**AI Project 1 Report**

**Team 32**

* **Problem Description**

The agent Jon Snow is required to kill all white walkers on a grid that is randomly generated. The agent uses dragon glasses to kill white walkers where it can kill only white walkers standing in adjacent cells to Jon’s current position. At every step, Jon can kill all neighboring white walkers using one dragon glass, yet if he moves to a different position he must use another dragon glass. A dragon stone is the place on the grid where Jon gets the dragon glasses, where Jon can have a specific glass capacity that is randomly generated upon starting the game. The agent supposed to find the solution to the problem using the implemented search algorithms. The optimal solution is to kill all white walkers with the least number of glasses.

* **Implementation of the search-tree node ADT**

A search tree node is implemented such that the search problem ADT can create new states and add them to the nodes, then enqueue these nodes according to any of the search algorithms implemented. A tree node has an instance of state that represents the current state, the depth of the node, the parent, and the path cost from the root to the current node.

* **Implementation of the search problem ADT**

The general search problem ADT implements a general search function that can solve any search problem. A search problem inherits from the general search problem the signatures of abstract functions that are in turn implemented based on the problem description. The specific search problem along with the required search strategy are both passed to the general search function. The function creates a queue and constructs the initial tree node. After nodes are expanded, enqueuing technique is determined by the strategy parameter. The goal test is applied on each node. If the node is a goal node, then it is returned. If the queue is empty, the function will return failure (null).

* **Implementation of the “SaveWesteros” problem**

SaveWesteros class is a child of the search problem ADT. It inherits and implements all the abstract functions from the parent class. These functions are the eight different search strategies, the expand function, the goal test function, and the path cost function. The savewesteros problem calls the general search function to solve the problem.

* **Main functions implemented**

1. Each strategy function enqueues the nodes in a different manner as explained in class. The BF function enqueues at the end while DF enqueues at the front. ID enqueues in a BF manner with a cutoff value that is iteratively increased until a solution is found. UC enqueues in order of increasing costs. Greedy function enqueues in order of increasing heuristic value, while the A\* function enqueues in order of increasing sum of path cost and heuristic value.
2. The expand function expands each node by applying all operators on the current state. If a state is not valid (ie. Jon moves outside the grid) the child node will be null, otherwise the child node will be constructed with its new state and parameters and then added to the array of nodes that is returned by the function.
3. The goal test function checks on the goal Boolean attribute in the current state.
4. The path cost function sums the path cost at the parent node wit the path of the current operator that is expanded.
5. The search problem passes the grid and the required save strategy to the general search problem. If the general search problem returns a node, a solution is constructed using helper function GenerateSolution(ArrayList<Snode>, Snode). This function also takes a boolean parameter (visualize) which when set to true, the solution is printed in the console as a sequence of grids according to the sequence of states in the constructed solution.
6. The print Solution function prints the sequence of states from the initial state to the goal state.

* **Various search algorithms**

Each of the different search algorithms required in the project description is implemented in a separate function that enqueues the new nodes at each level in the tree. The enqueuing techniques are implemented as discussed in the previous section, and are determined according to the strategy passed to the search problem in savewesteros class.

* **Heuristic functions**

Admissibility Proof:

**Assume N is the total number of white walkers initially in the grid. Then:**

&

,

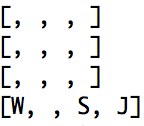
=

**So** and the constant will not affect.

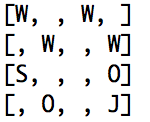
1. The second heuristic function calculates the distance form the agent to the closest white walker. This function can be proved admissible since the agent will need to reach the nearest walker on the grid and perform at least one more operation which is Stab (Kill the white walker).

* **Three running examples**

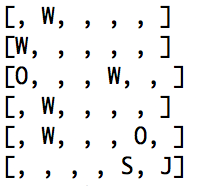
1. Trivial 4x4 grid with a single white walker



1. More complex 4x4 grid with 4 white walkers and 2 obstacles



1. Grid of size 6x6 with 5 white walkers and 2 obstacles (based on a random generated grid).



* **Performance Comparison**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Grid |  | BF | DF | ID | UC | AS1 | GR1 | AS2 | GR2 |
| 1 | Completeness | yes | yes | yes | yes | yes | yes | yes | yes |
| Optimality | 4 steps / 1 glass | 35 steps / 1 glass | 4 steps / 1 glass | 4 steps / 1 glass | 4 steps / 1 glass | 4 steps / 1 glass | 4 steps / 1 glass | 4 steps / 1 glass |
| expanded nodes | 16 | 38 | 43 | 34 | 34 | 12 | 14 | 4 |
| 2 | Completeness | yes | yes | yes | yes | yes | yes | yes | yes |
| Optimality | 20 steps / 2 glasses | 34 steps / 2 glasses | 20 steps / 2 glasses | 20 steps / 2 glasses | 20 steps / 2 glasses | 26 steps / 3 glasses | 20 steps / 2 glasses | 26 steps / 2 glasses |
| expanded nodes | 294 | 41 | 1164 | 545 | 544 | 132 | 301 | 246 |
| 3 | Completeness | yes | yes | yes | yes | yes | yes | yes | yes |
| Optimality | 17 steps / 4 glasses | 125 steps / 5 glasses | 17 steps / 4 glasses | 17 steps / 4 glasses | 17 steps / 4 glasses | 23 steps / 4 glasses | 17 steps / 4 glasses | 48 steps / 4 glasses |
| expanded nodes | 7656 | 135 | 18234 | 64206 | 62866 | 497 | 32591 | 382355 |

In the first two examples, the number of expanded nodes is maximum at the Iterative Deepening since the same nodes get expanded at every iteration until the shallowest solution is reached. Moreover, Uniform Cost and A\* strategies expand far more states than Breadth First does since BF always searches for the shallowest solution not taking optimality into account. We can also see that Depth First expands relatively few states yet in all three examples the solution was far away from being optimal.

Note: All search procedures luckily were able to find solutions to these three examples, yet Depth First and Greedy strategies are not always granted to find solutions for other random grids for their incompleteness.

* **How to run the code**

1. Run the code using eclipse. No user input is required for random grid generation. In case one of the three example grids is desired, find the grid initialization in the main method of saveWesteros and insert the number of the desired grid. You can alternate between the different search strategies through the passed string parameter in calling the search method (also found in the main method at saveWesteros)
2. In the console, you can see the grid that is generated at first. As long as the code is running, you can see the current queue length and the number of expanded nodes. Once a goal state is expanded, the solution will be printed in the console if the visualize parameter is set to true. The solution is represented as a sequence of states. All states are separated using dashed lines. In each state, the current grid state, the operator, and the array of current walker positions are printed. All states from the initial state to the goal are printed in order.