Play Ms. Pac-Man using an advanced reinforcement learning agent

Nikolaos Tziortziotis Konstantinos Tziortziotis
Konstantinos Blekas

March 3, 2014

Abstract

Reinforcement Learning (RL) algorithms have been promising methods for designing intelligent agents in games. Although their capability of learning in real time has been already proved, the high dimensionality of state spaces in most game domains can be seen as a significant barrier. This paper studies the popular arcade video game Ms. Pac-Man and outlines an approach to deal with its large dynamical environment. Our motivation is to demonstrate that an abstract but informative state space description plays a key role in the design of efficient RL agents. Thus, we can speed up the learning process without the necessity of Q-function approximation. Several experiments were made using the multi-agent MASON platform where we measured the ability of the approach to reach optimum generic policies which enhances its generalization abilities.

Keywords: Intelligent Agents, Reinforcement Learning, Ms. Pac-Man

1 Introduction

During the last two decades there is a significant research interest within the AI community on constructing intelligent agents for digital games that can adapt to the behavior of players and to dynamically changed environments [1]. Reinforcement learning (RL) covers the capability of learning from experience [2–4], and thus offers a very attractive and powerful platform for learning to control an agent in unknown environments with limited prior knowledge. In general, games are ideal test environments for the RL paradigm, since they are goal-oriented sequential decision problems, where each decision can have long-term effect. They also hold other interesting properties, such as random events, unknown environments, hidden information and enormous decision spaces, that make RL to be well suited to complex and uncertain game environments.

In the literature there is a variety of computer games domains that have been studied by using reinforcement learning strategies, such as *chess*, *backgammon* and *tetris* (see [5] for a survey). Among them, the arcade video game Ms. Pac-Man constitutes a very interested test environment. Ms. Pac-Man was released in early 80's and since then it has become one of the most popular video games of all time. That makes Ms. Pac-Man very attractive is its simplicity of playing in combination with the complex strategies that are required to obtain a good performance [6].

The game of Ms. Pac-Man meets all the criteria of a reinforcement learning task. The environment is difficult to predict, because the ghost behaviour is stochastic and their paths are unpredictable. The reward function can be easily defined covering particular game events and score requirements. Furthermore, there is a small action space consisting of the four directions in which Ms. Pac-Man can move (up, down, right, left) at each time step. However, a difficulty is encountered when designing the state space for the particular domain. Specifically, a large amount of features are required for describing a single game snapshot. In many cases this does not allow reaching optimal solutions and may limit the efficiency of the learning agent. Besides, a significant issue is whether the state description can fit into the memory, and whether optimization can be solved in reasonable time or not. In general, the size of the problem may grow exponentially with the number of variables. Therefore working efficiently in a reinforcement learning framework means reducing the problem size and establishing a reasonable state representation.

To tackle these disadvantages several approximations, simplifications and/or feature extraction techniques have been proposed. In [6] for example, a rule-based methodology was applied where the rules were designed by the human and their values were learned by reinforcement learning. On the other hand, neural networks have been also employed for value function approximation with either a single or multiple outputs [7,8]. Further search techniques have been applied to developing agents for Ms. Pac-Man, including genetic programming [9], Monte-Carlo tree search [10, 11] and teaching advising techniques [12].

In this study we investigate the Ms. Pac-Man game since it offers a real time dynamic environment and it involves sequential decision making. Our study is focused on the designing of an appropriate state space for building an efficient RL agent to the MS. Pac-Man game domain. The proposed state representation is informative by incorporating all the necessary knowledge about any game snapshot. At the same time it presents an abstract description so as to reduce computational cost and to accelerate learning procedure without compromising the decision quality. We demonstrate here that providing a proper feature set as input to the learner is of outmost importance for simple reinforcement learning algorithms, such as SARSA. The last constitutes the main contribution of our study and it suggests the need of careful modeling of the domain aiming at addressing adequately the problem. Several experiments have been conducted where we measured the learning capabilities of the proposed methodology and its efficiency in discovering optimal policy in unknown mazes. It should be emphasized that, although different Pac-Man simulators have been applied within the literature and a direct head-to-head comparison of the performance is not practical, we believe that our method yields very promising results with considerable improved performance.

