Project 1 Report

Search-Tree Node DT:

- This data type represents a node in the search tree of a search problem
- It has the following instance variables:
    - state: an object of type State, representing the state this node corresponds to
    - parent: an object of type SearchTreeNode, representing the parent of this node
    - operator: an object of type Operator, representing the operator applied to get this node
    - depth: a variable to type int, representing the depth of the node in the search tree
    - pathCost: a variable of type String, representing the cost of the path from the root to this node
    - id: a variable of type int, representing a unique identifier for the node

- All the above instance variables have getter methods, in addition to a setter method for the pathCost.

Search-Problem ADT:

- This data type represents an abstract search problem
- It has the following instance variables:
    - operators: an array of Operator objects, representing the various operators that can be used in this problem
    - initialState: an object of type State, representing the initial state of the problem
    - orderedInsertNodeComparator: an object of type NodeComparator, to be used as a comparator in the case of Uniform cost search strategy
    - manhattanHeuristicNodeComparator: an object of type NodeComparator, to be used as a comparator in the case of Greedy using Manhattan Heuristic
    - euclideanHeuristicNodeComparator: an object of type NodeComparator, to be used as a comparator in the case of Greedy using Euclidean Heuristic
    - totalCostEuclideanNodeComparator: an object of type NodeComparator, to be used as a comparator in the case of A* using Euclidean Heuristic

    - totalCostManhattanNodeComparator: an object of type NodeComparator, to be used as a comparator in the case of A* using Manhattan Heuristic

- The class has a static variable expandedNodes which is a counter representing the number of expanded nodes
- The class has a static variable nodeId which is a counter incremented whenever a node is created and assigned as a unique identifier to the node
- The class has the following instance methods:
    - nextState(State oldState, Operator action): a transition function that takes an old state and action and returns the next state
    - goalTest(State state): a function that checks whether a state is a goal state or not
    - pathCost(SearchTreeNode node): a function that returns the cost of the path from the root to the node passed as an argument
    - isRepeatedState(State state): a function that checks whether a state is a repeated state or not
    - resetRepeatedStates(): a function that resets the repeated states stored, needed only for iterative deepening search

- The class has the following static methods:
    - BF(SearchProblem problem): a method for the breadth-first strategy, it calls the generalSearch method with the queuing function ENQUEUE_AT_END
    - DF(SearchProblem problem): a method for the depth-first strategy, it calls the generalSearch method with the queuing function ENQUEUE_AT_FRONT
    - ID(SearchProblem problem): a method for the iterative deepening strategy, it keeps on incrementing the depth and calling the generalSearch method with the queuing function ENQUEUE_AT_FRONT and the depth, the depth is incremented by the following formula: depth = depth + 1 + (depth * 0.1), this makes the increment relative to the depth of the tree, which cuts execution time significantly compared to just incrementing by 1.
    - UC(SearchProblem problem): a method for the uniform cost strategy, it calls the generalSearch method with the queuing function ORDERED_INSERT
    - GR1(SearchProblem problem): a method for the greedy strategy using the manhattan heuristic, it calls the generalSearch method with the queuing function MANHATTAN_HEURISTIC

- GR2(SearchProblem problem): a method for the greedy strategy using the euclidean heuristic, it calls the generalSearch method with the queuing function EUCLIDEAN_HEURISTIC
- AS1(SearchProblem problem): a method for the A* strategy using the manhattan heuristic, it calls the generalSearch method with the queuing function MANHATTAN_TOTAL
- AS2(SearchProblem problem): a method for the A* strategy using the euclidean heuristic, it calls the generalSearch method with the queuing function EUCLIDEAN_TOTAL
- generalSearch(SearchProblem problem, QueuingFunction queuingFunction): this method represents the core of all search algorithms, it initializes a root node, creates a nodes queue with a comparator based on the queuing function passed, and adds the root node to the queue. It then iterates over the nodes queue, removing its front and checking whether it is a goal node or not using the goalTest method, if it is a goal, it is returned, otherwise the node is expanded using the expand method and the new nodes are added to the queue based on the queuing function passed, using the updateQueue method.
- generalSearch(SearchProblem problem, QueuingFunction queuingFunction, int depth): an overloaded generalSearch method that takes depth as an argument, it is used by the iterative deepening strategy, it performs the same functionality as the previous method, with the only difference that it expands a node only if its depth is less than the depth argument passed.
- expand(SearchProblem problem, SearchTreeNode node, Operator[] operators): a method that expands a node, for each operator in the problem, the current state and the operator are passed to the transition function, to get the new state. If a new state is returned, a node is constructed using this state, and added to the list of nodes that will be returned.
- updateQueue(Queue<SearchTreeNode> oldQueue, ArrayList<SearchTreeNode> expandedNodes, QueuingFunction queuingFunction): a method that updates the queue using the nodes returned from expansion, if queuing function passed is ENQUEUE_AT_FRONT, then a new queue is created and the expanded nodes are inserted in it first, then the rest of the old queue, otherwise, the expanded nodes are inserted directly in the old queue, and the order will be handled by the comparator of the priority queue if it has one.

