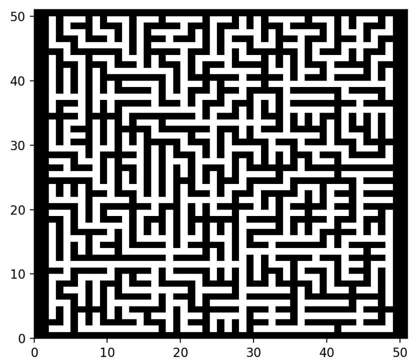
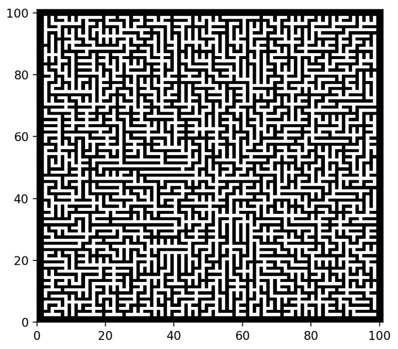
1. Introduction

We were quite interested in taking the planners we had developed earlier in the course and applying them to a much more difficult problem space. In particular, we noticed that the problems explored in previous problem sets did not have too many obstacles and did not require the planning trees generated by the EST/RST algorithms to change direction very much. In order to explore a more difficult path-planning problem, we settled on attempting to develop solvers in continuous space that would be tasked with going through mazes from a randomly generated starting node to a randomly generated goal node. We began with this problem and eventually extended to a problem in which the planner would encounter locked doors, which could only be unlocked after stumbling upon keys scattered throughout the maze. This was partially done to explore how much our planners exhibited breadth within the generated mazes, but also to take a stab at solving a dynamic path planning problem.

2. Setting up the problem space

While we will attempt to restrict our discussion of aspects of the project not directly related to path planning, it is helpful to highlight how the maze is actually generated. We explored quite a few options for maze generation and found/implemented several different approaches, but eventually determined Prim's maze generation algorithm to be the best at introducing a sufficiently complicated maze with one single shortest path from some set start node to goal node. A few examples of generated mazes are shown below, and were quickly highlighted in an earlier report:

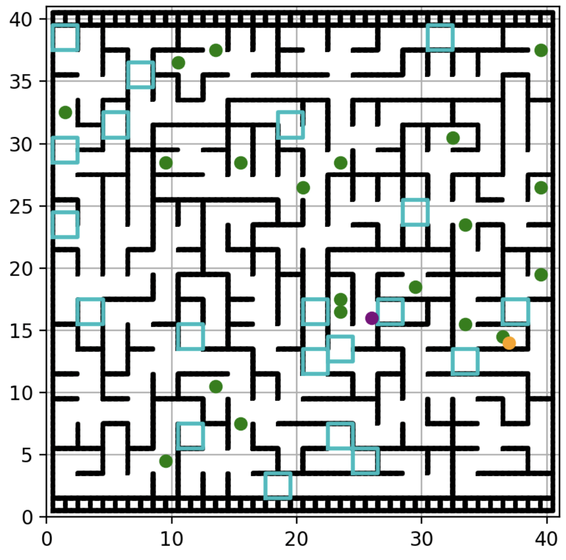




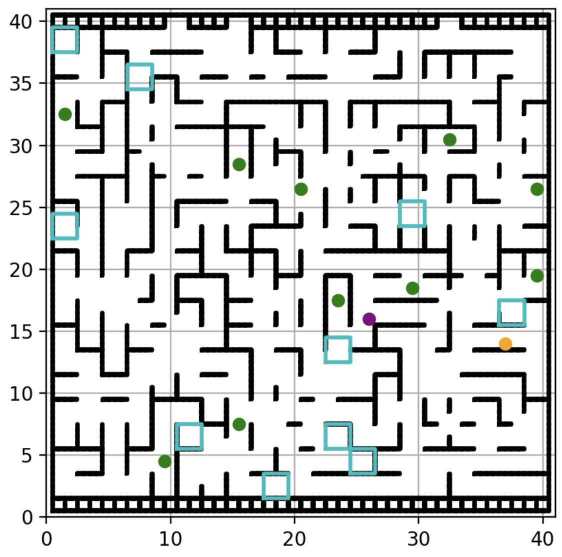
After the maze is generated, we randomly sample points in the grid to generate a start node, a goal node, key locations, and locked door locations. In the case of our project, there are an equal number of keys and locks, which are specified by a **num_keys** input. Let *K* be a list of keys and *L* a list of locks. For all *i*, we say that *K[i]* unlocks *L[i]*. Before lock *L[i]* is unlocked, it is treated the same as a wall by the planner, and cannot be crossed unless there exists a path around the locked door.

