A* Path Planning for Dynamic Agents with Changing Motivations

Matthew Desaulniers
Colorado School of Mines
Robotics MS Student
Golden, USA
mdesaulniers@mines.edu

Kai Mizuno
Colorado School of Mines
Computer Science MS Student
Golden, USA
kmizuno@mines.edu

Emil Muly
Colorado School of Mines
Robotics MS Student
Golden, USA
emuly@mines.edu

***Abstract*—Many traditional path planners are used to find the shortest trajectory from some beginning state to goal state. This results in a static agent that moves through an environment a certain way every time. This behavior is not sufficient in modeling the behavior of human-like agents with internal motivations. In this paper, we present our work on developing an augmented version of A* path planning that allows an agent to determine a path to its goal that best aligns with its internal motivations. The agent has internal states that can be impacted by its environment, follows paths that align with its current internal state, and attempts to accomplish secondary goals that also align with its internal state. Tests of our algorithm in a square-grid space were conducted. Results will be discussed comparing our Modified A* with traditional A* with a special focus on contextual application.**

I. INTRODUCTION

As robots and autonomous virtual agents take on more dynamic roles and capabilities, there will be a need to allow them to be more human-like and flexible in the way they plan and navigate the world.

For instance, an autonomous vehicle driving near a charging station may decide to opportunistically stop as it is passing, rather than wait for some specific charge level. Alternatively, the same vehicle may run much lower than usual given the occupant communicates they are in a rush. The problem of juggling priorities and/or taking advantage of context to gain something that was not otherwise a primary goal is what this work aims to solve.

In this paper, we go over a modified version of the A^* path planning algorithm we have implemented in C++ that determines paths through a space based on an agent's internal motivations. The modified A^* algorithm described herein functionally augments the traditional implementation in three ways:

1) The node transition costs are dependent on the internal states of an Agent that is traversing the graph.
2) The Agent internal states change dynamically as it moves from node to node.
3) Secondary goals are dynamically pursued depending on proximity to the Agent's original path to goal and the Secondary Goal's internal state.

A brief discussion of relevant, related work will be followed by a more formalized problem description. The completed algorithm and methodology will then be detailed along with experiments and validation of the algorithm's functionality. The resulting data will then be discussed along with conclusions about the contexts in which the proposed algorithm may be applied efficiently.

II. RELATED WORK

There have been several cases of researchers looking into how to apply traditional path planners to more specific applications. For example, Chen et. al. [1] presented research on how to use q-learning to generate more realistic paths for autonomous cargo ships. They saw that existing planners do not account for the dynamic complexities of autonomous cargo ships. For example, most planners result in sharp turns that are not realistic for ships to perform. They present their augmented q-learning algorithm as an alternative path planner that allows them to integrate the complexities of autonomous ships into the reward q-learning functions which result in more realistic paths. This is an example of an application area where traditional path planners were not sufficient in modeling the agent.

Another example of this can be seen through the use of planning in video games. Almost every modern video game is implementing some kind of high level planning, however the fields of research and the gaming industry do not often have complete overlap [3]. Video games often require a level of complexity in unconstrained environments such that many academic planners may not apply [2]. In noted unique applications, the video game F.E.A.R. famously used the STRIPS (Stanford Research Institute Problem Solver) for controlling its non-player agents [4] [2]. Even though it was released in 2005, it is still not the norm for industry to use academic planners for agent planning and as such is an indicator that more work should be done in the field of dynamic path planning for human-like agent representations.

Research papers like the ones mentioned above inspired our team to look more closely at this problem of traditional path planners not properly representing dynamic agents. We became interested in creating an algorithm that addresses similar problems that these other papers address, without having the algorithm be quite as context-specific.

III. PROBLEM DESCRIPTION

Our group was interested in investigating ways that existing path planning algorithms we have talked about in class, such as Dijkstra and A*, can be evolved to incorporate decision making processes that account for more than one goal and a changing internal state of an agent. After examining related research, we concluded that traditional path planners do not properly account for the dynamic complexities most agents would have. In particular, we were interested in how to model a human-like agent that determines its path through a space based on its internal motivations. This problem we are trying to solve closely resembles how an agent would behave in a real world context. An agent should have a main objective they are trying to achieve in a given space, however, a space is rarely dedicated to that single objective. Instead, there will be many other potential tasks that can be done in this space which may or may not apply to the agent's overall goal, and the contents of the space the agent encounters will affect the agent's path planning decisions in real-time. In these cases, an ideal path would not be whatever gets the agent to the goal the fastest, since the agent is expected to consider changing environmental variables that may require an updated trajectory. Thus, planning algorithms should take these environments and internal states into account when generating paths to the goal.