MissionImpossibleState:

- This class represents the state of the mission impossible problem
- It inherits from the State abstract class
- It has the following instance variables:
    - ethanPosition: a variable of type String, representing the row and column numbers of ethan on the grid
    - totalDamage: a variable of type int, representing the total damage incurred to the members since the root node
    - totalDeaths: a variable of type int, representing the total number of dead members
    - members: a String array, containing strings representing the members, each member is represented as “row,column,health,flag”, flag is one of ‘r’, ‘c’ and ‘d’ representing remaining, carried and dropped respectively.

- The class has getters and setters for all these instance variables

MissionImpossible:

- This class represents the Mission Impossible problem
- It inherits from the SearchProblem abstract class
- It has the following instance variables in addition to the ones inherited from SearchProblem abstract class:
    - width: a variable of type int, representing width of the grid
    - height: a variable of type int, representing height of the grid
    - submarinePosition: a variable of type String, representing the row and column numbers of the submarine in the grid
    - truckCapacity: a variable of type int, representing the maximum number of members the truck can carry
    - numberOfMembers: a variable of type int, representing the total number of members
    - statesHistory: an object of type HashSet<String>, used to store states to check for repeated states

- It has the following instance methods:
    - goalTest(State state): a method inherited from SearchProblem, it is implemented to check whether ethan and submarine are in the same position, and whether all members are dropped
    - pathCost(SearchTreeNode node): a method inherited from SearchProblem, it is implemented to get the path cost of a node, by

getting the number of deaths and the total incurred damage from the state corresponding to the node.

- isRepeatedState(State state): a method inherited from SearchProblem, it is implemented to check whether a state is repeated by checking if it exists in statesHistory hashset, if it does not exist, then it is added to the hashset. States are stored in the HashSet in the following format: "ethanRow,ethanColumn;member1Row,member1Column....,memberkRow,memberkColumn;carriedMembersCount"
- resetRepeatedStates(): a method inherited from SearchProblem, it clears the statesHistory hashset and adds the initial state again.
- nextState(State oldState, Operator action): a method inherited from SearchProblem, a transition function, it takes the old state and the operator as arguments, and returns the new state, by calculating the new position of ethan, the new position, health and flag of each member, the new total deaths, the new total damage. It makes sure this new state is not a repeated state by calling isRepeatedState.
- nodeToString(SearchTreeNode node): a method that takes the goal node, and iterates over all nodes from this goal node to the root node to get the plan, it also gets the healths of each member and the total number of deaths from the goal node, and formats a string with the required output format.
- membersCount(String[] members): a method that takes the members array and determines the remaining members and the number of carried members, and returns a string with the following format: "member1Row,member1Column;....memberkRow,memberkColumn;carriedMembersCount", this is used as part of the formatted state stored in statesHistory hashset.

- The class has the following static methods:
  - genGrid(): it generates a random grid along with random members at random positions, a random submarine position and a random ethan position. The generated grid is between 5×5 and 15×15 and the number of generated IMF members is between 5 and 10. For every IMF member, a random starting health between 1 and 99 is generated, the number of members the truck can carry “c” is generated as well.
  - solve(String grid, String strategy, boolean visualize): a method that initializes the MissionImpossibleProblem using the grid passed. It calls the correct search strategy method based on the strategy passed. It calls the nodeToString method with the goal node returned to format the solution with the required output format and returns the solution. If visualize

argument is passed as true, the visualize method is called to sideeffect a visual representation of each step in the plan in the solution.
- visualize(SearchTreeNode node, MissionImpossible problem): a method that loops over all nodes from the goal node to the root node, and constructs a grid representing the state of each node, and displays each grid by calling the displayGrid method.
- displayGrid(String[][] grid): a method that takes a grid and prints a visual representation of the grid.

