3D Tic-Tac-Toe Agent

Anuj Kakde (20CS30005)

***Abstract*— There are 2 parts of this question, 2D version and 3D version**

A. 2D Version

The 2 dimensional TicTacToe used an environment created by me which is very similar to the gym_environment. The logic behind the curtains is the simple minimax algorithm in which one player tries to maximise the score of the board while the other one minimises the score.

The Game.py has the necessary code to play the game in the terminal. The program has two functions named humanplay and AIplay which are called from the play function. These functions plays the move selected by them (user in case of human and minimax in case of AI) and displays the board in the terminal. The play function runs the gameplay.

1) 3D Version: **The 3 dimensional version of TicTacToe is implemented by the help of reinforced learning. In this we basically make the AI agent play against itself many times, each time rewarding or punishing based on the outcome of a particular game. It then backtracks the actions taken to reach a certain outcome and updates the q-value of the state accordingly. The AI then stores the information of the usefulness of the move in a file called "Q-Table.txt". This completes the training of our AI agent.**

Now the AI agent is ready to play against human. This can be done by running the Game3D.py file which uses the gym_environment. In each game the AI agent takes help of the data collected after playing lots of games to decide which move to play.

I. INTRODUCTION

The task involved making of Artificial Intelligent agents which can play 2D and 3D TicTacToe optimally. It is a game with no graphics and one which can be played in the terminal. The learning curve for this question was quite steep but equally informative and interesting.

Implementation of this problem required a deep understanding of the Minimax algorithm and reinforcement learning. The environment for this problem was already given for the 3D version while I created a new environment for the 2D part of this problem.

The working of this program requires gym environment pre-installed in the system.

II. PROBLEM STATEMENT

This task has two parts. First one is to make a 2D version of Tic-Tac-Toe while the second part is to make its 3D version.

In the first one, we were required to write an AI agent to play 2D Tic-Tac-Toe game using minimax algorithm. Minimax algorithm basically is just an exhaustive search of the game space.

The second part is basically the extension of the first part. Here we needed to write an agent to play 3-D Tic-Tac-Toe.

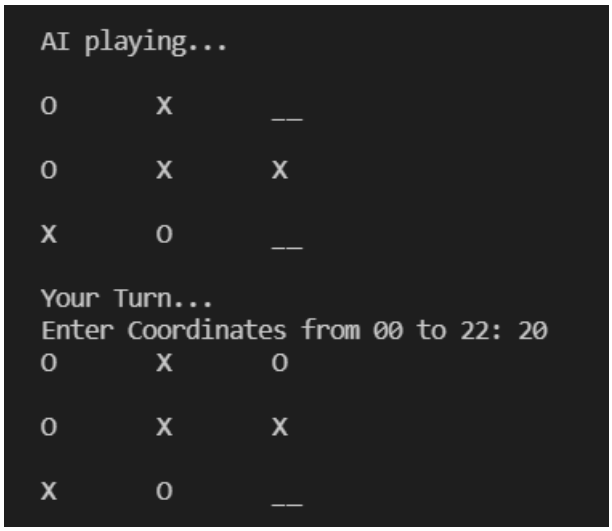

Fig. 1. 2D Tic-Tac-Toe

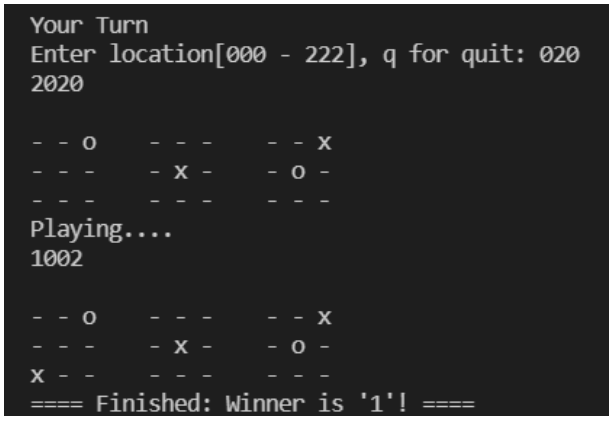

Fig. 2. 3D Tic-Tac-Toe

It is harder to use the Minimax algorithm for this part as the search space is much larger than the previous one. We would have to look into ways of optimising Minimax algorithm. Else we may also implement more advanced methods like Q-Value Learning. This game has been made in the gym-tictactoe environment.

The gym-tictactoe environment is basically a game environment specially made for tictactoe. It has functions necessary to implement the game.

Images from the original gameplay of 2D and 3D are shown in Fig 1 and Fig 2.

III. Related Work

For the implementation of 2D version, I needed to learn the minimax algorithm. While for the 3D version, I decided to use the reinforcement learning technique as it's approach was more efficient than the exhaustive search performed by the minimax algorithm.

Learning to use the gym environment was also essential before beginning to solve the problem. The gym environment is made specially for 3D version so for making the 2D version of the game I made my own environment similar to the gym environment.

IV. INITIAL ATTEMPTS

Initial attempts to solve the problem involved more of logic developing. I only read about the minimax algorithm and implemented it completely by myself. It took a lot of debugging but was successful at the end.

I decided to use reinforcement learning to solve the 3D part from the beginning. Understanding the logic behind it was quite simple but its implementation took some time.

Also initially I tried to implement 2D version in 3D environment but that did not work so I had to create a different environment for 2D.

V. Final Approach

This section is divided into two parts, namely 3D Version and 2D Version

A. 3D Version

1) ***General Setup****:* In the Game3D program we initially import packages such as sys, random, pickle that will be used subsequently. We also import the gym_tictactoe environment to use its methods. Next we open the Q-values.txt file which has the data of q-values of each state. Then we initialise env as the game environment.

