CS310 Practicals 4-6: Dots and Boxes

The aim of this practical is to beat a simple rule-based AI player for the game of Dots and Boxes by using depth-limited minimax search with alpha-beta pruning. This practical is worth 10% of your overall mark.

Instructions for Setting up Dots and Boxes in Eclipse

First you need to set up the dots and boxes code in eclipse. Ask a demonstrator to give you a hand with this if needs be.

1. Download the following file:

https://www.dropbox.com/s/k56rmw772a8k14t/dnb.zip

2. Create a new project in Eclipse and give it a suitable name, e.g. "Dots and Boxes".

3. Right click on the new project and select the 'import' option. From the dialog box, choose 'General -> Archive file' and then click 'next'. Click 'browse' and locate the dnb.zip file that you downloaded earlier, then click 'finish'.

4. Patch: right click on src/controller/GameController.java, select 'Team -> Apply Patch', select 'URL' and put 'http://rodgers.it/GameController.patch' in the field (without the quotes), then click 'Finish'.

5. Right click on src/dnb.DotsAndBoxes.java and select 'Run As -> Java Application'.

The game should now run. Select your players and hit "Go!" to run the game.

Practical 4: A Minimax-Based Player

Your next job is to implement the basic minimax algorithm to decide what move to make next (see Lecture 7, Slides 11 and 12):

http://www.cis.strath.ac.uk/CS310/Lecture7.pdf

Create a new player by locating the package 'players'. Right click on this package and choose 'New -> Class'. Give your new player a name such as 'MyPlayer' and put 'players.AbstractPlayer' in the Superclass field (replacing 'java.lang.Object'), then click 'Finish'.

You should now have a new Class file in front of you with two methods that need implementing. The first is makeMove() - you need to implement the minimax algorithm (without alpha-beta pruning and a depth limit at this stage) to decide which move the player should make to maximise its score. You have access to a clone of the current game state, so take a look at the API for GameState objects.

Here are the methods of the GameState objects that you may find useful:

int getPlayer() - the GameState object includes a field to say whose turn it is next, so you use this method to access this information. Returns 1 for Player 1 and 2 for Player 2.

List<line> getRemainingLines() – get a list of all of the possible remaining moves (i.e. lines joining two dots) in the game.

int addLine (Line line) – add a line into a game state. When you are running the minimax algorithm, make sure the game state you're adding the line to is a clone. The method returns an integer giving the number of boxes completed (0, 1 or 2) or -1 if the move is illegal.

GameState clone() – in your minimax algorithm, you will be needing a copy of the current game state to play with. This copy must not reference the original. This method returns a true deep copy of the game state.

List<GameState> expand() – this method returns a list of all game states reachable from the current state in a single move. If the game state is a terminal state then this list will be empty.

int getValue() – this method returns the current value of the game from the point of view of Player 1. For example, a value of -2 means that Player 2 is ahead by 2 boxes.

The second method you need to implement just returns a string to identify your player in the main menu; best to keep it short (something like "MiniMax Player" is fine).

To see if your method is working correctly, choose the game position Test 1 from the main menu. Your player should take the available box and then choose what is known as the "double cross" move.

This practical is worth 3% of your overall mark. When you have finished this practical, show your solution to a demonstrator.

Practical 5: alpha-beta pruning

Try running Test 2 using your minimax player. You should find that it cannot return a move in a reasonable amount of time because there are too many nodes to search. Now add alpha-beta pruning to your minimax algorithm (see Lecture 8, slides 14-19) and test your algorithm using Test Case 2. It should be able to solve it in a reasonable amount of time now. This practical is worth 3% of your overall mark.

http://www.cis.strath.ac.uk/~johnl/CS310/Lecture8.pdf

Practical 6: beating Easy AI

Implement depth-limited minimax search (see Lecture 9, slides 7-8) with an evaluation function (Lecture 8, slide 9). Test your player by playing it against Easy AI - your aim is to beat it. This is worth 4%: 2% for depth-limited search, 2% for creating an evaluation function that can beat Easy AI.

http://www.cis.strath.ac.uk/~johnl/CS310/Lecture9.pdf

Note: even if you didn't manage to get alpha-beta pruning (Practical 5) working, you can still have a go at this practical and get credit for all or part of it.

John Levine and Phil Rodgers
18th March 2013