Multi Penalty Algorithm for Local Predator Prey Planning

1st Karan Sutradhar
Robotics Program
A. James Clark school of Engineering
College Park, Maryland

2nd Cristian Bueno
Robotics Program
A. James Clark school of Engineering
College Park, Maryland

3rd Brain Matejevich
Robotics Program
A. James Clark school of Engineering
College Park, Maryland

***Abstract* - This paper focuses on implementing an algorithm like the A* and the potential field algorithm in a predator-prey scenario. This algorithm determines the best cost function for prey attempting to escape a predator. The proposed cost function balances the cost of being close to the predator with the cost of being far from its goal. The more the steps taken by the predator to catch the prey, the smarter the prey is deemed to be. The results were very consistent with twelve combinations of different weights tried at three different predator speeds.**

Keywords - predator, prey, cost function, A* & node

I. Introduction

In this project, we will attempt to implement a simple pseudo potential field method for both the predator and the prey. We plan to do this by assigning a “cost for prey” and “cost for predator” to each node. “Cost for predator” will be initialized to a high number and will be replaced by a low number if visited by the prey. The predator will take the path with the least cost, taking into consideration movement cost (1 or √2), a heuristic for distance to prey, and “cost for predator”.

Additionally, between each time step the entire set of nodes’ “cost for predator " will be increased so that nodes visited more recently by the prey take preference. A similar process will follow for “cost for prey”. All node’s “cost for prey " will initialize to 0 and when visited by a predator is set to a high number. Before each time step the entire set of nodes’ “cost for prey” will decrease by 1 with a floor of zero.

The prey will follow a cost function considering movement cost, “cost for prey”, cost to predator and cost to goal (center). This is very similar to potential field methods where there is an attractive force assigned to goals and a repulsive force assigned to obstacles.

The above method will give prey that attempts to both avoid the predator and be as close to the center as possible. The predator will take the shortest path to the current location of the prey. We will explore improving the prey’s cost function by tuning the weights for cost to goal and cost to pred. Considering the distance to the prey will give a path like A*. Once we add “cost for predator” to the function the predator should follow the path the prey has taken.

II. Ease of Use

A. Definitions

- Agent - an actor in the scene which will execute action given to it by a path planner
- Node - represents a possible configuration of an agent
- Discrete - finite or countably infinite
- Cost function - a function that represents the cost of an action or set of actions taken
- Potential field - is any physical field that obeys Laplace’s equation, the magnetic field

- Predator - Agent which is trying to catch the prey agent
- Prey - Agent attempting to avoid predator agent

III. Background

The A* algorithm is considered as one of the best-known path planning algorithms, which can be used in many different environments. The A* algorithm combines a heuristic searching criterion and searching based on the shortest path criteria to find not only a feasible solution but also an optimal path [5]. Every possible move on a discrete cell or map is evaluated using these considerations and returns a different cost to visit each cell in the space of the map. Later, these values determine which path the robot will follow, choosing the path with the lowest cost.

By itself, searching for the optimal path towards a static goal can be arduous, now imagine searching for a goal that constantly moves. For this reason, a predator-prey application can be considered as one of the most challenging applications for path planning algorithms. Computing a feasible path can be both time and CPU intensive, furthermore, if the predator needs to anticipate the possible trajectory of the prey multiple time steps in advance. This means that for every time step the predator must recalculate the trajectory towards its prey. “Off-line and incremental path planning algorithms are not able to handle moving targets in real-time, and most of the on-line search algorithms are specifically designed for partially observable environments with static targets” [1].

Many predator-prey research papers focus on making the predator or predators smarter. Some research has been done to implement teams of predators and develop their teamwork in capturing prey [1,3]. Other research provides abilities to some predators in multi-agent teams to see if having heterogeneous teams with different abilities is of benefit [2]. Finally, there is research-based on all existing methods to make them more efficient [4].

IV. Methodology

During our implementation the following assumptions were made; Both the predator and prey agents are considered point robots capable of localizing within the map, neither the predator nor the prey robot has any dynamic constraints in their movement, the path generated will be explored in 8-action space (Fig. 1).

Given two different random starting points, one for the prey and the other for the predator, both agents will have knowledge of the other’s location. Our algorithm dictates the best movement for the predator to hunt the prey while giving the prey the best move to escape capture. Utilizing the cost function, the predator will choose the optimal path towards the prey.