The remaining of this paper is organized as follows: In section 2 we give a brief description of the Ms. Pac-Man game environment. Section 3 describes the background of the reinforcement learning schemes and presents some preliminaries about the general temporal-difference (TD) scheme used for training the proposed Ms. Pac-Man agent. The proposed state space structure is presented at section 4 while the details of our experiments together with some initial results are illustrated in section 5. Finally, section 6 draws conclusions and discusses some issues for future study.

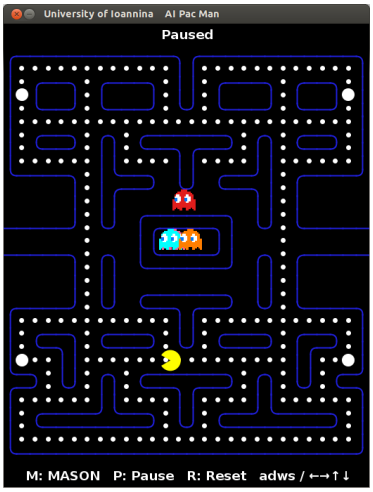


Figure 1: A screenshot of the Pac-Man game in a typical maze (*Pink maze*)

2 The game of Pac-Man

Pac-Man is an 1980s arcade video-game that reached immense success. It is considered to be one of the most popular video games to date. The player maneuvers Ms. Pac-Man in a maze that consists of a number of dots (or pills). The goal is to eat all of the dots. Figure 1 illustrates a typical such maze. It contains 220 dots with each of them to worth 10 points. A level is finished when all the dots are eaten ('win'). There are also four ghosts in the maze who try to catch Ms. Pac-Man, and if they succeed, Pac-Man loses a life.

Four power-up items are found in the corners of the maze, called power pills, which are worth 40 points each. When Ms. Pac-Man consumes a power-pill all ghosts become edible, i.e. the ghosts turn blue for a short period (15 seconds), they slow down and try to escape from Ms. Pac-Man. During this time, Ms. Pac-Man is able to eat them, which is worth 200, 400, 800 and 1600 points, consecutively. The point values are reset to 200 each time another power pill is eaten, so the player would want to eat all four ghosts per power dot. If a ghost is eaten, it remains hurry back to the center of the maze where the ghost is reborn. Our investigations are restricted to learning an optimal policy for the maze presented at Fig. 1, so the maximum achievable score is $220 \times 10 + 4 \times 40 + 4 \times (200 + 400 + 800 + 1600) = 14360$. [1]

In the original version of Pac-Man, ghosts move on a complex but deterministic route, so it is possible to learn a deterministic action sequence that does not require any observations. In the case of Ms. Pac-Man, randomness was added to the movement of the ghosts. Therefore there is no single optimal action sequence and observations are necessary for optimal decision making. In our case ghosts moved randomly in 20% of the time and straight towards Ms. Pac-Man in the remaining 80%, but ghosts may not turn back. Ms. Pac-Man starts playing the game with three lives. An additional life is given at 10000

[1] In the original version of the game, a fruit appears near the center of the maze and remains there for a while. Eating this fruit is worth 100 points.

points.

It must be noted that, although the domain is discrete it has a very large state space. There are 1293 distinct locations in the maze, and a complete state consists of the locations of Pac-Man, the ghosts, the power pills, along with each ghosts previous move and whether or not it is edible.

3 Reinforcement learning

In the reinforcement learning (RL) framework an agent is trained to perform a task by interacting with an unknown environment. While taking actions, the agent receives feedback from the environment in the form of rewards. The notion of RL framework is focused on gradually improving the agent's behavior and estimating its *policy* by maximizing the total long-term expected reward. An excellent way for describing a RL task is through the use of Markov Decision Processes.

