2nd Lt David Crow

ENG/20M

Assignment #2 – Search and Game Tree Search

Artificial Intelligence - CSCE 523

Turnin: E-mail me a zip file containing your typed solution to questions 1 and 2, and your program and report for question 3.

1. (30 points) For the following maze, show the lists of open and visited nodes (with their associated costs) for each cycle of the listed search algorithms. The start cell is , and the goal cell is . The agent can move north, south, east, and west. The agent expends 1 point moving south or west, and 2 points moving north or east.
   1. Decide on a heuristic estimator function and write the function out.

A simple heuristic estimator function is one in which the estimated cost of the path to the goal from a given cell is the Manhattan distance from that cell to the goal. However, we can limit the penalty for moving east by giving a greater estimate to those cells that are in a column to the left of the goal’s column. Effectively, we’ll double the effect of the x-displacement in the estimate function for cells that are left of the goal column.

Formally,

// Assume is of type

// Assume we can access a ’s coordinates with and

Heuristic()

Let

If

Return

* 1. Decide on a method to break ties (label all cells with letters and break alphabetically, etc.) and write that out.

I’ve labeled the cells with letters from the alphabet. We’ll break ties alphabetically. We could arrange the letters in a way that leads to the goal, but then it’s less of a *search* and more of an *I-know-how-to-get-to-the-goal*. However, because we are “arbitrarily” assigning letters to these nodes, we can say that, luckily, the lettering started in the bottom right corner. I’m doing this to (potentially – I don’t actually know that it helps very much) make my searches in parts (c) and (d) a little shorter.

Then perform the following searches on the space (follow the format from class):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

* 1. Beam search with a beam size of 2

|  |  |
| --- | --- |
| Frontier | Explored |
| S | - |
| wx | S |
| qv | Swx |
| u | Swxqrvy |
| pt | Swxqrvyu |
| lo | Swxqrvyupt |
| hk | Swxqrvyuptlo |
| bg | Swxqrvyuptlohk |
| ca | Swxqrvyuptlohkbg |
| d | Swxqrvyuptlohkbgca |
| G | Swxqrvyuptlohkbgcad |

Goal found!

* 1. IDA\* search

Threshold:

|  |  |
| --- | --- |
| Frontier | Explored |
| S5 | - |
| w5x8 | S |
| q5v8x8 | Sw |
| v8x8 | Swq |

Threshold: 8

|  |  |
| --- | --- |
| Frontier | Explored |
| S5 | - |
| w5x8 | S |
| q5v8x8 | Sw |
| v8x8 | Swq |
| x8u11 | Swqv |
| r8u11y11 | Swqvx |
| m8s11u11y11 | Swqvxr |
| i8n11s11u11y11 | Swqvxrm |
| e10j11n11s11u11y11 | Swqvxrmi |

Threshold: 10

|  |  |
| --- | --- |
| Frontier | Explored |
| S5 | - |
| w5x8 | S |
| q5v8x8 | Sw |
| v8x8 | Swq |
| x8u11 | Swqv |
| r8u11y11 | Swqvx |
| m8s11u11y11 | Swqvxr |
| i8n11s11u11y11 | Swqvxrm |
| e10j11n11s11u11y11 | Swqvxrmi |
| j11n11s11u11y11f13 | Swqvxrmie |

Threshold: 11

|  |  |
| --- | --- |
| Frontier | Explored |
| S5 | - |
| w5x8 | S |
| q5v8x8 | Sw |
| v8x8 | Swq |
| x8u11 | Swqv |
| r8u11y11 | Swqvx |
| m8s11u11y11 | Swqvxr |
| i8n11s11u11y11 | Swqvxrm |
| e10j11n11s11u11y11 | Swqvxrmi |
| j11n11s11u11y11f13 | Swqvxrmie |
| n11s11u11y11f13 | Swqvxrmiej |
| s11u11y11f13 | Swqvxrmiejn |
| u11y11f13 | Swqvxrmiejns |
| p11y11f13t14 | Swqvxrmiejnsu |
| l11y11f13o­14t14 | Swqvxrmiejnsup |
| h11y11f13k14o­14t14 | Swqvxrmiejnsupl |
| y11b13f13g14k14o14t14 | Swqvxrmiejnsuplh |
| b13f13g14k14o14t14 | Swqvxrmiejnsuplhy |

