Tag on Graphs

Christopher Hoyt and Ryan Humble
Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA, 94305

We investigated a simplified game of tag played out on a square grid. In the game, the player attempts to avoid an adversary (or multiple adversaries) with the additional complications that both the player and the adversary have some chance of faulty movements (dubbed as the jitter probability) and the player is only able to observe the adversary with some probability on any given turn (dubbed as the radar accuracy). We find that the player's survivability increases with larger board size, fewer adversaries, and a larger radar accuracy. However, we find that the survivability is not monotonic with respect to increasing jitter probability.

I. Nomenclature

G	=	game graph
$\mathcal{S}_G$	=	state set of graph G
$\mathcal{A}_G$	=	action set of graph G
$\mathcal{A}_G(s)$	=	actions of graph G available from state $s \in \mathcal{S}_G$
T_G	=	transition function of graph G
P	=	(PO)MDP problem
$\mathcal{S}_P$	=	state space of problem P
$\mathcal{A}_P$	=	action space of problem P
O_P	=	observation space of problem P
R_P	=	reward function of problem P
T_P	=	transition function of problem P
O_P	=	observation function of problem P
γ_P	=	discount factor of problem P
U	=	utility

II. Introduction

We pose the following game: one player is tasked with avoiding some number of (potentially) co-operating adversaries for as long as possible while traveling on a graph G. The game ends when one of the adversaries "tags" the player by occupying the same state $s \in \mathcal{S}_G$ as the player. Broadly speaking, there are two perspectives from which this problem setting can be considered. From the perspective of the player, the researcher attempts to evade the adversaries for as long as possible. From the perspective of the adversaries, the researcher seeks to "tag" the player in the minimal number of turns.

In either case, the resulting problem is one of optimal decision making. The difficulty of finding an optimal strategy for this decision problem can be quite variable though, as this problem setting is particularly expressive. Researchers are free to choose:

1) the type, size, and general characteristics of the graph
2) the number and strategy of the player/adversaries
3) the information available to both the player and the adversaries

Naturally then, this problem, or near variants, has been extensively studied in the decision making community.

A. Related work

Several well-known games are variants of this general problem setting. For example, both Pac-Man and Scotland Yard (and their many sequels) are games in which the player loses whenever an adversary catches them. Fig. 1 shows the game board for Pac-Man. Each board clearly shows the set of nodes $\mathcal{S}_G$ and actions $\mathcal{A}_G(s)$ available from each state $s \in \mathcal{A}_G$ that make up the game. However, each game has additional rules and objectives that depart from the simple

game proposed here.

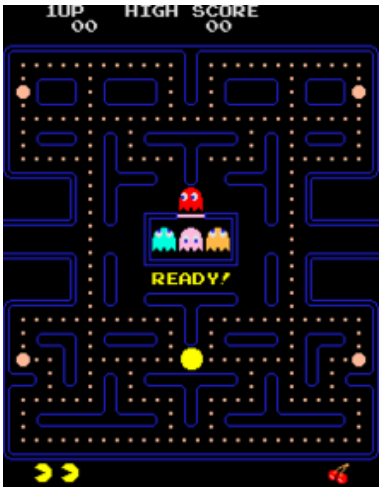

Fig. 1 Pac-Man game graph.

Due to the popularity of Pac-Man, there has been extensive research on finding optimal strategies in that game setting. The survey of the last two decades of academic research into Pac-Man and related variants in [1] categorizes the various approaches taken for defining a Pac-Man controller as follows:

1) Rule based (offline): primarily based on if-then clauses
2) Tree search and Monte Carlo (online): uses tree search techniques
3) Evolutionary algorithms (mostly offline): largely genetic algorithms
4) Artificial neural networks (offline): can be configured as policy networks given just the screen capture
5) Reinforcement learning (offline): largely Q-learning or other forms of temporal difference learning

The setting for almost all of these approaches uses a full-observability model, meaning Pac-Man and the ghosts can always observe each other. [2] explores three different partial observability models, applied to both Pac-Man and the ghosts: (i) line-of-sight, (ii) forward-facing line-of-sight, and (iii) radius-based. The author finds significant scoring differences between the different observability models and the various Pac-Man controllers, but generally the forward-facing line-of-sight model was the hardest to overcome.