A *Markov Decision Process* (MDP) [13] can be supposed as a tuple ($\mathcal{S}$, $\mathcal{A}$, P, R, γ), where $\mathcal{S}$ is a set of states; $\mathcal{A}$ a set of actions; $P : \mathcal{S} \times \mathcal{A} \times S \to [0, 1]$ is a Markovian transition model that specifies the probability, $P(\boldsymbol{s}, a, \boldsymbol{s}')$, of transition to a state $\boldsymbol{s}'$ when taken an action a in state $\boldsymbol{s}$; $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function for a state-action pair; and $\gamma \in (0, 1)$ is the discount factor for future rewards. A *stationary policy*, $\pi : \mathcal{S} \to \mathcal{A}$, for a MDP is a mapping from states to actions and denotes a mechanism for choosing actions. An *episode* can be supposed as a sequence of state transitions: $< \boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_{\mathrm{T}} >$. An agent repeatedly chooses actions until the current episode terminates, followed by a reset to a starting state.

The notion of *value function* is of central interest in reinforcement learning tasks. Given a policy π, the value $V^{\pi}(\boldsymbol{s})$ of a state $\boldsymbol{s}$ is defined as the expected discounted returns obtained when starting from this state until the current episode terminates following policy π:

$$V^{\pi}(\boldsymbol{s}) = E\left[\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t) | \boldsymbol{s}_0 = \boldsymbol{s}, \pi\right] . \tag{1}$$

As it is well-known, the value function must obey the *Bellman's equation*:

$$V^{\pi}(\boldsymbol{s}) = E_{\pi}\left[R(\boldsymbol{s}_t) + \gamma V^{\pi}(\boldsymbol{s}_{t+1}) | \boldsymbol{s}_t = \boldsymbol{s}\right], \tag{2}$$

which expresses a relationship between the values of successive states in the same episode. In the same way, the state-action value function (Q-function), $Q(\boldsymbol{s}, a)$, denotes the expected cumulative reward as received by taking action a in state $\boldsymbol{s}$ and then following the policy π,

$$Q^{\pi}(\boldsymbol{s}, a) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t) | \boldsymbol{s}_0 = \boldsymbol{s}, a_0 = a\right] . \tag{3}$$

In this study, we will focus on the Q functions dealing with state-action pairs $(\boldsymbol{s}, a)$.

The objective of RL problems is to estimate an optimal policy π^* by choosing actions that yields the optimal action-state value function Q^*:

$$\pi^*(\boldsymbol{s}) = \arg\max_{a} Q^*(\boldsymbol{s}, a). \tag{4}$$

Learning a policy therefore means updating the Q-function to make it more accurate. To account for potential inaccuracies in the Q-function, it must perform occasional exploratory actions. A common strategy is the ϵ-greedy exploration, where with a small probability ϵ, the agent chooses a random action. In an environment with a capable (reasonably small) number of states, the Q-function can simply be represented with a table of values, one entry for each state-action pair. Thus, basic algorithmic RL schemes make updates to individual Q-value entries in this table.

One of the most popular TD algorithms used in on-policy RL is the SARSA [4] which is a *bootstrapping* technique. Assuming that an action a_t is taken and the agent moves from belief state $\boldsymbol{s}_t$ to a new state $\boldsymbol{s}_{t+1}$ while receiving a reward r_t, a new action a_{t+1} is chosen (ϵ-greedy) according to the current policy. Then, the predicted Q value of this new state-action pair is used to calculate an improved estimate for the Q value of the previous state-action pair:

$$Q(\boldsymbol{s}_t, a_t) \leftarrow Q(\boldsymbol{s}_t, a_t) + \alpha\delta_t, \tag{5}$$

where

$$\delta_t = (r_t + \gamma Q(\boldsymbol{s}_{t+1}, a_{t+1}) - Q(\boldsymbol{s}_t, a_t)) \tag{6}$$

is known as the one step temporal-difference (TD) error. The term α is the learning rate which set to some small value (e.g. $\alpha = 0.01$) and can be occasionally decreased during the learning process.

