2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 27 - October 1, 2021. Prague, Czech Republic

Learning to Play Pursuit-Evasion with Visibility Constraints

Selim Engin, Qingyuan Jiang, and Volkan Isler

2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-1714-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IROS51168.2021.9635959

***Abstract*— We study the problem of pursuit-evasion for a single pursuer and an evader in polygonal environments where the players have visibility constraints. The pursuer is tasked with catching the evader as quickly as possible while the evader tries to avoid being captured. We formalize this problem as a zero-sum game where the players have private observations and conflicting objectives.**

One of the challenging aspects of this game is due to limited visibility. When a player, for example, the pursuer does not see the evader, it needs to reason about all possible locations of the evader. This causes an exponential increase in the size of the state space as compared to the arena size. To overcome the challenges associated with large state spaces, we introduce a new learning-based method that compresses the game state and uses it to plan actions for the players. The results indicate that our method outperforms the existing reinforcement learning methods, and performs competitively against the current state-of-the-art randomized strategy in complex environments.

I. Introduction

Recent advancements in Reinforcement Learning (RL) have led to progress in many different fields, including Atari games [1] and Go [2]. While these methods are successful in single-agent scenarios with static environments and perfect information, they do not readily generalize to settings where multiple agents interact in an environment with imperfect observations. Consequently, there is an increasing interest in developing algorithms for multi-agent problems.

One way to approach this problem is to control the agents using a central unit that computes actions from the joint state-action spaces of all the agents [3]. However, in many settings the partial observations and communication constraints of the agents limit the usage of this approach. An alternative method is to use decentralized agents whose policies are conditioned on only the individual observations.

One of the key challenges when using decentralized agents is *non-stationarity* [4]. In a multi-agent game, the actions taken by an agent affect the global state and the rewards received by the rest of the agents. However, this invalidates the Markovian and stationarity assumptions of most RL algorithms [5]. The paradigm of centralized training with decentralized execution is designed to tackle the non-stationarity problem [6]. In this approach, the critics are trained together with information (observations and actions) from all agents. On the other hand, the policies are learned using information only from their corresponding agents, allowing decentralized execution at test time.

In this paper, we focus on the game of pursuit-evasion where the pursuer and evader have line-of-sight visibility constraints. Due to these constraints, the players obtain partial observations from their surroundings and may not have access to information from their opponents. An instance of the game is shown in Figure 1, displaying the pursuer (red circle) and its visible area in a polygonal environment. Since the evader is not visible, the pursuer needs to locate its adversarial opponent and capture it. To address the challenges associated with limited visibility, we propose a method where each agent maintains a belief state for the possible locations of the other agent and uses it along with the partial observations to compute its actions.

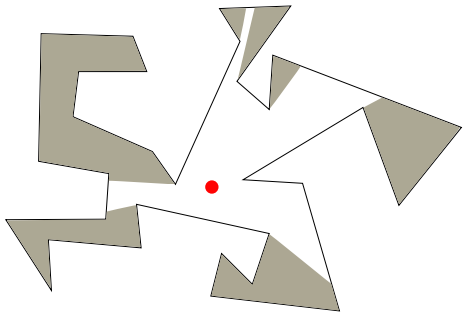

Fig. 1: **Pursuit-evasion with visibility constraints:** The pursuer (red circle) needs to clear the unseen regions of the environment (shaded areas) to locate and capture the evader.

The usage of a belief state is crucial for the performance of our method. Imagine that the pursuer visits and covers one of the shaded areas, and then returns to its original position in Figure 1. A naïve state representation might fail to capture the actual state of the game. For instance, if it includes only the position of the agent and local observations, then the state will be unable to indicate that the player has already covered an area and does not need to visit there again. In contrast, the belief state provides a means for representing the past history of the game and satisfies the Markovian property.

