4. Program DFS and BFS solution for the farmer problem is based off the solution in sections 13 and 14 of the textbook “AI Programming Help”.

The state of the problem is set up by the functions from line 0 to 67, these functions produce the moves, the representation of the problem state and checking whether a current state is legal or not.

The functions that make up the general basis of BFS and DFS are defined by the functions from line 69 to 99. They are used to represent the steps necessary to perform either of these search algorithms in a general sense, building a tree like record the possible states and their descendants.

The 2 functions that make up BFS are defined from 104 to 121, breadth-first and run-breadth. BFS is run by traversing all sets of states in order of adjacency until the desired end state is found, constructing a path of the possible transitions to this state along the way. This is accomplished by maintaining \*open\*as First in First Out queue, as states are traversed, they are added to \*open\*. Run-breadth simply takes the start state and the goal state and sets up the \*open\*, \*closed\* and \*goal\* values to run the algorithm.

Similarily, the 2 functions that make up DFS are defined from 123 to 141. They are identical, except for DFS is implemented by using a stack for \*open\* rather than a FIFO queue. This is simply done by changing the order of the append values on lines 132 – 134 for DFS and lines 113-114 for BFS.

Sample Output:

1. Function Calls: (print (reverse (run-breadth (make-state 'e 'e 'e 'e) (make-state 'w 'w 'w 'w))))

(print (reverse (run-depth (make-state 'e 'e 'e 'e) (make-state 'w 'w 'w 'w))))

Output:

((E E E E) (W E W E) (E E W E) (W W W E) (E W E E) (W W E W) (E W E W)

(W W W W))

((E E E E) (W E W E) (E E W E) (W W W E) (E W E E) (W W E W) (E W E W)

(W W W W))

2. Function Calls: (print (reverse (run-breadth (make-state 'e 'w 'e 'e) (make-state 'w 'w 'w 'w))))

(print (reverse (run-depth (make-state 'e 'e 'w 'e) (make-state 'w 'w 'w 'w))))

Output:

((E W E E) (W W E W) (E W E W) (W W W W))

((E E W E) (W W W E) (E W E E) (W W E W) (E W E W) (W W W W))

3. Function Calls: (print (reverse (run-breadth (make-state 'e 'w 'e 'w) (make-state 'w 'w 'w 'w))))

(print (reverse (run-depth (make-state 'e 'w 'e 'e) (make-state 'w 'w 'w 'w))))

Output: ((E W E W) (W W W W))

((E W E E) (W W E W) (E W E W) (W W W W))