Our objective was to design a planner for some agent, where it has a primary goal and a set of potential secondary goals to reach. The agent should seek out these secondary goals if they are within reach, however, the agent should be actively working toward the primary goal. Whether a secondary goal is within reach will be determined by a distance value associated with each secondary goal. Along the path to the primary goal, the agent will encounter changes to their internal state, which will affect the path that the agent views as the most optimal to get to the primary goal. This can be summarized by the following algorithmic requirements:

1) The path to the goal must be the path most aligned to the agent's internal state.
2) The agent should consider pursuing secondary goals on its way to the primary goal, if those secondary goals align with its internal state.
3) The agent's internal state should be changed by the environment as it moves through the space, then it must be able to reassess the best path to the goal based on its new internal state.

IV. METHODOLOGY

A. General Approach

As the agent moves through a state space, it needs to adapt to changes in its internal state and assess secondary goals that move into its vision range. These dynamic changes require the agent to repeatably reassess the path it wants to take to the primary goal. This requirement led to our team prioritizing run-time when picking which path-planning algorithm to implement. We identified A* as the best candidate for our algorithm due to it fitting this fast run-time requirement. The primary downside of traditional A* is the possibility of not always producing the shortest path to the goal due to its heuristic function, however, our group was not concerned with this downside since our objective for this algorithm does not include having the shortest possible path. Detours and slightly longer paths are acceptable if it means the path successfully aligned with the dynamic properties of the agent and helped the agent achieve its goals.

We then augmented A* to account for two additional components: internal states of the agent and state space nodes, and the presence of secondary goals in the state space.

1) Internal States: Our algorithm simulates internal motivations of agents through the use of internal states. A simulation using our algorithm can identify an n-ary list of internal states. These internal states are float values that can represent the mood or motivations of the agent present in the simulation. Both the agent and each state space node will have their own version of this list of internal states.

The agent uses these lists of internal states to help it decide which path to take in the space. It will consider the internal states of its neighbors, and prioritize neighbors that have similar internal states to itself. This behavior allows the agent to pick the path that best aligns with its current motivations.

Additionally, as the agent moves around the space it can be affected by the environment. Each node has the potential to change the agent's internal state, which is impacting the agent's motivations as it navigates the space. These changes in motivation can potentially lead to the agent changing how it wants to path through space.

2) Secondary Goals: A simulation running our algorithm will start with a list of possible secondary goals. These goals are nodes that the agent *may* want to reach on its way to the primary goal. The agent has a vision range indicating how far away from its current position it can see secondary goals. When a secondary goal is within vision range, the agent will assess whether that secondary goal is worth moving to. This assessment is done using the agent's list of state difference thresholds. The list of state difference thresholds indicates how much difference between the internal states of the agent and secondary goal that the agent is willing to accept. For example, consider an agent with an internal state list of $[10, 10, 10]$ and state difference thresholds of $[1, 1, 1]$. This implies the agent will only consider secondary goals where all 3 internal states are in the range of 9 to 11.

When the agent finds a secondary goal within its vision range that satisfies its internal state requirements, then it will attempt to move down a path that detours to the secondary goal on the way to the primary goal.

B. Algorithms

Our modified A* algorithm functions the same as a traditional A* algorithm, except in how it calculates the "Cost-to-Go" (G-Cost) value. Our method of calculating G-Cost is shown in Algorithm 1. The algorithm looks at the state list for both the agent and the potential neighboring node. For each state, it gets the difference between the agent and node's

version of that state then adds it to a cumulative weight. After summing up all these differences, the algorithm scales the cost to have equal bounds as the heuristic H-Cost then returns the value. The heuristic H-Cost is calculated through a Manhattan distance calculation.

