Mid-Term Report on Game-Player-Bot Project

Muskan Paliwal
Avikalp Kumar Gupta

September 30, 2019

Contents

1 Abstract

The project is about game dynamics, decision processes and decision-making automation wherein the goal is to implement Reinforcement Learning on a game called "Terminal" which is a very complex game with "2 player zero sum simultaneous game" with so many moving parts! Throughout the whole project I have been mentored by Avikalp Kumar Gupta(IIT Kanpur Alumni, 2016).Before starting with Terminal game, I was asked to start with some simple two player games like Tic-Tac-Toe, so the report is all about the implementation on Tic-Tac-Toe game.

2 Introduction

The project is more about implementation involving a lot of learning in coding skills, debugging, object oriented programming concepts and their importance, getting comfortable with open source version control system Git and moving to the real learning part it includes creating dummy bot of a particular game, applying MiniMax algorithm on the game and later applying Reinforcement Learning. The main goal is to see how well can reinforcement learning methods (like Q-learning) work in the same circumstances and what are the differences in the performance, training time and resource requirements between Q-learning and NEAT(a genetic algorithm).

3 Related Work

As stated in Introduction section, I went through multiple iterations of trials and errors.Here is the work I did in this project:
1. As I have to apply algorithms on Terminal game so my first step was to learn playing it and to learn writing algorithms for playing the game automatically.
2. I went through the GitHub repository of the game and learned what all the bots do in it. Also, learned what "Rewards" algorithm does and wrote my understanding of the algorithm in a doc file.
3. Next step was to start some real learning by going through NEAT(a genetic algorithm). In order to understand what NEAT does, I went through various blogs and did a lot googleing followed by some YouTube videos too. Somehow I was able to understand the Crossover and Mutation part but the Selection part still seemed tough for me to understand. NEAT being a complex algorithm remained out of my reach of mind and I was assigned to drop it here and start working on some simpler games like Tic-Tac-Toe and so did I.
4. I started working on Tic-Tac-Toe where I initially learned to create a Tic-Tac-Toe game. I created the game in python but didn't use object-oriented concepts of python in the code. So, Avikalp taught me the importance of object-oriented programming concepts and I wrote number of codes to create a perfect one with classes and objects so that I don't have to write a different code from starting whenever there are some changes either in game strategy or type of players to be made.
5. After successfully creating the game, I implemented a dummy bot which looks at the current state of the board and submits its move. Last part was to apply MiniMax algorithm where Minimax is a recursive algorithm which is used to choose an optimal move for a player assuming that the other player is also playing optimally. This is all about the project till now.

3.1 How did I become comfortable with GitHub?

As I was working remotely with Avikalp, So I was suggested to use GitHub and I started using it. Using open source distributed version control system while working includes the following:
1. What is open source distributed version control system? How is it different from centralized version control system?
2. What is repository and how to create repository?

3. How to clone repository in our local device?
4. How you can go back to the changes you made a week ago in your code using commit command?(This is the most important thing about Git. While creating a project using Git is beneficial just because of it. For every change, commit your changes to remote repository as well as local repository which will save all your changes with showing a part of code before changes and a part of code after changes.)
5. How to write a Readme file ?

3.2 Learning of object-oriented concepts

Throughout the whole project I was asked to write the whole code in object-oriented paradigm as if I use classes and objects in the code then I don't have to write a different code next time if I do any changes in game strategy or anything else. Therefore, I have created a different class for Game which contains game strategy and different classes for AI-player(submits it's move using MiniMax algorithm), Human player(submits it's move using user-input method.), and Dummy-player(submits it's move by looking at the current state of board.) Each of these classes are independent of each other.

3.3 What is MiniMax algorithm?

Minimax is an artificial intelligence applied in two player games, such as tic-tac-toe, checkers, chess and go. This games are known as zero-sum games, because in a mathematical representation: one player wins (+1) and other player loses (-1) or both of anyone not to win (0).

3.3.1 How does the algorithm works?

The algorithm search, recursively, the best move that leads the Max player to win or not lose (draw). It consider the current state of the game and the available moves at that state, then for each valid move it plays (alternating min and max) until it finds a terminal state (win, draw or lose).

