Minimax on TicTacToe

Ali Hamdani

February 2017

1 abstract

In this project I am going to analyze two different methods to solve adversarial problems, mini max and alpha beta pruning. I will also be talking about the heuristic function that I used to plug into the algorithms. For small problems like tic tac toe the mini-max function was enough to search through all the states and find the optimal solution however for Ultimate tic tac toe the mini max function was too time consuming considering the number of possible states the board could be in is a lot larger than in tic tac toe. I used alpha beta pruning to solve this problem since it does not require as much memory.

2 Introduction

Tic Tac toe is a very popular game amongst children because of its simple rules and short game time. This is also a good reason why we should use Tic Tac Toe as the game we should test different problem solving strategies on. The first method that I am going to discuss is minimax.

3 Data Structures

The data structures that I used in this program were split up into three groups Board, Game, and the AI. The reason why I split up the game and the AI is that I like to think of the AI as another player playing the game.

3.1 Board

I defined the board by a two of variables, a 2d character array and a turn counter. The board had a lot of methods of checking itself for errors and other mismatched cases so I did not have to worry about it when I designed the game. The board also holds a lot of methods that help the AI determine what move to make. As I said before the AI is another player so it looks at the board and determines the best possible outcome. The board is also a little more robust than was needed because I need certain information about hte board like the last move placed to accurately implement the NineBoard data structure. The biggest problem I had when doing this project was actually cloning issues. I realized that java usually assignes points to objects and assignments of objects so when I changed what I though to be an assignment of an object I acutally changed the original object. This was solved by heavy use of the clonable implementation.

3.2 Game

This data structure is self explanatory. Since I keep track of most things in my board data structure I don't need much in the game class besides how the game operates. This mean I need to keep track of how the game begins and how the game ends and how the game is played. So I have a lot of methods determine how a player sets a piece and which player is the X player (human vs machine).

3.3 AI

This is probably my favorite part of the project. The AI class holds the method that determines the best move based on the algorithms that I described in the abstract.

4 Methodology

4.1 Minimax

When writing a machine player the first thing we need to specify is what the machine is trying to do. In this case as with a lot of game solving problems the machine is trying to win the game and to win the game he needs to get three in a row. However in this environment the machine is not the only actor, there is in fact another player that the machine does not confer that also has the same goal but the two goals can not coexist. Either the machine wins or the actor wins. One way to determine which move to make is to enumerate through all the possible states in the list and take the one that looks the most promising but that would take an enormous amount memory and would not be very efficient. It also dismisses a crucial fact of the problem, when one person wins the other person loses and both players want to win.

Keeping that in mind suppose without loss of generality that you are playing the X player. Then that means every time you reach a terminal state that has three X's in a row that is a win for you and when you reach a terminal state that has three O's in a row that means a loss for you. We can model this as +1 and -1. Then that means than on each turn either X is trying to maximize its gains and O is trying to minimize. This will result in a min-max dynamic because you want the best possible option regardless of what O chooses. If you assume that O will choose the best option possible then you are also making the best option possible. I coded this into my program with the following code:

```
public int minimax(Board board,int depth , boolean max) throws CloneNotSupportedException
{
if(board.PlayerOscore() > 999)
{
return -1*board.PlayerOscore();
}
else if(board.PlayerXscore() > 999)
{
return board.PlayerXscore();
}
else if(depth > 8)
{
return 0;
}
if(max)
{
int bestVal = Integer.MIN_VALUE;
Board[] children = board.PossibleMoves();
for(int i = 0; i < children.length; i++)
{
int test = minimax(children[i] , children[i].turn, false);
bestVal = Math.max(test,bestVal);
}
return bestVal;
}
else
{
int bestVal = Integer.MAX_VALUE;
Board[] children = board.PossibleMoves();
for(int i = 0; i < children.length; i++)
{
int test = minimax(children[i] , children[i].turn, true);
bestVal = Math.min(test,bestVal);
}
return bestVal;
}
}
```

As I will discuss there are only 9! number of states that the board could be in, it is not very difficult for a computer to check through all of the possible states and determine which one would be the best choice to make. If we had a more complex board for instance a Ultimate tic tac toe board then we could not run minimax algorithm because it would take too much time.