This modification lends itself to the creation of several maze configurations, irrespective of how parameters are set, in which case mazes cannot be solved. We had to take caution in these situations and designed a simple DFS-based approach that would check to see that a randomly generated maze could indeed be solved and, if not, we would continue regenerating the maze.

Taken together, the problem space looks like this after maze generation is complete:



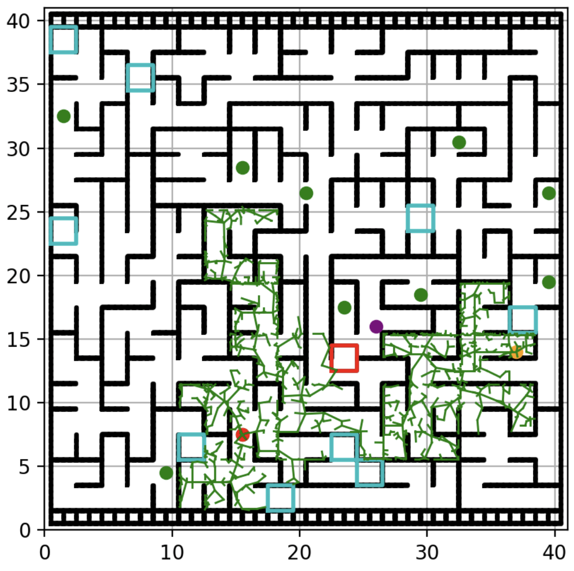
In this representation, each blue box represents a locked door, each green dot represents a key, the gold-colored dot represents the start node, and the purple dot represents the goal node. In this case, there are 20 keys/locks spread out across the 41 by 41 grid. In addition to all of this, we introduced a difficulty setting, which ranges from 0 to 1. The difficulty setting represents the probability that an internal wall in the maze is randomly removed. In the case shown above, difficulty is set to the maximum of 1. Below, an example is shown of the same maze with difficulty 0.9, and **num_keys** restricted to 10:



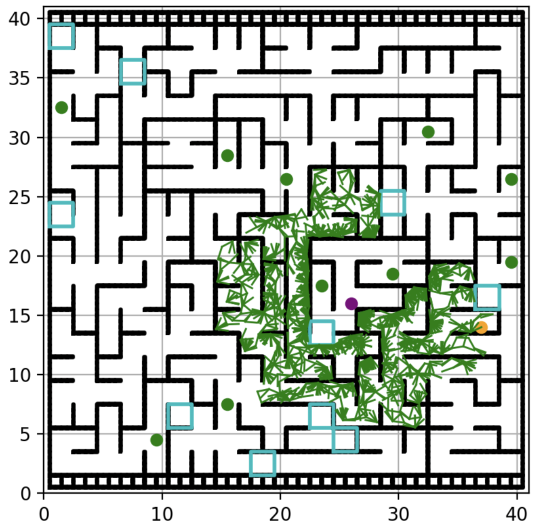
While these details may not seem particularly important now, they are the parameters of the problem space that we will compare in our discussion of the various planners below.

3. Path planning (unmodified)

We started with two path planners: a relatively simple RRT planner and an equally basic EST planner, both of which were not too dissimilar from what we had implemented previously in a problem set. They both seemed to struggle quite a lot, with the example of the "vanilla" RRT planner shown below on the maze presented above, with difficulty 1.0:

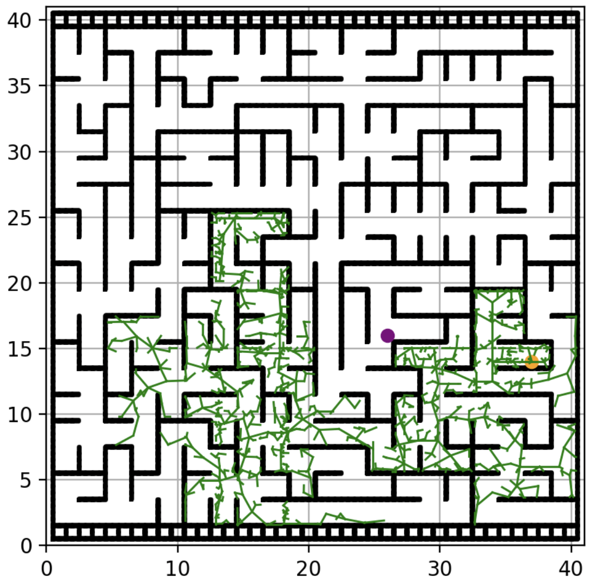


In fact, the RRT did not manage to find a path in this scenario (with 1500 nodes and 6207 steps). Below, we have an example of default EST attempting this complex problem as well. The default EST planner also did not manage to find the goal node after creating 1500 nodes:

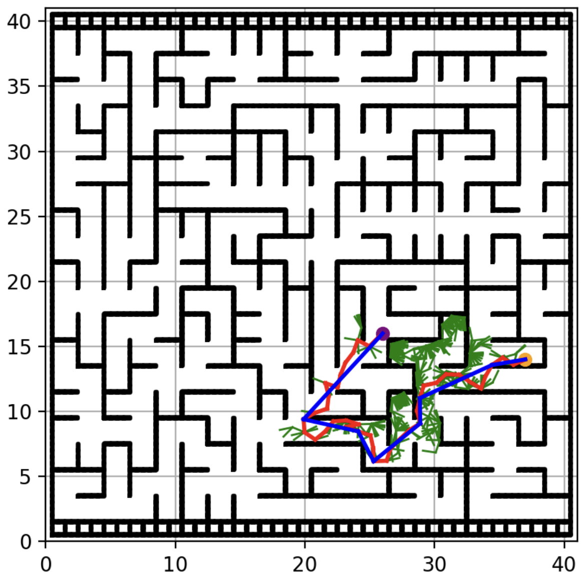


In fact, while these planners were especially bad at the locked doors problem, they also struggled quite a bit when presented with a maze without locked doors (shown below).

Default RRT



Default EST



These findings gave root to our final approaches for path planning, which are described below.

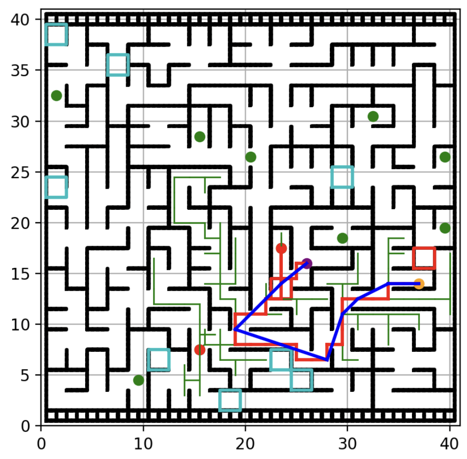
4. Approach

When considering the results from the default planners, we noticed quite a few issues with how the original path-planning algorithms worked. Firstly, in this grid-based problem, only right-angle turns are ever necessary to solve the maze. The original planners seemed to make small turns that would strictly increase the complexity of the tree without providing any value with regards to finding keys and/or the goal node. We reasoned, then, that if we pushed either an RRT or EST planner to only make movements that would grow off of the tree in right-angled or near-right-angled directions, the tree would be able to converge much faster and perform much better.

Additionally, in the case of EST, nodes were originally sampled from the tree by considering the number of nodes close to themselves (**numnear**) as well as the distance to the goal node. However, there are cases in which the goal node is close by but blocked by locked doors, so we would likely need to bias the growth node sampling technique by also somehow considering the location of keys scattered around the map. In addition to this, it is possible for EST to develop a cluster of leaf nodes next to a locked door which is later unlocked. Due to the **numnear** factor, the unlocked region would not be explored due to this existing cluster. As such, we figured that we should also slightly modify EST to account for this dynamic algorithm.

5. Technical Details

We started by modifying RRT to use only the x or y component of the new sampled node and keep the other component the same as the growth node. This made the RRT model grow only in 90-degree angles, so we dubbed this RRT variant RRT 90. This suited the nature of the maze problem better as there were fewer wasted turns. What we then noticed was that with only 90-degree turns, the planner would often go back in the same direction it had come from. To limit this, we restricted the planner to only grow in a direction that it hadn’t come from. That helped it to explore the space better and have a higher chance of solving the maze within 1500 nodes.