III. Model and Methods

In this paper, we will act as the player in our game, attempting to find an optimal strategy to evade our adversaries. In order to find such a strategy, we first explore how to represent tag on graphs as a Markov Decision Process (MDP) or the more general Partially observable Markov Decision Process (POMDP).

A. Game graph

We first define the basic game graph that both the player and the adversaries will play on. To allow for the game to be played on arbitrary graphs, we define a simple interface for a game graph:

- $states_G()$: returns $\mathcal{S}_G$
- $actions_G(s)$: returns $\mathcal{A}_G(s)$
- $d_G(s1, s2)$: returns distance between the states $s1$ and $s2$
- $T_G(s, a)$: returns probability distribution over states reachable from $s \in \mathcal{S}_G$ by taking action $a \in \mathcal{A}_G(s)$

We only created one concrete game graph, `GridGraph`, which was strongly motivated by the `GridWorld` example in [3]. A `GridGraph` of size n defines a nxn square regular grid with 4 actions (left, up, down, right). The distance function is defined to be the Manhattan distance between the two states (plus a small fractional amount of the Euclidean distance to act as a tiebreaker). Lastly, the transition function uses the same jitter mechanism as the `GridWorld` example, meaning the probability of moving in the intended direction is $1 - 3 * jitter$ and probability of moving in the other directions is $jitter$.

B. Adversaries

To limit the scope of the paper, we define adversaries with defined strategies, meaning they are not themselves intelligent agents that are learning an optimal strategy of their own. All of the adversaries we consider follow a prescribed strategy, motivated by that of the ghosts in Pac-Man:

1) Choose a target state $t \in \mathcal{S}_G$
2) Choose the action a that moves the adversary closest to t in expectation. Tie breaks are chosen by a pre-defined ordering of the actions (left, up, down, right).

The adversaries then only differ in how they choose a target t. We define the adversary interface:

- $target(adv, g, pState, pNextAction, advState)$: returns target t from adversary state $advState$ with player state $pState$ and next player action $pNextAction$

Note that we allow the adversary to choose a target based on the player's next action. This allows adversaries to "predict" where the player is going while still respecting the Markov assumption.

Again inspired by the ghosts of Pac-Man, we created 5 adversaries:

- `StochasticAdversary`: chooses a random target state from the set of game stats $\mathcal{S}_G$
- `ChaseAdversary` (like Red Ghost): chooses the player's state as the target
- `AmbushAdversary` (like Pink Ghost): chooses the target as the player's "predicted" state in parameter $lookAhead$ steps by extrapolating the player's next action $pNextAction$
- `FeignAdversary` (like Orange Ghost): chooses the player's state if the player is further away than parameter $feignDist$; otherwise behaves like the wanderer
- `PatrolAdversary`: walks (or returns to) a defined patrol loop (simple cycle in the graph) unless the player is closer than parameter $pursueDist$

C. Tag as a MDP

We now have the building blocks to describe our tag game as a MDP. Specifically, `TagMDP` is defined from a graph G and an array of n adversaries $[adv_1, \ldots, adv_n]$. We define the MDP P as follows:

- $\mathcal{S}_P = \mathcal{S}_G^{n+1}$
- $s \in S_P$ is $(pState, advState_1, \ldots, advState_n)$
- $\mathcal{A}_P = \mathcal{A}_G$
- $R_P(s, a)$: return 1
- $T_P(s, a)$: returns probability distribution built from $T_G(s, a)$ and defined adversary targeting
- γ_P: returns 0.99 (gives perfect score as 100)
- $isterminal_P(s)$: returns $true$ if player is caught

Note that we can easily define a generative MDP version instead of the explicit version described above.

To solve `TagMDP` and get a good strategy for the player, we initially considered value iteration. We suppose that the player can still continue to evade the adversaries from any state that is not terminal and therefore define an initial guess:

$$V^{(0)}(s) = \begin{cases} 0 & isterminal_P(s) \\ \frac{1}{1-\gamma_P} & \text{otherwise} \end{cases} \tag{1}$$

While value iteration was incredibly useful in testing early implementations of the code, we found that it was only feasible for small graphs with 1 or 2 adversaries because the state space grows very quickly with respect to the grid size and the number of adversaries.