Comparators:

- OrderedInsertNodeComparator: compares between path costs of nodes, used in the priority queue of uniform cost search
- ManhattanHeuristicNodeComparator: compares between manhattan heuristic costs of nodes, used in the priority queue of Greedy 1.
- EuclideanHeuristicNodeComparator: compares between euclidean heuristic costs of nodes, used in the priority queue of Greedy 2.
- TotalCostManhattanNodeComparator: compares between total costs of nodes (manhattan heuristic cost + path cost), used in the priority queue of A* 1.
- TotalCostEuclideanNodeComparator: compares between total costs of nodes (euclidean heuristic cost + path cost), used in the priority queue of A* 2.
- In all comparators, priority in comparison is given to number of deaths, followed by damage, followed by insertion order.

Heuristics:

- Manhattan Heuristic: The heuristic function estimates the cost from the current node to the goal node, it starts by initializing total damage and total deaths with 0. Then for each remaining member on the grid, it calculates the manhattan distance between this member and ethan, multiplies this distance by 2 to get the cost of moving to this member, adjusts the cost such that the damage of the member does not exceed 100, and updates the total damage and total deaths accordingly. After repeating these steps for each remaining member, the total deaths and damage are returned.
- Euclidean Heuristic: Exactly the same as the first heuristic but uses euclidean distance instead of manhattan distance to calculate the distance between ethan and the member.
- Both heuristics are admissible, that is they never overestimate the cost to the nearest goal, that is because the estimated cost is calculated by calculating the distance from ethan’s position to every remaining member, neglecting the fact that ethan will move from one member to another, and assuming he will only

move from his position to every member. The actual cost however would be incurred by ethan moving from one remaining member to another. So the estimated cost will always be less than the actual cost.

Examples:

- Input:
  "9,9;8,7;5,0;0,8,2,6,5,6,1,7,5,5,8,3,2,2,2,5,0,7;11,13,75,50,56,44,26,77,18;2"
- BF:
  - Complete
  - Not optimal
  - Expanded Nodes: 123376
  - Max CPU utilization: 43%
  - Max RAM usage: 28 MB
  - Output:
    right,up,up,up,up,up,up,up,up,carry,left,carry,left,left,left,left,left,left,left,do
    wn,down,down,down,down,drop,right,right,right,right,right,right,right,up,up,
    up,up,carry,left,down,carry,left,left,left,left,left,left,down,down,down,drop,ri
    ght,right,right,right,right,right,right,carry,left,carry,left,left,left,left,left,drop,right,ri
    ght,right,right,right,up,up,up,carry,left,left,left,carry,left,left,down,down,dow
    n,drop,right,right,right,down,down,down,carry,left,left,left,up,up,up,drop;6;2
    9,91,100,100,100,100,100,100,40;123376

- DF:
  - Complete
  - Not optimal
  - Expanded Nodes: 847
  - Max CPU utilization: Could not report due to very fast execution
  - Max RAM usage: Could not report due to very fast execution
  - Output:
    left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,up
    ,up,left,left,left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,
    right,up,up,left,carry,left,left,left,left,left,left,left,down,down,right,right,right,ri
    ght,right,right,right,right,down,down,left,left,left,left,left,left,left,left,down,do
    wn,right,right,right,right,right,right,right,right,down,down,left,left,left,left,left,
    carry,left,left,left,up,up,up,drop,right,right,right,right,right,right,right,right,up,
    up,left,left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,r
    ight,up,carry,left,left,left,left,left,left,left,left,down,down,right,right,right,right,
    right,right,right,right,down,down,left,left,left,left,left,left,left,left,down,drop,ri
    ght,right,right,right,right,right,right,right,right,up,up,left,left,left,left,left,left,left,left,
    up,up,right,right,right,right,right,right,right,carry,left,left,left,left,left,left,left,d

own,down,right,right,right,right,right,right,right,right,down,down,left,left,left,left,left,left,left,left,drop,right,right,right,right,right,right,right,right,up,up,left,left,left,left,left,left,up,carry,left,left,up,up,right,right,right,right,right,right,right,down,down,left,left,left,left,down,left,down,left,left,left,down,down,right,right,right,right,right,right,up,carry,left,left,left,left,left,left,drop,right,right,right,right,right,right,right,right,up,up,left,left,left,up,carry,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,down,down,left,down,left,down,left,left,left,left,left,left,down,drop,right,right,right,right,right,right,right,right,right,up,up,left,left,up,carry,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,down,down,down,left,down,left,left,left,left,left,left,left,down,drop,right,right,right,right,right,carry,left,left,left,left,left,drop;8;100,100,100,100,100,100,100,100,98;847

