ConnectM Computer Game

Russell Plenkers and Phillip Byram

Introduction

Connect 4 is a zero-sum two player adversarial game wherein players take turns dropping discs down columns to make stacks of pieces. The goal of the game is to Connect 4 of the player's discs in either a column, row, or a diagonal. Additionally, an abstraction was created to allow the Connect 4 game to be played as Connect M, so that a number other than 4 could be specified as a win condition for the game. The game board size should also be variable so that connecting higher numbers of tiles in a direction is still feasible. For purposes of simplicity, the board dimensions will be represented by a single variable, indicating both the width and the height.

Before being able to create an artificially intelligent player to play against a human player in this game, the first step is to create the game with two human players. After the game is playable by two humans sharing a console and a keyboard, the work on the Artificial player can begin.The Artificial player has two notable requirements to function. The first being the Utility function which evaluates the utility of a given board. The second being a way to implement that utility function to make decisions about the best moves to make.

The Game

The game is implemented by a Game class which creates a board with two players in an alternating queue. At this point in time both players are human and use the same input device.

Piece is an enum which contains the definitions for the different kind of pieces that can be placed in the board which are X, O, and NONE. Piece also supplies a way to get the name of the Piece.

Board is an interface that defines a board as something that can, most significantly, check for a win and place a piece. A Board also always has a getSpaces function which returns a two dimensional array of Pieces. Before the Game begins the Spaces on the board should all have the NONE piece. Board.didWin returns what piece has won the game and if noone has won yet, it returns the NONE piece. Board.placePiece takes in a Piece and an integer column and returns a success condition. There is also some error checking in placePiece to determine if the column was full. Board is implemented via the class QuadraticBoard which specifies a board with equal width and height. It implements didWin by checking each subboard of size M in the Board for an array of all one type of piece in each direction.

The Player is also an interface that knows if it isTurn and what the nextMove is. nextMove returns the success of the move (usually derived from Board.placePiece). Usually the implementation of a Player knows about the Board the game is being played on which is passed in the initialization. InputPlayer is the human implementation of Player that uses System.in.

The Artificial Player

The ArtificialPlayer is an implementation of the Player interface, so that the Game class is able to treat both InputPlayer and ArtificialPlayer as just a Player. ArtificialPlayer's implementation of nextMove is the core mechanic of the AI functionality in ConnectM.

The Utility Function

Before ArtificialPlayer can implement nextMove, it needs to be able to evaluate a Board's "goodness." This is defined as the Utility of a Board. In a static class Utility, a calculate method is defined which simply takes in a Board, which piece is 'mine', and which piece is my opponents.

If the board contains a win for 'my Piece' or 'their Piece', the calculate method returns positive or negative infinity, respectively. Otherwise, the calculate method iterates through all the sub-boards (similarly to board.didWin), and awards utility points to certain conditions. Finally, calculate determines the utility points for both players and finally returns the difference between 'my points' and 'their points' as the utility for that board.

Calculate awards a utility point for each piece in each piece array (of size win condition) that has 'my piece' and takes a point away for each space with 'their piece'. Additionally any of those arrays that have no enemy pieces, a point is given for each NONE piece and more and more points are given each time 'my piece' is found and 'their piece' has not been found in this piece array.

The Search Tree

ArtificialPlayer implements a search tree which provides an action that is to be used as the return for nextMove. The tree is a data structure created via recursive calls to each of the two players anticipated moves. The purpose of the game tree is to calculate all permutations of possible moves for both the human player (min) and the AI (max), thus enabling the AI to choose its best possible move. The min player is assumed to be trying to minimize the utility of the board using only logical moves, while the max player should be maximizing the utility of the board. By default the game tree has a depth of 5 nodes deep and is comprised of StateNodes which have alpha and beta values used to prune the tree along the way if a branch is discovered as unexplorable.

The search algorithm starts by placing the max player piece in column 0. Then the min player places it's piece in column 0. This back and forth continues until 5 (by default) moves ahead of the current game board are performed at which point the utility of that posibile future board is returned. The alpha or beta value is set (depending on who's turn it was when the desired depth was reached) from the utility, and then the neighboring option (placing all columns as 0 and then the last move being colum 1) is explored. This method of tree exploration via recursion continues until all options of the board have been explored. However, if alpha is ever greater than or equal to beta, positive or negative infinity (depending on if this player is min or max) is returned so as to cut-off or prune the remaining neighbors of that branch. Additionally if a win is detected on the board, the search stops short of the default depth of 5.

If the utility of all actions returned is negative infinity (meaning max cannot win if min continues to play logically) then a random column is selected, hoping the opponent makes a mistake in a future move.