A practical application of the pursuit-evasion problem is automated cinematography. Suppose that a mobile robot is tasked with capturing images of an animal, whose body dynamics are to be recorded. The animal might be running away, or even hiding behind the trees. Pursuit-evasion is useful to study such scenarios when the movement of the target is modeled as adversarial. Finding strategies for the pursuer can enable us generating trajectories for the camera that ensure maintaining a certain distance with the target and capturing good-quality images. While we study a different version of this problem, many of the challenges such as planning under uncertainty and target tracking are common

This work was supported in part by NSF grants #1617718 and #2022894. All authors are with the University of Minnesota. {engin003, jian0345, isler}@umn.edu.

in both settings.

To summarize, our contributions in this paper are:

- We introduce novel belief state and action representations for the players to reason about the unseen parts of the environment that improve the learning performance.
- We present a curriculum learning technique to train the agents by repetitively playing them against each other in increasingly complex environments.
- We show that training by playing against an expert strategy significantly improves the player performances.
- We compare our method against baseline approaches in a series of experiments, including out-of-domain scenarios, and demonstrate its generalization capability.

II. Related Work

We focus our related work on the fields most relevant to our method. We first go over the the problem of pursuit-evasion, and then discuss previous work on multi-agent RL.

A. Pursuit-Evasion

The pursuit-evasion problem has been studied since 1930's. In the original formulation, there is a lion and a human in a circular arena, where the goal of the lion is to catch its prey as quickly as possible, whereas the human escapes to avoid being a meal. Pursuit-evasion since then has been studied in many different settings. Some of these settings include trees [7], graphs [8], polygons [9], [10] and surface of polyhedrons [11]. The work of [12] introduced a differential game formulation to the problem, which has led to game-theoretic approaches for pursuit-evasion [13].

Traditionally pursuit-evasion has been referring to two related, but different sub-components of the problem: 1) locating the evader, and 2) capturing the evader. When the players have constrained observations of their opponents, such as line-of-sight visibility (e.g. lidar, camera), they need to search the environment to locate the opponent. The problem of planning paths to eventually have an unobstructed view of the evader was first introduced in [14]. The seminal work of [9] showed that $\Theta(\log n)$ pursuers are necessary and sufficient to detect an unpredictable evader in a simply-connected polygon of n vertices. Since then, many studies have addressed variations of this problem [15]–[17]. For example, in [15] the robots have limited field of view, and in [16] they obtain potentially erroneous measurements.

There are a handful of studies that worked on capturing the evader while being subject to visibility constraints. A randomized algorithm was introduced in [10] that can locate the evader using a single pursuer, and an extension of this strategy allows two pursuers with line-of-sight visibility to capture the evader in simple-connected polygons. Later, [18] showed that a single pursuer can locate and capture the evader in monotone polygons.

B. Multi-agent Reinforcement Learning

There has been a significant interest for solving problems involving a group of agents with conflicting or cooperative objectives, using multi-agent RL [19]. One of the earlier works advocating the paradigm of centralized training of decentralized policies, [6] proposed to use a single centralized critic with all joint actions and state information for training and use the counterfactual baseline as the advantage function. Later, [20] improved the overall performance by introducing a mixing network to represent a monotonic value function. However, both these methods focus on cooperative tasks such as the StarCraft II micromanagement problem [21]. In [19], this setting is extended to competitive modes with continuous actions and included explicit communication between agents by learning a centralized critic for each agent.

III. Notation and Formulation

In this section we start by defining key concepts used in our algorithm, then present the game formulation.

We denote the polygonal workspace by W with boundary ∂W. In this paper, we focus on simply-connected polygons, meaning that there is no hole inside the polygon. We measure the complexity of a polygon by its number of *reflex* vertices. A vertex in a polygon is called reflex if its internal angle is greater than 180°. The Euclidean distance between two points $x, y \in \mathbb{R}^2$ is denoted by $d(x, y)$. In a polygon W, the geodesic distance between x and y is denoted by $d_W(x, y)$. We use the top-down images of some of the geometric entities in our method, and denote the image of a geometry by $I(\cdot)$.