The modified A* algorithm is used by the agent when it is trying to find a potential path from its current position to the primary goal. This process is represented by Algorithm 2. This algorithm is run whenever the agent needs to reevaluate which path to take to the goal. First, the agent looks for secondary goals within its vision range. Any viable secondary goals are compiled into a list. Viable secondary goals are defined as secondary goals that are within vision range and have internal states within the agent's state thresholds. The agent then loops through the compiled secondary goals in order of their G-Cost. If it finds that the primary goal is closer than the best secondary goal, the agent will return a path straight to the primary goal. If the agent finds a secondary goal that it can successfully walk to and still reach the primary goal, it will return a path that detours through that secondary goal on the way to the primary goal. If no reachable secondary goals are found, then the algorithm returns a path straight to the primary goal.

The high-level navigation of the space is represented by Algorithm 3. The agent decides the best possible path through the space using Algorithm 2, then takes a step along that path. After taking this step, the agent evaluates how its internal state was changed due to taking that action. The agent then finds a new best possible path through the space based on its new location and state, and repeats the process until it reaches the primary goal.

Function: getGCost
Input : Agent State List (A_S), Neighbor State List (N_S)
Output : The "Cost to Go" from agent's current position to the neighbor. Used by A*
$weight \leftarrow 0$;
for $i \leftarrow 0$ **to** $A_S.size$ **do**
  $weight \leftarrow weight + |A_S[i] - N_S[i]|$;
end
$weight \leftarrow weight * scaling\_factor$;
return $weight$

Algorithm 1: Get "Cost to Go" Value

Function: findPath
Input : Agent Position (A_p), Primary Goal (p_goal)
Output : The path from the agent's position to the primary goal
$viable\_secondary\_goals \leftarrow getSecondaryGoals()$;
for $s\_goal \in viable\_secondary\_goals$ **do**
  if $s\_goal.h\_cost < p\_goal.h\_cost$ **then**
    return $aStar(A_p, p\_goal)$
  end
  $path\_to\_s\_goal \leftarrow aStar(A_p, s\_goal)$;
  $remaining\_path \leftarrow aStar(s\_goal, p\_goal)$;
  if $path\_to\_s\_goal.size \neq 0$ **and** $remaining\_path.size \neq 0$ **then**
    return $path\_to\_s\_goal \cup remaining\_path$
  end
end
return $aStar(A_p, p\_goal)$

Algorithm 2: Find a Potential Path

Function: navigateSpace
Input : Agent (A), State Space (SS) Primary Goal (p_goal)
Output : The finalized path the agent took through the space
$finalPath \leftarrow [A.position]$;
while $A.position \neq p\_goal$ **do**
  $potentialPath \leftarrow findPath(A.position, p\_goal)$;
  $nextPosition \leftarrow potentialPath[1]$;
  $A.state = A.state + nextPosition.changeValues$;
  $A.position = nextPosition$;
  $finalPath.append(nextPosition)$;
end
return $finalPath$

Algorithm 3: Navigate a State Space

C. Example

The general steps from the algorithms we just outlined can be visualized in Appendix A: Map Traversal Example which walks through the steps associated with traversing an 8x8 grid-space graph with two potential secondary goals and one primary goal. This example was executed by the algorithm described above and the resulting traversal shows the effectiveness of the algorithm at allowing the agent to dynamically respond to updates in its internal state.

V. RESULTS

A. Experiments

In our experimental design we wanted to keep the implementation generic enough such that it was still applicable to the broader idea at hand, but specific enough that any reader may easily understand how to implement the algorithm themselves.

In this design certain parameters were used to generate the test grid space graphs for traversal. The exact fixed parameters for this experiment are detailed below:

- Agent/Node State Dimensions: 3
- Secondary Goal Density: 5%
- Obstacle Density: 12%
- Agent Vision Distance: 5 spaces
- Min/Max Agent states: 0/10
- Secondary State Threshold Values: random value between 0 and 7

All three node dimensions were scaled linearly. If each node on the grid space is described by its Cartesian coordinates, then the node states for node at position (x,y),

$$s_{x,y} = \begin{bmatrix} x & y & \frac{x+y}{2} \end{bmatrix} \tag{1}$$

In this sense the planner is correlating the Agent states with 3 different ramp functions. The goal state is always located at the peak of the ramps and start state located at the lowest point. For instance, for a 10 by 10 grid space, the start state would be node (0,0) and the end state at (9,9). This ensures consistency across tests.

Since the Agent is always traversing towards the goal state no matter where it is, the goal state does not necessarily need to be located at some global maximum. It was done in this case simply to create an easy to visualize setup. The only real difference is the path taken chosen by the Agent due to it's dynamic internal state.

