Machine Problem 2 of CS 165A (Winter 2019)

University of California, Santa Barbara

Assigned on February 7, 2019 (Thursday)
Due at 11:59 pm on March 14, 2019 (Thursday)

Notes:

- Be sure to read "Policy on Academic Integrity" on the course syllabus.
- Any updates or correction will be posted on the course announcements page and piazza, so check them occasionally.
- You are allowed to discuss about this machine problem with your classmates, and you need to declare all their names in your report. However, you must do your own work (code and report) independently.
- You are required to implement all the details of these tasks by yourself, so you cannot invoke other existing codes or tools. However, some mathematical tools, such as numpy and scipy, are fine.
- Use Python for this assignment.

---

1 Background

In this assignment you will write a computer program to play the game, dots and boxes. `http://en.wikipedia.org/wiki/Dots_and_boxes` Starting with an empty grid of dots, players take turns, adding a single horizontal or vertical line between two unconnected adjacent dots. A player who completes the fourth side of a 1×1 box earns one point whereas a player who completes two grids of size 1×1 simultaneously earns 2 point . The game ends when no more lines can be placed. The winner is the player with higher points.

2 Problem Statement

1. You should implement the minimax search algorithm and alpha-beta pruning on a box grid of size 6×6, as shown in Figure 2.

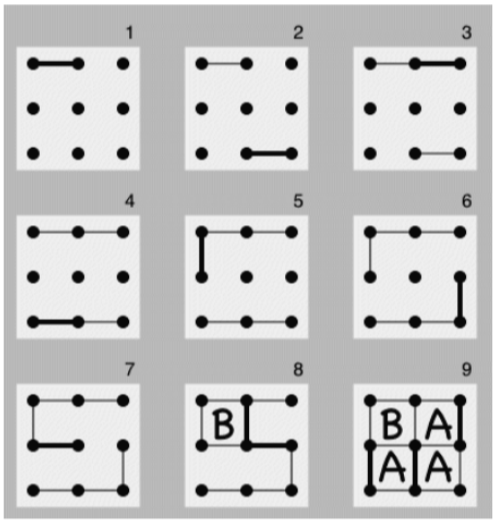

Figure 1: The Dots and Boxes game

2. Since the search tree can be very deep, you should implement the cut-off test strategy and go only till a certain depth. You need to design a evaluation function for a given configuration. (That's the key for your computer player's strength).

3. Try different cut-off depths and set a depth so that each move takes proper time on your computer. Your code will be evaluated both on your final score as well as the time taken for making a move.

3 Task Description

3.1 Prerequisites

You will need to have Python 3 installed in your system. Apart from python you will need the Pygame package installed in your system.

Pygame: Pygame is a python based platform used for developing games. You need to have Pygame installed before setting up the algorithm. Here is a link that will help you in installing Pygames: `https://www.pygame.org/wiki/GettingStarted`.

3.2 Implementing Minimax

Given below are the details that are required for implementing minimax.

State of the game: The entire grid structure is represented as one horizontal matrix

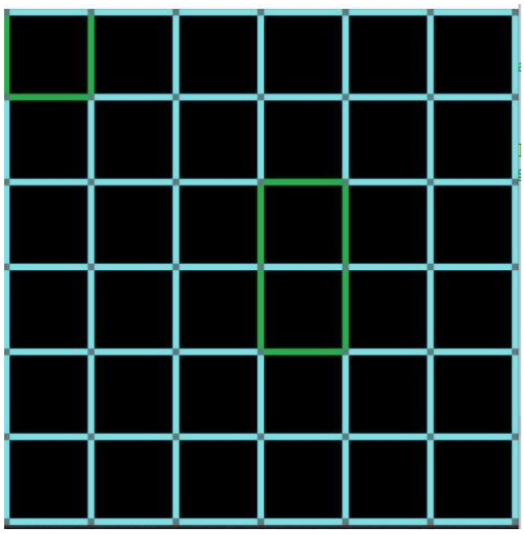

Figure 2: A sample grid comprising of horizontal and vertical edges. The ones displayed in the color green represent the moves that have been made. Competing the box on the upper left corner gives a score of 1 and completing the box in the middle will gives a score of 2.

`h_matrix` and one vertical matrix `v_matrix`.Both the matrices are Boolean in nature and are used for the storing the edges of the grids. The edge is set as True if a move has already been made.The horizontal edge between the ith row and the jth column is stored in the position `h_matrix[i][j]`. Similarly, the vertical edge between the ith row and the jth column is stored at the position `v_matrix[i][j]`.

The state will generally be used for determining the current state of the matrix as well as predicting the future states.