Therefore, we turned our attention to online solvers. For this project, we decided to use Monte Carlo Tree Search (MCTS). For MCTS, we estimate the value at the leaves of the tree with a simple function instead of with rollouts of a random policy. Specifically, we estimate:

$$\tilde{V}(s) = V^{(0)}(s) \cdot \min\left\{1, \frac{\min_i d_G(pState, advState_i)}{threshold}\right\} \tag{2}$$

This encodes an incentive to stay further away away from the adversaries even as we explore only a limited depth of the MCTS. For the implementation, we leveraged the Julia package `MCTS.jl`.

D. Tag as a POMDP

We extend this to a POMDP, `TagPOMDP`, where the player has partial observability of the adversaries, regulated by the parameter $radarAccuracy$. Specifically, we consider a stochastic observation model where the adversaries' states are observed perfectly with probability $radarAccuracy$ and completely unobserved with probability $1 - radarAccuracy$. Thus we additionally define:

- $O_P = \mathcal{S}_P \cup \{(s, none) : s \in \mathcal{S}_G\}$
- $O_P(sp)$: returns sp with probability $radarAccuracy$; otherwise, return $(pState, none)$ for $sp = (pState, \dots)$

We solve `TagPOMDP` using the online Partially Observable Monte Carlo Planning (POMCP) algorithm first introduced in [4] and implemented in the Julia package `BasicPOMCP.jl`. The solver "combines a Monte-Carlo update of the agent's belief state with a Monte-Carlo tree search from the current belief state" [4]. To account for the partial observability, we needed to specify a belief updater. Since the state space can be intractably large which forces us to use a generative model, we decided to use a particle filter. Specifically, we used `SIRParticleFilter` from the Julia package `ParticleFilters.jl`, which implements sequential importance resampling. As with MCTS, we estimate the value at the leaves of the tree with a simple estimation function.

E. Parameters

For the majority of the data collection, several key parameters were fixed. Unless otherwise stated, the game was played on a 10 by 10 sized `GridGraph` with only a single `ChaseAdversary`. The jitter parameter was set at 0.1 (so the probability of traveling in the intended direction is 0.7), and the radar accuracy was set at 0.2. We set the starting game state with the player in the middle of the grid (uses rounding logic to place at even index) and the adversaries in the corners (ordering them as bottom left, top right, top left, bottom right).

Since we are in the `TagPOMDP` setting with partial observability, we use the POMCP algorithm. We use 500 queries and a total depth of 20, along with an exploration constant of $c = 15$. For the policy evaluation, we used simulated rollouts, accumulating the discounted rewards until $\gamma_P^{numSteps} < 0.1$.

The `AmbushAdversary` was set so that the *lookAhead* parameter was set to three spaces. The `FeignAdversary` was set with a *feignDist* of 4 spaces. Finally, the `PatrolAdversary` was set to patrol from position (3,3) to (3,8) to (8,8) to (8,3) and back again with a *pursueDist* of 3.

IV. Results

To start, we note that the policy used creates utility scores that are extremely variable, as in Figure 2. The distribution of scores is roughly Gamma, noted by the large peak of scores in the (20,25] bin and the tail that appears to decay exponentially. In addition, we notice that the 95% confidence interval for the mean of this data is (36.9, 42.0). This large interval implies that unless we have hundreds of samples, the subsequent results should be taken with a grain of salt.

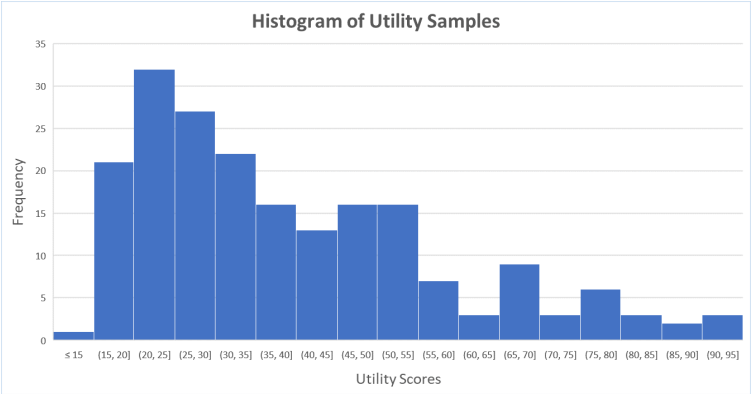

Fig. 2 Histogram of 200 random utility samples given a jitter probability of 0.1 and a radar accuracy of 0.2. Game played on a 10x10 grid, and with a single chaser adversary.

