J Control Theory Appl 2011 9 (3) 391–399
DOI 10.1007/s11768-011-0272-3

A model-based approximate λ-policy iteration approach to online evasive path planning and the video game Ms. Pac-Man

Greg FODERARO, Vikram RAJU, Silvia FERRARI

Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708, U.S.A.

Abstract: This paper presents a model-based approximate λ-policy iteration approach using temporal differences for optimizing paths online for a pursuit-evasion problem, where an agent must visit several target positions within a region of interest while simultaneously avoiding one or more actively pursuing adversaries. This method is relevant to applications, such as robotic path planning, mobile-sensor applications, and path exposure. The methodology described utilizes cell decomposition to construct a decision tree and implements a temporal difference-based approximate λ-policy iteration to combine online learning with prior knowledge through modeling to achieve the objectives of minimizing the risk of being caught by an adversary and maximizing a reward associated with visiting target locations. Online learning and frequent decision tree updates allow the algorithm to quickly adapt to unexpected movements by the adversaries or dynamic environments. The approach is illustrated through a modified version of the video game Ms. Pac-Man, which is shown to be a benchmark example of the pursuit-evasion problem. The results show that the approach presented in this paper outperforms several other methods as well as most human players.

Keywords: Approximate dynamic programming; Reinforcement learning; Path planning; Pursuit evasion games

1 Introduction

Although simple in appearance and gameplay, the single player video game Ms. Pac-Man offers a challenging representation of a pursuit-evasion problem that requires extended foresight, quick decision-making and a high degree of adaptability. Like many computer games, Ms. Pac-Man presents an excellent benchmark for testing intelligent algorithms because it is characterized by simple rules and objectives but offers challenging environments and tasks [1]. The pursuit-evasion family of games describes a predator and prey scenario where the objective of one group is to evade a second group in the environment, which has the goal of tracking and catching the first group. This type of game is analogous to several real-world applications, such as robotic path planning [2, 3], mobile-sensor applications [4], and path exposure [5, 6].

In Ms. Pac-Man, the player assumes the role of the evader and must navigate a maze to visit several target locations ('dots') while avoiding a team of pursuing adversaries with individualized strategies. There is a pre determined pattern of dots scattered in each maze that Ms. Pac-Man must eat, and when all of the dots have been eaten, the player advances to the next level, which involves a more difficult maze, a new set of dots, and faster adversaries. A screenshot of the game's first maze is shown in Fig. 1.

The existing methodologies addressing this problem are diverse, but all have fallen far short of expert human players [7–12]. The previous approaches use fairly short-term greedy strategies and fail to effectively consider future game states, which is an essential capability needed for success.

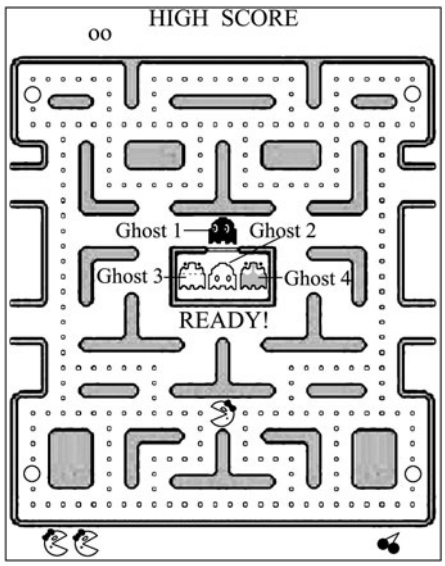

Fig. 1 Screenshot of Level 1 game maze.

In addition, classical dynamic programming methods are not feasible due to the computational complexity caused by the game's large state space. This paper proposes the combination of online learning with prior knowledge through modeling to choose long-term decisions, and the frequent reevaluation of these decisions allows for quick reactions to unexpected changes in the environment.

The model-based approximate λ-policy iteration methodology presented in this paper gives an approach for determining optimal paths online for an evader in a pursuit-

Received 9 November 2010; revised 23 March 2011.
This work was supported by the National Science Foundation (No.ECS 0925407).

evasion problem where an agent must balance the tasks of avoiding a group of pursuers while visiting several target locations within a region of interest. Cell decomposition is used to transform the obstacle-populated Eucledian space, representing the environment where the agent is allowed to travel, into a finite set of convex cells, and the resulting cell decomposition is used to construct a decision tree. Combining the decision tree with prior knowledge of the system and online approximate λ-policy iteration, an optimal strategy can be computed, which achieves the objective of minimizing the risk of being caught by an adversary and maximizing a reward associated with visiting the target locations [13, 14]. This methodology is illustrated through a modified version of the video game Ms. Pac-Man, which is shown to be a benchmark example of the pursuit-evasion problem, and it is directly compared with a model-free neural player trained with Q-learning. The results show that the model-based approximate λ-policy iteration approach outperforms several other methods as well as most human players.

2 Problem formulation and assumptions

The path planning problem considered in this paper is to find the optimal paths of a single mobile agent with position or state, x_{p}, that travels in a two-dimensional Euclidian workspace denoted by $\mathcal{W} \subset \mathbb{R}^2$. The agent must navigate through the workspace and collect a set of distributed objects or visit several points of interest within $\mathcal{W}$ while simultaneously avoiding collisions with a group of N actively-pursuing adversaries with states denoted by x_{G}^I, where I corresponds to an adversary's index, and a collision is defined as any instant where $x_{\mathrm{p}} = x_{\mathrm{G}}^I$ for all I. The optimal paths are then those that minimize the risk of encountering an adversary while maximizing the number of goal positions achieved. The workspace geometry and the positions of the points of interest or distributed objects are assumed to be known a priori, and the approach can be extended to higher dimensional workspaces, assuming the system is within the limits of computational feasibility. The behaviors or control laws of the adversaries are also assumed to be known, and their positions are assumed to be observable in real time.