Threshold: 13

|  |  |
| --- | --- |
| Frontier | Explored |
| S5 | - |
| w5x8 | S |
| q5v8x8 | Sw |
| v8x8 | Swq |
| x8u11 | Swqv |
| r8u11y11 | Swqvx |
| m8s11u11y11 | Swqvxr |
| i8n11s11u11y11 | Swqvxrm |
| e10j11n11s11u11y11 | Swqvxrmi |
| j11n11s11u11y11f13 | Swqvxrmie |
| n11s11u11y11f13 | Swqvxrmiej |
| s11u11y11f13 | Swqvxrmiejn |
| u11y11f13 | Swqvxrmiejns |
| p11y11f13t14 | Swqvxrmiejnsu |
| l11y11f13o­14t14 | Swqvxrmiejnsup |
| h11y11f13k14o­14t14 | Swqvxrmiejnsupl |
| y11b13f13g14k14o14t14 | Swqvxrmiejnsuplh |
| b13f13g14k14o14t14 | Swqvxrmiejnsuplhy |
| c13f13g14k14o14t14a16 | Swqvxrmiejnsuplhyb |
| d13f13g14k14o14t14a16 | Swqvxrmiejnsuplhybc |
| f13G13g14k14o14t14a16 | Swqvxrmiejnsuplhybcd |
| G13g14k14o14t14a16 | Swqvxrmiejnsuplhybcdf |
| g14k14o14t14a16 | SwqvxrmiejnsuplhybcdfG |

Goal found!

1. (20 points) For the following Light-Up puzzle, show the sequence of variable assignments during backtracking with forward checking; examine cells in alphabetical order. Show assignments by writing the forward-checking table process (14 columns: step number, value, value, …, value, backtrack (list the constraint violation that causes the backtrack)).

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 1 | 2 |  |
|  |  | 0 |  |
|  |  |  |  |

For each variable at each time step, “l” denotes that “light” is an option, “n” denotes that “no light” is an option, “***L***” denotes that “light” is the selected value, and “***N***” denotes that “no light” is the selected value.

When a cell has no eligible assignments remaining, I will note the constraint violation in the “Backtrack” column. Additionally, I will note constraint violations when any cells are not illuminated. Finally, I will note constraint violations when the correct number of lightbulbs are not placed/cannot be placed next to the 0, 1, or 2 blocks.

When I say the correct number of lightbulbs “cannot be placed,” I mean to say that forward-checking here can identify a violated n-block constraint once an essential “light” assignment is removed as an option (e.g., if “light” is not an option for *c*, then forward-checking will identify that the 2-block constraint will be violated).

We can implement this by adding these extra constraints: both *c* and *f* must be “light,” either *b* or *e* must be “light,” and neither *h* nor *k* can be “light.”

