Polytechnic Institute of New York University

Artificial Intelligence

A 4 x 5 Tic-Tac-Toe Game

By

Gopikrishna Sathyamurthy

#0533465

Important Pages

Page No.

- Evaluation Function 2
- Cut-off function 4
- Compiling and Running the Project 7

Table of Contents

Important Pages ii
1. Introduction 1
2. Project Design 2
2.1. The Main Form Interface 2
2.2. Board Class 2
2.2.1. Board Status 2
2.2.2. Evaluate Board (Evaluation Function) 2
2.3. Game Tree Node Class 3
2.4. Alpha-Beta Search 3
2.4.1. Depth Limit Function (Cut-off function) and Difficulty of Game 4
2.5. Game Tree 5
2.5.1. Depth Limit 5
2.5.2. Expand Node 5
2.5.3. Sub Tree Generator 5
2.5.4. Next Best Move 5
2.5.5. Immediate Resulting 5
2.6. Features 6
2.6.1. Initial Move 6
2.6.2. Statistics Interface 6
2.6.3. Difficulty Interface 6
2.6.4. Progress Control 6
2.6.5. Application Settings 6
3. Project Implementation 7
3.1. Specifications 7
3.2. Compiling and Running the Project 7
References 8

1. Introduction

The project implements a typical Interactive M-N-K (4-5-4) Game for a person to play against a computer. A simple m-n-k game is an abstract board game in which two players take turns in placing a stone of their color on an m x n board, the winner being the player who first gets k stones of their own color in a row, horizontally, vertically, or diagonally.

In the project, the player uses O to mark and CPU uses X to mark on the board of 4 x 5 grid sizes. The player and the CPU take turns to make these marks and whoever gets the first four consecutive marks wins the game.

The project is implemented using Alpha-Beta Search Tree with Pruning to get the next best move for X every time the player marks an O on the board. The Alpha-Beta Search algorithm works on a Minimax Tree where the player and CPU are alternatively given minimum and maximum chances, respectively to win or to complete the board.

2. Project Design

The project contains following modules to accomplish the successful implementation of the game.

2.1. The Main Form Interface

This is the main interface of the program that allows the player and the CPU to make their moves. This interface contains 4 x 5 = 20 Picture Boxes that will be filled in with a picture of X or O depending on the current player's turn.

This calls the CPU to make a move every time the player has made a move by creating a Game Tree and searching for the next best move.

2.2. Board Class

This class has a 2D-array of 4 x 5 sizes that stores the state of the board. The states are defined by an enumerated type State with values of X, O and N representing CPU's move, player's move and a null placeholder respectively.

2.2.1. Board Status

Board Status is an enumerated type with values of Incomplete, Success, Draw and Failure, each telling the current status of the board. The Status is determined by first checking on a win for either X or O, in row, column, diagonal wise (both 45 and 135 degrees). These are individually done by checking every possible series that can be consecutively formed on the board.

2.2.2. Evaluate Board (Evaluation Function)

For a successful board i.e. if X wins or O wins or for a complete the board, the evaluate function is not called. Instead, the board is assigned with a +200 for X's win, -200 for O's win and a zero for Draw. When the board is Incomplete and needs to be evaluated as a terminal node in the Game Tree, the evaluation function is used.

Evaluation function for this game is extended from a formula given in the book *Artificial Intelligence, a Modern Approach by Stuart Russell and Peter Norvig (page 198)*. The evaluation function given was for a 3 x 3 tic-tac-toe game, which is as follows:

$$Eval(state) = X_1 - 3O_2 + 3X_2 - O_1$$

Where X_n represents the numbers of n consecutive X's on the board and O_n represents the numbers of n consecutive O's on the board.

By extending this formula to a 5 x 5 which covers the current problem of 4 x 5, the project implements an evaluation function as follows. Considering the formula for 3 x 3 boards, being in the form of 1-3-3-1 for $n = 3$ in the Pascal's triangle, the project selects the form for $n = 5$ for the 5 x 5 boards in the same Pascal's triangle.

$$1\ \ 3\ \ 3\ \ 1$$

$$1\ \ 4\ \ \ 6\ \ \ 4\ \ 1$$

$$1\ \ 5\ \ 10\ \ 10\ \ 5\ \ 1$$

Forming a formula of $X_1 - 5O_2 + 10X_3 - 10O_3 + 5X_2 - O_1$, which evaluates for a consecutive 1 X to 3 X's or 1 O to 3 O's and returns the evaluated value.

2.3. Game Tree Node Class

This class inherits the properties of built-in Tree Node class in C#. This allows the project to perform all basic tree operations and extend the class to have project-specific fields like alpha, beta and node values for each Tree Node in the Game Tree. Every Tree Node has an instance of board to allow Alpha-Beta Search in the tree.

2.4. Alpha-Beta Search

This module implements the basic Alpha-Beta Search with Pruning starting with the root node of the Game Tree. The utility function here uses the Evaluate function from the Board class to evaluate the terminal nodes of the tree. The utility function assigns a -200 if O wins and a +200 if X wins and 0 if the board is Draw and assigns the value received from the Evaluate function in any other cases i.e. for incomplete boards.