This problem can be put in the context of a benchmark problem taken from the video game Ms. Pac-Man, of which the details are provided in Section 1. The agent's (Pac-Man's) state and control are represented by the 2×1 vectors:

$$x_{\mathrm{p}} = [x_{\mathrm{p_x}} \;\; x_{\mathrm{p_y}}]^{\mathrm{T}}, \tag{1}$$

$$u_{\mathrm{p}} = [u_{\mathrm{p_x}} \;\; u_{\mathrm{p_y}}]^{\mathrm{T}}, \tag{2}$$

where the subscript p corresponds to Pac-Man, $x_{\mathrm{p_x}}$ and $x_{\mathrm{p_y}}$ are Pac-Man's x and y coordinates in pixels, and Pac-Man's controls, denoted by $u_{\mathrm{p_x}}$ and $u_{\mathrm{p_y}}$, signify the direction of Pac-Man's movement (or attempted movement, if facing a wall) in the x and y directions, respectively. The adversaries' (ghosts') states and controls, x_{G}^I and u_{G}^I, are defined in an identical manner, where the subscript G corresponds to the ghosts. The workspace is defined as a maze encountered in the game, as shown in Fig. 1. Let $\mathcal{W}$ have an inertial frame of reference, $F_{\mathcal{W}}$, such that all possible xy-coordinates are in the positive orthant and are always greater than zero. The geometries of Pac-Man and the ghosts can be thought of as rectangular in shape such that their widths only allow for bidirectional movement along a straight path length. Therefore, each point within the maze has a set of admissible actions, $U[x(t_k)] \subset \mathcal{U}$, where

$$\mathcal{U} = \{a_1, a_2, a_3, a_4\} \equiv \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\} \tag{3}$$

is the space of all possible actions or control values for both Pac-Man and the ghosts, and t_k is a discretized instant in time such that $t_i \leqslant t_k \leqslant t_f$. According to the coordinate frame convention, $u_{\mathrm{x}} = +1$ denotes motion to the right, $u_{\mathrm{x}} = -1$ represents motion to the left, $u_{\mathrm{y}} = +1$ corresponds to upward motion, and $u_{\mathrm{y}} = -1$ coincides with downward motion.

Since the workspace is a rectangular grid of pixels, there are a finite number of positions possible in the game. Also, as shown below in Section 4, the ghosts' decisions depend only on the current state and action of Pac-Man. An exception is during the first few seconds of the game when the ghosts make random decisions before they begin to chase Pac-Man, but the associated risk is low enough that the player may use this time to virtually move freely about the maze. Neglecting this period of randomness, the game may be formulated as a Markov decision process (MDP).

An MDP can be represented by a 4-tuple $\mathcal{M} = \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}$ denoting a sequential decision model, where $S = \{s_1, s_2, \ldots, s_n\}$ is the finite set of all feasible state values, $A = \{a_1, a_2, \ldots, a_n\}$ is the set of possible actions, $\mathcal{T}$ is the transition function where $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to P(\mathcal{S})$, and $\mathcal{R}$ is the reward function such that $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. $\mathcal{T}$ describes the probability of an action a_i executed at state x_j resulting in state x_l. The reward function determines the immediate reward obtained as a result of performing action a_i at state x_j. From the MDP, a policy π can be derived, which maps the states to corresponding actions.

In the game, the player's goal is to achieve the highest possible score by eating various objects while evading four pursuing ghosts. If the ghosts catch Pac-Man, a 'life' is lost, and when the player is out of lives, the game ends. Therefore, the agent's performance is based on the score earned before being caught by a ghost. Points are scored as follows: Each dot eaten is worth 10 points. Each power pill, which temporarily causes the ghosts to retreat and allows Pac-Man to send the ghosts back to their starting locations by eating them, is worth 50 points when eaten. When a power pill is active and the ghosts are fleeing and vulnerable, eating a ghost earns 200×2^n points for the nth ghost eaten during a single power pill's time interval. In other words, if all four ghosts are eaten in succession, $200 + 400 + 800 + 1600 = 3000$ points would be earned. Bonus 'fruits' which are objects that move randomly across the maze and give the player extra points if the objects are eaten, are not considered for this problem and have been removed from the game.

3 Background

Approximate dynamic programming (ADP) is a highly recognized approach used to find control strategies that op-

timize a desired performance metric for complex nonlinear and stochastic dynamic systems. Such systems often suffer from what is known as the 'curse of dimensionality' and cannot be solved through conventional optimization methods or dynamic programming due to computational complexity. ADP overcomes this by representing the system as a parametric structure and performing the optimization incrementally. Approximate policy iteration by temporal difference (TD) learning and Q-learning are two ADP algorithms applied here, where Q-learning is only implemented for comparison. Temporal difference learning uses a model of the system to predict future performance values and then derive a policy that optimizes those values. Q-learning computes an action-value function that provides estimated performance values of the possible actions in a given state. One of the major advantages of Q-learning is that it does not have a need for a system model; however, in this paper, it is shown that the inclusion of prior knowledge through a model used in temporal difference learning can be utilized to significantly reduce the complexity of the optimization and increase controller effectiveness.

3.1 Temporal difference theory

By formulating the problem as an MDP, a policy can be derived that maps the states to corresponding actions. Through learning, the optimal policy, π^*, can be found such that the value function, $V^\pi(X_k)$ is always maximized, where

$$V^\pi(X_k) = \mathrm{E}\{\sum_{i=0}^{\infty} \gamma^i r_{k+i} | \pi, X_k\}. \tag{4}$$

The reward received at i time steps into the future is given by r_{k+i}. The effect that future rewards have on the present decision is based on the discount factor, γ, where a large discount factor results in a policy that emphasizes on the long term effect, and vice versa. By definition, an optimal policy is guaranteed for an MDP, but it does not have to be unique.

Temporal difference learning seeks to find the optimal policy, π^*, by computing a prediction function, P, of the total reward expected from a given state such that

$$P_k(X_k) \approx \sum_{i=0}^{\infty} \gamma^i r_{k+i} \tag{5}$$

with minimal error for all k. For large systems, P_k can be represented as the weighted linear function:

$$P_k(X_k) \approx \sum_{j=0}^{n} w_k^j x_k^j, \tag{6}$$

where $X_k = [x_k^1 \ x_k^2 \ \cdots \ x_k^n]$ is a vector of state variables at time step k, $W_k = [w_k^1 \ w_k^2 \ \cdots \ w_k^n]$ is a vector of weight parameters, and the super scripts correspond to the variables' indices within the vectors. The temporal difference learning algorithm modifies the weights at each iteration to reduce the error in the prediction function according the following rule [15]:

$$\begin{aligned} w_{k+1}^j &= w_k^j + \alpha[r_{k+1} + \gamma P_k(X_{k+1}) - P_k(X_k)]x_k^j \\ &= w_k^j + \alpha\delta_{k+1}x_k^j, \end{aligned} \tag{7}$$

where the temporal difference error term is

$$\delta_{t+1} = [r_{k+1} + \gamma P_k(X_{k+1}) - P_k(X_k)]. \tag{8}$$

3.2 Q-learning theory

The optimal policy of an MDP is a fixed point on the Bellman equation. This can be determined iteratively with value iteration or policy iteration approaches. In value iteration, $V(X_k)$ is the total expected discounted reward accumulated by a policy beginning at X_k. The Q function of a state-action pair, $Q(X_k, U_k)$, is the total expected discounted reward accrued by a policy that produces $U_k = \pi(X_k)$. Then, the Bellman equation can be rewritten such that the state-action value function is

$$Q(X_k, U_k) = E\{R(X_k, U_k) + \gamma V(X_{k+1})\}, \tag{9}$$

$$V(X_{k+1}) = \max_{U_{k+1} \in A} Q(X_{k+1}, U_{k+1}). \tag{10}$$

If Q and V satisfy the above Bellman equation, then the optimal greedy policy is

$$\pi^*(X_k) = \arg\max_{U_k \in A} Q(X_k, U_k). \tag{11}$$

The value-iteration approach uses equation (9) to find Q and V iteratively, which can then be used to find π^*.

