Tic-Tac-Toe game report

Ying Yi

Contents

Abstract

Tic-tac-toe game is a classical adversarial game in a 3x3 gird. Two players in this game will alternatively make moves and each player has information about opponent decisions. If we choose one player to be AI, it will make optimal choice for each turn. For maximizing its performance, the most common strategy is minimax algorithm using alpha-beta pruning. However, I want to use heuristic function as evaluation method to quantify the goodness of each choice. Minimax algorithm will let computer do the best move in each state. Choosing a good heuristic function may also make computer perform well. Also, heuristic function is easy to implement and help save the computation. In this case, there is no significant time-consuming difference between two strategies since it is a simple game without too much computation. Thus, I will focus on their performances by comparing the results of game after implementing each algorithm.

Introduction

In my project, we choose X to be computer and O to be user. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. Players may soon discover that best play from both parties to a draw. We want to design a strategy implemented by computer to make a best move considering current state. Thus, we use minimax algorithm as the best choice. Tic-tac-toe requires you to lookahead one opponent's move to ensure that you will not loss. That is, you need to consider your opponent's move after your next move, which is also the essence of minimax algorithm. And we try to use heuristic function to do the same thing. In my program, I need to create a game board for computing next move and placing it. Then, I can implement strategies and see the result.

Approach

First strategy is heuristic evaluation function on board. In this strategy, we need to formula a heuristic evaluation function, which returns a relative score, e.g., +100 for computer-win, -100 for opponent-win, 0 for neutral, and a number in between to indicate the relative advantage of the computer vs. the opponent.

In Tic-Tac-Toe, a possible heuristic evaluation function for the current board position is:

- +100 for EACH three in a line for computer.
- +10 for EACH two in a line (with an empty cell) for computer.
- +1 for EACH one in a line (with two empty cells) for computer.
- Negative scores for opponent, i.e., -100, -10, -1 for EACH opponent's 3-in-a-line, 2-in-a-line and 1-in-a-line.
- 0 otherwise (empty lines or lines with both computer's and opponent's seeds).

For Tic-Tac-Toe, compute the scores for each of the 8 lines (3 rows, 3 columns and 2 diagonals) and obtain the sum. To implement this strategy, I need to compute the score for all the valid moves, and place X at the position with the highest score.

Second strategy is minimax algorithm.

In Tic-Tac-Toe, the principle of minimax is to minimize the maximum possible loss. As an illustration, suppose that there is only one move for available position per player in each turn.

Furthermore, an evaluation function has been defined, which returns +1 if the computer wins, -1 if the computer loses, and a score in between to reflect the relative advantage of the computer. Computer (or the maximizing player) is represented by cross. The user (or the minimizing player) is represented by circle. We limit the search tree by pruning unnecessarily searching parts. And we recursively call the minimax algorithm and return the result until we reach the leaf nodes.

We use alpha-beta pruning in minimax algorithm to reduce the number of nodes that needs to be evaluated in the search tree, which is also an efficient way to implement the algorithm since it returns the same move as minimax would but prunes away branches that cannot possibly influence the final decision. In the algorithm, two parameters are needed: an alpha value which holds the best MAX value found for MAX node; and a beta value which holds the best MIN value found for MIN node. As illustrated, the remaining children can be aborted if alpha > beta, for both the alpha cut-off and beta cut-off. Alpha and beta are initialized to -∞ and +∞ respectively.

Related Work

1. App class

Run tic-tac-toe game by creating a new object Game.

2. Game class

Implement a sequence of methods:

initializeGame(): Set up the game board including 9 empty cells.

displayBoard(): display the game board.

Your board may look like this ('-' represents an available location)

```
- - X
- O -
- - X
```

chooseAlgorithmType(): we need to choose one of two algorithms: minimax (using alpha beta pruning) and heuristic.

your board will look like this

```
- - -
- - -
- - -
Which algorithm? 1 – minimax ; 2 – heuristic
```

makeFirstMove(): we should let computer choose a random position when computer makes a move first.

Your board will look like this if you choose computer first

```
Who starts? 1 - Computer ; 2 - User
1
- - -
- - -
- - X
User move:
```

playGame(): When the board is running (no draw and no one wins), we read in the user's input of position choice and display the board. Then, let computer make a move using the chosen algorithm (minimax or heuristic) and display the board finally.

checkStatus(): It is used to determine whether it is computer win or user win or a draw and print out the result.

3. Cell class

Store cell values, including row number x, column number y, minimaxValue and heuristicValue. We can also print out minimaxValue and heuristicValue for each cell.

4. Player class

We use enumerate method to create 3 different players: COMPUTER, USER and NONE.

5. Constant class

This class is used to set the size of game board which is 3.