- ID:
    - Complete
    - Not optimal
    - Expanded Nodes: 991704
    - Max CPU utilization: 56%
    - Max RAM usage: 21MB
    - Output:
      left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,up,up,left,left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,up,up,left,carry,left,left,left,left,left,left,left,down,down,right,right,right,right,right,right,right,right,down,down,left,left,left,left,left,left,left,left,down,down,right,right,right,right,right,right,right,right,down,down,left,left,left,left,left,carry,left,left,left,up,up,up,drop,right,right,right,right,right,right,right,right,up,up,left,left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,up,carry,left,left,left,left,left,left,left,left,down,down,right,right,right,right,right,right,right,right,down,down,left,left,left,left,left,left,left,left,down,drop,right,right,right,right,right,right,right,right,up,up,left,left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,carry,left,left,left,left,left,left,left,down,down,right,right,right,right,right,right,right,right,down,down,left,left,left,left,left,left,left,left,left,drop,right,right,right,right,right,right,right,right,right,up,up,left,left,left,left,left,up,carry,left,left,up,up,right,right,right,right,right,right,right,right,down,down,left,left,left,left,down,left,down,left,left,left,down,down,right,right,right,right,right,right,up,carry,left,left,left,left,left,left,drop,right,right,right,right,right,right,right,right,up,up,left,left,left,up,carry,right,carry,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,down,down,down,left,down,left,left,left,left,left,left,left,down,drop,right,right,right,right,rig

ht,carry,left,left,left,left,left,drop;8;100,100,100,100,100,100,100,100,98;99
1704

- UC:
  - Complete
  - Optimal
  - Expanded Nodes: 27448
  - Max CPU utilization: 33%
  - Max RAM usage: 15 MB
  - Output:
    left,up,up,up,carry,left,carry,left,left,left,left,left,drop,right,right,up,up,up,car
    ry,left,left,down,down,down,drop,right,right,right,right,right,right,up,up,up,c
    arry,right,right,up,up,carry,left,left,left,left,left,left,left,left,down,down,down,
    down,down,drop,right,right,right,right,right,right,right,up,up,up,up,up,carry,
    down,carry,left,left,left,left,left,left,left,down,down,down,down,drop,right,rig
    ht,right,right,right,up,up,up,carry,left,left,down,down,down,down,down,dow
    n,carry,left,left,left,up,up,up,drop;4;89,81,83,100,68,100,62,100,100;27448

- GR1:
  - Complete
  - Not Optimal
  - Expanded Nodes: 30355
  - Max CPU utilization: 31%
  - Max RAM usage: 10 MB
  - Output:
    up,up,up,left,up,up,up,carry,left,up,up,right,right,carry,down,left,left,down,l
    eft,left,left,up,up,left,left,down,down,down,right,down,right,right,right,right,ri
    ght,right,right,down,down,left,left,left,left,left,left,left,left,up,drop,right,right,r
    ight,right,right,right,right,up,up,up,up,up,carry,left,down,carry,left,left,l
    eft,left,left,left,left,down,down,down,down,drop,right,right,right,right,right,ri
    ght,carry,left,carry,left,left,left,left,left,drop,right,right,right,right,right,up,up,
    up,carry,left,left,left,carry,left,left,down,down,down,drop,right,right,right,do
    wn,down,down,carry,left,left,left,up,up,up,drop;7;100,27,100,100,100,100,
    100,100,44;30355

- GR2:
  - Complete
  - Not Optimal
  - Expanded Nodes: 15500
  - Max CPU utilization: 29%

- Max RAM usage: 10 MB
- Output:
  up,up,up,up,left,up,up,carry,left,down,down,down,carry,up,up,up,right,up,up,left,left,left,left,down,left,down,down,left,down,down,drop,right,right,up,up,right,right,right,right,up,right,up,right,up,carry,left,carry,left,left,left,left,left,left,left,down,down,down,down,down,drop,right,right,right,right,right,right,right,up,up,up,up,carry,left,left,down,carry,left,left,left,left,left,down,down,down,drop,right,right,right,right,right,right,carry,left,left,left,down,down,down,carry,left,left,left,up,up,up,drop,right,right,up,up,up,carry,left,left,down,down,down,drop;6;99,27,100,100,80,100,100,100,100;15500

