Project III Report for COM S 4720 Spring 2025: Pursue-Escape Planning Algorithm Development with Uncertainty

Tanvi Mehetre[1] and Tejal Devshetwar[2]

I. INTRODUCTION

Project III extends Project II which was a complex multi-agent setting involving pursuit and evasion among three agents namely: Tom, Jerry and Spike by introducing probabilistic transitions into the pursue-escape planning environment. The agent roles do not change and are follows:

- Tom is trying to catch Jerry
- Jerry is trying to catch Spike
- Spike is trying to catch Tom

The key difference that we have built upon is the uncertainty. Each agent's intended actions can be altered according to the dynamic probability values $[p_1, p_2, p_3]$, which define the chances of the selected action being rotated left by 90°, executing it in the same place as now, or rotating it right by 90°, respectively. This requires the agent to plan their moves as well as account for the probabilistic deviations.

II. IMPLEMENTATION

A. System Overview

We implemented a custom planner that is both fast and responsive to the uncertainty present in each task. We changed our approach from project II where we used a Minimax approach, but here we apply a Heuristic Greedy Planner with Uncertainty Awareness.

B. Key Components

1) Probability Management:

- The planner maintains a dynamic probability vector $[p_1, p_2, p_3]$ where:
  - p_1: Probability of rotating the chosen action by the agent left by 90°.
  - p_2: Probability of executing the intended action by the agent without modification.
  - p_3: Probability of rotating the chosen action by the agent right by 90°.
- These probabilities are updated before each move through the `update_probabilities()` method, which receives the latest values directly.

2) Action Space Definition:

- The agent considers 9 possible actions:
  - Moving in 8 directions: North, South, East, West, and the four diagonals. The 8 directions are stored in the arrays as follows [-1, 0] is North, [1, 0] is South, [0, -1] is West, [0, 1] is East, [-1, -1], [-1, 1], [1, -1], [1, 1] are the diagonals.
  - Staying in the current position [0, 0] is staying still.
- These are represented using numpy arrays for efficient mathematical operations.

3) Uncertainty Handling and Expected Action Computation:

- For each possible action, the planner computes an expected action vector to model the effects of uncertainty:

$$\text{Expected Action} = p_1 \cdot \text{Left}(a) + p_2 \cdot a + p_3 \cdot \text{Right}(a) \tag{1}$$

- Here, a is the original action vector, and the `Left()` and `Right()` functions apply a 90° rotation using simple linear transformations:
  - `Left(a)`: $[-a_y, a_x]$
  - `Right(a)`: $[a_y, -a_x]$
- The expected result vector is rounded to the nearest integer to align with the grid that we are placed in.

4) Heuristic Evaluation Function:

- For every candidate move, a heuristic score is computed to aid with the decision making for the move. The heuristic function encourages closing in on the pursued agent while avoiding the pursuer. The score calculation is given below:

$$\text{Score} = -d_{\text{target}} + 2.0 \cdot d_{\text{threat}} \tag{2}$$

- Where:
  - d_{target}: Euclidean distance to the pursued agent.
  - d_{threat}: Euclidean distance from the pursuer.
- The weighting factor of 2.0 that has been used by us amplifies the importance of evasion, ensuring the agent favors safety unless a highly rewarding pursuit opportunity is present.
- This heuristic that has been used here effectively balances the risks and rewards, promoting strategic behavior.

5) Valid Move Filtering:

- Before moving onto the next move, the planner verifies the validity of the move using the `_is_valid()` method:

[1]Tanvi Mehetre is a student in the Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, United States tanvim@iastate.edu

[2]Tejal Devshetwar is a student in the Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, United States tejal@iastate.edu

- Checks if the move violates any boundaries and does not overstep.
- Ensures that the next move to be taken does not result in a collision with obstacles (cells where `world[r, c] != 0`).

6) Action Selection Process:

1) Retrieve the latest dynamic probabilities $[p_1, p_2, p_3]$.
2) For each possible action the following is to be done:
   - Computation of the expected action under uncertainty is conducted.
   - Validating the next resuting position is done.
   - If the validation is successful, the heuristic score is evaluated.
3) Selection of the action with the highest heuristic score is set as the final move to be made.

This process ensures real-time decision-making that respects uncertainty and keeps the agent competitive.

III. TIME COMPLEXITY

The planner we implemented uses a Heuristic Greedy Algorithm with Uncertainty Awareness. Instead of using a multi step lookahead search we based our decisions on immediate heuristic evaluations.

A. Per-Move Analysis

- **Action Evaluation Loop:** 9 actions (8 for movement and one for staying in place) are evaluated by the planner.
- **Expected Action Calculation:** The time complexity is $O(1)$ as simple arithmetic operations are performed in the _expected_action method. For each action it performs constant-time vector rotations and weighted averaging based on dynamic probabilities given to us.
- **Validity Check:** The _is_valid method performs a check on the boundary and an obstacle check. This is done using a single array lookup. This also runs in $O(1)$ time.
- **Heuristic Evaluation:** The _heuristic method computes Euclidean distances between two points. This method involves basic arithmetic and square root calculations which also run in constant time.

The algorithm evaluates a fixed number of actions and each one involves a constant amount of computation. The total per-move time complexity is:

$$\mathcal{O}(9 \times 1) = \mathcal{O}(1)$$

B. Scalability

The algorithm's runtime is independent of grid size and remains constant per decision. This makes it highly scalable and reliable under different environment complexities. This enables the planner to meet the strict time constraints for each game while maintaining responsiveness.

IV. CONCLUSIONS

In this project, we implemented an effective heuristic planner that handles probabilistic action outcomes through expected value modeling. This method is fast and is suitable for tight runtime constraints while delivering results.

The addition of uncertainty to the planner's decision-making process shows how simple greedy strategies can be enhanced to perform well even in random environments. We were able to build an agent that adapts dynamically to the and performs consistently by prioritizing low computational cost and robust decision logic.

REFERENCES

"Greedy Algorithm." GeeksforGeeks, GeeksforGeeks, 12 Feb 2025, www.geeksforgeeks.org/greedy-algorithms-set-1-introduction/.

"Uncertainty Quantification in AI." Towards Data Science, 5 March 2025, www.towardsdatascience.com/uncertainty-quantification-in-ai-2025/.