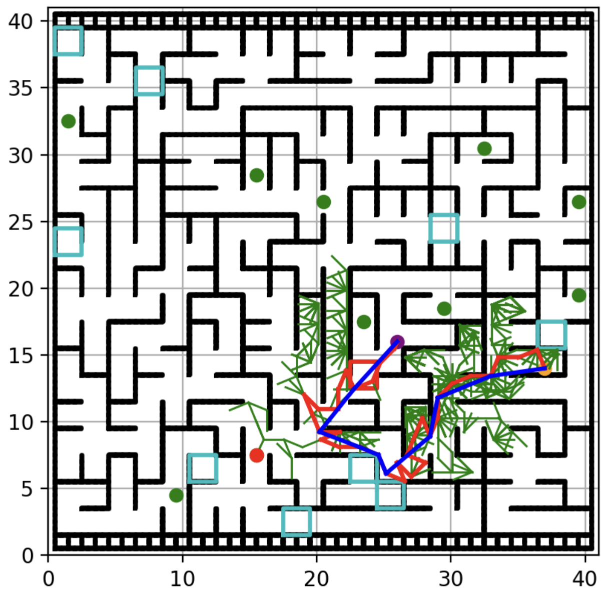
We then modified EST using our observations as noted above. In particular, we learned about a variation of RRT which randomly removes a leaf node in the explored tree for every iteration – this technique is called RRT* FN. To address the problem of leaves clustering around locked doors that later become unlocked, we introduced a segment of the code that would draw from a set of known leaf nodes in the tree and randomly remove one of them. Additionally, to bias the EST planner towards the location of keys, we modified how sampling occurred to use the following calculation:

$$\begin{aligned} dist_i &= \sqrt{(x_i - x_g)^2 + (y_i - y_g)^2} \\ keydist_i &= \min\{\sqrt{(x_i - x_{k_j})^2 + (y_i - y_{k_j})^2}, j \in [1, N_k]\} \\ i_{min} &= \arg\min_i (c_1 * numnear + c_2 * keydist_i + c_3 * dist_i) \\ x_{sample} &\leftarrow x_{i_{min}} \\ y_{sample} &\leftarrow y_{i_{min}} \end{aligned}$$

In this expression, each x_i and y_i corresponds to the x and y coordinates of a node within the tree, (x_g, y_g) represents the position of the goal, and each x_{k_i} and y_{k_i} represents the x and y coordinates of keys *i* for all keys 1 through N_k. We use the same **numnear** array as before, which uses a kdtree to get the number of nodes within 1.5*dstep distance of each node in the tree. We chose $c_1 = 1$, $c_2 = 2$, and $c_3 = 3$ in our final implementation. Essentially, this pushes us to grow from nodes that are close to the goal, are close to keys, and are in relatively unexplored regions. In other words, these positions would appear to be the perfect places to grow the tree.

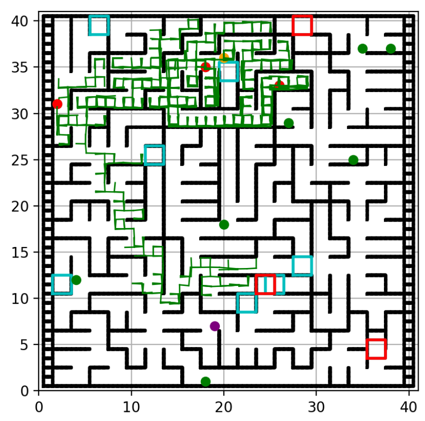
Additionally, we added a modification such that every time we add to the tree and a leaf node is naturally created, we add the node to a list of leaf nodes maintained within the planner. If we ever choose to grow from that node, we remove it from the leaf nodes list. Otherwise, on every iteration that there is more than one node in the tree, we randomly sample from the list of leaf nodes and remove one leaf node, if it is still in the tree. This, we hoped, would avoid the risk of clusters of leaf nodes forming near a locked door which should be crossed to eventually arrive at the goal.

We also introduced a modification to the EST that would sample the angle from which to grow from the tree from a normal distribution (as before), and then round this angle to the closest multiple of $\frac{\pi}{4}$. The result of these modifications is shown for the map and number of keys presented in the previous section:

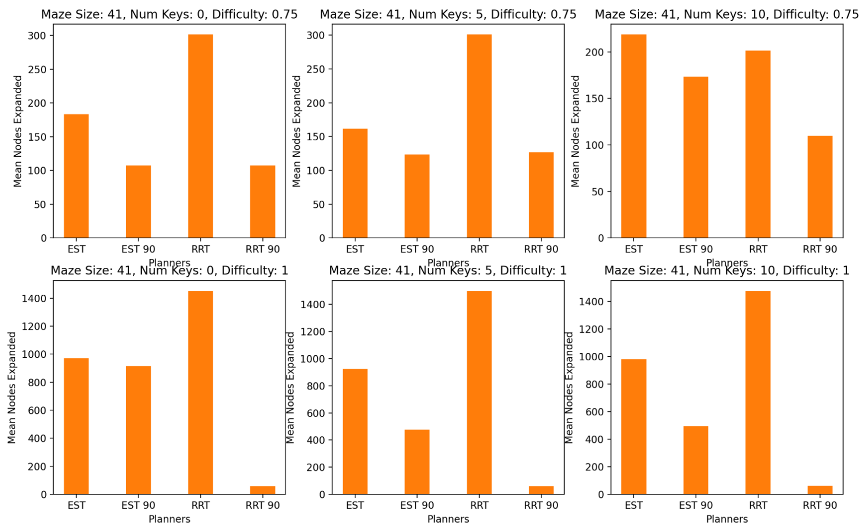


Even so, we found these modifications to yield slower convergence as compared to the aforementioned RRT modifications.

We then tried another EST planning technique similar to RRT 90. In this case however, we made it so the planner would grow within a certain tolerance of 90 degrees, so it did not need to be perfect. It looked as shown in the image below.



Finally, using all 4 of our planners, we decided to test how they performed on mazes of varying size, number of keys and locks, and maze difficulty. We implemented seeding to make the mazes consistent, then we ran each planner on each maze 20 times using each combination of variables. The results are shown below. Note that if a planner was unable to solve a maze, it was given a score of 1500 nodes, the maximum we allowed, for that attempt.



Upon analyzing our final results which varied the number of keys (0, 5, or 10) and the difficulty (.75 or 1), and averaged the number of nodes in the tree in the 20 runs per maze, we can see that RRT 90 is significantly better than the other planners with a difficulty of 1. It is also comparable to EST 90 for difficulty of .75. All mazes except for RRT 90 did significantly worse for mazes with more keys. This makes sense as RRT 90 includes developments to prevent it from exploring spaces it has already gone, so it is more likely to spread out and find keys.

Interestingly, we can see that RRT is much worse or comparable with EST on all of these parameters. This may be because our EST had some developments to help it explore as well, but it is hard to be sure. In general, we know that RRT is better at exploring more open space with a few big obstacles while EST is better at narrower spaces like this maze.

Overall, the improved models RRT 90 and EST 90 performed better than their predecessors, so our implementations were effective in reducing the number of nodes in the tree and thus the computation required of our computers.

6. Contributions / Lessons Learned

The primary thing we learned from this project was the importance of understanding your problem space. We first started by implementing two standard planners, but they were ill-suited for our mazes as shown by their performance in the above graphs. Our best-performing planner was RRT 90 which we made specifically to move within the maze and not regress on itself. Through our experimentation and understanding of the drawbacks of the old RRT/EST approaches, we were able to devise new approaches that worked much better than the approaches previously explored in class when applied to this specific problem space, which we found to be particularly interesting due to its difficulty. Some key takeaways include that if the problem space is known to have walls in certain configurations, planners can be configured to take advantage of that and reduce the complexity of the exploration tree. Furthermore, randomly removing leaves from a generated tree can be particularly beneficial in a dynamic environment.

In a general sense, we learned how to develop planners that were much more adept at solving this dynamic problem that we introduced for this project as compared to previously explored methods in problem sets.

7. Conclusion

For those who may wish to do a similar project in the future, we would recommend following a similar approach to what we did. Understanding the limitations of a planner in the space allowed us to improve its performance more than if we had started by trying something new. Having our simulation code to examine performance over many mazes at once helped with our analysis as well.

One area in which we would have liked to improve our planners would be in the pathfinding. While several of our planners were adapted to be very efficient at finding keys and the endpoint in the mazes, the path they produced did not necessarily always follow the path a person would have to take to get through the maze. For instance, if a key and the goal were acquired on different paths of the tree, it would show the final path as only going down the branch with the endpoint on it. We believe that animating a person following the path a person would have to take through the maze would be an interesting challenge to tackle.

One of the complexities of this problem is that we don't currently have a way to tell which keys were necessary to make it to the endpoint. Not every key needs to be picked up to reach the goal. We would need to add a way of checking which lock and key pairs were used to reach the goal.

8. Final Project Video Link

Google Drive link to the project video:
https://drive.google.com/file/d/12Qz16qqWcVXxXxWc8K_W7Y5NbrSMmVXc/view?usp=sharing