Mark Demore II

CSCE523

Assignment #2

1. A) Heuristic Estimator: Manhattan distance, weighted with cost of move direction

*int heuristicEstimator(int moveCount)*

*dx = pos.x – goal.x*

*dy = pos.y – goal.y*

*heur = moveCount*

*if(dx < 0)*

*heur += abs(dx)*

*else*

*heur += 2\*dx*

*if(dy < 0)*

*heur += abs(dy)*

*else*

*heur += 2\*dy*

*return heur*

B) Tie Breaker: favor move to South/West

*boolean tieBreaker(Move A, Move B)*

*int dx = A.x – B.x*

*int dy = A.y – B.y*

*if(dx > dy)*

*if(dx > 0)*

*return B*

*else*

*return A*

*else*

*if(dy < 0)*

*return B*

*else*

*return A*

C) Beam Search, Width 2

**OPEN: VISITED:**

F2

F3, F1 F2

F4, G3 F2, F3

E5, G5 F2, F3, F4

D5, G5 F2, F3, F4, E5

C5, D4 F2, F3, F4, E5, D5

B5, D5 F2, F3, F4, E5, D5, C5

B4, C5 F2, F3, F4, E5, D5, C5, B5

Goal, B5 F2, F3, F4, E5, D5, C5, B5, B4

B2, B4 F2, F3, F4, E5, D5, C5, B5, B4, **GOAL**

D) IDA\*, Threshold = minCost of children

**OPEN: VISITED:**

F2

F3, G2, F1 F2

F4, G3, F2 F2, F3

F5, G4, F3 F2, F3, F4

E5, F4, G5 F2, F3, F4, F5

D5, F5 F2, F3, F4, F5, E5

C5, D4, E5 F2, F3, F4, F5, E5, D5

B5, D5 F2, F3, F4, F5, E5, D5, C5

B4, C5, A5 F2, F3, F4, F5, E5, D5, C5, B5

Goal, B5, A4 F2, F3, F4, F5, E5, D5, C5, B5, B4

B2, B4, A3 F2, F3, F4, F5, E5, D5, C5, B5, B4, **GOAL**

1. Light-Up Puzzle:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Step Number** | ***a*** | ***b*** | ***c*** | ***d*** | ***e*** | ***f*** | ***g*** | ***h*** | ***i*** | ***j*** | ***k*** | ***l*** | **Backtrack** |
| Start |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | √ | x | x | x | x |  | x |  | x |  |  |  | Either *b* or *e* must have a light bulb |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | x | √ | x | x |  |  |  |  |  |  |  |  | Both *c* and *f* must have light bulbs |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | x | x | √ | x |  |  |  |  |  |  |  |  |  |
| 6 | x | x | √ | x | √ |  | x |  | x |  |  |  |  |
| 7 | x | x | √ | x | √ | √ | x | x | x |  |  | x |  |
| 8 | x | x | √ | x | √ | √ | x | x | x | √ | x | x |  |

1. Compile Instructions: Open the IntelliJ project and click run. (I believe the only files I altered were LOABoard and MinimaxSearch).

Description:

1. Implementation: I implemented my alpha-beta minimax using a recursive helper function. The helper is fed each child of the current board state and evaluates subsequent states using depth, a Boolean determining the maximizing player, and the alpha and beta values to prune different subtrees.
2. Heuristic: My heuristic makes use of the Euler number and Quad counts that are calculated elsewhere in the program. Based on Winand’s evaluation function, the heuristic favors a smaller Euler number and Quads of 3 or 4 pieces, with a smaller preference for Quads of 2 pieces. The Euler heuristic takes thee Euler number and scales it using tanh, it is then subtracted from one. The Quad heuristic increases by 2 with every quad of 3 or 4 pieces, and by 1 with every quad of 2 pieces, it is then scaled with tanh. The final heuristic value returned for each board is the tanh of the sum of the Euler and Quad heuristics. It returns a -1 if the board is a loss, and a 1 if the board is a win, in place of the heuristic.
3. Experiences: The heuristic is not terribly good at winning, but not totally ignorant. I attempted scaling the Euler and Quad heuristics in relation to each other by doubling them before adding for the final heuristic calculation, but there was no observable performance increase. I also attempted subtracting the opponents corresponding heuristic as part of the calculation, but this incurred a substantial time cost at depths greater than 3.