The players are assumed to have an omni-directional line-of-sight sensor, meaning that they can obtain location measurements from their 360° surrounding at each time step. A player sees its opponent only if the line segment joining their locations does not intersect the polygon boundary. The visible area of a player i, also known as the visibility polygon, is denoted by $V^i \subseteq W$. Thus, the pursuer p sees the evader e if $e \in V^p$ and vice versa. Finally, we assume that the players have holonomic mobility with equal bounded speeds.

A. Game Formulation

We are given a simply-connected polygon W and a capture radius c. The positions of the players at time t are denoted by p_t and e_t for the pursuer and evader, respectively. Each time step, the players make their moves simultaneously. The pursuer captures the evader if $d(p_t, e_t) \leq c$ and p sees e. The pursuer wins the game if it can capture the evader in some finite time T, and loses otherwise.

IV. Background

In this section we present background on multi-agent games which will be useful for describing our method.

We formulate the pursuit-evasion problem as a Partially Observable Markov Game (POMG) [22], an extension of the Markov Decision Process (MDP) with partial observations and multiple players. Each player is an agent that senses and acts within the environment in discrete time steps. The Markov game is defined by a tuple $\{\mathcal{N}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{N}}, \{\mathcal{R}^i\}_{i\in\mathcal{N}}, \mathcal{T}, \gamma\}$ where $\mathcal{N}$ denotes the set of N agents, $\mathcal{S}$ is the set of possible states of all agents, $\mathcal{T}$ is the transition probability $\mathcal{S}\times\mathcal{A}\times\mathcal{S} \rightarrow [0, 1]$, and $\gamma \in [0, 1]$ is the discount factor. In this paper, we have 2 players and denote the set of players by $\mathcal{N} = \{p, e\}$.

Since the agents cannot fully observe the state, they have partial observations of the game state separately $\mathcal{O}^p, \mathcal{O}^e$, which are private to the players as long as they do not see each other. The players take actions using their policies π^p, π^e, with parameters θ^p and θ^e, respectively. The collection of actions $\mathcal{A} : \mathcal{A}^p \times \mathcal{A}^e$ and the current game state determines the next state according to $\mathcal{T}$ and provides rewards as a function of the state and agent's action $r^i : \mathcal{S} \times \mathcal{A}^i \times \mathcal{S} \to \mathbb{R}$.

The objective of each agent i is to maximize its total expected return $G_t^i = \sum_{k=0}^{T} \gamma^k r_{t+k+1}^i$ over time horizon T by optimizing the policy π_{θ^i}. We formulate the pursuit-evasion problem as a competitive zero-sum Markov Game. Players have opposite rewards at each step t, that is, $r^p(s, a, s') + r^e(s, a, s') = 0$.

V. Learning to Play Pursuit-Evasion

We present a new method called Pursuit-Evasion with Belief States (PEBS) in this section. An overview of our method is shown in Figure 2.

A. State representations

Since the agents have partial access to the global game state, the selection of the state representation for the players has high significance. Moreover, it is not straightforward to decide which representation is ideal for the learning algorithms.

PEBS uses a combination of an observation vector and a 2D spatial map that characterizes the state of the game for each player. This spatial map is an image of multiple channels with each channel corresponding to a specific geometry. We denote this image at time t by $\mathcal{I}_t$. The observation vector o_t, on the other hand, includes the players own position p_t (or e_t if the player is the evader), the visibility flag indicating whether the opponent is visible at time t, and the opponent's position if it is visible (otherwise, a vector of zeros is used for the opponent position).