An additional mechanism that can be employed is that of *eligibility traces*. This allows rewards to backpropagate to recently visited states, allocating them some proportion of the current reward. Every state-action pair in the Q table is given its own eligibility value (e) and when the agent visits that pairing its eligibility value set equal to 1 (*replacing traces*, [14]). After every transition all eligibility values are decayed by a factor of $\gamma\lambda$, where $\lambda \in [0, 1]$ is the trace decay parameter. The TD error forward proportional in all recently visited state-action pairs as signalised by their nonzero traces according to the following update rule:

$$Q_{t+1}(\boldsymbol{s}, a) \leftarrow Q_t(\boldsymbol{s}, a) + \alpha\delta_t e_t(\boldsymbol{s}, a) \qquad \text{for all } \boldsymbol{s}, a \tag{7}$$

where

$$e_{t+1}(\boldsymbol{s}, a) = \begin{cases} 1 & \text{if } \boldsymbol{s} = s_t \text{ and } a = a_t \\ 0 & \text{if } \boldsymbol{s} = s_t \text{ and } a \neq a_t \\ \gamma\lambda e_t(\boldsymbol{s}, a) & \text{otherwise} \end{cases} \tag{8}$$

is a matrix of eligibility traces. The purpose of eligibility traces is to propagate TD-error to the state-action values faster so as to accelerate the exporation of the optimal strategy. The specific version, known as SARSA(λ) [4], has been adopted for the learning of the Ms. Pac-Man agent.

4 The proposed state space representation

The game of Ms. Pac-Man constitutes a challenging domain for building and testing intelligent agents. The state space representation is of central interest for an agent, since it plays a significant role in system modeling, identification,

and adaptive control. At each time step, the agent has to make decisions according to its observations. The state space model should describe the physical dynamic system and the states must represent the internal behaviour of system by modeling an efficient relationship from inputs to actions. In particular, the description of the state space in the Ms. Pac-Man domain should incorporate useful information about his position, the food (dots, scared ghosts) as well as the ghosts. An ideal state space representation for Ms. Pac-Man could incorporate all these information that included in a game snapshot, such as:

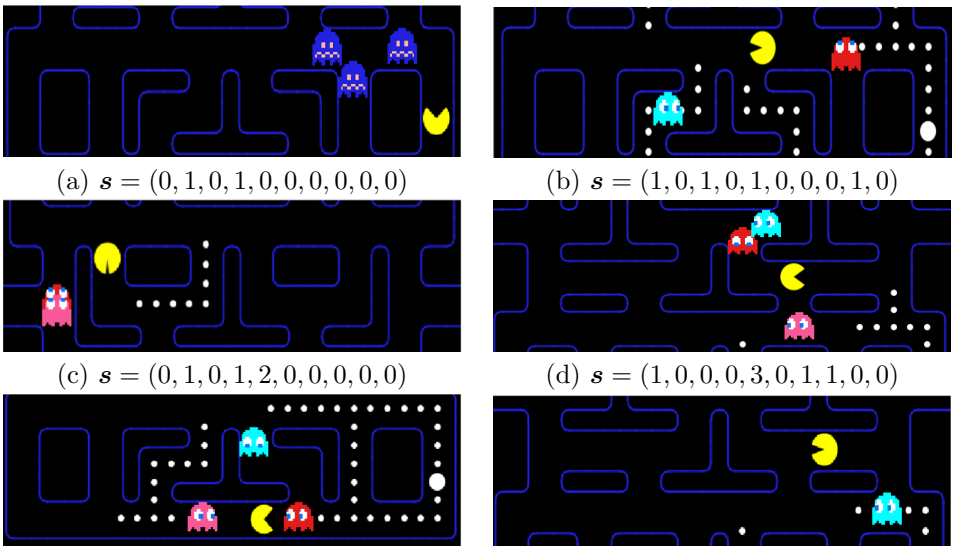
- the relative position of Ms. Pac-Man in the maze,
- the situation about the food (dots, power pills) around the agent,
- the condition of nearest ghosts.

Although the state space representation constitutes an integral part of the agent, only little effort has been paid in seeking a reasonable and informative state structure. As indicated in [6], a full description of the state would include (a) whether the dots have been eaten, (b) the position and direction of Ms. Pac-Man, (c) the position and direction of the four ghosts, (d) whether the ghosts are edible (blue), and if so, for how long they remain in this situation. Despite its benefits, the adoption of such a detailed state space representation can bring several undesirable effects (e.g. high computational complexity, low convergence rate, resource demanding, e.t.c), that makes modeling of them to be a difficult task.