If the transfer function $\mathcal{T}$ of the MDP is known, then value-iteration can be used to find π^*. However, if $\mathcal{T}$ is not known, Q-learning can be employed instead of value-iteration to learn an approximate state-action function $Q(X_k, U_k)$ that is iteratively updated by the rule:

$$\begin{aligned} Q(X_k, U_k) \leftarrow & \alpha(X_k, U_k) \times [r_{k+1} \\ & + \gamma \max_U Q(X_{k+1}, U) - Q(X_k, U_k)]. \end{aligned} \tag{12}$$

As displayed in equation (12), the learning rate, $\alpha_k(X_k, U_k)$, and the discount factor, γ, affect the generation of the new reward at each iteration. The learning rate determines the speed at which new information is assimilated by the controller and evaluated in order to execute a given action. The discount factor affects the controller's consideration of future rewards. At every iterative refreshment of the game state, the expected reward is evaluated for the range of actions U to give the next Q value for the time step $k+1$ [15].

Each state upon which an action is undertaken provides the player with a specific positive or negative reward. The ultimate goal of this player is therefore to optimize the reward by conducting the best possible action at each given state. In a situation where the system is as complex as it is in the game, with numerous variables and motives affecting the possible reward, a Q-function is not easily obtained. In this case, it is required to approximate the Q-function using an artificial neural network.

4 Mathematical models

Although model-free techniques exist in approximate dynamic programming (e.g., Q-learning), if the system behavior is known (or partially known) and can be represented mathematically, the amount of information that needs to be approximated and learned by the controller can be decreased. Here, we have derived models of the ghost adversaries in Ms. Pac-Man for use in the approximate λ-policy iteration method for computing an optimal control policy. The model was constructed using estimated ghost strategies found on various Pac-Man websites and was improved through trial and error.

Let $I_{\mathrm{G}} = \{I | I = r, p, b, o\}$ denote the ghosts' index set, where r, p, b, and o represent each of the four ghosts in the game. While in active pursuit, each of the ghosts chases Pac-Man in an individualized manner by utilizing different rules for choosing a set of target locations, denoted by x_{T}^{I}, which guide their decisions. The positions of the targets are functions of time and Pac-Man's state, where Pac-Man's position and control are represented by (1) and (2). The ghosts then share an identical algorithm for moving to their separate targets. The laws employed for determining the ghosts' targets are discussed as follows:

For Ghost 1, $I = r$, the target is assigned as the location of Pac-Man. This causes Ghost 1 to often chase the player from behind.

$$x_{\mathrm{T}}^{r}(t_k) = x_{\mathrm{p}}(t_k). \tag{13}$$

For Ghost 2, $I = p$, the target is set as the position slightly in front of Pac-Man, and the resulting behavior is an adversary that tries to attack from the front.

$$x_{\mathrm{T}}^{p}(t_k) = x_{\mathrm{p}}(t_k) + A_i d \ \text{ for } \ u_{\mathrm{p}}(t_k) = a_i, \tag{14}$$

where $d = [32 \ \ 32]^{\mathrm{T}}$ in units of pixels, and

$$\begin{cases} A_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \ A_2 = \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}, \\ A_3 = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}, \ A_4 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}. \end{cases} \tag{15}$$

For Ghost 3, $I = b$, and the target is a reflection of Ghost 1's position about Ghost 2's target. This causes Ghost 3 to seem like it attempts to guess Pac-Man's future paths.

$$x_{\mathrm{T}}^{b}(t_k) = [2 \cdot x_{\mathrm{R}}(t_k) - x_{\mathrm{G}}^{r}(t_k)], \tag{16}$$

where the reflection point, $x_{\mathrm{R}}(t_k)$ is,

$$x_{\mathrm{R}}(t_k) = x_{\mathrm{p}}(t_k) + A_i e, \ \ e = [16 \ \ 16]^{\mathrm{T}}. \tag{17}$$

For Ghost 4, $I = o$, the target is set at the bottom left corner of the maze if Pac-Man is near, and if Pac-Man is far away, the target becomes the location of Pac-Man itself. The orange ghost is the least threatening, as it often keeps its distance from Pac-Man, but its seemingly unpredictable behavior sometimes causes it to get in the way of the player's path unexpectedly.

$$x_{\mathrm{T}}^{o}(t_k) = \begin{cases} x_{\mathrm{B}} & \text{for } \|x_{\mathrm{G}}^{o}(t_k) - x_{\mathrm{p}}(t_k)\| \leqslant c, \\ x_{\mathrm{p}}(t_k) & \text{for } \|x_{\mathrm{G}}^{o}(t_k) - x_{\mathrm{p}}(t_k)\| > c, \end{cases} \ \forall k, \tag{18}$$

where $c = 80$ pixels, x_{B} denotes the position vector of the bottom left corner of the game maze, and $\| \cdot \|$ is the Euclidean norm.

After the target positions are calculated, the ghosts all utilize a common rule for moving toward their separate targets:

$$u_{\mathrm{G}}^{I}(t_k) = \begin{cases} a_i = H\{B\} \circ \mathrm{sgn}\{D\}, \\ \quad \text{for } a_i \in U_{\mathrm{G}}^{I}[x_{\mathrm{G}}^{I}(t_k)]; \\ a_j = H\{C\} \circ \mathrm{sgn}\{D\}, \\ \quad \text{for } a_i \notin U_{\mathrm{G}}^{I}[x_{\mathrm{G}}^{I}(t_k)], \ a_j \in U_{\mathrm{G}}^{I}[x_{\mathrm{G}}^{I}(t_k)]; \\ a_k = U_{\mathrm{G}}^{I}\{1\}, \\ \quad \text{for } a_i \notin U_{\mathrm{G}}^{I}[x_{\mathrm{G}}^{I}(t_k)], \ a_j \notin U_{\mathrm{G}}^{I}[x_{\mathrm{G}}^{I}(t_k)], \end{cases} \tag{19}$$

where $\circ$ denotes the Schur product, $H\{\cdot\}$ represents the Heaviside function, and U_{G}^{I} is the set of admissible actions for ghost I.