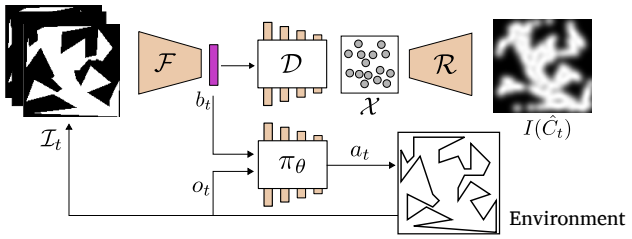

Fig. 2: **PEBS overview:** Each agent receives a partial observation o_t and an image $\mathcal{I}_t$ in each iteration of the game. The agent employs a belief network $\mathcal{F}$ that encodes $\mathcal{I}_t$ to a belief state b_t. The belief state is used together with o_t as input to the policy network π_θ to compute actions a_t. To supervise the belief states, we decode b_t with $\mathcal{D}$ for mapping to a set of 2D points $\mathcal{X}$ which is then rendered with $\mathcal{R}$ to generate an estimate image of the contaminated region.

The first channel of $\mathcal{I}_t$, denoted by $I(W)$, is a top-down binary image of the polygonal region W that determines the movable space in the environment. $I(W)$ remains constant throughout an episode. The second channel corresponds to the self position of the player, indicating the location of the player within the environment. We denote this channel by $I(p_t)$ if the player is the pursuer, otherwise by $I(e_t)$. Finally, the last channel is what we call the *contaminated region*, borrowing the terminology from previous work [9], [15]. The contaminated region C_t refers to the set of possible locations of the opponent. Since the players have visibility constraints, the opponent's position may not be available to the player at time t. Therefore, each player maintains an image corresponding to all possible locations the opponent might occupy at t. When there is direct visibility the contaminated region shrinks to a small patch in the image, and the region grows at each time instant the opponent is occluded. We denote the image of the contaminated region by $I(C_t)$. Note that whereas $I(W)$ is the same for both the pursuer and evader, the image channels $I(p_t)$ and $I(C_t)$ are maintained separately. An instance of an environment configuration and the image channels the pursuer receives is shown in Figure 3, and a sequence of images obtained throughout a trajectory is shown in Figure 5. We use 128×128 dimensional images in our experiments.

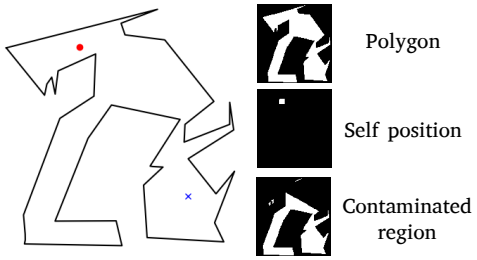

Fig. 3: For a given configuration (left), the pursuer (●) and evader (×) both receive a 3-channel image, separately. On the right, the image received by the pursuer is shown.

B. Belief state generator

Given the environment image $\mathcal{I}_t$, each agent separately generates a belief state using an encoding function $\mathcal{F}$, which is a Convolutional Neural Network (CNN). $\mathcal{F}$ embeds the input images into a latent space that characterizes the belief state distribution of the agent. Each player uses its encoder network $\mathcal{F}$ to compute a vector corresponding to the belief state as $b_t = \mathcal{F}(\mathcal{I}_t)$.

One way we can use the belief state is directly as an input to policy network π_θ without any regularization of the latent space. However, we find that this approach is not able to encode information about the contaminated region effectively to explore the environment as shown in Section VI. Instead, we propose to use another network to decode the belief state into the 2D workspace and supervise it with image losses, which we explain next.

C. Differentiable renderer

After generating the latent belief state b_t, we use a decoder network $\mathcal{D}$ to map b_t to the 2D plane, $\{x_i\}_{i=1}^N = \mathcal{D}(b_t)$, $x_i \in \mathbb{R}^2$. The point set $\mathcal{X} = \{x_i\}_{i=1}^N$ coordinates are then normalized with respect to the size of the image canvas. We

represent each point x_i as an isotropic Gaussian with mean x_i and a fixed variance σ_x^2.