2.4.1. Depth Limit Function (Cut-off function) and Difficulty of Game

This function calculates the depth at which the visiting of the nodes to be cut off and start evaluating the terminal nodes for their values. The depth is increased exponentially with respect to the number of remaining nodes. The difficulty of the game is tied to cut-off function so as to determine how deep the search is to be done so that the difficulty for the player is as desired. If easy mode is the depth is less and for high difficulty, the depth is more, so the chance of X winning is increased.

Also to manage the system memory, the cut-off function is determined such that it generates only a number of nodes which comes under the handling limit of the system. Eventually one can say that at hard mode, the highest depth is reached such that the number of nodes generated is within the limits of system memory.

The depth is exponentially increased with decrease in the number of empty cells on the board to have a fair-play between CPU and player. The cut-off function is calculated based on the depth that a system can handle; starting with 4 for 19 possible moves (19 empty cells) after the first move is made. The capacity is exponentially increased till the number of remaining cells is 9 and remains at 9.

Subtracting 4 from all the values, an exponential increase is obtained from 0 to 5 and value of 4 is used as difficulty constraint. So, on hard to easy scale, this value goes from 4 to 2 thus limiting the depth of search and limiting the chances for X.

The formula used for calculating the depth or cut-off criteria is as follows:

$$depth = L + \begin{cases} 0, & x < 0 \\ \left(e^{\frac{\log(5)}{8}}\right)^{x}, & 0 \le x \le 8 \\ 5, & x > 8 \end{cases}$$

Where L is the difficulty level and x goes from -3 → 8 while n (number of empty cells) goes from 19 → 9.

2.5. Game Tree

Game tree is the one that is used to do everything in the project. Game Tree is refreshed every time player makes a move.

2.5.1. Depth Limit

The cut-off function is used to limit the number of nodes generated at a time in the Game Tree and also to limit the depth that is expanded.

2.5.2. Expand Node

This section checks the depth, and the status of the board to be incomplete to call the function that generates all the child nodes for the given node.

2.5.3. Sub Tree Generator

This section is called by Expand Node. It generates and returns a list of nodes each containing a board with a possible move on top of the parent node. The returned value is then assigned as the list of child nodes in the parent node that is being currently expanded.

2.5.4. Next Best Move

After generating the Game Tree, till the given depth or till the complete of board, the Game Tree calls the Alpha Beta Search to get the next best move for the root node. The next best move is then returned to Main Form Interface to perform the move for CPU on the board.

2.5.5. Immediate Resulting

Sometimes, the move is only a single step to win the entire board and it is really expensive to create a Game Tree and perform Alpha-Beta Search. So this function generates next winning move and returns to the Main Form Interface to perform the winning move. This increases quicker response and better performance and also the chance for X to win.

2.6. Features

The project also provides few features.

2.6.1. Initial Move

The project allows the user to make a choice on playing first or second, and in the case of latter, a random move is done by CPU instead of using expensive Alpha-Beta Search. This is reliable because the initial value for all the positions in an empty board will have the same value.

2.6.2. Statistics Interface

In the menu strip of the game, one can select the option “Statistics” under Game menu, and view the wins, loses, draws and winning percentage of the player till now. The values can also be reset to initial values.

2.6.3. Difficulty Interface

In the menu strip of the game, one can select the option “Difficulty” under Game menu, to change the difficulty level for the game. Once changed it is always set to the latest value entered until it is changed.

2.6.4. Progress Control

The player can exit or start a new game as desired, by accessing “Exit” or “New Game” from Game menu or by pressing “Alt + F4” or “Ctrl + N” keys, respectively.

2.6.5. Application Settings

The application holds four fields under $Properties.Settings.Default$ that stores the values of difficulty, wins, loses and draws. These can be accessed under Properties in Visual Studio.

The above listed are modules that are implemented and used by the project to successfully implement the 4-5-4 Tic-Tac-Toe Game.

3. Project Implementation

3.1. Specifications

- Language: C#
- Platform: .NET Framework 4.5
- Developer Environment: Visual Studio 2012

3.2. Compiling and Running the Project

This project was developed in Visual Studio 2012, on .NET Framework 4.5. So, the minimum requirement for compiling the project is to be done on the given environment or an equal one. The following steps can be used to run the program.

1. Open the **TicTacToe.sln** or **TicTacToe.csproj** (inside the Project folder) file in Visual Studio 2012, from the Project Directory.
2. Go to Debug→Start Debugging or hit F5 or press the small green play button on the tool strip to run the application.
3. The output file can be found under Project folder/bin/Debug, if the configuration is set to debug, or under Project folder/bin/Release, if the configuration is set to release.

References

1. Artificial Intelligence, a Modern Approach, 3rd Edition by Stuart Russell and Peter Norvig
2. Wikipedia.org/MNK-game
3. Wikipedia.org/Tic-tac-toe
4. Google: Ole Miss Lecture Slides, slides 4 to 7