* See code files for complete method and class definitions.

```java
// A Node object that stores information corresponding to a gridworld cell.
public class Node {
    int x,y;
    int g;
    double h;
    double f;
    Node tree;
    int search;
...
...
}

// A tracker object that stores information on the current search.
public class SearchTracker {
    Search.Direction direction;
    boolean repeated;
    Node agent;
    int counter;
    boolean finished = false;
    Node path = null;
...
...
}

/**
 * Moves the agent along the presumed path until it encounters a blocked space
 * or the goal. Updates the agent's view of the gridworld (agentWorld) if it
 * encounters a blocked space. Updates the agent's position to the space
 * before the block on the presumed path if a blocked space is found, or to
 * the goal if no blocked spaces are found.
 * @param presumedPath
 * @param agentWorld The gridworld the agent sees.
 * @param gridworld The fully visible gridworld.
 */
public static Node moveAgent(SearchTracker tracker, Node presumedPath, int[][] agentWorld,
                                 int[][] gridworld) {...}
/**
 * Returns a tree path in reverse order. If the path tree is null, the aStarSearch algorithm
 * failed and this method will return the path as is.
 * @return The reversed path
 */
public static Node reversePath(Node path) {...}

/**
 * @param gridworld The underlying gridworld the agent must travel through.
 * @param direction Whether to perform forwards or backwards Repeated A* search.
 * @param heuristic The heuristic formula to use.
 * @return The Node with the tree path the agent can travel to the goal. If the
 * goal is impossible to reach, a Node with a null tree pointer*/
public static Node repeatedAStarSearch(int[][] gridworld, Direction direction,
Type heuristic) {
    // Create a duplicate gridworld object with no information on blocked spaces.
    // Update this object as the agent discovers blocked spaces.
    int dim = gridworld.length-1;
    int[][] agentWorld = new int[gridworld.length][gridworld.length];

    // Initialize the closed list, open list, and NodeComparator objects.
    // The NodeComparator tells the PriorityQueue how to compare Nodes.
    Comparator<Node> nodeComparator = new NodeComparator();
```

* See code files for complete method and class definitions.

```
    PriorityQueue<Node> openList = new PriorityQueue<Node>(nodeComparator);
    ArrayList<Node> closedList = new ArrayList<Node>();

    SearchTracker t = new SearchTracker(direction, true, null, 0);

    // The coordinate pair (symmetrical) the agent will travel towards.
    int goal;

    // Initialize values based on the direction the agent will travel.
    if(direction == Direction.FORWARD) {
        t.agent = new Node(0, 0, 0,
                heuristicCalc(heuristic, 0, 0, dim, dim), null);
        t.path = new Node(0, 0, 0,
                heuristicCalc(heuristic, 0, 0, dim, dim), null);
        goal = dim;
    } else { // BACKWARD
        t.agent = new Node(dim, dim, 0,
                heuristicCalc(heuristic, dim, dim, 0, 0), null);
        t.path = new Node(dim, dim, 0,
                heuristicCalc(heuristic, dim, dim, 0, 0), null);
        goal = 0;
    }

    Node presumedPath;
    Node temp;

    // Execute the Repeated A* Search algorithm.
    while(!t.finished) {
        t.counter ++;
        t.agent.search = t.counter; //
        t.agent.g = 0;
        agentWorld[t.agent.x][t.agent.y] = 0;
        agentWorld[goal][goal] = Integer.MAX_VALUE;
        closedList.clear();
        openList.clear();

        // temp is a Node linked list that contains a path from the agent’s
        // current position to the goal assuming there are no obstructions.
        temp = aStarSearch(agentWorld,
                openList, closedList, heuristic, t);
        temp.search = t.counter;
        presumedPath = reversePath(temp);
        if(openList.isEmpty()) { // Failure.
            break;
        }
        t.agent = moveAgent(t, presumedPath, agentWorld, gridworld);
    }
    // If the agent reached the goal we return its travel path.
    // Else, we return a node with a null tree pointer to signify failure.
    return (t.agent.x == goal && t.agent.y == goal)? t.path :
            new Node(-1,-1, -1, -1, null);
}
```