According to the above discussion, in our study we have chosen carefully an abstract space description that simultaneously incorporate all the necessary information for the construction of a competitive agent. More specifically, in our approach the state space is structured as a 10-dimensional feature vector, $\boldsymbol{s} = (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10})$ with discrete values. Its detailed description is given below:

- The first four (4) features $(s_1, \ldots, s_4)$ are binary and used to indicate the existence (1) or not (0) of the wall in the Ms. Pac-Man's four wind directions (north, west, south, east), respectively. Some characteristic examples are illustrated in Fig. 2; state vector $(s_1 = 0, s_2 = 1, s_3 = 0, s_4 = 1)$ indicates that the Pac-Man is found in a corridor with horizontal walls (Fig. 2(a)), while state values $(s_1 = 1, s_2 = 0, s_3 = 1, s_4 = 0)$ means that Ms. Pac-Man is located between a west and east wall (Fig. 2(b)).

- The fifth feature s_5 suggests the direction of the nearest *target* where it is preferable for the Ms. Pac-Man to move. It takes four (4) values (from 0 to 3) that correspond to north, west, south or east direction, respectively. The *desired target* depends on the Ms. Pac-Man's position in terms of the four ghosts. In particular, when the Ms. Pac-Man is going to be trapped by the ghosts (i.e. at least one ghost with distance less than eight (8) steps is moving against Ms. Pac-Man), then the direction to the closest *safer* exit (*escape direction*) must be chosen (Fig.2(d)). In all other cases this feature takes the direction to the closest dot or frightened ghost. Roughly speaking, priority is given to neighborhood food: If a edible (blue-colored) ghost exists within a maximum distance of five (5) steps, then the ghost's direction is selected (Fig.2(a)). On the other hand, this feature takes the



(a) $\boldsymbol{s} = (0, 1, 0, 1, 0, 0, 0, 0, 0, 0)$ (b) $\boldsymbol{s} = (1, 0, 1, 0, 1, 0, 0, 0, 1, 0)$

(c) $\boldsymbol{s} = (0, 1, 0, 1, 2, 0, 0, 0, 0, 0)$ (d) $\boldsymbol{s} = (1, 0, 0, 0, 3, 0, 1, 1, 0, 0)$

(e) $\boldsymbol{s} = (1, 0, 1, 0, 3, 0, 1, 0, 1, 1)$ (f) $\boldsymbol{s} = (1, 0, 1, 0, 1, 0, 0, 0, 0, 0)$

Figure 2: Representative game situations along with their state description

direction that leads to the nearest dot (Fig.2(c,f)). Note here that for calculating the distance as well as the direction between Ms. Pac-Man and *target*, we have used the known A* search algorithm [15] for finding the shortest path.

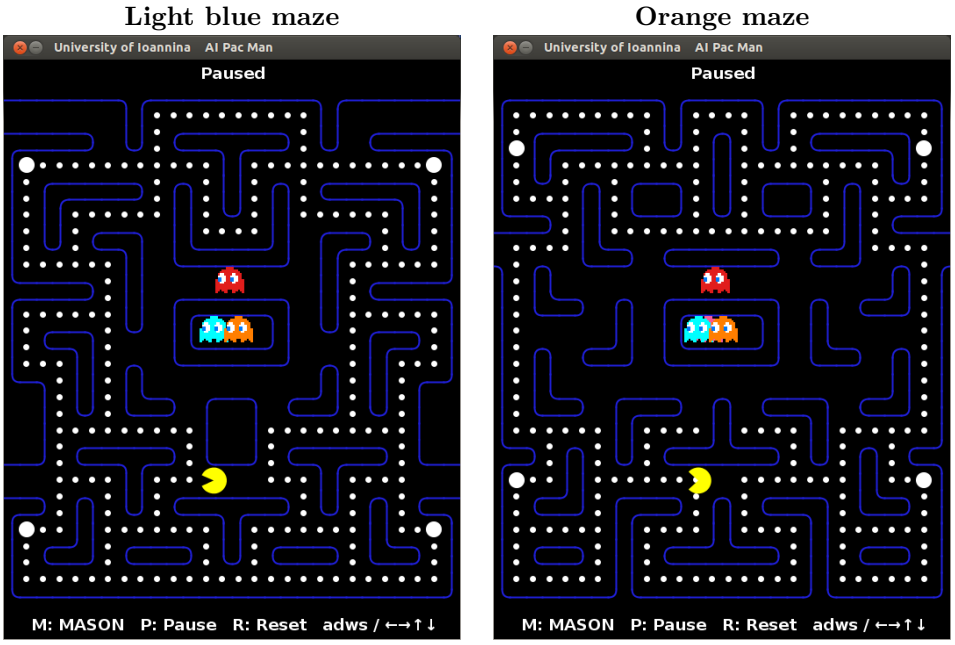
- The next four features $(s_6, \ldots, s_9)$ are binary and specify the situation of any direction (north, west, south, east) in terms of a *direct* ghost threat. When a ghost with distance less that five steps (5) is moving towards pac-man from a specific direction, then the corresponding direction takes the value of 1. An example given in Fig.2(d) where the Ms. Pac-Man is approached threateningly by two ghosts. More specifically, the first ghost approaches the agent from the east ($s_7 = 1$) and the other from the south direction ($s_8 = 1$).

