Alex Hall

UMBC ID: Hall9

CMCS 341

Project 4 - Tic-Tac-Toe

P4 Answers

1. (2 points) How did you represent a board in your code? Why did you choose this representation?

Answer:

I made a class called board and it holds the serialized representation of the 3x3 array for that config, it also holds the wins, losses, ties, probability and times played for that config. It also has a value called weight which is simply weight = losses-wins. The serialized representation of the board is converted to binary. What this mean is, 'X' is 01 and 'O' is 10, also an empty space on the board is 00. Putting all those binary bits together starting from top left as being the 2 most significant binary bits with in the larger 18 bit binary number.

2. (3 points) Describe the hash function you used.
How well does your hash function distribute the game boards in the hash table?
Provide an analysis of your hash table statistics to support your claim.

Answer:

The hash function works but having a key and data at each location in the hash table. So, I give the hash function the serialized representation of the board which it then using modulo with the hash table size to calculate an address to place that data. My hash has 8 – 1. In other words, for any config on a board it really has 8 configs that are exactly all like. By means of symmetry (by rotating and mirroring). I find the lowest serialized binary representation among those 8 symmetric configs and I hash those.

```
The random player (X) just moved:
.       .       .
.       .       X
.       .       .

Potential Moves (found in hashed) =  serializedBoard:96  %P:0.0 Wins:0 Ties:1 Losses:2 P:3 Weight:2
Potential Moves (found in hashed) =  serializedBoard:72  %P:0.0 Wins:0 Ties:0 Losses:2 P:2 Weight:2
Potential Moves (found in hashed) =  serializedBoard:6  %P:87.47 Wins:14050 Ties:728 Losses:1284 P:16062
Weight:-12766
Potential Moves (found in hashed) =  serializedBoard:1152  %P:0.0 Wins:0 Ties:1 Losses:1 P:2 Weight:1
Potential Moves (found in hashed) =  serializedBoard:516  %P:0.0 Wins:0 Ties:0 Losses:1 P:1 Weight:1
Potential Moves (found in hashed) =  serializedBoard:96  %P:0.0 Wins:0 Ties:1 Losses:2 P:3 Weight:2
Potential Moves (found in hashed) =  serializedBoard:72  %P:0.0 Wins:0 Ties:0 Losses:2 P:2 Weight:2
Potential Moves (found in hashed) =  serializedBoard:6  %P:87.47354003237456 Wins:14050 Ties:728 Losses:1284
P:16062 Weight:-12766
```

```
The learning player (O) just moved:
.            .            .
.            .            X
.            .            0
In the past, this move has led us to a win 87.47 of the time.
```

You can see that multiple "serializedBoard" numbers are the same. Which really mean, for this config there is really only 5 moves rather than 8.

3. (5 points) At some point your smart player will have to decide to choose a previous move that has led to win or try a new move which may eventually lead to a higher winning percentage. What criteria did you use to determine whether the smart player should repeat a previous move or try a new move?
How did changing this criteria affect the learning curve of the smart player.

Answer:

I use two Lists, one list contains the previous moves (or found in the hash table), the other list contains new moves (or moves not found in the hash table). If the pervious moves list is empty then get a random move from the new move list. If the pervious moves list AND the new move list are both NOT empty then I get the highest probability from the previous moves list. If that probability is below a minimum probability (which 60% worked well) then I get the one with the lowest weight.

This criteria is the brains of the AI in terms of which is the best move to go. I started out only comparing probability, but it would never go to new moves, so going to new moves early within the 18000 games run had a hug affect with the learning curve, the smart player has to see more and more moves to truly find the best move to win.