$$B = \begin{bmatrix} |x_{\mathrm{Gx}}^{I}(t_k) - x_{\mathrm{Tx}}^{I}(t_k)| & |x_{\mathrm{Gy}}^{I}(t_k) - x_{\mathrm{Ty}}^{I}(t_k)| \\ |x_{\mathrm{Gy}}^{I}(t_k) - x_{\mathrm{Ty}}^{I}(t_k)| & |x_{\mathrm{Gx}}^{I}(t_k) - x_{\mathrm{Tx}}^{I}(t_k)| \end{bmatrix}, \tag{20}$$

$$C = \begin{bmatrix} |x_{\mathrm{Gy}}^{I}(t_k) - x_{\mathrm{Ty}}^{I}(t_k)| & |x_{\mathrm{Gx}}^{I}(t_k) - x_{\mathrm{Tx}}^{I}(t_k)| \\ |x_{\mathrm{Gx}}^{I}(t_k) - x_{\mathrm{Tx}}^{I}(t_k)| & |x_{\mathrm{Gy}}^{I}(t_k) - x_{\mathrm{Ty}}^{I}(t_k)| \end{bmatrix}, \tag{21}$$

$$D = \begin{bmatrix} |x_{\mathrm{Tx}}^{I}(t_k) - x_{\mathrm{Gx}}^{I}(t_k)| \\ |x_{\mathrm{Ty}}^{I}(t_k) - x_{\mathrm{Gy}}^{I}(t_k)| \end{bmatrix}. \tag{22}$$

Since it can be seen in (19) that the ghosts will not choose an action opposite to their current action, they will not reverse direction on a path and will only effectively make decisions when they encounter intersections where there are three or more directions in which they can move.

5 Numerical verification of model

By recording the position of Pac-Man and the ghosts during gameplay on an emulated Ms. Pac-Man game found in [16] with a simple screen-capture program, it is possible to verify the models of the ghosts' behaviors by comparing these positions with those generated by the equations described above. This is done by setting the initial positions of Pac-Man and the ghosts in a simulated game to the positions recorded at some arbitrary instant during the real game. The simulated game is then run for a period of time, and the resulting trajectories of the ghosts are compared to those observed from the real game. If the ghost trajectories produced by the model match those recorded from the real game, then the model effectively represents the ghost behaviors. An example of the comparison is shown in Fig. 2.

It can be seen that the simulated ghosts behave very similarly to the ones in the real game. When the initial state of the simulated game was set to match the real game's state at an arbitrary instant, the decisions chosen and the resulting paths were almost always identical. The small number of errors that are present are predicted to be caused by slight imprecisions in the screen-capture approach when extracting the game state. These may have been due to the small amount of computation time needed to process the image or the inability to know the exact positions referenced by the game compared to the character images it displays. Fig. 3 shows how the path comparison of the light blue ghosts in Fig. 2 (c) appears with respect to time, and it can be seen that there is a small amount of error present. To calculate a numerical approximation of the model's accuracy, the simulated game was given an initial state from the real game and run until a ghost's decision from the simulated game differed from the corresponding ghost in the real game. The number of correct decisions was counted, and the process was repeated several times. After 10 runs, the model of the ghost behaviors correctly evaluated 818 decisions out of a total of 829. Therefore, the approximate accuracy of the model is 98.7%.

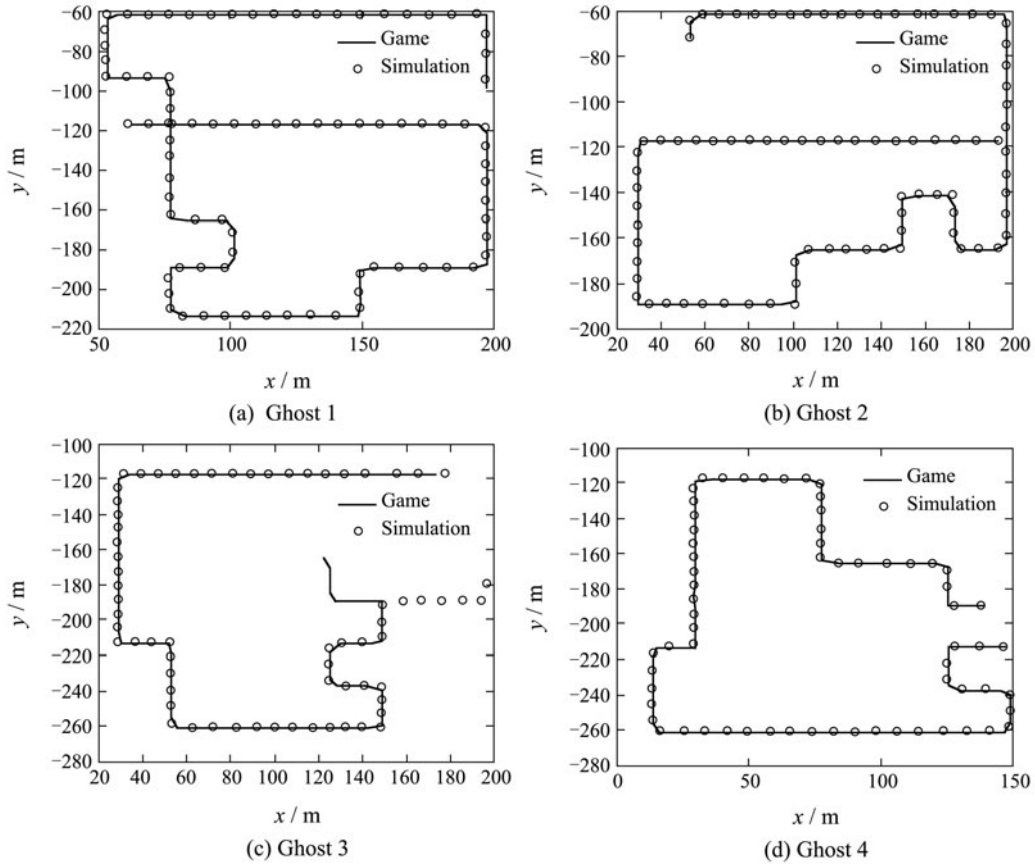

Fig. 2 Comparisons between the ghosts' paths observed in the Ms. Pac-Man game and those created using the derived model.

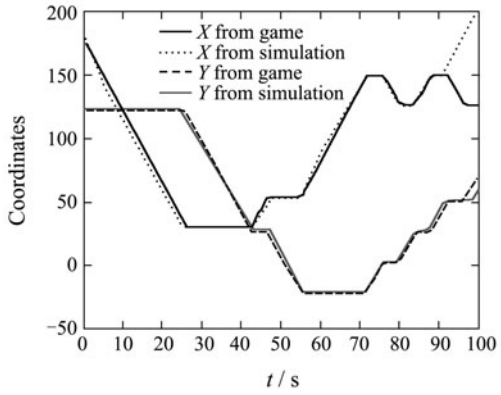

Fig. 3 Comparison between Ghost 3's paths in the Ms. Pac-Man game and using the derived model.