We also note that none of the samples derived have a perfect score of 100, which makes sense considering that a perfect score can only be acquired only on an infinite play-time horizon. The maximum value found in this particular collection of samples was 90.08. Given the reward function and the discount factor, we can derive the equality

numSteps = $\ln(1 - 0.01 * U)/\ln(0.99)$ or $(1 - 0.99^{\text{numSteps}})/0.01 = U$. Thus, the maximum number of steps we have seen is 230. Given that 230 is the smallest integer k such that $0.99^k < 0.1$, the observed maximum is consistent with the setup with the problem.

A. Graph Size

Now, we examine the impact of the size of the grid on the player's utility. In Figure 3, we notice that as the grid size increases from a 3 by 3 grid to a 10 by 10 grid, the utility is strictly increasing. This is fairly intuitive for a number of reasons. If the board size is larger, then there is a larger initial distance between the player and the adversaries. In addition, a larger board size means more options to maneuver around the adversary, which increases the players survival time and thus the total utility as well.

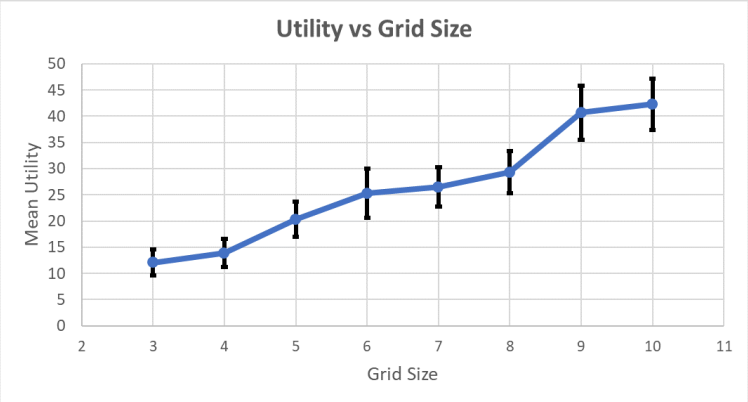

Fig. 3 A plot of the average utility of the player against a single adversary given varying grid sizes. The means are of 60 datapoints with a constant jitter probability of 0.1 and a radar accuracy of 0.2. Error bars represent the 95% confidence intervals.

B. Jitter Probability and Radar Accuracy

Next, we look at the impact of the jitter probabilities and the radar accuracy on the player utility, which is presented in Figure 4. Here, we note that although there is some variability, generally the trend is that larger radar accuracy values corresponds to larger mean utilities. This makes sense: if we are able to observe the position of the adversary more readily, then we are better able to avoid them.

However, we do notice several interesting features of the plot which come as a bit of a surprise. First, we note that we always obtain the maximal score with a jitter probability of 0, regardless of the radar accuracy. One optimal strategy is to travel to the top boundary and then choose the "up" action to stay in the same state. If we were to color all of the grid spots in a chessboard-like fashion, hitting the wall effectively places the player and the adversary on opposite colors, preventing the adversary from ever landing on the same space as the player. Since this does not depend on the player's ability to detect the adversary's location, this strategy works and is optimal for any radar accuracy.

Another interesting point is that the utility is not monotonic with respect to the jitter probability. This is because an increase in the jitter probability has a dual effect: the player has a more difficult time trying to move away from the adversary (so the utility decreases), but the adversary also has a harder time moving towards the player (so the utility increases).

C. Adversaries

Finally, we look at the effect of adversaries on the player's utility, shown in Figure 5. We find that there is a fairly clear partial ordering of the utilities based on the adversary (or group of adversaries). There is also an ordering of the adversaries in terms of AI difficulty: stochastic, feign, ambusher, patroler, and then chaser in order of increasing AI difficulty. Adding a new adversary to the game, or changing the adversary to one with a higher difficulty always decreases the utility (or increases how fast the adversaries can catch the player).