Move: Each move is a vector with three entries, while the first two entries represent the coordinates or the row and column location , the third entry represents whether the move is horizontal or vertical. 0 is used for denoting the vertical edges while 1 is used for denoting the horizontal edges.

Examples

$[0, 2, 0]$ - The vertical edge between the first row and the third column. (Remember, that things in computer science start from 0.)

$[3, 4, 1]$ - The horizontal edge between the fourth row and the fifth column.

Scoring: For completing one block each player will be given one points whereas a simultaneous completion will result in two points.

3.3 Helper functions

Here is a list of functions that can be useful while implementing the minimax algorithm.

1. `current_state()`: The function returns the current state of the system in the form of `h_matrix` and `v_matrix`.
2. `game_ends(h_matrix,v_matrix)`: Returns True if the game is over and False otherwise.
3. `next_state(move,h_matrix,v_matrix)`: Returns the future states and corresponding score given the present state and the move.
4. `increment_score(move,h_matrix,v_matrix)`: Given a move and the state of the system, the function returns the increment in the score because of the respective move. The increment can be 0, 1, or 2.
5. `make_move(self,move,player_id)`: The function is used for making a move. Given that there are two players, the id of the player will either be 0 or 1. (You can always set the id to 1 for this assignment.)
6. `list_possible_moves(h_matrix ,y_matrix)`: The function lists all the possible moves given the state of the game in terms of horizontal and vertical matrices.

3.4 Files Details

Here are the files that you will working on.

1. `Practice.py`: The file is to allow you to understand the basic working of the game and try the different functions that you will be using for implementing minimax and alpha beta pruning. Read the comments on top of file and uncomment some of the sample moves given right at the bottom.
2. `File1.GUI.py`: This is where you will be writing your final code. You need to make changes to the following functions defined in the file. Your algorithm will play against a simple heuristic defined as player 1.
   (a) minimax(self)): You will implement the minimax algorithm in this function. You can change the number of input parameters of the function and the output of the function must be the optimal move made by the function. **Note:** Please be careful of the depth and set an optimal depth of your minimax algorithm. You may have to implement two functions for calculating the minimum and maximum successor of a given node.

(b) alphabetaprunning(self): You will implement the alpha beta pruning algorithm in this function. You can change the number of input parameters of the function and the output of the function must be the optimal move made by the function. **Please report and print the value of alpha and beta.**

(c) evaluate(self): Come up with your own evaluation strategy for minimax. Try to make the best guess of the winner at a reasonable depth given the state of the system.

(d) player(self): Call the minimax and alpha beta pruning in this function to compete with the other heuristic. The function should return the optimal move. **Note: Please dont make any changes to any other functions apart from this function as we will be running these functions from your code onto our own files.**

3. `sampleboxesandgrids.py`: Implementation of the non GUI version of the game. You will use this if there is some problem with installing Pygame. You also need to implement all the functions mentioned.

4 Submission Guidance

Once you have finished developing your code, put your *boxes and grids* folder and your report into a directory named `mp2`. Use the CSIL `turnin` command to submit. Note that all directory and file names are case sensitive. For this assignment you need to use the following command:

```
% turnin mp2@cs165a mp2
```

Once it's successfully turned in, you will see a confirmation message on screen like:

```
*** Turnin OF mp2 TO cs165a COMPLETE! ***
```

Otherwise, please check your network or other possible issues.

5 Grading Policy

We will evaluate your submission according to both implementation and report. If plagiarism is identified, no scores will be given to this assignment and your department will be notified.

For implementation, we are interested in its correctness, performance, and code style.

For report, we are interested in your evaluation strategy of the minimax, your alpha beta pruning parameter, time taken by your algorithm to make a move.

The total points of this project is 100 points, which are distributed as:

1. Correctness of your minimax algorithm (25 points)
2. Correctness of your alpha-beta pruning algorithm (25 points)
3. Your algorithm is able to beat the Player 1. (25 points)
4. Report (25 points)
5. Your agent beats the more advanced agent written by TA (bonus 5 points)

Important: You have a constraint on run time. Your individual move must take less than **1 minute CPU time on the CSIL server**. A run time greater than 1 minute will result in a penalty of 1 point per second, until the points are deducted to 0 for this MP2.

6 Tournament

We will hold the Boxes and Grids knock-out tournament after the due time. Participation in the Tournaments can gain additional credit (and fame!).

Match setup: A single match between two players consists of 2 games. Each player starts first once. After two games, the one with higher total score wins. If there is a tie, the faster player wins.

Please visit the course website for further tournament information updates.

Have fun!