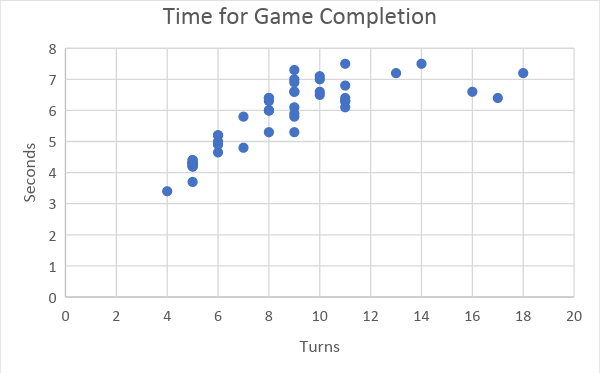
**Results:**

Minimax/Game Tree:

We expected this system to run very quickly as it is a standard method for creating AI for games. A large benefit of this system is that as the game develops, the computation time gets smaller and smaller as fewer and fewer choices become available. Due to this tapering, we expected the computation time to remain consistent when playing against someone at a given skill level. Our results lined up with these predictions.



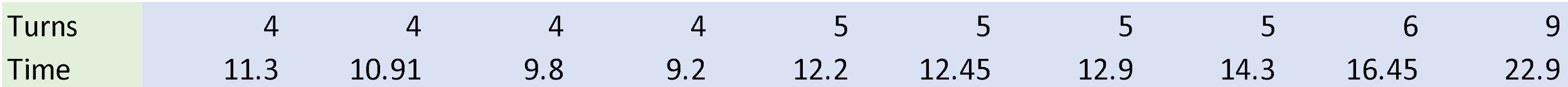
|  |  |  |
| --- | --- | --- |
|  | **Turns** | **Time (s)** |
| **Mean** | 7.935883 | 5.623328 |
| **Median** | 8 | 6 |
| **Mode** | 5 | 4.3 |
| **STDEV** | 3.227686 | 1.125824 |
| **Min** | 4 | 3.4 |
| **Max** | 18 | 7.5 |



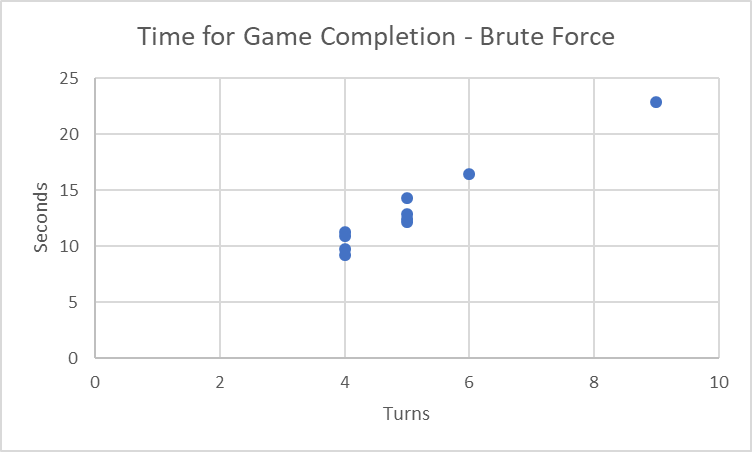
The Minimax/Game Tree AI never took longer than 7.5 seconds, even when the number of turns nearly doubled. This fell right in line with our prediction of the computation time tapering off as the total number of remaining states got smaller. It is important to also keep in mind that these numbers represent total computation time for the whole game, meaning that in an 10 turn game, the AI typically took .7 seconds to determine the best choice and make their move. The playability of this AI is very high, as a .7 delay is extremely quick, especially when compared to a human player.

Brute Force:

As expected, our brute force AI was extremely slow when compared to the minimax one. The min time of the brute force algorithm was 1.7 seconds slower than the max of the minimax, and the max time of the brute force algorithm was over 3 times that of the minimax.



|  |  |  |
| --- | --- | --- |
|  | **Turns** | **Time** |
| **Mean** | 4.93914 | 12.79197 |
| **Median** | 5 | 12.325 |
| **Mode** | 4 | - |
| **STDEV** | 1.523884 | 3.998782 |
| **Min** | 4 | 9.2 |
| **Max** | 9 | 22.9 |



An important note here is that even though this algorithm took twice as long on average, it took fewer turns for the game to reach an end.

**Improvements:**

As our data indicates, we would yield the largest improvement in efficiency by sticking to more efficient heuristics such as minimax. We could further increase the efficiency through methods such as alpha-beta pruning, but in the context of connect four, the gains may not be large enough to justify the man hours needed to implement it. On the other hand, if we were going to scale this AI to take on larger game, we would absolutely need to implement alpha-beta pruning to reduce our wasted searches.

The brute force method could be improved primarily through better coding. When dealing with an algorithm that makes so many comparisons, any optimizations we could make to the comparison loops or even variable initializations could have a modest impact on the overall runtime.

**Conclusion:**

Game Trees appear to be the best method for solving games like connect four. The tree does not need to be extremely large due to the limited number of legal plays per turn, and the code to implement them is not extremely complex. Comparing something that efficient to brute forcing a solution shows how far we have come in field of algorithms. It also emphasis the fact that many of our modern amenities like GPS directions and the Internet would not be possible at all if we had to brute force out way through every problem.