- AS1:
  - Complete
  - Optimal
  - Expanded Nodes: 24472
  - Max CPU utilization: 28%
  - Max RAM usage: 12 MB
  - Output:
    up,up,up,left,carry,left,carry,left,left,left,left,left,drop,right,right,up,up,up,carry,down,down,down,left,left,drop,right,right,right,right,right,right,up,up,up,carry,right,up,up,right,carry,left,left,left,left,left,left,left,left,down,down,down,down,down,drop,right,right,right,right,right,right,right,up,up,up,up,up,carry,down,carry,left,left,left,left,left,left,left,down,down,down,down,drop,right,right,right,right,up,up,up,carry,left,left,down,down,down,down,down,down,carry,left,left,left,up,up,up,drop;4;89,81,83,100,68,100,62,100,100;24472

- AS2:
  - Complete
  - Optimal
  - Expanded Nodes: 25112
  - Max CPU utilization: 29%
  - Max RAM usage: 13 MB
  - Output:
    up,up,up,left,carry,left,carry,left,left,left,left,left,drop,right,right,up,up,up,carry,down,down,down,left,left,drop,right,right,right,right,up,right,up,up,right,carry,up,right,up,right,carry,left,left,left,left,left,left,left,left,down,down,down,down,down,drop,right,right,right,right,right,right,right,up,up,up,up,up,carry,down,carry,left,left,left,left,left,left,left,down,down,down,down,drop,right,rig

ht,right,right,right,up,up,up,carry,left,left,down,down,down,down,down,dow
n,carry,left,left,left,up,up,up,drop;4;89,81,83,100,68,100,62,100,100;25112

- Input:
  "11,11;7,7;8,8;9,7,7,4,7,6,9,6,9,5,9,1,4,5,3,10,5,10;14,3,96,89,61,22,17,70,83;5"
- BF:
  - Complete
  - Not optimal
  - Expanded Nodes: 305725
  - Max CPU utilization: 31%
  - Max RAM usage: 126 MB
  - Output:
    left,left,left,carry,left,left,left,down,down,carry,right,right,right,right,carry,rig
    ht,carry,right,carry,right,up,drop,left,left,up,carry,left,up,up,up,carry,right,rig
    ht,right,right,right,up,carry,down,down,carry,left,left,down,down,down,drop
    ;4;50,9,100,100,89,40,77,100,100;305725

- DF:
  - Complete
  - Not optimal
  - Expanded Nodes: 1157
  - Max CPU utilization: Could not report due to very fast execution
  - Max RAM usage: Could not report due to very fast execution
  - Output:
    left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,rig
    ht,right,up,up,carry,left,left,left,left,left,left,left,left,left,down,down,right,ri
    ght,right,right,right,right,right,right,right,right,down,down,left,left,left,left,left,
    left,left,left,left,left,down,down,right,right,right,right,right,right,right,carry,left
    ,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,right,
    right,up,up,left,left,left,left,left,up,carry,left,left,left,left,left,up,up,right,right,ri
    ght,right,right,right,right,right,right,right,down,down,left,left,left,down,left,do
    wn,left,left,left,left,left,left,down,down,right,right,right,right,right,right,right,ri
    ght,drop,left,left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,
    right,right,right,right,up,carry,left,left,left,left,left,left,left,left,left,left,down,do
    wn,right,right,right,right,right,right,right,right,right,right,down,down,left,left,l
    eft,left,left,left,left,left,left,carry,left,up,up,right,right,right,right,right,right,righ
    t,right,right,right,down,down,left,left,left,left,left,carry,left,left,left,left,left,up,
    up,right,right,right,right,right,right,right,right,down,drop,left,left,left,left,up,c

arry,left,left,left,left,up,up,right,right,right,right,right,right,right,right,right,right,righ
t,down,down,left,left,left,left,down,down,carry,left,left,left,left,left,left,up,up,r
ight,right,right,right,right,right,right,right,right,down,drop,left,left,up,carry,left,left,l
eft,left,left,left,up,up,right,right,right,right,right,right,right,right,right,right,do
wn,down,left,left,down,drop;9;100,100,100,100,100,100,100,100,100;1157