6. Board class

Define several methods, including minimax algorithm and heuristic function.

```
public boolean isWinning(Player player)
```

Checking whether there is a player (computer or user) having won the game. Going over 3 columns, 3 rows and 2 diagonals to check.

```
public boolean isRunning()
```

By checking whether the board is running, strategies can be determined to implement or not. If there is a player wining or there is a draw, return false. Else, return true.

```
public List<Cell> getEmptyCells()
```

Return all available cells so that we can implement minimax algorithm or heuristic function.

Heuristic function:

First define functions to get the heuristic score of game board.

```
public int getHeuristicScore()
```

This function directly evaluates heuristic value of the board based on current state. And it is implemented by evaluating 8 lines of the board by calling function evaluateLine() and sum it up. There are 3 rows, 3 columns and 2 diagonals to be evaluated.

```
private int evaluateLine(int row1, int col1, int row2, int col2, int row3,
int col3)
```

This function will go over each cell, that is (row1, col1), (row2, col2) and (row3, col3) and then return evaluated score which is calculated by specific method.

Then define a function to update heuristic value of each cell based on current state.

```
public void callHeuristic()
```

Clear all cell heuristic values and call heuristic() function to reset it.

```
public void heuristic()
```

Finding all available (empty) cells first by using getEmptyCells() function and then calculate its current heuristic value by make a move for one available cell. After storing this value, this point should be reset and try other available cells. At last, rootValues will store all heuristic values of available cells.

Finally, implement heuristic function by using previous method.

```
public Cell getBestMoveHeuristic()
```

It chooses the cell with greatest heuristic value and return the cell.

Then, we need to define two main algorithms or strategies.

Minimax algorithm:

First define functions to get the minimax score of game board.

```
public int minimax(int depth, Player player, int alpha, int beta)
```

This function uses backtracking method. First checking status of board, return +1 if computer wins, return -1 if user wins and return 0 if it is a draw. Traversing all available cells and making a move for one cell, changing player's turn to its opponent and call minimax() function again with depth increment and storing the score of current state. Storing each cell with minimax value in rootValues if depth is 0. Resetting the point and access other available cells at the end of for loop. And then, updating alpha score if current score larger than alpha and updating beta score if current score smaller than beta. If it is an alpha-cut case or a beta-cut case (alpha greater than beta when it is user's turn or computer's turn), we can prune the game tree to reduce search space and make it faster. Last, returning maximal score of all potential states if it is computer's turn and return minimal score of all potential states if it is user's turn.

Then define a function to update minimax value of each cell based on current state.

```
public void callMinimax(int depth, Player player, int alpha, int beta)
```

It is to clear all cell minimax values and call minimax() function to reset it.

```
public Cell getBestMoveMinimax()
```

Choosing the cell with largest minimax value and return it.

Implementation

I use JAVA programming to create the game on eclipse. There are too many ways to implement the game since user can choose several positions to make a move each turn. I just show an example of playing this game.

```
Which algorithm? 1 - minimax ; 2 - heuristic
2
Who starts? 1 - Computer ; 2 - User
1

- - -
- - -
X - -
User move:
1 1

- - -
- O -
X - -
Cell values: (0, 0) heuristicValue: 10
Cell values: (0, 1) heuristicValue: 1
Cell values: (0, 2) heuristicValue: 1
Cell values: (1, 0) heuristicValue: 9
Cell values: (1, 2) heuristicValue: 1
Cell values: (2, 1) heuristicValue: 9
Cell values: (2, 2) heuristicValue: 10

X - -
- O -
X - -
User move:
0 1

X O -
- O -
X - -
Cell values: (0, 2) heuristicValue: 1
Cell values: (1, 0) heuristicValue: 91
Cell values: (1, 2) heuristicValue: 2
Cell values: (2, 1) heuristicValue: 19
Cell values: (2, 2) heuristicValue: 10

X O -
X O -
X - -
Computer has won
```

Conclusion

It is obvious that minimax algorithm will always get the best choice based on current situation and computer will not loss in any cases, it will only win the game or make a draw. As for heuristic function, it performs well in most cases although it may lead computer to a loss sometimes. There is no significant difference on time complexity between these two algorithms since this game requires too small computation to notice the difference.

There is an interesting phenomenon, that is computer may not choose the position which leads to a direct win. The reason is when using minimax algorithm, there is several positions having same minimax value in some cases and computer will choose the first available position with maximal minimax value. However, when using heuristic function, this phenomenon will not happen since the position which leads to a direct win will defiantly have the greatest heuristic value which is chosen by computer.

References

1. WIKIPEDIA
2. http://www3.ntu.edu.sg/home/ehchua/programming/index.html
3. https://kartikkukreja.wordpress.com/