6 Methodology

The methodology presented in this paper for computing an optimal strategy for an automated player of Ms. Pac-Man can be summarized as follows: The workspace $\mathcal{W}$ is decomposed into rectangloids using a line-sweeping approach. From the decomposition, a connectivitiy tree, $\mathcal{T}$, is formed using the adjacency relationships between cells and the agent cell position, $\kappa_{\rm p}$. Each branch in the connectivity tree represents a path extending from $\kappa_{\rm p}$ and corresponds to a sequence of possible actions. The policy used by the agent to choose the optimal path given an initial state is evaluated via a temporal difference-based approximate λ-policy iteration method.

6.1 Cell decomposition and connectivity tree

Cell decomposition is a well-known robotic path planning method used for obstacle avoidance [17, 18]. The approach decomposes a workspace into a finite set of non-overlapping convex polygons, known as cells, such that each cell represents a subset of the workspace in which the agents and adversaries can move freely without colliding with an obstacle. In classical cell decomposition, this can be obtained by using a line-sweeping algorithm and constructing a one-dimensional representation of the free-space geometry known as a connectivity graph.

The workspace of Ms. Pac-Man, $\mathcal{W}$, is decomposed, as shown in Fig. 4, using an approach that has an added property compared to classical decomposition, namely, a unique set of admissible actions from (3) is associated with each cell, and Pac-Man and the ghosts can perform those actions anywhere inside the cell. Let κ_j denote a cell in the decomposition, and I_κ represent the index set of all cells in the decomposition of $\mathcal{W}$ and in the corresponding connectivity graph denoted by $\mathcal{G}$ and illustrated in Fig. 5.

Definition 1 A connectivity graph, $\mathcal{G}$, is a non directed graph where the nodes represent rectangloid cells in the cell decomposition, and two nodes κ_i and κ_j in $\mathcal{G}$ are connected by an arc (κ_i, κ_j) if and only if the corresponding cells are adjacent in the decomposition.

Several methods can be used to search $\mathcal{G}$ for sequences of adjacent cells, connecting possible paths for Pac-Man within the maze. Based on a maximum feasible distance, the connectivity graph can be pruned and transformed into

a decision tree, which is a graphical representation of all possible paths from $\kappa_{\rm p}$. After visiting a cell, Pac-Man can only move to an adjacent cell, creating a causal process that can be represented as follows.

Definition 2 The connectivity tree $\mathcal{T}$ associated with $\mathcal{G}$ and the agent's cell position, $\kappa_{\rm p}$, is a tree graph with $\kappa_{\rm p}$ as the root and branches with length L. The nodes represent cells where travel is possible, and a branch represents a non-reversing path extending from $\kappa_{\rm p}$.

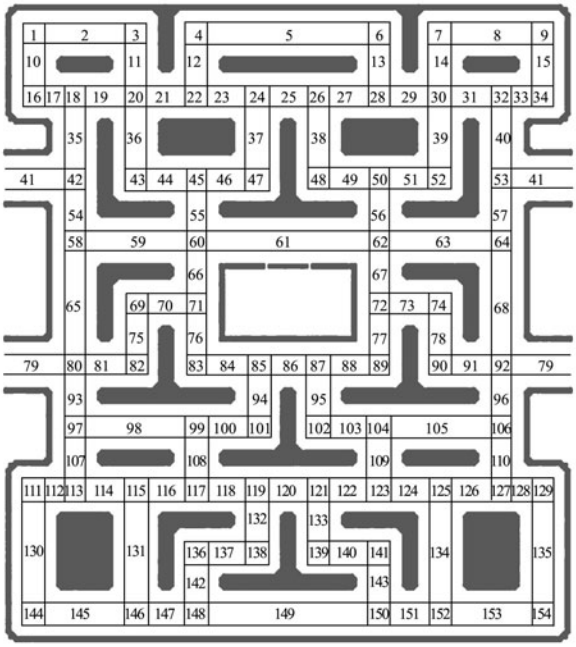

Fig. 4 Cell decomposition of the Level 1 maze.

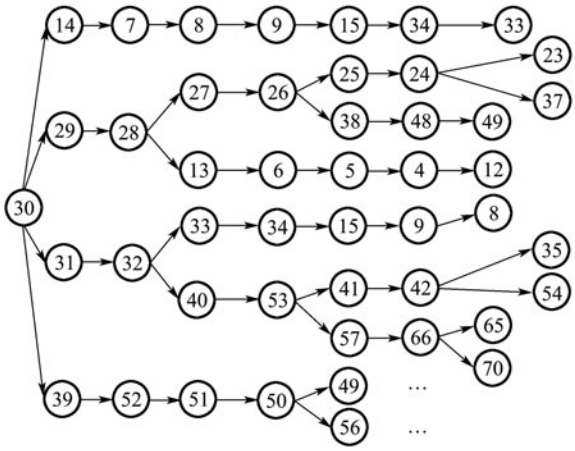

Fig. 5 Connectivity tree of the cell decomposition.

Based on the cell properties, it can be seen that a unique action value is attached to each arc in $\mathcal{T}$. Therefore, the set of admissible actions is a function of $x_{\rm p}$ and of $u_{\rm p}(t_k-1)$, and it is denoted by $U_{\rm p}[x_{\rm p}(t_k)] \subset \mathcal{U}$. Since the agent is not allowed to immediately reverse direction in $\mathcal{T}$, the set of admissible actions is given by the complement of $u_{\rm p}(t_{k-1})$ in the set of arcs attached to the cell occupied at time t_k, in $\mathcal{G}$.

For the example workspace, $\mathcal{W}$, in Fig. 4, the connectivity tree is illustrated in Fig. 5. Since the size of the tree used in this paper is very large, only the first few layers of the tree are shown, where layers are defined by their adjacency position relative to the initial cell. For the example problem, the connectivity tree is limited to prohibit decisions involving reversals in the middle of a cell or actions resulting in no movement. However, these capabilities can be added by incorporating tree reflections or additional repeating nodes.

6.2 Optimal agent strategy

Based on the above problem formulation, we seek an optimal strategy μ^* defined as a sequence of functions,

$$\mu^* = c_i, \ldots, c_F, \tag{23}$$

where each function c_k maps the state $x_{\rm p}$ into an admissible decision,

$$u_{\rm p}(t_k) = c_k[x_{\rm p}(t_k)] \ \text{ for } \ k = i, \ldots, F \tag{24}$$

and maximizes the reward-to-go,

$$J_{i,F}[x_{\rm p}(t_i)] \equiv \sum_{k=i}^{F} \alpha_k \mathcal{L}[x_{\rm p}(t_k), u_{\rm p}(t_k)] \tag{25}$$

from the present time, t_i, up to the final time, t_F, over the finite horizon $[t_i, t_F]$, where $\mathcal{L}[\,\cdot\,]$ is Pac-Man's instantaneous reward associated with visiting a cell $\kappa_j \in \mathcal{G}$, and α_k is a discount factor that is defined as an exponential function of k, such that future rewards are discounted, compared to immediate ones.