4.2 Heuristic

Before I begin to talk about alpha beta pruning I need to talk a little bit about the heuristic that I used. For alpha beta pruning you want to be able to return a score for the state that you are currently in so that you can evaluate which states are better. In minimax the evaluation was either 1(X wins), 0(draw), or -1(O wins). How can we decide with a different board. The heuristic that I used would count the X's that occupy the row, column, and diagonal, it would also count the number of empty spaces. So if a row had 1 X and two empty space then I assigned 1 point for that board, if a row and two X's and 1 empty space then I would assign 10 points, if it had three in a row then 1000. This heuristic is based on the fact that empty spaces and X's have a potential for a game winning move, if a row has an O in then it does not serve towards winning the game.

4.3 Alpha Beta Pruning

Alpha beta pruning is heavily reliant on the heuristic that you chose. In this case I chose a heuristic that was based on the potential of winning a game. The following illustration from Wikipedia shows the process of alpha beta pruning.

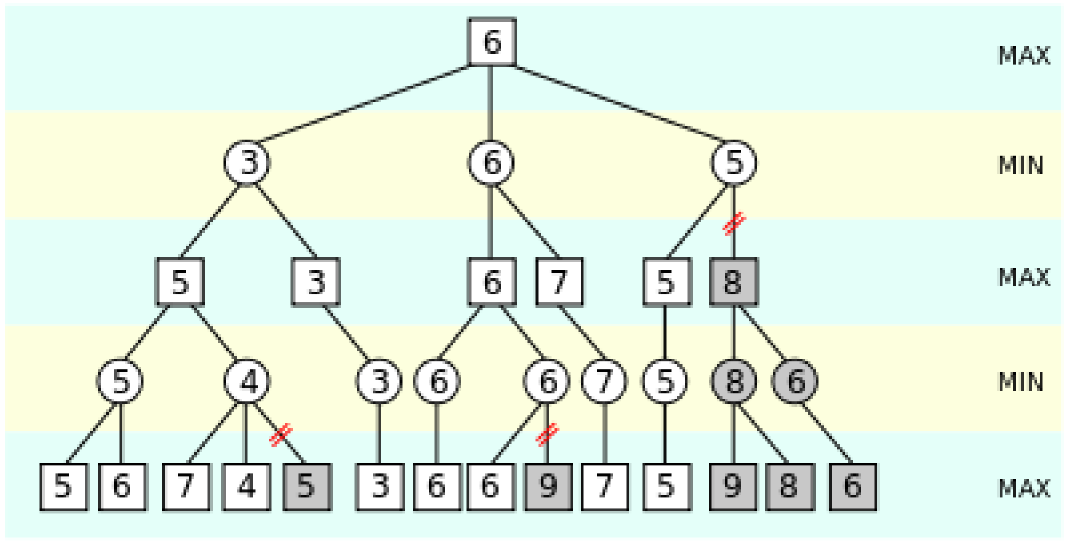

Figure 1: Alpha Beta Prune

As you see in the graph this algorithm will return the best score that it can in the depth determined by the creator of the machine.

5 Conclusion

Overall I think for nontrivial problems alpha beta pruning is definitely much wiser than a blanket mini max approach due to its compactness. I also want to point out that Alpha Beta algorithm grew exponentially as the depth increased. I personally had it set the depth to 7 even that begin to get comber some because the AI took a couple of seconds to evaluate all the possible states it could be in.

6 How to run program

All you really need to do is have all the files in one folder and run the test file. The main method should prompt you through the different options.

7 Bibliography

https://en.wikipedia.org/wiki/AlphaArtificial Intelligence: A Modern Approach