File for the algorithms

Luke Garrigan

registration: 100086495

1 Introduction

Algorithm 1 Manhattan Distance

```
1:  procedure MANDIST(state)                          ▷ The current puzzle configuration
2:      total ← 0
3:      puzzleLength ← state.size()
4:      dimensions ← √puzzleLength
5:      for i ← 1, puzzleLength do                    ▷ Loops through each tile of the puzzle
6:          tileValue ← state[i]
7:          expectedRow ← (tileValue − 1) ÷ dimensions
8:          expectedCol ← (tileValue − 1) mod dimensions
9:          rowNum ← i ÷ dimensions
10:         rowNum ← i mod dimensions
11:         total ← total+ | expectedRow − rowNum | + | expectedCol − colNum |
12:     return total                                  ▷ The heuristic is the total
```

Algorithm 2 Iterative Deepening A Star

```
1:  state                                          ▷ The current puzzle configuration
2:  g(state)                                       ▷ The cost to reach the current state
3:  h(state)                  ▷ Estimated cost of the cheapest path from state to goal
4:  f(state) ← g(state) + h(state)
5:  neighbours(state)   ▷ Expands possible moves from current state ordered by g + h

6:  procedure IDASTAR(state)
7:      bound ← f(state)
8:      while not solved do                        ▷ Loops until a solution is found
9:          bound ← DFS(state, bound)              ▷ Performs a bounded depth-first search

10: procedure DFS(state, bound)
11:     if f(state) > bound then
12:         return f(state)
13:     if h(state) == 0 then          ▷ No more moves needed to reach goal state
14:         return solved
15:     min ← ∞
16:     for neighbour in neighbours(state) do
17:         temp ← DFS(neighbour, bound)
18:         if temp < min then
19:             min ← temp
20:     return min                          ▷ Returns the smallest of the neighbours
```

Algorithm 3 Breadth-First Search

```
1:  procedure BFS(state)
2:      s ← empty set
3:      q ← empty queue
4:      q.add(state)
5:      q.enqueue(state)
6:      while q.size() > 0 do
7:          currentState ← q.dequeue()
8:          if currentState == goal then
9:              return current
10:         for neighbour in neighbours(currentState) do
11:             if neighbour is not in s then
12:                 s.add(neighbour)
13:                 q.enqueue(neighbour)
```

Algorithm 4 Is Current State Solvable

```
1:  procedure IsSolvable(state)
2:      stateLength ← state.size()
3:      gridWidth ← √stateLength
4:      row ← 0                                   ▷ The current row we are on
5:      blankRow ← 0                              ▷ The row with the blank tile
6:      for i ← 1, puzzleLength do
7:          if i mod gridWidth == 0 then
8:              row++
9:          if state[i] == 0 then
10:             blankRow ← row
11:             continue
12:         for j ← i+1, puzzleLength do
13:             if state[i] > state[j] and state[i] ≠ 0 then
14:                 parity++
15:     if gridWidth mod 2 == 0 then
16:         if blankRow mod 2 == 0 then
17:             return parity mod 2 == 0
18:         else
19:             return parity mod 2 ≠ 0
20:     else
21:         return parity mod 2 == 0
```

Progress report

Description of project: aims, motivation	First	2.1	2.2	3	Fail
Description and understanding of issues and problems addressed in the project	First	2.1	2.2	3	Fail
Achievement so far according to what is reasonably expected for the type of project	First	2.1	2.2	3	Fail
Discussion and justification of changes to project aims, scope, workplan	First	2.1	2.2	3	Fail

Quality of writing

Clarity, structure correctness of writing	First	2.1	2.2	3	Fail

Comments

Supervisor: supervisor

Markers should circle the appropriate level of performance in each section. Report and evaluation sheet should be collected by the student from the supervisor.