SWINBURNE UNIVERSITY OF TECHNOLOGY

COS30002

Tic Tac Toe

Task 3, Week 2

Matthew Coulter
S102573957

12/03/2020

Tutor: Tien Pham

Table of Contents

Design 1

The first design for the AI was not intended to be very smart, but was more a development of the original random number generator. The idea was that, instead of doing a random number over the whole board, the AI will choose moves based upon what the AI chose. The algorithm works by choosing a random number out of the adjacent and opposite squares that the AI just chose.

Here is the code:

```
def get_ai_move(self):
        '''Get the AI's next move '''
        # A simple dumb random move - valid or NOT!
        adjacentOptions = []
        # Note: It is the models responsibility to check for valid
moves...
        if (self.move == 0):
            adjacentOptions = [1,3,4]
        if (self.move == 1):
            adjacentOptions = [0,2,3,4,5]
        if (self.move == 2):
            adjacentOptions = [1,4,5]
        if (self.move == 3):
            adjacentOptions = [0,1,4,6,7]
        if (self.move == 4):
            adjacentOptions = [0,1,2,3,5,6,7,8]
        if (self.move == 5):
            adjacentOptions = [1,2,4,7,8]
        if (self.move == 6):
            adjacentOptions = [3,4,7]
        if (self.move == 7):
            adjacentOptions = [3,4,5,6,8]
        if (self.move == 8):
            adjacentOptions = [4,5,7]
        #if can complete row, complete row
        return adjacentOptions[randrange(len(adjacentOptions))]
```

Whilst this AI did not play much better than the complete random number generator, it did allow me to learn how the game was written for me to better plan out my version 2.

Design 2

Summary

Design 2 was much more thought out. I originally attempted making the ultimate algorithm based on cases but i felt as though this wasn't flexible enough, meaning that the AI would always play the same move according to what move was just played. Instead, I designed my version of a self learning AI, meaning that each game played is recorded, including the move order and whether it was a win or loss.

An example record looks like this:

126435870x
257086341tie
217364050

My AI for every move will search through the txt file containing all records for the games, and look for a matching game. It will only look for games that have led to a win, then play the next move that allowed that win to happen. If there are multiple winning games found with different moves, it will collate all of those available moves and choose a random one. If no winning games are found, it will then do the exact same process but for games that are tied. If wins or losses are found, it will find all the moves that lead to a loss and avoid them.

Here is the code for that.

```
def get_ai_move(self):
    '''Get the AI's next move '''
    avoidMoves = []
    potMoves = [0,1,2,3,4,5,6,7,8]
    with open('data.txt') as f:
      datamoves = f.readlines()

    for i in datamoves:
      if (self.moves in i):
        if ('o' in i): #take a win
          if (len(self.moves) < len(i)):
            if (i[len(self.moves) + 1] == 'o'):
              return i[len(self.moves)]
            if (self.moves in i):
              if (i[len(self.moves)] not in potMoves):
                potMoves.insert(-1,int(i[len(self.moves)]))
        if ('x' in i): #avoid a loss
          if (len(self.moves) < len(i) + 1):
            if (i[len(self.moves) + 2] == 'x'):
              if (i[len(self.moves)] not in avoidMoves):

avoidMoves.insert(-1,int(i[len(self.moves)]))
```

```
if (len(avoidMoves) > 0):
    for x in avoidMoves:
        if (x in potMoves):
            potMoves.remove(x)
for z in potMoves:
    if (self.board[int(z)] != ' '):
        potMoves.remove(int(z))
if (len(potMoves) > 0):
    return potMoves[randrange(len(potMoves))]
else:
    return randrange(9)
```

The reason I consider this ‘self learning’ is because everytime the game is played, it will record the game in the above format. Hence, I was able to build up a total of 12115 by versing this AI against a random number generator over prolonged hours. These are all different games that this AI will use to try and win.

Challenges Faced

One of the main challenges I faced was trying to figure out what move I should use to train my AI. Should I just use random numbers or the best move possible? Well, through trials of both methods, which included using Mihai’s AI as a ‘Perfect AI’, I found that it was better to go with the random number generator for one reason: it will eventually play every single possible game of tic tac toe. The total number of games possible is actually 255168, so my AI is only currently playing with a 5% knowledge of tic tac toe.

This leads me to the other problem: trying to record all of these games. Whilst I was able to ensure that no duplicate games would be played, the rate of which games were played slowed down dramatically. At the beginning, I was able to get 2000 in 5 minutes, but by the end, this same time led to only 100 more games. This must be a bug with my algorithm somewhere, but it has already gone through enough iteration that this bug is something beyond my current knowledge, and the actual methods for the type of AI will eventually be taught in this course.

Extension: AI vs AI

As previously mentioned, I played my AI against my friend Mihai’s ‘perfect’ AI. As expected, my AI of only 5% knowledge was crushed by his producing the following results:

(Mihai) X Wins: 157 = 78.5%
(Me) O Wins: 2 = 1%
Ties: 41 = 20.5%

These results are over a sample of 200 games. The games were recorded in the file: 200GamesResult.txt, found in the repository.