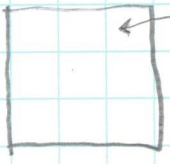
Tic Tac Toe Reinforcement Learning

- Tic Tac Toe is the simplest game I could think of for us to learn how to make the computer learn by playing against itself.
- Several notable programs have learned how to play at near champion / champion level using similar techniques to what we will learn in this section. Some are
  - > TD gammon - Backgammon by Tesauro
  - > Alpha go
  - > etc.

Concerning Tic Tac Toe -

There are lots of states. A rough and incorrect number is $3^9 = 19,683$ ← ouch



← each square can be empty, X or O → 3 states

9 squares

so 3^9

- In practice, there seems to be about 5,000 or so unique states, but that is still a large number — a couple of orders of magnitude more than the:
  - 6 State example
  - the cheese and pit game
  - ghost chase game
- Another problem here is that it is not immediately clear how to set up the
  - Matrix - action at current state leads to the next state

    Why? For each state in the game, especially the early states, there are many actions available, leading to many next state. Each successive move, reduces this problem
  - Immediate - the goal state is loaded with immediate reward. In tic tac toe there are 8 ways to win the game, with many paths to get there resulting in a

multitude of goal states

qTable - a one to one match up with qMatrix that holds the delayed reward values associated with the moves at each state. Since we are unsure about defining qMatrix, these issues will impact how we set up qTable.

We will retain the functionality of qMatrix, qTable, and qImmediate by using a

<u>state dictionary</u>

What is a state dictionary?

Each team, X and O will record each game. After game is over the dictionary will be updated.

What is a dictionary?

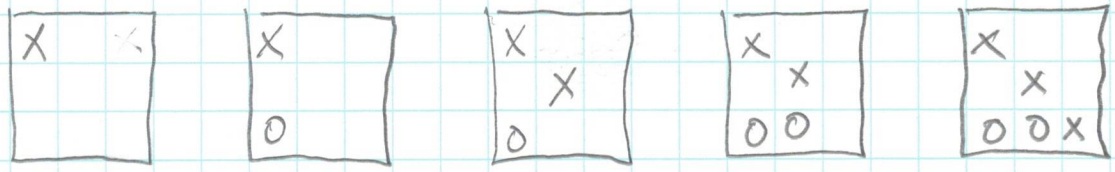
In python it is a data type that pairs a string with a value

myDictionary = {}

```
myDictionary["biff"] = 100
myDictionary
{'biff': 100}
myDictionary["biff"]
100
biff in myDictionary
True
```

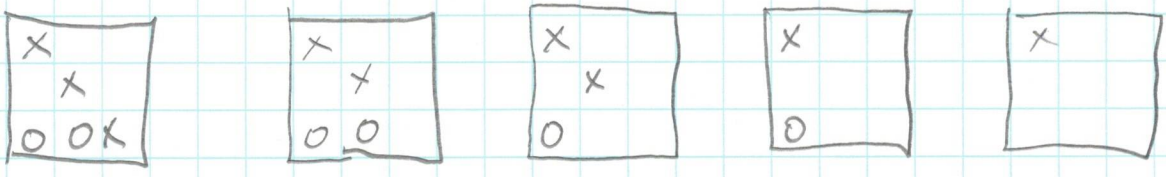
- So our plan will be to put each game board into the dictionary as a string, with the value associated with being the reward value
- About reward – if a player wins a game it gets positive reward – for the game each state leading to the win gets delayed reward based on the discount factor
  - if a player looses, they get negative reward, similar to above
  - a cat game results in some negative reward for each player

Example



X wins !

> We will reverse the game and figure out the reward, discount = .9



100 90 ≈81 ≈72 ≈63

* no learn rate used here

Update is based on the standard q learning delayed reward equation

$$qTable[state, action] = qTable[state, action] + ln * (immediate\ reward + discount * qTable[nextState, best\ action] - qTable[state, action])$$

- for tic tac toe only the last state (win state) gets immediate reward
- the $qTable[nextState, best\ action]$ is equivelent to cumreward because the game sequence at this time has already been determined - we end up with:

we reversed the game sequence, we are updating based on a game sequence that has already occured.

$$qTable[state, action] = qTable[state, action] + ln * (discount * cumReward - qTable[state, action])$$

Pseudo Code

```
xstates = {}
ostates = {}
xstates, ostates = numTTTexVsOh(ngames, xstates, ostates)

numTTTexVsOh(ngames, xstates, ostates)
    for(j=0; j<ngames; j++) {
        winner, game = tttExVsOh(xstates, ostates)
        if (winner is X) {
            updateDict(xstates, game, reward=100, lr, discount)
            updateDict(ostates, game, reward=-100, lr, discount)
        }
        else if (winner is O) {
            ....
        }
        else if (catgame)
            updateDict(xstate -- reward=-20
                       --)
            updateDict(ostates -- reward=-20
                       --)
```

15,000 works well

```
updateDict(states, game, reward, lr, discount)
    nGame = reversed(game)
    # for each board in the game, see if it is
    # in the dictionary
    # if not, put it in, otherwise update it's
    # reward state.
    currReward = reward
    for board in nGame:
        bString = str(board)
        # if the board is not in dictionary
        # put it in, initial reward is 0
        if (bString in states) == False
            states[bString] = 0
        # update reward
        states[bString] = states[bstring] +
                lr * ((discount * currReward)
                    - states[bstring])
        currReward = states[bString]
        # the next board back will get
        # currReward as its starting place.
```

```
## ExVsOh(xstates, ostates)
    board = initBoard()
    turn = X
    play = True
    game = []
    while (play) {
        game.append(board)
        moves = getMoves(board)
        chance = getRnd()
        if (exploit(chance) == True)
            if (turn is X's)
                find board + move is
                xstates with higest
                reward
                > that is the best move
            else if (turn is O's) {
                ....
        else {
            select a random move
        }
        board = board + move
        win, winner = checkForWin(board)
        if (win)
            play = False
    return winner, game
```

moves is ← an array of row, col pairs of available spots in board

How to do this.

- Write a Dimple person against computer tic tac toe first
  - computer picks randomly from available states
  - win / loss / cat are detected

- Next implement
  ttt ExVs Oh
  then
  Num ttt ExVs Oh and update Dict
  check for win, get Moves, etc
  will be in simple tic tac toe

- Implement ttt ComputerVs Person
  In this one computer uses dictionary to play person