Each test was run three times and resulting data averaged together. All random variables were drawn from uniform distributions, except some small noise, $noise \sim \mathcal{N}(0, \frac{1}{2MaxWeight})$, added to the node change values to ensure less predictability in the Agent.

The choice of three state dimension was arbitrary chosen to illustrate a fill implementation.

B. Evaluation

Our algorithm implementation was tested using the following objective metrics:

1) Total run time (from start state to primary goal)
2) # of Secondary goals reached
3) Maximum A^* search time. This is the time it takes A^* to find the goal from it's current position.

Table I shows results collected from the experiments. The labels in the first row of the table are the respective grid size for the results in the column below it. It is important to note that the grid sizing are increasing by the square when interpreting this data.

TABLE I: Results of experiments for different grid sizes using modified A^*. n=3

	10x10	20x20	30x30	50x50
Total Time (s)	0.0801	0.1456	0.5506	3.7876
Max A* Time (s)	0.0035	0.0127	0.0232	0.0766
Goals Reached	0.3333	1	3.3333	7.6667

Table II below shows data from the normal operation of A^*. This data simply timed how long it took to find a path from start to finish as in a standard planner. Comparisons will be discussed in the conclusion.

TABLE II: Results of experiments using standard A^*. n=3

	10x10	20x20	30x30	50x50
Max A^* Time (s)	0.0025	0.0109	0.0210	0.0617

As shown in tables I and II, while our proposed algorithm has a much longer total run time its time for new path generation (Max A^* Time (s)) remains the same. This shows that although the entire simulation is running longer, *each step* takes a comparable amount of time. We find this to be an acceptable result because the use case of this planner implies actions being taken by the Agent as it traverses through the graph. Our planner helps the agent choose how to take each step around the space, but the agent's role in the space and the actions it takes interacting with the space will determine the total time it takes to reach the primary goal. Thus, we are less concerned about total run time than we are concerned about the maximum A^* run-time, which we can see is remaining fast and comparable to traditional A*.

Additionally, the stats on the number of goals reached scales alongside the increasing graph size, indicating the algorithm consistently is able to guide the agent along an optimal path regardless of the type of graph. Put these findings together and we can say with confidence that our modified A^* algorithm can very quickly generate an effective path through a space, and allows the agent to continue to find those optimal paths at whatever pace it desires.

VI. CONCLUSIONS

Our team created an algorithm that uses an augmented version of A* to account for the dynamic properties of an agent with internal motivations. The algorithm incorporates internal states of agents and nodes along with the introduction of secondary goals to help create a more realistic human-like path through a space. We have shown that an agent can successfully navigate a space using our algorithm using a path that aligns with its changing motivations while accomplishing goals that also align with those motivations. We have also shown that our algorithm can generate a potential path at acceptable speeds, allowing agents to use these potential paths at whatever pace they choose to move through the space.

The novel introduction of accounting for internal motivations of the agent can allow this project to be applied to a wide array of potential applications. Any scenarios that require a human-like agent can make use of our algorithm to make more realistic paths. For example, robots meant to emulate humans or AIs mimicking human behavior in simulations or video games would likely benefit from the ability to path based on internal motivations. We also believe our algorithm would be able to be applied to situations with dynamic environments, where the nodes in the space are changing and can affect the agent in various unpredictable ways.

There are a number of potential future steps our work in this project. One avenue is to attempt to implement a similar algorithm using different types of path planners. Our group was interested to see if we could get even more human-like behavior through the use of q-learning, however, we did not have time to attempt an implementation. Another avenue we wished to explore was the development of more types of heuristics for our A* implementation. We currently are using Manhattan distance with the assumption the agent is traversing square-space, however, we are interested in learning how to apply the algorithm to more types of environments such as hexagon-space. Lastly, we are interested in testing our algorithm in more real-world contexts to see how the complexities we have theorized about actually behave in an real-life.

REFERENCES

[1] C. Chen, X.-Q. Chen, F. Ma, X.-J. Zeng, and J. Wang, "A knowledge-free path planning approach for smart ships based on reinforcement learning," Ocean Engineering, vol. 189, p. 106299, Oct. 2019, doi: 10.1016/j.oceaneng.2019.106299.