- ID:
  - Complete
  - Not optimal
  - Expanded Nodes: 1226249
  - Max CPU utilization: 59%
  - Max RAM usage: 43 MB
  - Output:
    left,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,rig
    ht,right,up,up,carry,left,left,left,left,left,left,left,left,left,down,down,right,ri
    ght,right,right,right,right,right,right,right,right,down,down,left,left,left,left,left,
    left,left,left,left,left,down,down,right,right,right,right,right,right,right,carry,left
    ,left,left,left,left,left,left,up,up,right,right,right,right,right,right,right,right,right,
    right,up,up,carry,left,left,left,left,left,left,left,left,left,left,down,down,right,righ
    t,right,right,right,right,right,right,right,right,down,down,left,left,left,left,left,lef
    t,left,left,left,carry,left,up,up,right,right,right,right,right,right,right,right,down,
    drop,left,left,left,left,up,carry,left,left,left,left,up,up,right,right,right,right,right,
    up,carry,left,left,left,left,left,down,down,right,right,right,right,right,down,righ
    t,carry,right,down,right,down,down,left,left,up,carry,left,carry,right,right,righ
    t,up,drop;9;100,100,100,100,100,100,100,100,100;1226249

- UC:
  - Complete
  - Optimal
  - Expanded Nodes: 7350
  - Max CPU utilization: Could not be reported due to very fast execution
  - Max RAM usage: Could not be reported due to very fast execution
  - Output:
    left,carry,down,down,carry,left,carry,right,right,carry,right,up,drop,left,left,le
    ft,left,up,carry,right,up,up,up,carry,left,left,left,left,down,down,down,down,d
    own,carry,right,right,right,right,right,right,right,right,right,up,up,up,up,up,up,
    carry,down,down,carry,left,left,down,down,down,drop;2;32,39,98,97,73,88
    ,63,100,100;7350

- GR1:

- Complete
- Not Optimal
- Expanded Nodes: 3169
- Max CPU utilization: 31%
- Max RAM usage: 6 MB
- Output:
  left,left,down,down,carry,left,up,up,carry,right,down,down,right,carry,right,carry,left,left,left,left,left,left,carry,right,right,right,right,up,up,up,up,up,right,right,right,right,right,down,down,left,down,left,down,drop,left,left,up,carry,left,up,up,up,carry,right,right,right,right,right,up,carry,down,down,carry,left,left,down,down,down,drop;5;44,19,100,100,69,66,100,100,100;3169

- GR2:
  - Complete
  - Not Optimal
  - Expanded Nodes: 4789
  - Max CPU utilization: 27%
  - Max RAM usage: 8 MB
  - Output:
    left,carry,left,left,carry,right,right,down,down,left,carry,up,up,up,up,up,carry,down,down,down,down,right,down,right,carry,left,left,left,left,left,left,left,up,up,up,up,up,right,right,right,right,right,right,right,right,down,down,down,down,drop,left,left,left,left,left,left,left,down,carry,right,right,right,right,right,carry,right,right,right,right,up,up,up,up,up,up,carry,down,down,carry,left,left,down,down,down,drop;4;62,11,98,100,81,100,49,100,100;4789

- AS1:
  - Complete
  - Optimal
  - Expanded Nodes: 4070
  - Max CPU utilization: 26%
  - Max RAM usage: 4 MB
  - Output:
    left,carry,down,down,carry,left,carry,right,right,carry,up,right,drop,left,left,left,left,up,carry,right,up,up,up,carry,left,left,left,left,down,down,down,down,down,carry,right,right,right,right,right,right,right,right,right,up,up,up,up,up,up,carry,down,down,carry,left,left,down,down,down,drop;2;32,39,98,97,73,88,63,100,100;4070

- AS2:

- Complete
- Optimal
- Expanded Nodes: 4718
- Max CPU utilization: 26%
- Max RAM usage: 4 MB
- Output:
  left,carry,down,down,carry,left,carry,right,right,carry,up,right,drop,left,left,left,left,up,carry,up,up,right,up,carry,left,left,down,down,down,left,down,left,down,carry,right,right,right,right,right,right,right,right,right,up,up,up,up,up,up,carry,down,down,carry,left,left,down,down,down,drop;2;32,39,98,97,73,88,63,100,100;4718

- As observed from the number of expanded nodes, CPU utilization and RAM usage, there are some fluctuations, but the general trend is that as the number of expanded nodes increase, both the CPU utilization and RAM usage increase.