CSEN 901: Introduction to Artificial Intelligence
Projct 1: Water Sort

Project Report

Abdelrahman Aboelkehir, Hossain Ghoraba,
Mahmoud Aboelenein, Omar Hesham

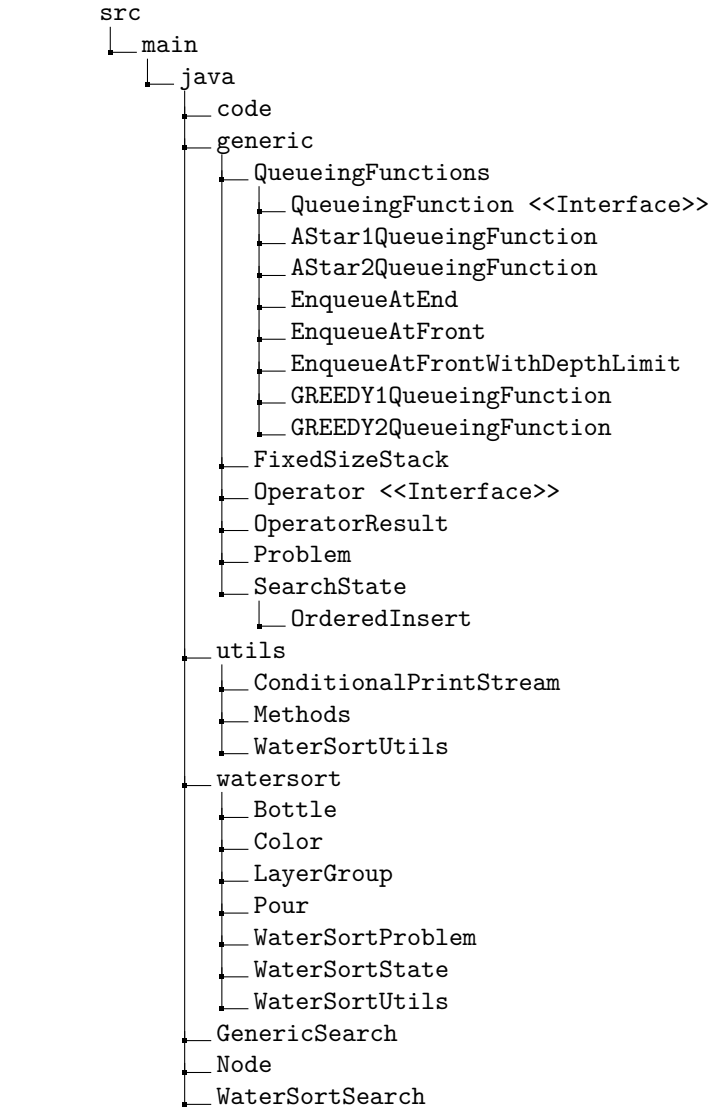
18.10.2024

1 Introduction

This project is the implementation of a search agent to solve the water sort puzzle. You can find a description of the water sort puzzle in the document titled **Project Description.pdf**. The agent is implemented in Java. The search the agent performs does not prune any branches early, nor does it perform any checks for whether the current node can or can not lead to a solution. A branch in the search is terminated only when there are no more possible nodes to expand from the leaf of that branch (in this probelm, this means there are no possible pour combinations to apply to the current state). The search terminates when either **a.** A node selected for expansion is identified as a goal state, or **b.** No more nodes on the search frontier can be expanded.

2 Project Hierarchy

Here is a hierarchy of all the project files:



3 Class Diagram

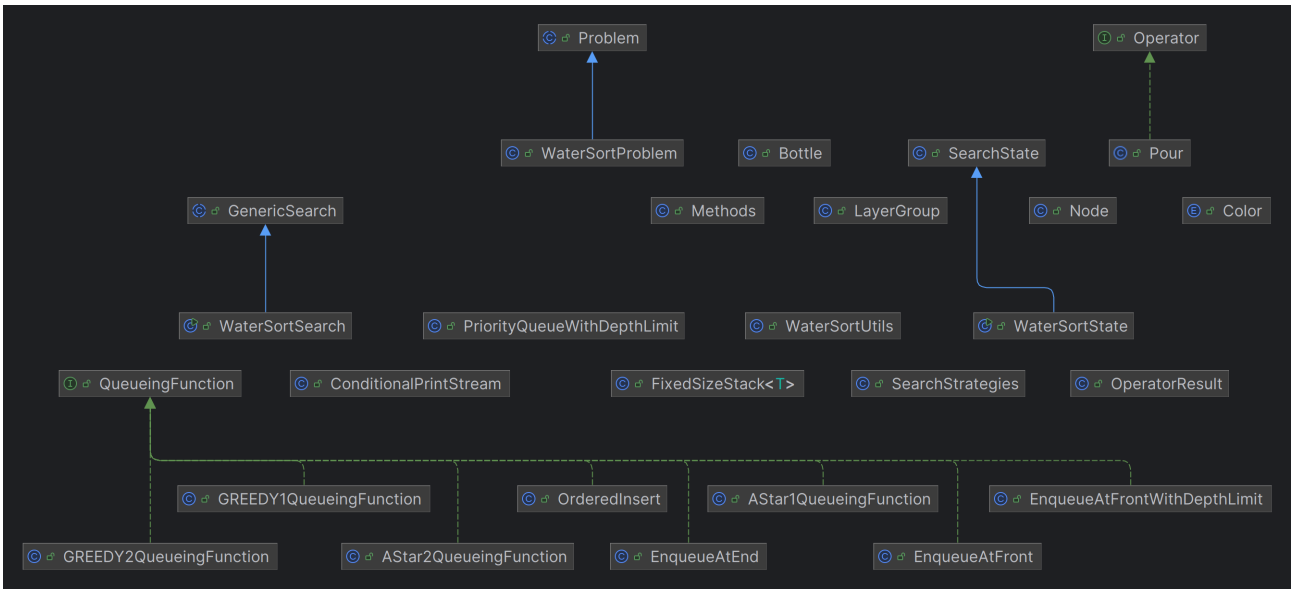


Figure 1: Class Diagram

4 Implementation of the Search Algorithms

Our implementation for all search algorithms closely follows the description of the *GeneralSearch* algorithm outlined in lecture 2 of the course:

Algorithm 1 GENERAL-SEARCH(problem, QING-FUN)

return a solution, or failure
$nodes \leftarrow$ `MAKE_Q`(`MAKE_NODE`(`INIT_STATE`($problem$)))
while true **do**
    if $nodesisempty$ **then**
        return failure
    end if
    $node \leftarrow$ `REMOVE_FRONT`($nodes$)
    if `GOAL_TEST`($problem$)(`STATE`($node$)) **then**
        return node
    end if
    $nodes \leftarrow$ `QING_FUN`($nodes$, `EXPAND`($node$, `OPER`($problem$)))
end while

All the algorithms use a Java `PriorityQueue` as their queue. The `PriorityQueue` in Java orders objects based on a given `Comparator`. The classes in the `QueueingFunctions` package return a `Comparator`, which, according to the search strategy being used, orders the elements in the queue in a certain manner. This `Comparator` is then used to initialize a queue.

For example, here is the queuing function for Breadth-First Search:

```
public class BFSQueuingFunction implements QueuingFunction {
  @Override
  public Comparator apply() {
    return new Comparator.comparingInt(Node::getDepth);
  }
}
```

In this case, when a `PriorityQueue` is created using the returned `Comparator`, all the nodes in the queue will be ordered in ascending order of depth.