Since the connectivity tree effectively amounts to a decision tree, as the tree is grown and the instantaneous rewards are computed and attached to each node, along with $\mathcal{L}$, the cumulative reward,

$$\begin{aligned} J_{i,k}[x_{\rm p}(t_i)] &\equiv \sum_{j=i}^{k} \alpha_j \mathcal{L}[x_{\rm p}(t_j), u_{\rm p}(t_j)] \\ &= J_{i,k-1}[x_{\rm p}(t_i)] + \alpha_k \mathcal{L}[x_{\rm p}(t_k), u_{\rm p}(t_k)] \end{aligned} \tag{26}$$

can also be computed for each node at each time step, t_k in $\mathcal{T}$ iteratively over time, where as it can be seen inFig. 5 that each node in the time step t_k denotes one of the possible values of Pac-Man's state $x_{\rm p}(t_k)$, and each outgoing arc denotes one of the possible values of Pac-Man's control $u_{\rm p}(t_k)$. Therefore, by the time the tree is completed, the cost-to-go, $J_{i,F}$, for each branch (i.e., each possible sequence of state and decision values) will be attached to the leaf of the branch (i.e., the last node, at t_F), and, thus the optimal branch can be determined simply by picking the largest value or $J_{i,f}$, namely, $J^*_{i,F}$. The optimal branch will then determine the optimal strategy μ^*, which will consist of the actions corresponding to the sequence of arcs in the optimal branch.

6.3 Approximate λ-policy iteration

In the game of Ms. Pac-Man, the ghosts are seen to exhibit random behavior for a short period of time after a power pill has been eaten, and Pac-Man has the ability to chase and eat the ghosts. Therefore, the agent must often determine the optimal set of actions with only a probabilistic knowledge of future game states. To handle the existance of stochastic effects in a system or a system with only a partial model available, the connectivity tree approach can be modified by adding learning components to the instantaneous reward function and training them through a temporal difference-based learning method called approximate λ-policy iteration adapted from that in [13, 14].

Due to the large state space associated with the game, the reward function is represented with a linear feature-based

approximation architecture of the general form,

$$\tilde{\mathcal{L}}(i,w) = w(0) + \sum_{a=1}^{A} w(a)\phi_a(i), \tag{27}$$

where $\phi_a, a = 0, 1, \ldots, A$ are features of the state i, and $w(a)$ are the components of the weight vector w. The weight $w(0)$ is a bias. The values of w give an approximation of the optimal reward-to-go function, $\mathcal{L}(i,w)$.

For the Ms. Pac-Man example, the reward function is approximated as

$$\begin{aligned}\tilde{\mathcal{L}}(i,w) = {} & w(0) + w(1)(1-z(i))R(i) \\ & + w(2)z(i)R(i) + w(3)V(i) \\ & + w(4)R_{\mathrm{PP}}(i),\end{aligned} \tag{28}$$

where $V(i)$ is the number of dots in the corresponding cell at the time Pac-Man would visit that cell, $z(i)$ is 1 if a power pill is in the cell and 0 otherwise, and $R(i)$ is the risk function that measures the distances from Pac-Man to the ghosts

$$R[x_{\mathrm{p}}(t_k), u_{\mathrm{p}}(t_k)] = \sum_{\ell \in I_{\mathrm{G}}} [|x_{\mathrm{p}}(t_k) - x_{\mathrm{G}}^{\ell}| - \rho_0]^2, \tag{29}$$

where $|\cdot|$ is the Manhattan distance, and ρ_0 is a user-defined parameter, such that when $[x_{\mathrm{p}}(t_k) - x_{\mathrm{G}}^{\ell}(t_k)] \to \rho_0, R \to 0$. The ghost states at the time corresponding to the layer of the tree are evaluated using the validated equations in Section 4. The risk function for ghosts after a power pill has been eaten, $R_{\mathrm{PP}}(i)$, has the same form as (29) but measures the distances to the fleeing ghosts. Note that $z(i)$ switches the terms with $R(i)$ on and off since the distance to the ghosts will have a negative reward during normal play to avoid being caught but have a positive reward when considering moving to a power pill because a smaller distance increases the chances of eating ghosts as they flee.

A policy maps a given state to a corresponding admissible action, and it can be written as

$$\mu_j(i) = \arg \max_{u \in U(i)} \sum_{\kappa_j \in \mathcal{G}} (g(i,u) + \alpha \mathcal{L}(i, w_j)), \ \ \forall i, \tag{30}$$

where w_j is the weight vector after j policy updates. The function $g(i,u)$ is a known immediate reward that results from the control u at state i. This is set as the corresponding increase in points that the agent achieves in the game (following the game rules described above) when the control u is applied from state i. The policy μ_j is updated by training over a batch of M games and modifying w_j. The number of games, M, is on the order of 100. Over each iteration, the weights are updated according to

$$\begin{aligned}w_{j+1} = \arg\max_{w} \sum_{m=1}^{M} \sum_{b=0}^{Bm} & [\tilde{\mathcal{L}}(i_{m,b}, w) \\ & - \tilde{\mathcal{L}}(i_{m,b}, w_j) - \sum_{c=b}^{Bm-1} \lambda^{c-b} d(i_{m,c}, i_{m,c+1})]^2,\end{aligned} \tag{31}$$

where $(i_{m,0}, i_{m,1}, \ldots, i_{m,Bm-1}, i_{m,Bm})$ denotes the sequence of states in the mth game of the batch, and $i_{m,Bm}$ is the termination state. The termination state occurs when the agent has been caught by a ghost, and the resulting terminal cost is $\tilde{\mathcal{L}}(i_{m,Bm}, w_j) = 0$. The temporal differences are

$$\begin{aligned}d(i_{m,c}, i_{m,c+1}) = {} & g(i_{m,c}, \mu_j(i_{m,c}), i_{m,c+1}) \\ & + \tilde{\mathcal{L}}(i_{m,c+1}, w_j) - \tilde{\mathcal{L}}(i_{m,c}, w_j).\end{aligned} \tag{32}$$

Training is implemented by simulating the game starting at the initial state shown in Fig. 1. The game ends when a ghost catches the agent. If all of the dots and power pills have been eaten by Pac-Man (clearing the maze), they are restored, and the game continues until the ghosts catch the agent. However, with each clearing of the maze, the ghost speeds are increased slightly, so the probability of the game terminating is 1. The policy iteration described is run until the policy converges to a stationary optimal policy.