[2] Neufeld, X., Mostaghim, S., Sancho-Pradel, D. & Brand, S. Building a Planner: A Survey of Planning Systems Used in Commercial Video Games. *IEEE Transactions On Games*. **11**, 91-108 (2019)

[3] Nau, D. Current Trends in Automated Planning. *AI Magazine*. **28**, 43 (2007), https://ojs.aaai.org/index.php/aimagazine/article/view/2067

[4] Tencé, F., Buche, C., Loor, P. & Marc, O. The Challenge of Believability in Video Games: Definitions, Agents Models and Imitation Learning. *CoRR*. **abs/1009.0451** (2010), http://arxiv.org/abs/1009.0451

VII. Appendix A: Map Traversal Example

Figure 1 shows the initial setup of the 2-D 8x8 grid space problem for the agent to navigate. The agent, seen shown in blue, starts at position (0,0). The agent's primary goal is shown in green at position (7,7). Additionally, this planning example has two secondary goals present shown in yellow in positions (0,4) and (4,7), as well as several impassable nodes which are displayed with hashes overlaid on top of them.

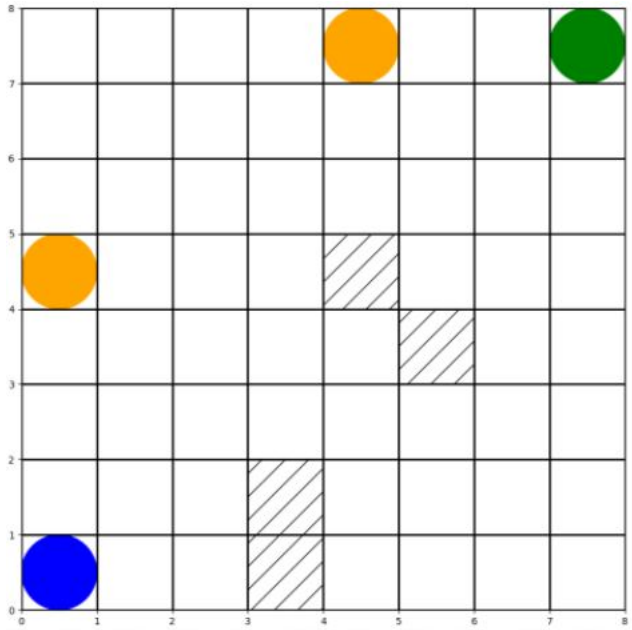

Fig. 1: Initial Navigation Setup

Figure 2 shows the initial path generated by the agent from its current position i.e. its starting state to the primary goal. When generating this path the agent considers three main things which will remain relevant throughout all subsequent path generations as the agent navigates the space. These are:

1) Vision Range
2) Internal State of the Nodes on the Path
3) Internal State of Secondary Goals

For this example the agent's vision range is 5 nodes. This means that when it generates its path to the primary goal it searches all nodes with a Manhattan Distance of 5 or less from its current location. One of the two secondary goals are within the agent's "vision" and therefore might be chosen for pursuit by the agent.

The other determining agent for whether or not a secondary goal is pursued by the agent is the internal state of the secondary goal in comparison to its own internal state. The agent's internal state in this example, at this location, is $[4.115, 6.468, 8.159]$ and its predetermined state threshold is $[1.823, 1.294, 2.632]$. This means that the agent will only chose to pursue a secondary goal if its internal state falls within the pursuit threshold that can be constructed using the current state of the agent as well as the agent's state threshold.

The upper and lower pursuit bounds at this location, for this example, are defined respectively as $[5.938, 7.762, 10.791]$, $[2.292, 5.174, 5.527]$ while the internal state of the secondary goal within the agent's vision range is $[0.0, 5.714, 2.857]$ since the internal state vector does not fall within the agent's pursuit bounds it will not be added to the agent's planned path for this iteration.

The internal state of neighboring nodes are then considered as the agent finds which nodes it can generate a path with that best matches its internal state. For example in the agent's current position the first two nodes it must chose between as it routes to the primary goal are nodes $[0, 1]$, and $[1, 0]$. These nodes have the following internal states respectively, $[0.0, 1.429, 0.715]$, and $[1.429, 0.0, 0.715]$ since node $(0, 1)$ has an internal state closer to the internal state of the agent it was added to the path. Using A* this method is continuously used to route a path to the primary goal during each iteration.