The cost function for the predator will take multiple inputs including; action cost, “cost for predator” which is low in nodes recently visited by the prey, and a distance function to the prey. The prey will have a similar cost function including, action cost, a distance function for both the predator and a goal (center of the map), and cost for prey.

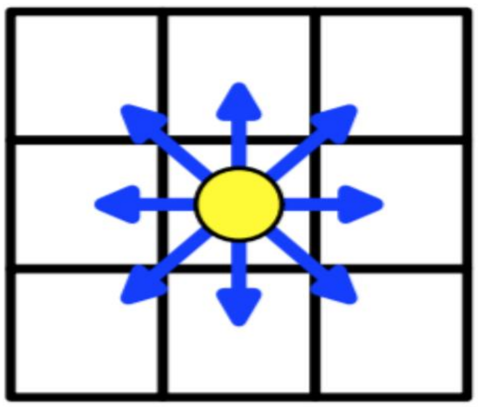

Fig. 1 Display of both the predator and prey's 8 action space for movement. Up, down, left and right have a cost of one while the diagonal motions have a cost of √2.

Using these assumptions, the following steps were followed in our implementation. First, we created a map on 200x200 to place the agents in. Then we define the 8-action set which will be used for both predator and prey. We implemented a heuristic to be used in both the predator and prey cost functions, the distance calculation used was Manhattan distance. We used this heuristic in weighted A* style for distance to prey, distance to predator, and distance to the goal.

Next, we create separate cost functions for predator and prey. Finally, we wrote a function to factor in each timestep. This function updates both the 'cost for predator' and "cost for prey" maps so they are time dependent.

Additionally, using the base of A*'s heuristic function, it will be easy to implement in our situation. For the predator and prey problem, a few extra considerations have been added to the overall cost function. The prey's cost will be highest in nodes that the predator has recently visited while the predator's cost will be lowest in places the prey has most recently visited.

Due to A* being computationally intensive (exploring all nodes) we have imposed a limit on the length of the path. Between each time step, the algorithm will search for the lowest cost node within one step of the current location. This is an extremely local path planner, only planning one step ahead. After this node is chosen for both the predator and prey the agents will execute one action along the path. Depending on the relative speed of the predator it will plan and execute actions at a faster rate than the prey. We tested equal speeds, twice prey speed and three times prey speed. To simulate three times prey speed the prey only takes a step for every three predator steps.

As mentioned above the cost functions will be different for the predator and prey as will the way each interacts with the map. The predator will be drawn to nodes recently visited by the prey while the prey will avoid nodes recently visited by the predator. Nodes are initialized with "cost for predator" at 255 while nodes recently visited by the prey have their "cost for predator" set to 155. Similarly, nodes are initialized with "cost for prey" to be one and are set to 100 when visited by the predator. After each timestep, the entire cost for the predator map is increased by 1 and the cost for prey map is decreased by 1. This low cost will be an attractive force like the potential field method. These costs will act as attractive and repulsive forces like potential field methods.

$$\boldsymbol{EQ.1 = cost_{action} + cost_{for\ pred} + cost_{to\ prey}}$$
$$\boldsymbol{EQ.2 = cost_{action} + cost_{for\ prey} + * cost_{to\ pred} + w_c * cost_{to\ goal}}$$

EQ.1 is the cost function of the predator agent while line EQ.2 is the cost function for the prey agent. Wp is the weight given to distance to the predator and W_c is the weight given to distance to the goal. Note W_p is negative.

In search of the "smartest" prey cost function, we tested 12 combinations for W_p and W_c against a constant predator cost function. For each of the 12 weight combinations, the number of steps taken before each capture was recorded and averaged over 15 trials at three different predator speeds. The list of W_c include 0.5,1.0 and 1.5 while the list of Wp explored include 0.5,1.0,1.5 and 2.0. This set of weights is interesting because there are multiple combinations where W_c and Wp are equal and all of these weights are small enough to still give preference to “cost for prey/predator” at close range.

V. Results

The first set of tests conducted assigned equal speeds to the predator and prey agents. This means that for every iteration both the predator and the prey agent move one step. Two trends are immediately apparent in Fig. 2, as the weight assigned to cost to the center is increased the time it takes to capture the prey is reduced and as the weight assigned to cost to predator is raised the time to capture also increases.

Number of Iterations				
	W_p = 0.5	W_p = 1.0	W_p = 1.5	W_p = 2.0
W_c = 0.5	88.93	252.8	323.2	391.07
W_c = 1.0	67.87	115.33	269.27	299
W_c = 1.5	66.67	85.6	102.47	264.47

Fig. 2 Chart of the average number of iterations before capture for 15 trials. W_c is weight for distance to center and W_p is weight for distance to the predator. This chart is for predator and prey with same speed.