The renderer $\mathcal{R}$ is a deterministic function that projects the points $\mathcal{X}$ onto the canvas while preserving gradients, similar to the *splatting* [23] operation:

$$\mathcal{R} : \mathbb{R}^{N\times 2} \rightarrow \mathbb{R}^{H\times W}, \mathcal{X}_t \mapsto \mathcal{R}(\mathcal{X}_t) = I(\hat{C}_t) \tag{1}$$

The pixel $(u, v) \in \mathbb{R}^2$ of the generated image for the contaminated region $I(\hat{C}_t)$ is computed by,

$$I(\hat{C}_t)_{uv} = \sum_{i=1}^{N} w_i \cdot f((u, v)|x_i, \sigma_x^2) \tag{2}$$

where $f(\cdot|x_i, \sigma_x^2)$ is a Gaussian radial basis function with mean x_i and variance σ_x^2. The scalar weights w_i are used to normalize the image intensities to lie within $[0, 1]$. In our experiments, we set $N = 32$ and $\sigma_x = 8$.

D. Curriculum through competitive games

While there is no weight sharing between the networks of the agents, they are trained by repetitively playing them against each other with conflicting objective functions.

PEBS exhibits two forms of curriculum for training the players. The first form is implicit, and emerges from the learning dynamics of two competing models. Figure 4 shows an example plot of the average returns of the players during training. We can see a learning regime where the pursuer initially wins most of the games and then evader starts to become more adversarial and makes the game more difficult for the pursuer. The oscillatory curves of the average returns indicate an autocurriculum by playing the pursuer and evader against each other. This is also demonstrated in the resulting trajectories of the players at different epochs.

We additionally use a second form of curriculum for the environment complexity, which is performed explicitly. We can measure the complexity of a polygonal environment with the number of reflex vertices, since a reflex vertex induces a critical boundary which determines regions in the environment the pursuer has to clear if they are not visible from the pursuer's location. Initializing with 5 reflex vertices, during training we increase the number of reflex vertices in the environment by 5 every 200 epochs, until it reaches 20. In our experiments, we find that starting with simple environments helps the training of the players, and leads to better learning of navigation and exploration skills.

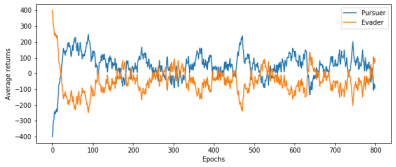

Fig. 4: Average returns of the players during training.

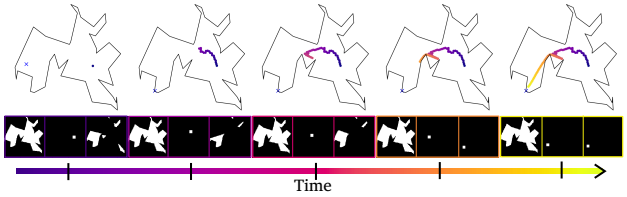

Fig. 5: The snapshots of the game at various time instants, along with the obtained images. After clearing the contaminated regions and locating the evader (×), the pursuer (colored circle) captures its opponent. Best viewed digitally.

E. Model architecture

Both the pursuer and evader have their own separate models, without any weight sharing between the networks of the players. The encoder network $\mathcal{F}$ is a CNN of 4 layers with instance normalization and ReLU activation between the layers. The decoder $\mathcal{D}$ is a Multi-Layer Perceptron (MLP) with a single fully connected layer. As our base RL algorithm, we use the Soft Actor-Critic (SAC) [24] method, where the policy network π_θ is a 3 layer MLP with parameters θ. The output of the policy network is a 2D coordinate in the workspace corresponding to the direction the player chooses to go in the next time step. To compute the next position, we find the angle φ to the 2D coordinate from the current position and take a step with direction φ.