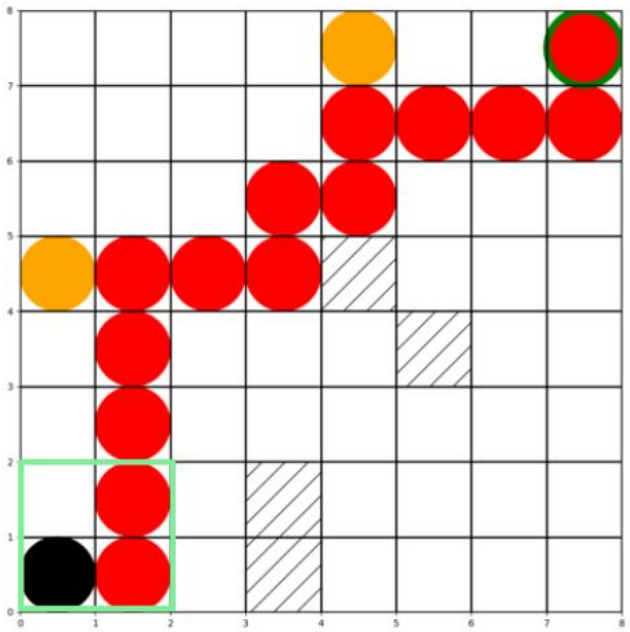

Fig. 2: First Step Path Generation

Figure 3 shows the agent after it has traversed the same example graph for three more steps. Though the agent is getting closer to the first secondary goal $(0, 4)$, and now within vision range of the second secondary goal $4, 7$ it has not generated a path through either secondary goal because neither goal's internal states falls within the pursuit threshold of the agent. Thus the agent continues to head straight to the primary goal.

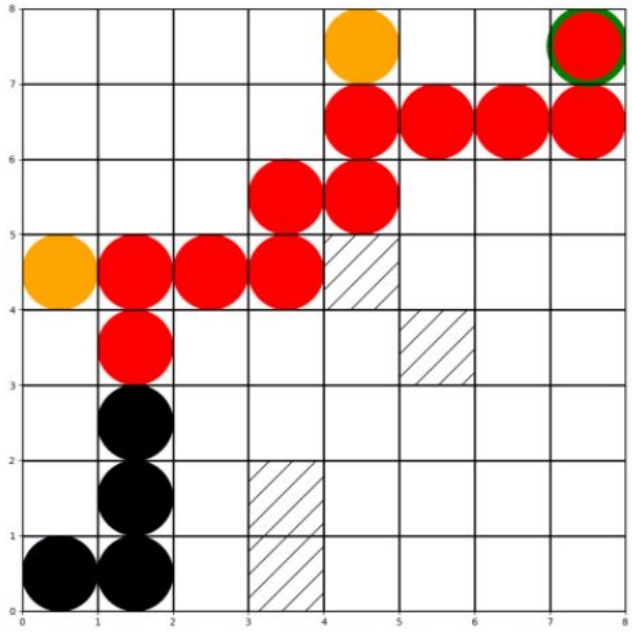

Fig. 3: Second Step Path Generation

Figure 4 continues to demonstrate the importance the pursuit bounds in determining which states to pursue. Even now as the generated path to the primary goal has updated and the agent is adjacent to the secondary goal $(0, 4)$. The agent does not add the goal to the planned path as its internal state does not fall within the required pursuit bounds.

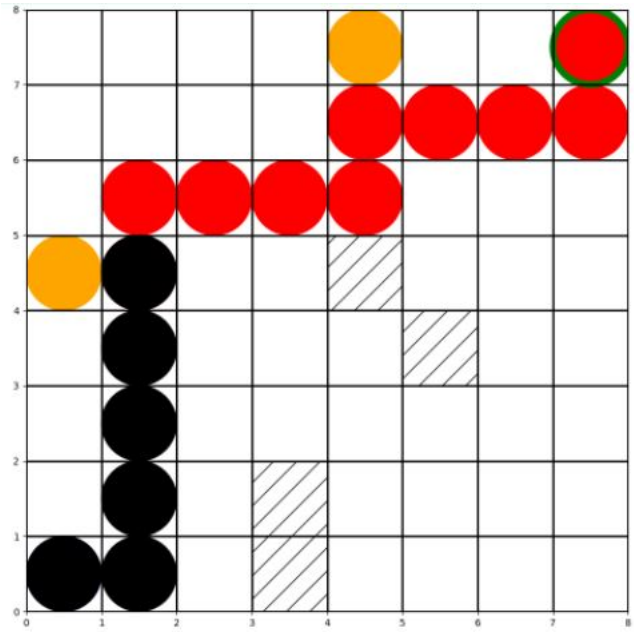

Fig. 4: Third Step Path Generation - Passing a Secondary Goal

Figure 5 shows how the agent passed by the secondary goal at $(0, 4)$ in pursuit of the primary goal as it was not desirable.