6.4 *Q*-learning player

A model-free neural player trained with Q-learning was constructed to be used only for comparison with the approximate λ-policy iteration architecture. The neural player does not use the mathematical models from Section 5, rather the connectivity tree from Section 6.1 is used with a shorter branch length. The state inputs to the artificial neural network are variables that are preprocessed using the connectivity tree in an attempt to pass the most relevant information possible to the controller. The decision by the agent is then which branch to take or, more specifically, which target cell to move toward. After some experimentation, the inputs were selected as follows:

1) If Ghost 1/2/3/4 is on the branch, $q(1)$ (4 variables). If true, $q(1) = 1$. If false, $q(1) = 0$.

2) Number of fleeing ghosts on branch, $q(2)$.

3) The Manhatten distance in pixels that Ms. Pac-Man would arrive at the target cell ahead of each ghost if the ghost takes the shortest path, $q(3-6)$ (four variables).

4) The Manhatten distance in pixels that Ms. Pac-Man would arrive at the target cell ahead of each fleeing ghost if the ghost takes the shortest path, $q(7-10)$ (four variables).

5) Branch length in pixels, $q(11)$.

6) Number of dots on branch, $q(12)$.

7) Number of power pills on branch, $q(13)$.

8) Euclidean distance from target cell to the nearest cell containing pills, $q(14)$.

A two-layer feed-forward sigmoidal neural network with 100 hidden neurons was used for the Q-function approximator. The network was trained with the MATLAB gradient descent adaptive learning algorithm, traingda.

7 Results

To illustrate the effectiveness of the model-based approximate λ-policy iteration approach presented in this paper, the method was run on an accurate simulation of the game and compared to the model-free neural player described in 6.4 and average human players. The game simulation was created in MATLAB and based on the verified equations in Section 4, the maze map from the first level, and a knowledge of the game mechanics. The simulation differs slightly from the real game in that some small features were removed, such as the appearance of the bonus 'fruits' which are objects that move randomly across the maze and give the player extra points if the object is eaten, and the slower movement is experienced by the ghosts when traveling through a 'tunnel' which is a wrap-around path that allows characters to quickly move to the opposite side of the maze. In addition, the ghosts' speeds have been altered manually and initially set to 95% of Pac-Man's speed, which are approximately the speeds seen in the fifth maze of the real game. Screenshots from the simulation's graphical output

during a single run are displayed in Fig. 6.

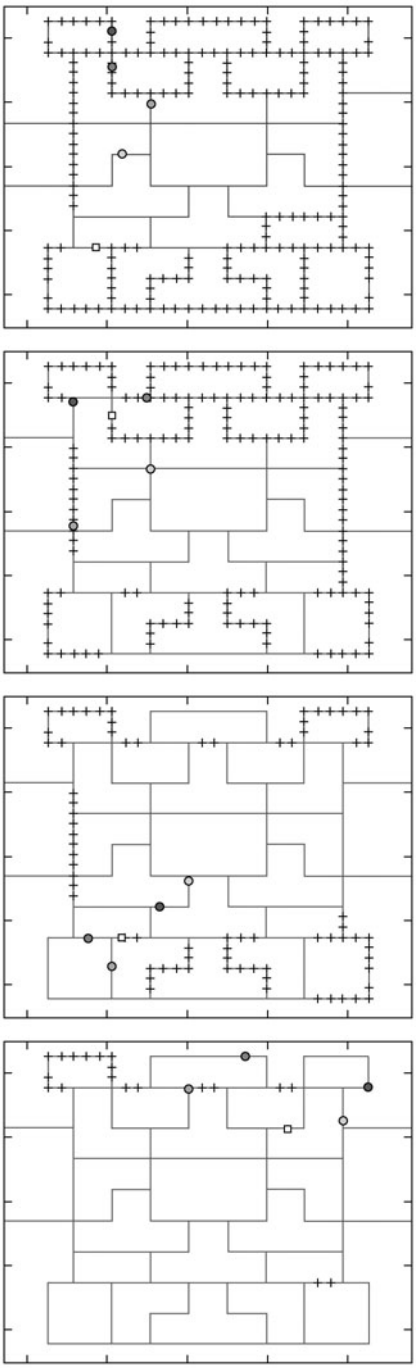

Fig. 6 Screenshots from the simulation's graphical output during a single run. The square circle represents Pac-Man, and the other circles correspond to the ghosts of similar color. The dots are represented by black crosses.

Like the training process described in Section 5, the game begins with the initial state shown in Fig. 1. If the agent is able to clear the entire maze, the dots and power pills are reset, the ghosts' speeds are increased by 5%, and the game continues until a ghost catches Pac-Man. Also, contrary to the real game, the agent is only given one 'life' (attempt) per game.

The objective of the agent is simply to earn the most points possible. Points are scored as follows: Each dot eaten is worth 10 points. Each power pill eaten is worth 50 points. When a power pill is active and the ghosts are vulnerable, eating a ghost earns 200×2^n points for the nth ghost eaten during a single power pill's time interval. In other words, if all four ghosts are eaten in succession, $200 + 400 + 800 + 1600 = 3000$ points would be earned.

After training, the game simulation was run 20 times using each approach, and the results are displayed in Table 1. Two human players were also asked to play the simulation with the same objectives, and the average of their scores is shown. The approach combining the connectivity tree and temporal difference-based policy iteration performed the evasion, collection, and pursuit objectives well, and its scores were much higher than that of the model-free Q-learning method and the human players. There are initially 220 dots in the maze, so note that the presented approach was capable of clearing at least one maze on most of its attempts before being caught by the ghosts. This level of success is significantly greater than what has been accomplished with most existing automated players that are not hand coded [7–10]. One of the major strengths of the method is its ability to plan optimal paths relatively far into the future, which is a common shortfall shared by the currently highest scoring programs [11, 12].

Table 1 Comparison of automated and human players over 20 partial game simulations.

	Average score	High score	Average dots eaten
λ-PI	4708	7710	281
Q-learning	2395	4340	152
Human	2279	3670	142

The methods were also tested with a slightly different game format to focus on the skills that may be more relevant to applications outside of Ms. Pac-Man. Since it is unlikely that evaders will need to occasionally chase their pursuers in the large majority of evasion problems, the power pills were removed from the maze, and the agents and human players were simply given the objectives to evade the ghosts and clear all dots from the maze. If the maze was cleared, the game was stopped and restarted. Like the other test, only one life was given, the bonus fruits did not appear, and the ghosts did not slow down when passing through the tunnels. The test was run at three different ghost speeds: 95%, 100%, and 105% of Pac-Man's speed. The results are shown in Table 2. Again, the model-based λ-policy iteration performed best. Impressively, it was able to clear 3 out of 20 mazes with the ghosts moving much faster than Pac-Man. Note that these conditions are not seen until the very late stages in the real game.

