For the first application of ITCS-3153-001, many teams had to produce an Artificial Intelligence that could solve a randomized maze structured like Figure 1 below. Zeros represented nodes that could not be traversed, and every other number 1-9 represented the step cost for each node.

1	5	2	4	1	5	0	0	4	0
1	5	0	2	4	1	0	2	5	0
1	0	5	5	1	2	1	4	4	3
2	1	2	2	0	0	2	4	4	0
2	0	4	2	0	0	1	5	1	2
0	3	4	1	3	0	4	2	1	3
1	0	3	5	2	5	1	0	4	3
2	4	0	1	0	2	4	2	4	0
0	4	5	5	2	5	3	1	4	3
3	0	3	3	5	5	3	4	1	1

Figure 1

To add on another stipulation, the AI had to be able to use different search algorithms to find the randomized goal node starting from a randomized initial state. This paper will focus on how Group 10 of the Fall 2021 class constructed their AI to solve randomized mazes using uninformed, informed, and local search algorithms.

It's important to recognize how the AI sees each position on the maze grid. To make it easier to visualize, Figure 2 describes how each node is positioned so that the AI can read the maze as a 2D List, with the first integer representing the "y" coordinate, and the second integer representing the "x" coordinate.

[0,0]	[0,1]	[0,2]	[0,3]	[0,4]	[0,5]	[0,6]	[0,7]	[0,8]	[0,9]
[1,0]	[1,1]	[1,2]	[1,3]	[1,4]	[1,5]	[1,6]	[1,7]	[1,8]	[1,9]
[2,0]	[2,1]	[2,2]	[2,3]	[2,4]	[2,5]	[2,6]	[2,7]	[2,8]	[2,9]
[3,0]	[3,1]	[3,2]	[3,3]	[3,4]	[3,5]	[3,6]	[3,7]	[3,8]	[3,9]
[4,0]	[4,1]	[4,2]	[4,3]	[4,4]	[4,5]	[4,6]	[4,7]	[4,8]	[4,9]
[5,0]	[5,1]	[5,2]	[5,3]	[5,4]	[5,5]	[5,6]	[5,7]	[5,8]	[5,9]
[6,0]	[6,1]	[6,2]	[6,3]	[6,4]	[6,5]	[6,6]	[6,7]	[6,8]	[6,9]
[7,0]	[7,1]	[7,2]	[7,3]	[7,4]	[7,5]	[7,6]	[7,7]	[7,8]	[7,9]
[8,0]	[8,1]	[8,2]	[8,3]	[8,4]	[8,5]	[8,6]	[9,7]	[8,8]	[8,9]
[9,0]	[9,1]	[9,2]	[9,3]	[9,4]	[9,5]	[9,6]	[9,7]	[9,8]	[9,9]

Figure 2

One might notice that the “y” is top down instead of bottom up. This is why the coordinate pair analogy does not exactly fit, but the Figure above is an accurate representation on how the AI looks at the maze. In this paper, we will be touching on a plethora of different search algorithms, the first of many being uninformed search algorithms. Uninformed search algorithms adopt the “bruce force” strategy, whereas the algorithms have to search a tree without any additional information. There are three types of uninformed search algorithms: Uniform-Cost search, Depth-First search, and Breadth-First search. Uniform-Cost search searches the tree based on the step cost of each individual node, trying to ultimately find the least costly path. Depth-First search expands based on the maximum depth of the tree. Lastly, Breadth-First search expands like Depth-First search does, but only going down the tree as far as the depth of the goal node. These algorithms all have different time complexities, however they are all considered functional so long as they can find the goal node; preferably within the least amount of path cost.

The Group 10 AI, for uninformed and informed search algorithms, after analyzing the grid, has four actions: Move left, right, up, and down. The AI can do these actions as long as the action does not result in expanding on a node with a value of 0 or result in going out of bounds. The AI will also check to make sure that the node it is expanding is not one that has been expanded before. Once it’s checked the actions it can take and checks if the node has already been expanded, it can expand the unexplored node. This process is looped over and over again until the AI finds the goal node.

When the AI runs the Breadth-First search algorithm, it starts out by expanding nodes around the starting point and searching for the goal node. It then expands the node with the least step cost if it could not find the goal node. It will continue to do so until it reaches the goal node. As for the depth first search, it will typically expand nodes that increase the distance from the starting point to find the goal node with the least path cost. While the algorithm works, it is not the best in finding the best solution as where it expands almost always depends on the distance for the start and not distance to the end, as well as the step cost instead of the path cost.

The next search algorithm that was programmed was the informed search algorithm. Informed search and uninformed search are very similar, with one distinguishing feature between the two of them. The informed search algorithm has a heuristic function which helps the agent make better decisions on which node to expand next. In other words, the AI is better informed. Of the two informed search algorithms, while a greedy search was an option, the assignment called only for the use of a A* search algorithm. This is because A* is more accurate. Although the A* and Greedy search algorithms are very similar, Greedy search algorithms only consider the heuristic cost, instead of both the heuristic and step cost. The A* algorithm is also very similar to the Uniform-Cost search algorithm, as both the A* and Uniform-Cost algorithms use path costs to determine which nodes they expand. However, the A* algorithm implements a heuristic function to make better decisions on which nodes to expand. It does so by plugging data into the equation below.

$$f(n) = h(n) + g(n)$$

$h(n)$ = heuristic cost (distance to goal node)

$g(n)$ = path cost

Figure 3

A* expands in a more optimal manner compared to the uninformed search algorithm. This is because it has more information and thus can expand nodes in a more optimal and efficient way.

Lastly, a local search algorithm had to be implemented. Local searches are different from classical searches as with classical searches, the solution is a path to the goal node. With local search, the solution itself is a state that eventually ends up being the gode node. So then, how does the AI know what is the “path” to take to find a solution? The AI is able to see the entire board or grid in the maze. It then uses an algorithm to determine what successor states would have the greatest possibility of getting to the goal node. It will loop this process until the AI gets to the state in which the current state is also the goal node.

Because of the nature of Simulated Annealing, this algorithm seemed to work as efficiently as the A* search when it comes to finding an optimal solution. This is because local search takes every single board possibility, and evaluates which board would create the best solutions. Of course, the value of T and the decay rate have a huge part in how well the algorithm operates or how complex the time complexity is, but from the overall test results, local search and A* were the best algorithms for finding, what most would call, the best solution for the maze search.