The same tests of 12 combinations of weights were conducted with the predator having twice the speed of the prey. This means the prey moved one step every other iteration. The same trends as above are seen again in Fig. 3.

Number of Iterations				
	W_p = 0.5	W_p = 1.0	W_p = 1.5	W_p = 2.0
W_c = 0.5	77.93	103.13	117.93	144.73
W_c = 1.0	64.33	94.87	108.07	131.67
W_c = 1.5	63.33	83.4	99.2	106.6

Fig. 3 This chart is for 12 combinations of predator and goal weights for a predator with twice the speed of the prey.

Finally, all 12 combinations were tested while giving the predator a three to one-speed advantage over the prey. The same two trends continue in this dataset, as Wp increases so do time to capture, and while W_c increases time to capture decreases. This can be seen in Fig. 4.

Number of Iterations				
	Wp = 0.5	Wp = 1.0	Wp = 1.5	Wp = 2.0
Wc = 0.5	80.33	99.93	111.6	133.07
Wc = 1.0	79.67	91.87	103.27	127.47
Wc = 1.5	74.67	80.53	89.8	97.53

Fig. 4 This chart shows the 12 weight combinations for a predator three times as fast as the prey.

There are a few other observations to be made both from the data as well as from viewing the simulations. Another observation in the data is that time to capture is not decreasing for W_p = 0.5 from one to three speed advantage. This is due to the prey giving little weight to the proximity of the predator while the prey is attracted to the center of the map where it can be caught. On average the predator started between 63 and 89 steps from the center point while the prey did little to avoid the predator.

The prey exhibited a few distinct emergent behaviors which could be observed through our experimentation. The first of which was when at equal speed to the predator the prey was always caught in a corner of the map or within a few nodes of the center of the map. When the predator had a speed advantage, the prey could be caught anywhere within the map. This behavior can be seen in Fig. 5.

Before being caught the prey typically displayed one of two patterns; x-shaped and spiral diamond. The x-shaped pattern consisted of avoiding the predator by going to one corner of the map, narrowly escaping in that corner, and moving towards the center, then escaping to a new corner. This process was repeated until the predator had gained enough ground to catch the prey in either the corner of the center. This pattern can be seen in Fig. 6.

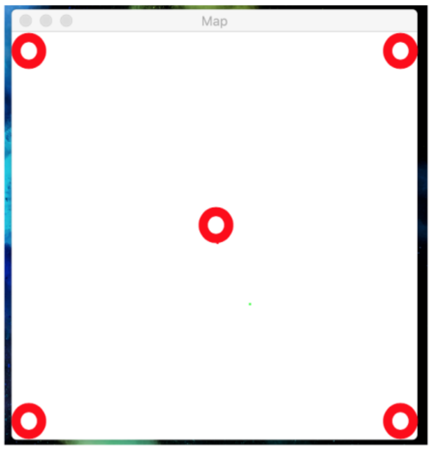

Fig. 5 Display of the common areas of capture when predator and prey have the same speed.

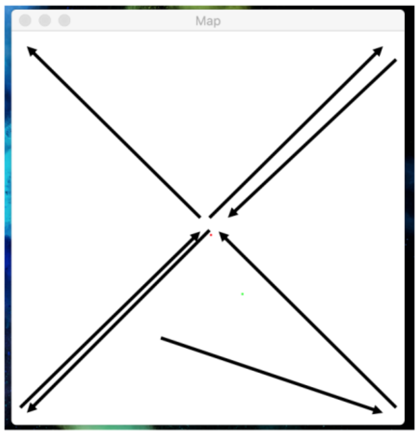

Fig. 6 This image displays an example of the x shaped path commonly taken by the prey.

The spiral diamond pattern is like the old DVD screensaver, bouncing around the map from one edge to another in a diamond shape. Eventually, the prey begins to spiral into the center before being caught near the center node. The spiral diamond motion can be seen in Fig. 7.