2) ***Agent Classes****:* We have two agents in this game, human agent and AI agent. Both the agents have their own seperate class. Both these classes have two functions each namely __init__ and act. The __init__ function initialises the object of this class and gives it a mark (identity).

The act function of HumanAgent asks for input from the user and returns the move if it is legal else prints that the move is illegal and prompts the user for a valid move.

The act function of AIAgent uses the q-values to play an optimal move. It first takes each move from the list of available moves and stores the q-value of the resulting state and the action in a dictionary named A. It then returns the move which results in a state with maximum q-value.

3) ***Gameplay****:* During the gameplay, the program first defines two players and then declares a variable done which checks if the game is over. Next we run a while loop until done is false. Next the program checks whose turn it is in the game and then calls that agent's act function. Then the program plays the returned move on the board and prints the board in the terminal. The program then sees if the game is over. If it is over then it prints the result and breaks out of the while loop else the loop continues.

4) ***gym_environment****:* The gym_environment needs the gym package installed to be used. In the tictactoe_env.py file which we imported in the Game3D.py file contains useful methods.

First we have a agent_by_mark function which takes agents and mark as input and returns the agent who has the given mark. Next we have after_action_state function which a state and action as input and returns the resulting state.

Then we have a class TicTacToeEnv. The env variable in Game3D.py is an object of this class. During initialization, the board is empty.

This class has a step method which takes an action as its input, checks if the move is valid and returns resulting state, reward of the state, variable done, and other info about the state. Then there are methods like reset which resets the board, render which displays the board in the terminal etc.

Some other method of this class include show_result which displays the result in the terminal, available_actions which returns the list of available positions where a move can be made etc..

5) ***Training program****:* The logic behind the training program is that the AI Agent should play several games with itself and find out a logical way to win. We can achieve this by using a reward system such that the AI is rewarded each time it wins while it is punished it time it loses. The AI will then remember the steps it took to reach the final state and allocate new q-values to the intermediate steps depending on the reward it received.

The training program stores this q-value data collected after playing a lot of games and stores it in a text file using the pickle module. Also each time the training program is called it will append to text file named "Q-value.txt" containing the training data Also in each training loop it uses the already collected data to make smarter moves.

Currently the "Q-value.txt" contains sufficient data to make optimal moves. It is collected after running the Triaining.py program 2 to 3 times.

The "Q-Values-Human.txt" is the same data stored in human readable format just to inspect how the data is actually being stored.

B. 2D Version

1) ***Game.py****:* In the 2D version of the game I have used the minimax algorithm. The Game.py program uses the environment made by me in the environment.py. In the code of this file, we one function "mm" which is basically the recursive function implementing the minimax algorithm.

In the minmax algorithm in AI, there are two players, Maximiser and Minimiser. Both these players play the game as one tries to get the highest score possible or the maximum benefit while the opponent tries to get the lowest score or the minimum benefit.

Every game board has an evaluation score assigned to it, so the Maximiser will select the maximised value, and the Minimiser will select the minimised value with counter moves. If the Maximiser has the upper hand, then the board

score will be a positive value, and if the Minimiser has the upper hand, then the board score will be a negative value.

Each time when the AI has to play, it does an exhaustive search of all possible scenarios and returns the best possible move.

The program also has two functions named humanplay and AIplay which are called from the play function. These functions plays the move selected by them (user in case of human and minimax in case of AI) and displays the board in the terminal.

The play function asks the user if he/she wants to play first and then acts accordingly. Then a while loop starts the game and continues till the game has reached a conclusion.

I have deliberately not written the condition in which AI loses as this case is never possible. The Minimax algorithm for 2D TicTacToe cannot be beaten.

*2) **Environment**:* The environment for the 2D version of TicTacToe is similar to that of the gym_environment of the 3D version with minor changes that needed to be done to convert the game from 3 dimensional to 2 dimensional.

VI. Results and Observation

The Minimax algorithm is a good choice for 2D version. I could have implemented the $\alpha - \beta$ pruning for solving the same question but that would have increased the complexity unnecessarily as Minimax algorithm is fast enough in case of 2D TicTacToe.

Having said that, the Minimax algorithm would have been a poor choice while implementing the 3D version of this question as it's game space is very large compared to the 2D version. Optimizations would definitely have helped but the method of reinforcement learning is easier and faster.

VII. Future Work

To improve the Minimax algorithm further methods such as $\alpha - \beta$ pruning but that would not have made much difference to the user experience of this game.

While it's manageable to create and use a q-table for simple environments, it's quite difficult with some real-life environments. The number of actions and states in a real-life environment can be thousands, making it extremely inefficient to manage q-values in a table. In such cases we can use neural networks to predict q-values for actions in a given state instead of using a table. Instead of initializing and updating a q-table in the q-learning process, we'll initialize and train a neural network model.

CONCLUSION

This was a very interesting problem as we had to make a game which we had all played while growing up. It was very satisfying to see the how the program could beat the person who created it :P

The task also involved a learning curve which was would be helpful in the future as this is what the foundation of Artificial Intelligence looks like.

References

[1] https://www.freecodecamp.org/news/minimax-algorithm-guide-how-to-create-an-unbeatable-ai/

[2] https://nestedsoftware.com/2019/07/25/tic-tac-toe-with-tabular-q-learning-1kdn.139811.html

[3] https://github.com/Cypre55/gym-tictactoe3d