VI. Analysis

In this section, we analyze our method and compare with earlier works through a series of experiments. We design our experiments around the following questions: **Q1)** How do existing methods (classical and learning methods) for pursuers perform against different evader types? **Q2)** How do the players perform when they are trained by playing against each other? **Q3)** Does it help to train the players by playing against an expert strategy? **Q4)** Do the learned policies generalize to out-of-domain and realistic environments?

Before reporting our findings, we present the baseline methods we compare against and the performance metrics used in the evaluations. We choose two primary considerations to solve the pursuit-evasion problem. The first is a classical algorithm, namely the randomized strategy. The second category consists of RL methods that use various types of state representations and training algorithms.

1) Randomized Lion's Strategy (RLS): Our first baseline is the randomized lion's strategy for the pursuer presented in [10]. This method is the current state-of-the-art for the pursuit-evasion problem with visibility constraints.

2) Multi-agent RL: MADDPG [19] is a multi-agent RL algorithm that uses the centralized learning with decentralized execution approach. The predator-prey task in [19] is similar to the game studied in this paper. In the predator-prey environment, multiple predators are tasked with hitting the prey by receiving partial observations. The observations are partial, however, not due to the visibility constraints but because they are expressed in the agent's local frame. Therefore, there is no private information in the setting of [19].

3) Learning from partial observations: The rest of the methods we compare against use an RL algorithm with decentralized critics and actors to compute actions. Specifically, we use SAC [24] as the base RL algorithm with the same set of hyperparameters as in our method.[‡]

We use two evaluation metrics to compare different baselines with our method. For a given episode termination timeout T, the first metric we use measures the Success Rate (SR) percentage of the pursuer capturing the evader before the episode terminates. We set $T = 150$ in all experiments.

The second metric measures the average time for the pursuer to capture the evader. We normalize the Capture Times (CT) to range between $[0, 1]$ by dividing them by T. We report the mean and standard deviation in both metrics.

A. Existing methods against various evader types

In our first set of experiments we investigate the performance of a state-of-the-art RL method against traditional algorithms in five different settings. These settings include using various types of evaders and information available to the players: 1) The pursuer has partial visibility and the evader remains static throughout an episode, 2) The players have full visibility (can see each other even when out of sight) and the evader moves away greedily, 3) the players have partial visibility and the evader uses the *rash* model [25], 4) The players have full visibility and use learned strategies, 5) The players have partial visibility and use learned strategies.

In the rash model, the evader hides and does not move until it is seen by the pursuer. Upon being visible, it picks a child node in the dual tree of the polygon and moves there.

	Capture time		Success rate	
Evader Type	RLS	SAC	RLS	SAC
1- Static (partial vis.)	0.30 (0.2)	0.48 (0.4)	1.0 (0.0)	0.67 (0.5)
2- Greedy (full vis.)	0.34 (0.2)	0.44 (0.3)	0.99 (0.1)	0.89 (0.3)
3- Rash (partial vis.)	0.51 (0.4)	0.76 (0.4)	0.68 (0.5)	0.33 (0.5)
4- Learned (full vis.)	0.32 (0.2)	0.41 (0.3)	0.99 (0.1)	0.89 (0.3)
5- Learned (partial vis.)	0.42 (0.3)	0.57 (0.3)	0.90 (0.3)	0.67 (0.5)

TABLE I: Capture times (lower is better for the pursuer) and success rates (higher is better for the pursuer) in previously unseen polygons with 40 vertices.

For the learned strategy we tested the SAC algorithm using different state representations and reported the results from the best performing one (Table I). We trained both agents against each other with the same state representations. We see that for the cases with full visibility (2 and 4) the success rates and performances are relatively close to that of the classical baseline (RLS). Whereas for partial visibility models (1, 3 and 5) the performances are much worse compared to the classical method even when the evader is static.

These results indicate that in problems with partial observations and complex dynamics, using an existing RL method without any modifications does not yield good performances. Next, we show how PEBS compares against the classical and learning-based baselines.