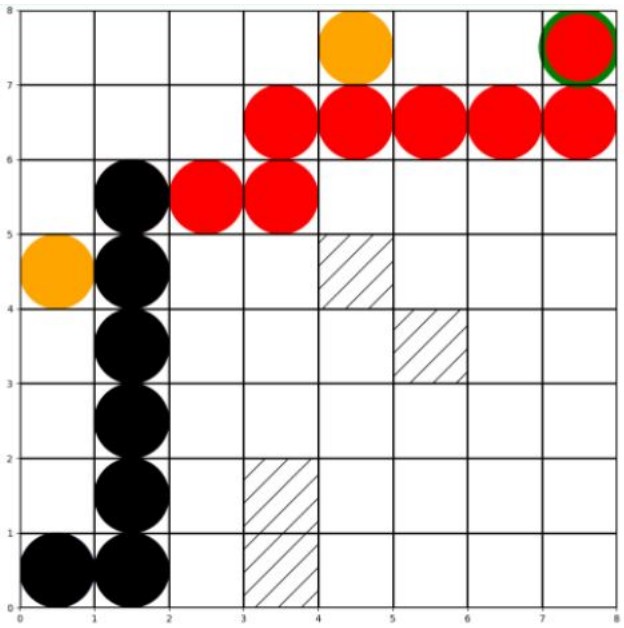

Fig. 5: Fourth Step Path Generation

Figure 6 shows how the agent added the secondary goal at $(4, 7)$ to its planned path because its internal state fell with in the pursuit bounds defined by the current agent state as well as the state threshold. The current agent state has updated to be $[5.313, 9.047, 9.935]$ and the state threshold remains the same as before (since it is constant throughout the path planning problem). This means that the agent's new upper and lower pursuit bounds were $[7.136, 10.341, 12.567]$, and $[3.49, 7.753, 7.303]$, respectively. The secondary goal's internal state was $[5.714, 10.0, 7.857]$ which is within the bounds and thus is added to the agent's path.

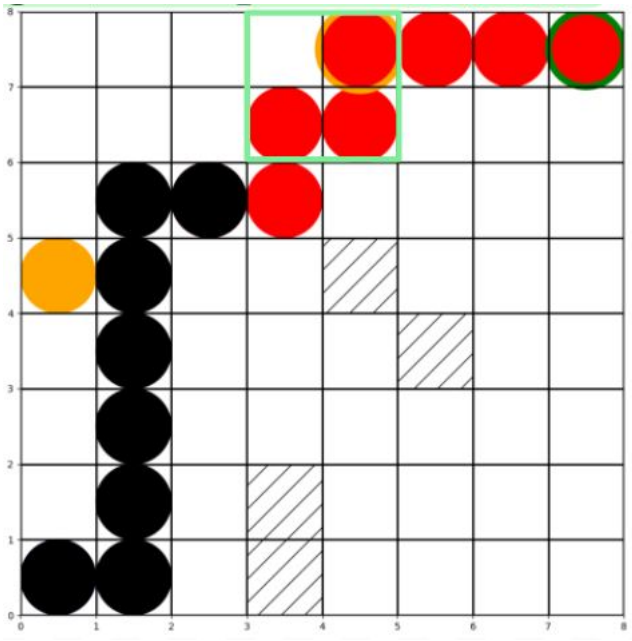

Fig. 6: Fifth Step Path Generation - routing through a secondary goal

Figure 7 shows the agent moving towards the primary goal with its path still including the secondary goal at $(4, 7)$.