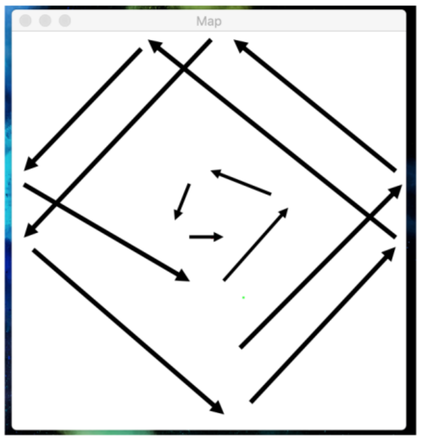

Fig. 7 This is an example of the prey following- the spiral diamond shape finally being captured at the center of the map.

One behavior of the predator that was observed was that unless making a timesaving move, like cutting a corner, the predator always followed the exact path that the prey had. This can be seen in Fig. 8. This is due to the “cost for predator” being so much lower. A move of this path would need to be much better in “cost to prey” to be chosen over the prey’s path.

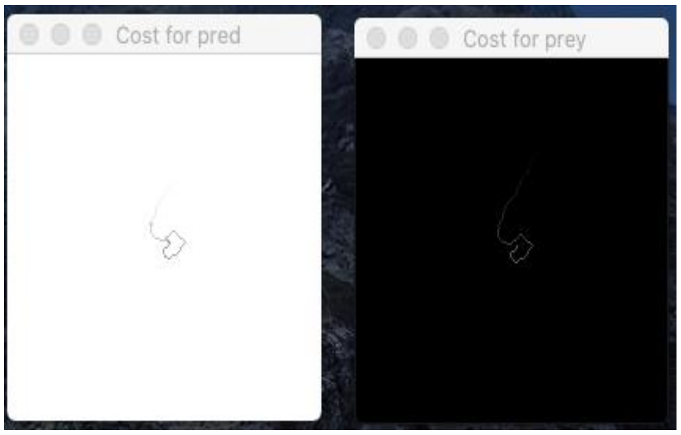

Fig. 8 Here the path shown in the “cost for predator” window is the path of the prey agent and the path in the “cost for prey” window is the path of the predator. The predator follows the exact path of the prey until it can capture the prey.

VI. Conclusion

After building a simple predator prey system we examined multiple weight combinations to create the smartest prey agent. We tested these combinations using predators of three different speeds. The results always followed the same pattern, the more consideration given to the predator location and the less given to the center location the longer it took for the prey to be captured. The "smartest" prey was able to avoid capture for an average of 391 iterations.

These weights introduced a few emergent behaviors including two common movement patterns and limited capture locations. When predators and prey have equal speeds, the capture of the prey can only happen in the corners of near the goal point of the prey (center of the map). This can be seen in Fig. 5. The common movement patterns can be seen in Fig. 6 & 7.

Our system is set up and primed to continue work on increasing the intelligence of both the predator and prey agents. At this point, a combination of A* and potential fields method is the best description of our predator-prey planner.

VII. Future Work

In the future, we plan to explore more weight combinations to see if the pattern continues. We will also consider adding obstacles to the map as well as multiple goals instead of just the center point. Additionally, work could be done in looking to make an even smarter prey by estimating predator direction or planning more steps at a time.

Acknowledgments

We'd like to thank Professor Monfaredi and our great TAs Sanket Goyal and Yashaarth Todi for a great semester. They provided a great environment for learning and adapted to the pandemic very well.

References

[1] Undeger, Cagatay & Polat, Faruk. (2010). Multi-agent real-time pursuit. Autonomous Agents and Multi-Agent Systems. 21. 69-107. 10.1007/s10458-009-9102-0.

[2] Silva, Arlindo & Neves, Ana & Goncalves, Teresa. (2013). Using Scout Particles to Improve a Predator-Prey Optimizer. 7824. pp 130-139. 10.1007/978-3-642-37213-1_14.

[3] Sun, Lijun & Lyu, Chao & Shi, Yuhui. (2020). Cooperative coevolution of real predator robots and virtual robots in the pursuit domain. Applied Soft Computing. 106098.10.1016/j.asoc.2020. 106098.

[4] Baier, Jorge & Botea, Adi & Harabor, Daniel & Hernández, Carlos. (2014). A Fast Algorithm for Catching a Prey Quickly in Known and Partially Known Game Maps. IEEE Transactions on Computational Intelligence and AI in Games. 7. 1-1. 10.1109/TCIAIG.2014.2337889.

[5] Hassan, Mahdi & Liu, Dikai. (2019). PPCPP: A Predator–Prey-Based Approach to Adaptive Coverage Path Planning. IEEE Transactions on Robotics. PP. 1-18.10.1109/TRO.2019.2946891.