[‡]Please refer to the appendix for more details on the baselines and experiments: `https://sites.google.com/umn.edu/pebs`.

B. Training by playing against each other

We analyze the selection of state representations for learning strategies to play the game. We compare the methods that train the pursuer and evader policies by playing against each other. In the first experiment (Table II), we evaluate the pursuers by using the evaders they were trained together. In the second (Table III), we evaluate the pursuer performances using the rash evader model.

We find that in both experiments, PEBS outperforms the rest of the learning-based methods. However, the classical algorithm's performance is still the best compared to all the end-to-end trained methods when evaluated with the rash evader (see Table III). We also see that the performances against the learned and rash evader do not match perfectly. One reason for this is, since the pursuer strategies are trained with the learned evader, they perform better against the learned behavior at test time. It can also be the case that the learned evader strategy is stuck at a saddle point which leads to good performance for the pursuer when evaluated against the learned evader. We observe this phenomenon in several cases (e.g. MADDPG and MLP (lidar)), where there is a large discrepancy between the performances in the two experiments.

The qualitative performance of PEBS can be seen from a sample game between the players shown in Figure 5. The colored circles indicate the trajectory of the pursuer while the blue crosses show the evader's position. In the bottom row, the obtained images are displayed. We see that the pursuer tries to cover the contaminated regions and as soon as it locates the evader, it moves towards and captures its opponent.

Pursuer Type	Capture time	Success rate
MADDPG[†]	0.566 (0.359)	0.62 (0.48)
MLP* (lidar)	0.659 (0.350)	0.55 (0.49)
MLP* (sampling)	0.575 (0.334)	0.67 (0.47)
CNN*	0.697 (0.334)	0.48 (0.50)
CNN+RNN*	0.686 (0.359)	0.48 (0.50)
PEBS*	**0.482 (0.330)**	**0.76 (0.42)**

TABLE II: Mean and std. of CT (↓) and SR (↑) in unseen polygons with 40 vertices against the *learned evader* trained together. Base algorithms: [†]DDPG [26], *SAC [24].

C. Training by playing against an expert

We then investigate the question whether training against an expert strategy helps the learned policies. To do so, we employ a two-step training procedure. We first train the evader against a pursuer using the randomized strategy. Then, we train a pursuer policy by playing against the pre-trained evader and continue the training of both agents. We find that this strategy helps the overall performances of both players. Using this training procedure, the SR of PEBS improves to 72% and it slightly outperforms the classical algorithm (RLS), as shown in the last row of Table III.

D. Generalization to real environments

In our final experiment, we analyze the generalization performance in out-of-domain environments. To do so, we use the indoor maps from the Gibson database of 3D spaces [27]

and evaluate the policies trained on polygonal environments without any retraining or fine-tuning. For each 3D model, we first compute its top-down image, then compute the contours of the image to get the boundary of the floor plan. Finally, we approximate the boundary of the environment as a polygon with at most 100 vertices. We discard the environments with multiple disconnected components from the dataset, since otherwise they may not be fully navigable. In the end, we have 78 scenes to test the methods.

Pursuer Type	Capture time	Success rate
RLS	**0.510 (0.37)**	**0.68 (0.46)**
MADDPG†	0.843 (0.31)	0.20 (0.40)
MLP* (lidar)	0.886 (0.28)	0.17 (0.37)
MLP* (sampling)	0.763 (0.36)	0.33 (0.47)
CNN*	0.766 (0.35)	0.33 (0.47)
CNN+RNN*	0.797 (0.34)	0.27 (0.44)
PEBS*	0.689 (0.37)	0.45 (0.49)
PEBS* (two-step)	**0.433 (0.39)**	**0.72 (0.44)**

TABLE III: Mean and std. of CT (↓) and SR (↑) in unseen polygons with 40 vertices against the *rash evader* model.