Because most of the cells will necessarily be assigned “no light,” I will always try “no light” before “light” when both options are available for a given variable.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Step | a | b | c | d | e | f | g | h | i | j | k | l | Backtrack |
| Initial | nl | nl | l | nl | nl | l | nl | n | nl | nl | n | nl |  |
| a: N | ***N*** | nl | l | nl | nl | l | nl | n | nl | nl | n | nl |  |
| b: N | ***N*** | ***N*** | l | nl | l | l | nl | n | nl | nl | n | nl |  |
| c: L | ***N*** | ***N*** | ***L*** | n | l | l | nl | n | nl | nl | n | nl |  |
| d: N | ***N*** | ***N*** | ***L*** | ***N*** | l | l | nl | n | nl | nl | n | nl |  |
| e: L | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | l | n | n | n | nl | n | nl |  |
| f: L | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | n | n | n | nl | n | n |  |
| g: N | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | n | n | nl | n | n |  |
| h: N | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | n | nl | n | n |  |
| i: N | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | ***N*** | nl | n | n |  |
| j: N | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | ***N*** | ***N*** | n | n |  |
| k: N | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | ***N*** | ***N*** | ***N*** | n |  |
| l: N | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | ***N*** | ***N*** | ***N*** | ***N*** | (j, k) illumination |
| l: | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | ***N*** | ***N*** | ***N*** |  | no options for l |
| k: | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | ***N*** | ***N*** |  | n | no options for k |
| j: L | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | ***N*** | ***L*** | n | n |  |
| k: N | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | ***N*** | ***L*** | ***N*** | n |  |
| l: N | ***N*** | ***N*** | ***L*** | ***N*** | ***L*** | ***L*** | ***N*** | ***N*** | ***N*** | ***L*** | ***N*** | ***N*** |  |

1. (50 points) Below are the rules for the game Lines of Action. For this part of the assignment, you are to write a board evaluator and an alpha-beta minimax search for the game. I have provided Java files to maintain the board state, to test the legality of a move, and to generate a list of possible moves as a Vector for one or all of the pieces in the class directory. You are responsible for completing LOABoard.heuristicEvaluation(), and writing a MinimaxAlphaBetaSearch class. Do not feel constrained by my code – if you feel that you need additional elements or different functionality, feel free to change it, but be sure to document the changes.

Notes:

* In the LOABoard class, set the BOARD\_SIZE to be 4 or 5 for easier debugging (mnkGame, i.e. TicTacToe is also included).
* In the LOACustomPanel class, set the SELF\_PLAY to true if you want to play against yourself to get a feel for the rules.

Lines of Action Rules:

1. The black pieces are placed in two rows along the top and bottom of the board, while the white pieces are placed in two files at the left and right side of the board (Figure 1).
2. The players alternately move, starting with Black.
3. A player to move must move one of its pieces. A move takes place in a straight line (up, down, left, right, and all four diagonals), exactly as many squares as there are pieces of either color anywhere along the line of movement (These are the Lines of Action).
4. A player may jump over its own pieces.
5. A player may not jump over the opponent’s pieces but can capture them by landing on them.
6. To win a player must move all their pieces on the board into one connected unit. The first player to do so is the winner. The connections within the group may be either orthogonal or diagonal. For example, in Figure 2 Black has won because the black pieces form one connected unit.



Figure 1: Starting board

1. If one player’s pieces are reduced by captures to a single piece, the game is a win for this player.
2. If a move simultaneously creates a single connected unit for both players, the player that moved wins.

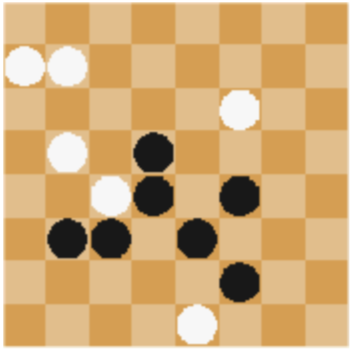


Figure 2: Black wins

Additional Resources:

* There are several articles on Lines of Action in the course directory under the subdirectory Articles. Some other links on the game:
  + [Lines of Action home page](http://boardspace.net/loa/)
  + [U of A GAMES Group Home Page (YL and MONA)](http://webdocs.cs.ualberta.ca/~darse/LOA/)
  + [Mark Winands (MIA)](https://dke.maastrichtuniversity.nl/m.winands/loa/)

Turn-Ins:

* Turn in should include your code, instructions for compilation, and a write-up describing your implementation, heuristic details, and experiences.

Stipulations:

* You are responsible for having your search halt at the set threshold depth, pruning the correct amount of space. For the threshold depth, note that one step is one player’s move – whether it is your move or your opponent’s – and not your program’s move and your opponent’s countermove.