- The last feature specifies if the pac-man is trapped (1) or not (0). We assume that the Ms. Pac-Man is trapped if there doesn't exist any possible escape direction (Fig.2(e)). In all other cases the Ms. Pac-Man is considered to be free (Fig.2(a, b, c, d, f)). This specific feature is very important since it informs the agent whether or not it can (temporarily) move in the maze freely.

Table 1 summarizes the proposed state space. Obviously, its size is quite small containing only $4 * 2^9 = 2048$ states. This fact allows the construction of a computationally efficient RL agent without the need of any approximation scheme. Last but not least, the adopted reasonable state space combined with the small action space speed up the learning process and enables the agent to discover optimal policy solutions with sufficient generalization capabilities.

Feature	*Range*	*Source*
$[s_1\ s_2\ s_3\ s_4]$	$\{0, 1\}$	Ms. Pac-Man view
s_5	$\{0, 1, 2, 3\}$	target direction
$[s_6\ s_7\ s_8\ s_9]$	$\{0, 1\}$	ghost threat direction
s_{10}	$\{0, 1\}$	trapped situation

Table 1: A summary of the proposed state space



(a) (b)

Figure 3: Two mazes used for evaluating the proposed RL agent

5 Experimental results

A number of experiments has been made in order to evaluate the performance of the proposed methodology in the Ms. Pac-Man domain. All experiments were conducted by using the MASON multiagent simulation package [16] which provides a faithful version of the original game. Due to the low complexity of the proposed methodology and its limited requirements on memory and computational resources, the experiments took place on a conventional PC (Intel Core 2 Quad (2.66GHz) CPU with 2GiB RAM).

We used three mazes of the original Ms. Pac-Man game illustrated in Figs. 1 and 3. The first maze (Fig. 1) was used during the learning phase for training the RL agent, while the other two mazes (Fig. 3) were applied for testing. In all experiments we have set the discount factor (γ) equal to 0.99 and the learning rate (α) equal to 0.01. The selected reward function is given at Table.2. It must be noted that our method did not show any significant sensitivity to the above

Event	***Reward***	***Description***
Step	-0.5	Ms. Pac-Man performed a move in the empty space
Lose	-35	Ms. Pac-Man was eaten by a non-scared ghost
Wall	-100	Ms. Pac-Man hit the wall
Ghost	$+1.2$	Ms. Pac-Man ate a scared ghost
Pill	$+1.2$	Ms. Pac-Man ate a pill

Table 2: The reward function for different game events

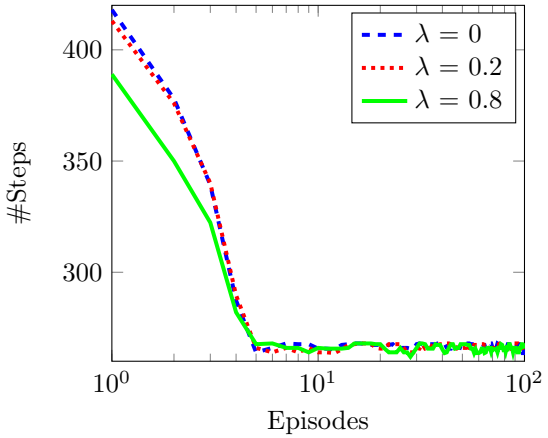


Figure 4: Learning progress of the agent at the pink maze without ghosts

reward values; however a careful selection is necessary to meet the requirements of the physical problem. In addition, we assume that an episode is completed either when all the dots are collected (*win*) or the Ms. Pac-Man is collided with a non-scared ghost. Finally, the performance of the proposed approach was evaluated in terms of four distinct metrics:

- Average percentage of successfully level completion
- Average number of *wins*
- Average number of steps per episode
- Average score attained per episode

The learning process follows a two-stage strategy. At the first phase, the agent is trained without the presence of ghosts. In this case the agent's goal is to eat all the dots and terminates the level with the minimum number of steps. During the second phase the agent is initialized with the policy discovered previously and the ghosts are entered into the same maze. Likewise, the agent's target is to eat all the dots, but now with the challenge of the 'non-scared ghosts avoidance'.

Figure 4 illustrates the depicted learning curve during the first phase, i.e. mean number of steps (after 20 different runs) that the agent needs to finish the episode by eating all the dots of the maze (Fig. 1). In order to study

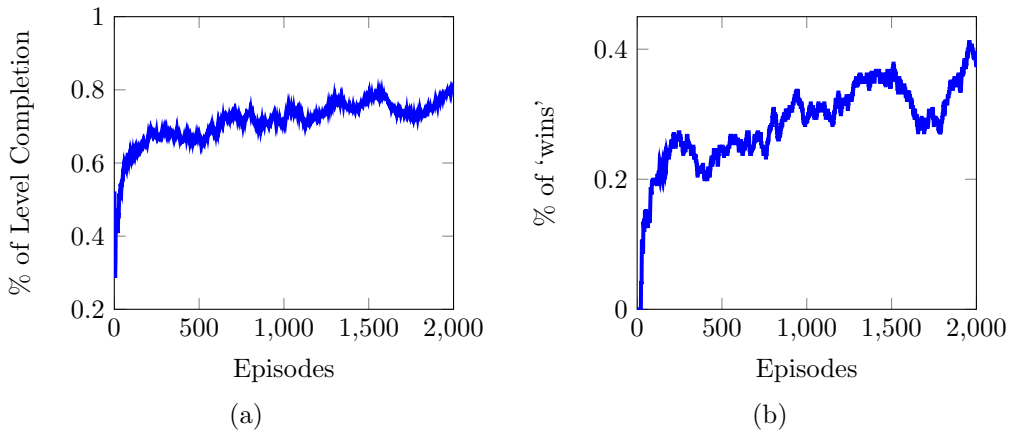


Figure 5: Learning progress of the agent at the pink maze with ghosts

Maze	***Level completion***	***Wins***	***# Steps***	***Score***
Pink maze (Fig. 1)	80% (±24)	40%	348.7 (±153)	2292.3 (±977)
Light blue maze (Fig. 3(a))	70% (±24)	33%	319.4 (±143)	2538.4 (±1045)
Orange maze (Fig. 3(b)	80% (±20)	25%	360.8 (±155)	2515.7 (±1011)

Table 3: Testing performance

the effectiveness of the eligibility trace (Eqs. 7, 8) to the RL agent, a series of initial experiments were made with three different values $(0, 0.2, 0.8)$ of the decay parameter λ. According to the results, the value of $\lambda = 0.8$ had shown the best performance, since it allows reaching optimal policy solution very quickly (260 steps in less than 100 episodes). We have adopted this value in the rest experiments. Note here that in all three cases the discovered policy was almost the same. Another useful remark is that the received policy is perfect, i.e. eating all 220 dots of the maze in only 260 steps (only 15% moves in positions with no dots).

The learning performance of the second phase is illustrated in Fig. 5 in terms of the (a) percentage of level completion and (b) number of wins (successful completion) in the last 100 episodes. As shown the method converges quite rapidly at an optimal policy after only 800 episodes. The Ms. Pac-Man agent manages to handle trapped situations and completes successfully the level at a high-percentage. We believe that the 40% of the level completion suggests a satisfactory playing of the pacman game.