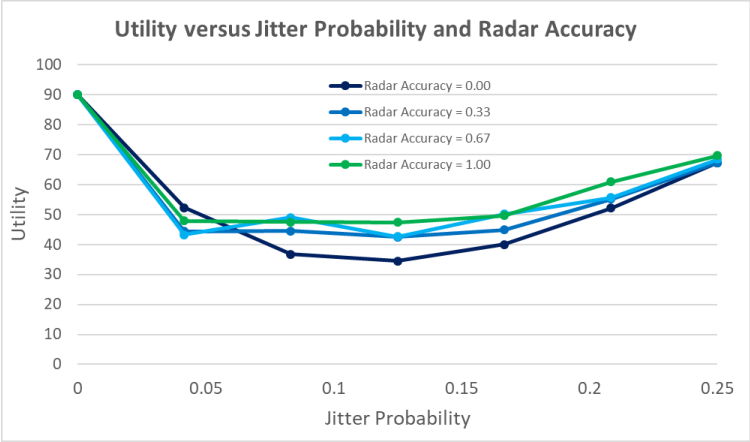

Fig. 4 A plot of the average utility given varying jitter probabilities and radar accuracy values. Each point represents the average of 140 samples.

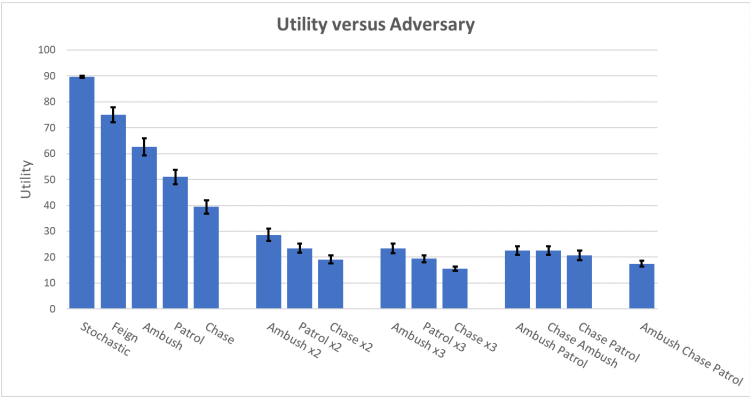

Fig. 5 A plot of the player's utility when playing against different adversaries or groups of adversaries. Each bar represents the mean utility of the player averaged over 200 samples. Error bars represent 95% confidence intervals.

V. Conclusion

In our game of tag, we were able to establish several key points of expected behavior. The player's utility increases with larger grid sizes as well as more frequent reveals of the opponent's location. In addition, the player's utility decreases with additional adversaries as well as when adversary AI's are swapped out for more competent AI's. However, there were a few surprises. For example, state observability has a relatively small impact on the utility compared to all of the other changed parameters of interest. Also, while jitter decreases the utility initially, the utility actually starts to increase with jitter past a certain point.

There are several additional considerations that could be added to the model. First, while the strategy of bumping into the boundary to effectively produce a null state is interesting, it would be interesting to add a significant penalty to dissuade the online model from ever choosing this action. In addition, it would be interesting to see how the player's utility changes with different board types (like irregular grids for example). It might also be interesting to test adversarial AIs that work in concert with one another.

Contributions

Chris and Ryan are both taking the class for 3 units. They collaborated on how to describe the problem of tag on graphs as a POMDP, specifically how they wanted to create both transition and observation uncertainty. They also jointly designed the `GridGraph` and the 5 adversaries.

Ryan wrote the majority of the Julia code specifying the game graphs, adversaries, and (PO)MDPs and did

proof-of-concept solutions with the listed methods above. He also wrote sections I, II, and III(a-d).

Chris analyzed the resulting (PO)MDPs, obtained data samples, and created the figures for the results section. He also wrote the abstract and sections III(e), IV, and V.

References

[1] Rohlfshagen, P., Liu, J., Perez-Liebana, D., and Lucas, S. M., "Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game," *IEEE Transactions on Games*, Vol. 10, No. 3, 2018, pp. 233–256. https://doi.org/10.1109/TG.2017.2737145.

[2] Williams, P. R., "Artificial intelligence in co-operative games with partial observability," Ph.D. Thesis, University of Essex, 2019.

[3] Kochenderfer, M. J., Amato, C., Chowdhary, G., How, J. P., Reynolds, H. J. D., Thornton, J. R., Torres-Carrasquillo, P. A., Ãœre, N. K., and Vian, J., *Decision Making Under Uncertainty: Theory and Application*, 1st ed., The MIT Press, 2015.

[4] Silver, D., and Veness, J., "Monte-Carlo Planning in Large POMDPs," *Advances in Neural Information Processing Systems 23*, edited by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Curran Associates, Inc., 2010, pp. 2164–2172. URL http://papers.nips.cc/paper/4031-monte-carlo-planning-in-large-pomdps.pdf.