Given that the ghost speeds in both tests were set to be considerably faster than the speeds seen in the first several levels of the game, it is shown that the performance of the presented method is very difficult for most human players to match. However, the approach has flaws that make it weaker than the average human player in some situations. For example, the method does not allow Pac-Man to reverse direction in the middle of a cell or to stop moving. This lack of capability caused many of its failures to occur when the agent moved itself from a safe situation into a dangerous one because it was not able to evaluate other options. This skill

is also essential to lure the ghosts near a power pill, so that many ghosts can be eaten in succession, and a large score can be earned. Additions can be made to the connectivity tree that would effectively address most of the shortcomings, assuming the computational demands would remain feasible.

Table 2 Performance of artificial and human Pac-Man players over 20 partial game simulations with power pills removed.

Ghost speed/%	Model-based approximate λ-policy iteration		Model-free Q-learning		Human player	
	Mazes cleared	Average dots eaten	Mazes cleared	Average dots eaten	Mazes cleared	Average dots eaten
95	18	212	1	144	4	161
100	14	197	0	109	1	105
105	3	137	0	84	0	88

8 Conclusions

This paper presents a model-based approximate λ-policy iteration method for optimizing paths online for a pursuit-evasion problem, where an agent must visit several target positions within a region of interest while simultaneously avoiding one or more actively pursuing adversaries. The methodology described utilizes cell decomposition to construct a decision tree and implements a temporal difference-based approximate λ-policy iteration to combine prior knowledge through modeling with online learning to achieve the objectives of minimizing the risk of being caught by an adversary and maximizing a reward associated with visiting target locations. The approach is illustrated through a modified version of the video game Ms. Pac-Man, which is shown to be a benchmark example of the pursuit-evasion problem. It was shown that the model-based approximate λ-policy iteration approach outperforms several other methods as well as most human players.

References

[1] S. M. Lucas, G. Kendall. Evolutionary computation and games. *IEEE Computer Intelligence Magazine*, 2006, 1(1): 10 – 18.

[2] J. Latombe. *Robot Motion Planning*. Boston: Kluwer Academic Publishers, 1998.

[3] Z. Sun, J. Reif. On robotic optimal path planning in polygonal regions with pseudo-euclidian metrics. *IEEE Transacions on Systems, Man, and Cybernetics – Part A*, 2007, 37(4): 925 – 936.

[4] D. Culler, D. Estrin, M. Srivastava. Overview of sensor networks. *Computer*, 2004, 37(8): 41 – 49.

[5] S. Megerian, F. Koushanfar, G. Qu, et al. Exposure in wireless sensor networks: Theory and practical solutions. *Wireless Networks*, 2002, 8(5): 443 – 454.

[6] V. Phipatanasuphorn, P. Ramanathan. Vulnerability of sensor networks to unauthorized traversal and monitoring. *IEEE Transactions on Computers*, 2004, 53(3): 365 – 369.

[7] S. Lucas. Evolving a neural network location evaluator to play ms. pac-man. *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games*, Piscataway: IEEE, 2005: 203 – 210.

[8] M. Gallagher, A. Ryan. Learning to play pac-man: An evolutionary, rule-based approach. *Proceedings of the Congress on Evolutionary Computation (CEC)*, Piscataway: IEEE, 2003: 2462 – 2469.

[9] P. Burrow, S. Lucas. Evolution versus temporal difference learning for learning to play ms. pac-man. *Proceedings of the 5th International Conference on Computational Intelligence and Games*, Piscataway: IEEE, 2009: 53 – 60.

[10] L. DeLooze, W. Viner. Fuzzy Q-learning in a nondeterministic environment: developing an intelligent ms. pac-man agent. *Proceedings of the 5th International Conference on Computational Intelligence and Games*, Piscataway: IEEE, 2009: 162 – 169.

[11] M. Emilio, M. Moises, R. Gustavo, et al. Pac-man: Optimization based on ant colonies applied to developing an agent for ms. pac-man. *Proceedings of IEEE Conference on Computational Intelligence and Games*, Piscataway: IEEE, 2010: 458 – 464.

[12] R. Thawonmas, T. Ashida. Evolution strategy for optimizing parameters in ms. pac-man controller ice pambrush 3. *Proceedings of IEEE Conference on Computational Intelligence and Games*, Piscataway: IEEE, 2010: 235 – 240.

[13] D. P. Bertsekas, S. Ioffe. *Temporal Differences-based Policy Iteration and Applications in Neuro-dynamic Programming*. Report LIDS-P-2349. Cambridge: Laboratory for Information and Decision Systems, MIT, 1996.

[14] A. Nedich, D. P. Bertsekas. Least-squares policy evaluation algorithms with linear function approximation. *Journal of Discrete Event Systems*, 2003, 13(1/2): 79 – 110.

[15] S. J. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach*. New York: Prentice Hall, 2003.

[16] *Ms. Pac-Man Game*. http://webpacman.com/.

[17] D. Zhu, J. C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 1991, 7(1): 9 – 20.

[18] K. Kadem, M. Sharir. An efficient motion planning algorithm for convex polygonal object in 2-dimensional polygonal space. *Courant Institute of Mathematical Science*, 1990, 5(1): 43 – 75.

Greg FODERARO received his B.S. degree in Mechanical Engineering from Clemson University, Clemson, SC in 2009 and is pursuing a Ph.D. degree at Duke University, Durham, NC and working in the Laboratory for Intelligent Systems and Controls. His primary research interests are underwater sensor networks, robot path planning, optimal control, spiking neural networks, and artificial intelligence. E-mail: greg.foderaro@duke.edu.

Vikram RAJU is a senior at Duke University, Durham, NC and pursuing his B.S. degree in Mechanical Engineering. He is working in the Laboratory for Intelligent Systems and Controls, and his main research interests include robot path planning and artificial intelligence. E-mail: vvr2@duke.edu.

Silvia FERRARI received her B.S. degree from Embry-Riddle Aeronautical University, Daytona Beach, FL, and her M.A. and Ph.D. degrees from Princeton University, Princeton, NJ. She is an associate professor of Mechanical Engineering and Materials Science with the Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC, where she directs the Laboratory for Intelligent Systems and Controls. Her principal research interests include robust adaptive control of aircraft, learning and approximate dynamic programming, and optimal control of mobile sensor networks. Dr. Ferrari is a member of ASME, SPIE, and AIAA. She is the recipient of the ONR Young Investigator Award (2004), the NSF CAREER Award (2005), and the Presidential Early Career Award for Scientists and Engineers Award (2006). E-mail: sferrari@duke.edu.