We find that most methods are able to generalize to these real maps well, since they were trained on a wide variety of environment shapes (see Table IV). Moreover, PEBS still outperforms other learning-based baselines in this experiment.

Pursuer Type	Capture time	Success rate
MADDPG†	0.661 (0.44)	0.358 (0.47)
CNN*	0.528 (0.44)	0.538 (0.50)
PEBS*	**0.482 (0.44)**	**0.589 (0.49)**

TABLE IV: Mean and std. of CT (↓) and SR (↑) in realistic indoor maps against the *rash evader* model.

VII. Discussion and Limitations

In this paper, we introduced a method for learning to play the pursuit-evasion game where the agents have visibility constraints. We showed that the existing state-of-the-art RL algorithms do not perform well when there are partial/private observations and complex dynamics. To address this problem, our method uses a compressed belief state for each player to reason about the possible locations of its opponent. Our experiments indicate that by maintaining a belief state, the agents are able to explore the environment better and improve their game playing performances. We also demonstrate our method's generalization capability on a dataset of real maps.

An important limitation of our method is not being able to significantly outperform the classical algorithm, even after the two-step training procedure with the expert. We believe that addressing this problem is crucial for solving other related games with partial observations.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[3] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, pages 183–221, 2010.

[4] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019.

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[6] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[7] Andreas Kolling and Stefano Carpin. Pursuit-evasion on trees by robot teams. *IEEE Transactions on Robotics*, 26(1):32–47, 2009.

[8] Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer, 1978.

[9] Leonidas J Guibas, Jean-Claude Latombe, Steven M LaValle, David Lin, and Rajeev Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry & Applications*, 9(04n05):471–493, 1999.

[10] Volkan Isler, Sampath Kannan, and Sanjeev Khanna. Randomized pursuit-evasion in a polygonal environment. *IEEE Transactions on Robotics*, 21(5):875–884, 2005.

[11] Narges Noori and Volkan Isler. The lion and man game on polyhedral surfaces with boundary. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1769–1774. IEEE, 2014.

[12] Rufus Isaacs. *Differential games*. Wiley, 1965.

[13] Sourabh Bhattacharya and Seth Hutchinson. A cell decomposition approach to visibility-based pursuit evasion among obstacles. *The International Journal of Robotics Research*, 30(14):1709–1727, 2011.

[14] Ichiro Suzuki and Masafumi Yamashita. Searching for a mobile intruder in a polygonal region. *Journal on Computing*, 21(5):863–888, 1992.

[15] Brian P Gerkey, Sebastian Thrun, and Geoff Gordon. Visibility-based pursuit-evasion with limited field of view. *The International Journal of Robotics Research*, 25(4):299–315, 2006.

[16] Nicholas M Stiffler, Andreas Kolling, and Jason M O'Kane. Persistent pursuit-evasion: The case of the preoccupied pursuer. In *Intl. Conference on Robotics and Automation (ICRA)*, pages 5027–5034. IEEE, 2017.

[17] Daigo Shishika and Vijay Kumar. Local-game decomposition for multiplayer perimeter-defense problem. In *Conference on Decision and Control (CDC)*. IEEE, 2018.

[18] Narges Noori and Volkan Isler. Lion and man with visibility in monotone polygons. *The International Journal of Robotics Research*, 33(1):155–181, 2014.

[19] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[20] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2018.

[21] Oriol Vinyals, Timo Ewalds, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[22] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[23] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus H. Gross. Surface splatting. In *Conference on Computer Graphics and Interactive Techniques, SIGGRAPH*. ACM, 2001.

[24] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[25] Alberto Quattrini Li, Raffaele Fioratto, Francesco Amigoni, and Volkan Isler. A search-based approach to solve pursuit-evasion games with limited visibility in polygonal environments. In *Intl. Conference on Autonomous Agents and Multiagent Systems*, pages 1693–1701, 2018.

[26] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[27] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson Env: real-world perception for embodied agents. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.