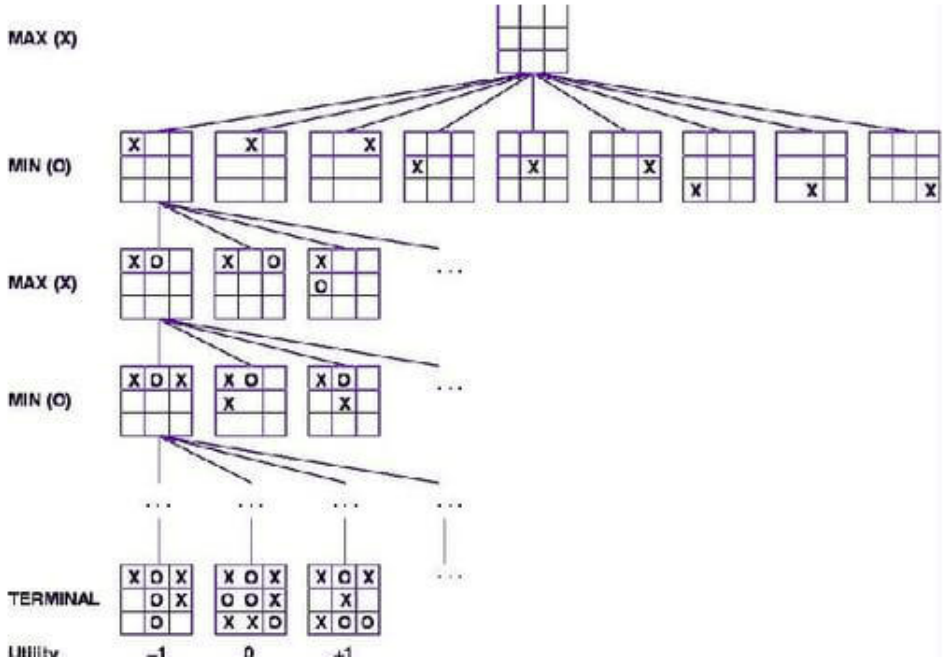
3.3.2 Understanding of the algorithm

The algorithm was studied by the book Algorithms in a Nutshell (George Heineman; Gary Pollice; Stanley Selkow, 2009). Pseudocode (adapted): There are two players involved in a game, called MIN and MAX. The player MAX tries to get the highest possible score and

MIN tries to get the lowest possible score, i.e., MIN and MAX try to act opposite of each other.

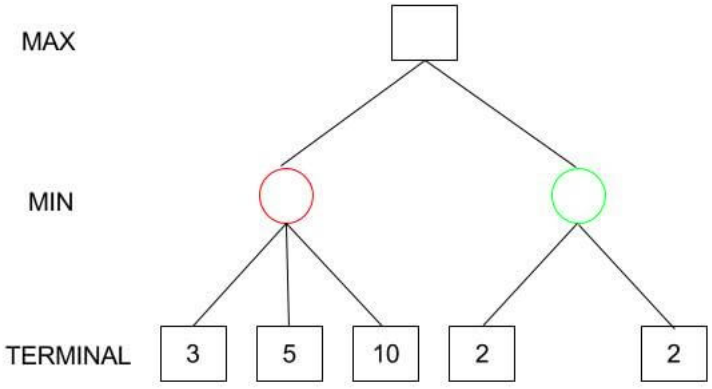
The general process of the Minimax algorithm is as follows:

Step 1: First, generate the entire game tree starting with the current position of the game all the way upto the terminal states. This is how the game tree looks like for the game tic-tac-toe.



Step 2: Apply the utility function to get the utility values for all the terminal states.

Step 3: Determine the utilities of the higher nodes with the help of the utilities of the terminal nodes. For instance, in the diagram below, we have the utilities for the terminal states



written in the squares.

Step 4: Calculate the utility values with the help of leaves considering one layer at a time until the root of the tree. Step 5: Eventually, all the backed-up values reach to the root of the tree, i.e., the topmost point. At that point, MAX has to choose the highest value.

In our example, we only have 3 layers so we immediately reached to the root but in actual games, there will be many more layers and nodes. So we have to evaluate MAX3,2 which is 3.

Therefore, the best opening move for MAX is the left node(or the red one). This move is called the minimax decision as it maximizes the utility following the assumption that the opponent is also playing optimally to minimize it.

To summarize, Minimax Decision = MAX(MIN(3,5,10),MIN(2,2))

= MAX(3,2)

= 3

3.4 Pseudocode for the algorithm

```
minimax(state, depth, player)

        if (player = max) then
                best = [null, -infinity]
        else
                best = [null, +infinity]

        if (depth = 0 or gameover) then
                score = evaluate this state for player
                return [null, score]

        for each valid move m for player in state s do
                execute move m on s
                [move, score] = minimax(s, depth - 1, -player)
                undo move m on s

                if (player = max) then
                        if score > best.score then best = [move, score]
                else
                        if score < best.score then best = [move, score]

        return best
end
```

4 Approach

Approach for the project was to start from simpler games than Terminal game as Terminal is very complex game with "2 player zero sum simultaneous game". So, I started with Tic-

Tac-Toe game which includes creating Tic-Tac-Toe, implementing dummy bot player which looks at the current state of the board and submits it's move, learning NEAT and MiniMax algorithm, and applying MiniMax on Tic-Tac-Toe. After getting comfortable this much, I would be applying all these on Terminal.

5 Results and Conclusion

Results and Conclusions I came up with are:
1.Using object-oriented concepts in our code make things much easier and make your long-long code readable very easily.
2. Using Git while working on any project can never let you worry about the changes we have made a day ago or a week maybe. All our changes are saved if we commit every single change to remote repository as well as local repository.
3. How MiniMax algorithm is used for two-player games to choose an optimal move for a player assuming that the other player is also playing optimally.

6 Future Work/Scope

Future work in this project can be applying Q-Learning on Tic-Tac-Toe as well as Terminal game.

7 References

Link to the GitHub repository of the project:- https://github.com/MuskanPaliwal/AI-Tic-Tac-Toe
For the whole project here is the list of references I used:
1. In order to learn object-oriented programming concepts in python:- https://www.youtube.com/watch?v=Z5JzLYMlist=PL-osiE80TeTsqhIuOqKhwlXsIBIdSeYtc
2. To get comfortable with GitHub:- https://www.youtube.com/watch?v=HVsySz-h9r4list=PL-osiE80TeTuRUfjRe54Eea17-YfnOOAx
3. To learn MiniMax:-
A. https://nicolodavis.com/blog/tic-tac-toe/
B. https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/
C. https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-

optimal-move/

D. https://www.youtube.com/watch?v=l-hh51ncgDI

E. https://www.youtube.com/watch?v=l-hh51ncgDIt=211s

F. https://www.youtube.com/watch?v=zp3VMe0Jpf8t=372s

4. To learn how to play Terminal game:-

A. https://terminal.c1games.com/

B. Link to the GitHub repository for Terminal Game: https://github.com/MuskanPaliwal/Terminal-Game-Player