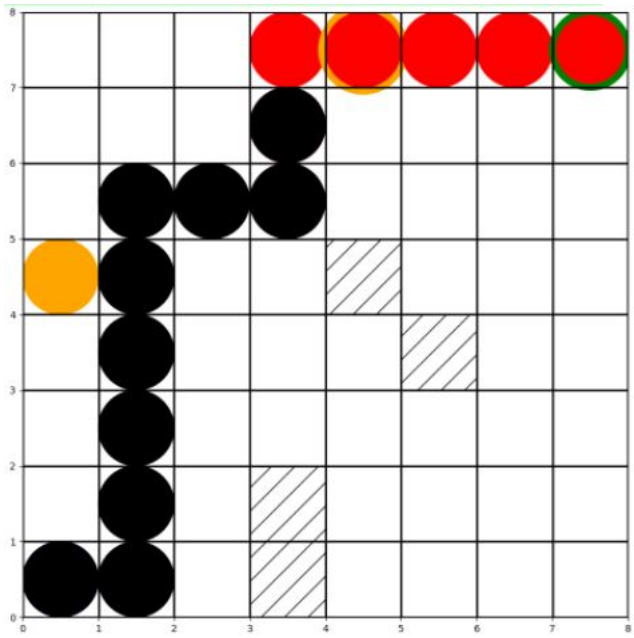

Fig. 7: Sixth Step Path Generation

Figure 8 shows the agent achieving the secondary goal located at $(4, 7)$ and continuing to move towards the secondary goal.

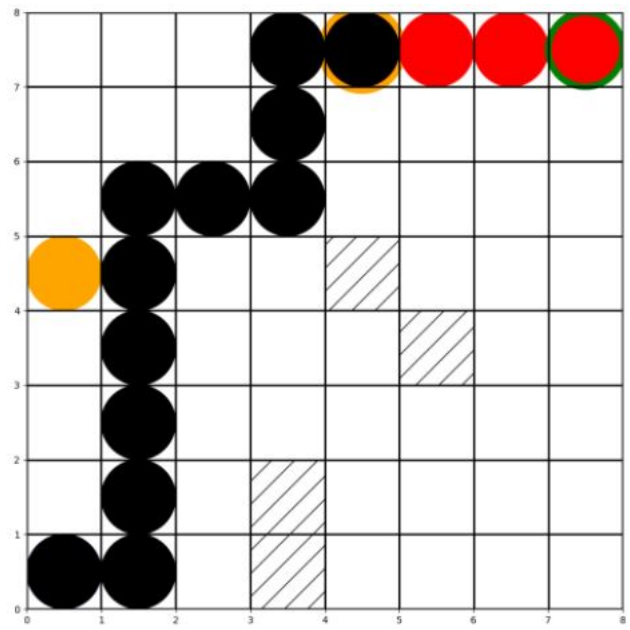

Fig. 8: Seventh Step Path Generation - Achieving a Secondary Goal

Figure 9 shows the agent achieving the primary goal and accordingly ending its path planning. This 8x8 example took 0.0115456s to plan and ended up traversing 15 nodes as well as achieving two of the three available goals (one primary goal and one of the two secondary goals).

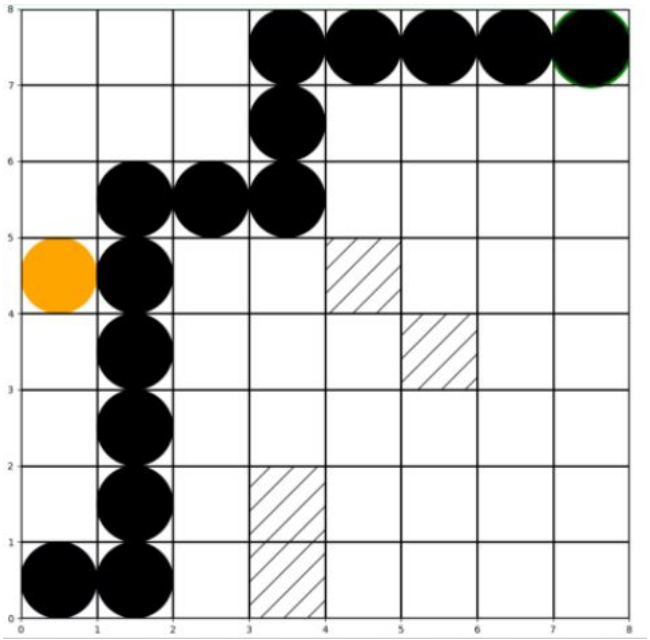

Fig. 9: Final Step - Achieving the Primary Goal