In order to measure the generalization capability of the proposed mechanism, we have tested the policy that was discovered during the learning phase into two unknown mazes (Fig. 3). Table 3 lists the performance of the fixed policy in three mazes, where the statistics (mean value and std) of the evaluation metrics were calculated after running 100 episodes. That is interested to note here is that the agent had shown a remarkable behavior stability to both unknown mazes providing clearly significant generalization abilities. Finally, the obtained policy was tested by playing 50 consecutive games (starting with 3 lives and adding

Mazes	***Average Scores***	***Max Score***
Pink maze (Fig. 1)	9665	20860
Light blue maze (Fig. 3(a))	12753	38840
Orange maze (Fig. 3(b)	11587	27620

Table 4: Ms. Pac-Man game score

a live at every 10000 points). Table 4 summarizes the depicted results where we have calculated the mean score together with the maximum score found in all three tested mazes. These particular results verify our previous observations on the generalization ability of the proposed agent that is managed to build a generic optimal policy allowing Ms. Pac-Man to navigate satisfactory at every maze.

6 Conclusions and future directions

In this work we have presented a reinforcement learning agent that learns to play the famous arcade game Ms. Pac-Man. An abstract but informative state space representation has been introduced that allows flexible operation definition possibilities through the reinforcement learning framework. Initial experiments demonstrate the ability and the robustness of the agent to reach optimal solutions in an efficient and rapid way.

There are many potential directions for future work. For example, in our approach the power-bills are not included in the state structure. Intuitively thinking, moving towards the power-bills can be seen gainful since it can increase the pacman's life as well as the score. However, there is a trade-off between searching (greedily for food) and defensive (avoiding the ghosts) abilities that must be taken into account. Another alternative is to investigate bootstrapping mechanisms, by restarting the learning process with previously learned policies, as well as to combine different policies that are trained simultaneously so as to achieve improved performance, especially in critical situations of the domain. Finally, we hope that this study will provide a foundation for additional research work in other similar game domains like Ms. Pac-Man.

References

[1] L. Galway, D. Charles, and M. Black. Machine learning in digital games: A survey. *Artificial Intelligence Review*, 29:123–161, 2008.

[2] R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[3] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Inteligence Research*, 4:237–285, 1996.

[4] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press Cambridge, USA, 1998.

[5] I. Szita. Reinforcement learning in games. In *Reinforcement Learning*, pages 539–577, 2012.

[6] I. Szita and A. Lorincz. Learning to play using low-complexity rule-based policies: Illustrations through ms. pac-man. *Journal of Artificial Intelligence Research*, 30:659–684, 2007.

[7] S. M. Lucas. Evolving a neural network location evaluator to play ms. pac-man. In *Proc. of IEEE Symposium on Computational Intelligence and Games (CIG05)*, pages 203–210, 2005.

[8] L. Bom, R. Henken, and M.A. Wiering. Reinforcement learning to train ms. pac-man using higher-order action-relative inputs. In *Proc. of IEEE Intern. Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 156–163, 2013.

[9] A. M. Alhejali and S. M. Lucas. Evolving diverse ms. pac-man playing agents using genetic programming. In *Proc. of IEEE Symposium on Computational Intelligence and Games (CIG10)*, pages 53–60, 2010.

[10] S. Samothrakis, D. Robles, and S. Lucas. Fast approximate max-n monte-carlo tree search for ms. pac-man. *IEEE Trans. on Computational Intelligence and AI in Games*, 3(2):142–154, 2011.

[11] K. Q. Nguyen and R. Thawonmas. Monte carlo tree search for collaboration control of ghosts in ms. pac-man. *IEEE Trans. on Computational Intelligence and AI in Games*, 5(1):57–68, 2013.

[12] L. Torrey and M. Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Intern. Conferecene on Autonomous Agents and Multi-agent Systems (AAMAS)*, pages 1053–1060, 2013.

[13] M. L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. Wiley, 2005.

[14] S. Singh, R. S. Sutton, and P. Kaelbling. Reinforcement learning with replacing eligibility traces. pages 123–158, 1996.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[16] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.