File for the algorithms

Luke Garrigan

registration: 100086495

1 Introduction

Algorithm 1 Manhattan Distance

procedure MANDIST($state$) ▷ The current puzzle configuration
$total \leftarrow 0$
$puzzleLength \leftarrow state.size()$
$dimensions \leftarrow \sqrt{puzzleLength}$
for $i \leftarrow 1, puzzleLength$ **do** ▷ Loops through each tile of the puzzle
$tileValue \leftarrow state[i]$
$expectedRow \leftarrow (tileValue - 1) \div dimensions$
$expectedCol \leftarrow (tileValue - 1) \mod dimensions$
$rowNum \leftarrow i \div dimensions$
$rowNum \leftarrow i \bmod dimensions$
$total \leftarrow total + \mid expectedRow - rowNum \mid + \mid expectedCol - colNum \mid$
return $total$ ▷ The heuristic is the total

Algorithm 2 Iterative Deepening A Star

```
state                        ▷ The current puzzle configuration
g(state)                     ▷ The cost to reach the current state
h(state)                     ▷ Estimated cost of the cheapest path from state to goal
neighbours(state)            ▷ Expands possible moves from current state ordered by g + h

procedure IDASTAR(state)
    bound ← h(state)
    solution ← null
    while solution == null do                  ▷ Loops until a solution is found
        solution ← dfs(state, bound)           ▷ Performs a bounded depth-first search
        bound ← bound+1
    return solution

procedure DFS(state, bound)
    if state == goal then
        return state                           ▷ Found the goal state
    for neighbour in neighbours(state) do
        f ← g(neighbour) + h(neighbour)        ▷ Estimated cost of the cheapest path
        if f ≤ bound then
            result ← dfs(next, f)
            if result ≠ null then
                return result
```

Algorithm 3 Breadth-First Search

```
procedure BFS(state)
    s ← empty set
    q ← empty queue
    q.add(state)
    q.enqueue(state)
    while q.size() > 0 do
        currentState ← q.dequeue()
        if currentState == goal then
            return current
        for neighbour in neighbours(currentState) do
            if neighbour is not in s then
                s.add(neighbour)
                q.enqueue(neighbour)
```

Algorithm 4 Is Current State Solvable

```
procedure IsSolvable(state)
    stateLength ← state.size()
    gridWidth ← √stateLength
    row ← 0                                        ▷ The current row we are on
    blankRow ← 0                                   ▷ The row with the blank tile
    for i ← 1, puzzleLength do
        if i mod gridWidth == 0 then
            row++
        if state[i] == 0 then
            blankRow ← row
            continue
        for j ← i+1, puzzleLength do
            if state[i] > state[j] and state[i] ≠ 0 then
                parity++
    if gridWidth mod 2 == 0 then
        if blankRow mod 2 == 0 then
            return parity mod 2 == 0
        else
            return parity mod 2 ≠ 0
    else
        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.