Reinforcement Learning

Data Mining Project

732A65

Rasmus Holm

January 17, 2018

Introduction

Reinforcement learning has made huge advances in the last couple of years, in particular with games such as when AlphaGo from Google DeepMind won against Lee Sedol in the game Go in 2016 [7]. It has also been possible to teach the computer to play Atari games [6] and this year at the largest eSports event in the world, the International 2017, OpenAI demonstrated that reinforcement learning can be used to defeat some of the best players in Dota 2 in 1 versus 1. [1]

Reinforcement is another large area in machine learning alongside supervised and unsupervised learning. The idea behind reinforcement learning is learning by interacting with an environment, which is how learning in nature happens very often. A child putting its hands on a hot stove will quickly feel a painful sensation and remember that was a terrible idea, and ultimately learn from it. Learning can also have positive effects such as when a mouse reaches the end of the labyrinth and finds a piece of cheese.

Reinforcement learning is a computational framework to this type of learning by having a so called *agent* interact with an *environment* and learn how to respond to different scenarios. The goal is to create learning methods that can learn an optimal *policy*, a mapping from the current sensory input to a decision in the form of an action, according to a behavior we want it to achieve. That might be from learning the computer to play tic-tac-toe to more complex behavior such as flying a helicopter where the agent do not posses full control of the environment.

In this report we are interested in investigating some of the fundamental building blocks of the reinforcement learning framework which will be presented in the next chapter. The questions we have posed are

1. Is the state representation important for learning?
2. Is it important for the reward function to reinforce the behavior we want the agent to learn?
3. Is punishing bad behavior equivalent to reinforcing good behavior?
4. Is it better if the reward function combines reinforcement and punishment?

[1]https://blog.openai.com/dota-2/

Theory

In this chapter we will formalize the reinforcement learning problem mathematically and define the methods to solve our particular reinforcement learning problem.

Framework

To begin the definition of the reinforcement learning problem we first need to define a few fundamental terms that are required in the following formalization. The basic idea is to learn from interactions in order to achieve a goal. The learner is called the *agent* which interacts with the *environment* by performing *actions* in a sequential manner. The agent is continuously interacting with the environment and the environment in turn responds based on those interactions in the form of a representation of its current *state*. Not only may the environment change its state in reponse to the agent's actions, it also give rise to *rewards*, numerical values that the agent is trying to maximize through its actions. Figure 1 shows the interaction cycle between the agent and the environment. A complete specification of an environment and how its rewards are determined defines a *task*, which is an instance of a reinforcement learning problem.

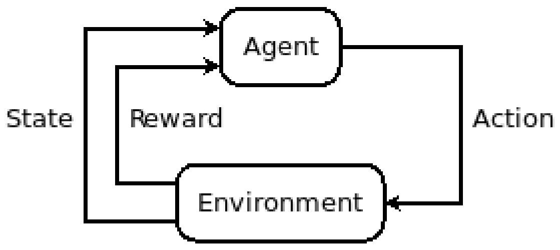


Figure 1: The agent-environment interaction loop.

This rather simple framework is actually abstract and flexible enough to be applied in many problems in many different ways. The actions could range from low level controls such as turning the wheels on the car to driving from point A to point B. Similarly, the state could represent current road surface condition to traffic flow in the nearby area. The reward will however always be a single numerical value since we require it to be comparable and have a well defined ordering to be able to determine the best action, the action that will maximize the total amount of reward the agent receives. We will come back to what that means in the following sections.

Snake

Before defining the *Markov decision process* we present our problem we want our agent to try to solve which is the game *Snake*[2] and its an old game with many variants. In this report, the game is played out as simple as possible with only one player, i.e., the agent and no obstructing environment. The goal of the

[2]https://en.wikipedia.org/wiki/Snake_(video_game)

game is that you play as a snake by controlling the head and the aim is to eat as many apples as possible without colliding with either the snake's body or the edges of the game board. At each time unit the snake moves one unit in the direction it is facing and the difficulty is that each time you eat an apple your body becomes longer by 1 unit and thus larger portion of the game board is covered by it. We have defined the final game score to be 100 per eaten apple. In order to reduce the state space the game board is played on a 5x5 grid which can be seen in figure 2.

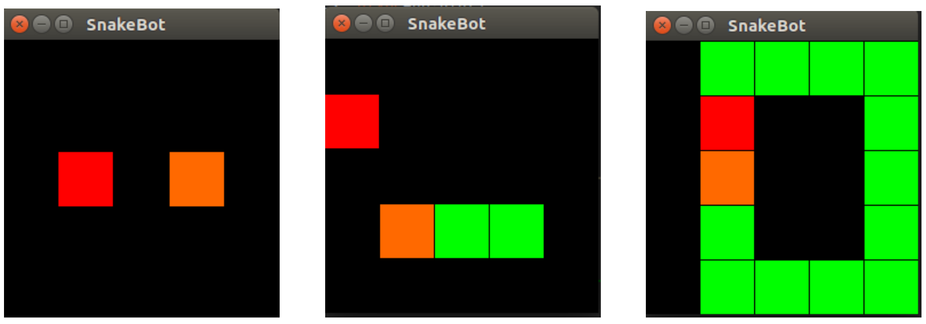


Figure 2: 5x5 board of Snake where the snake consists of the orange (head) and green parts (body) and the red part is the apple.

We will use Snake to further concretize the definitions below.

Markov Decision Process

We are going to limit our problem formalization to a special case with discrete time steps, finite state space, finite action space, fully observable state, and episodic tasks. It can be generalized to continuous time steps, infinite state space, infinite action space, partially observable state, and continuing tasks, but it is out of scope for this report. Bertsekas and Tsitsiklis give an in-depth coverage of reinforcement learning, also known as optimal control, in *Neuro-dynamic programming* [2]

Notation

Before continuing with the definition we need to present the most important notation that will be used from now on through the rest of the report.
$\mathcal{A}$: Action space
$\mathcal{S}$: State space
$A_t \in \mathcal{A}$: Action at time t
$S_t \in \mathcal{S}$: State at time t
$R_t \in \mathcal{R}$: Reward at time t
where $t \in \{0, 1, 2, \dots, T\}$ since we have discrete time steps. Capital letters with a subscript such as A_t are regarded as random variables while a lowercase letter such as a is a realisation of a random variable.

For snake the action space $\mathcal{A} = \{\text{North}, \text{Sourth}, \text{East}, \text{West}\}$ but the state space $\mathcal{S}$ is not defined yet. We will present different state representations in the method.

Dynamics

We will now formalize the reinforcement learning problem through the *Markov Decision Process* (MDP), in particular the *finite* MDP, by sticking to the limitations mentioned above. This formalization assumes that the problem satisfies the *Markov property* which means that the one-step dynamics of the problem satisfies the following

$$Pr(S_{t+1} = s', R_{t+1} = r | S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t) = Pr(S_{t+1} = s', R_{t+1} = r | S_t, A_t),$$

which in words says that the future state-reward pair is independent of the past actions, states, and rewards given the current action-state pair. That is also equivalent to saying that the current state and action captures all the relevant information from the history. To simplify this formula we write it as

$$p(s', r | s, a) = Pr(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$$

and it completely specifies the dynamics of the finite MDP, or equivalently the dynamics of the environment in which the agent is interacting. We can compute anything we might want to get from it such as the the *state-transition probabilities* and the *expected reward* for state-action pairs

$$\begin{aligned} p(s' | s, a) &= \sum_r p(s', r | s, a), \\ r(s, a) &= \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a). \end{aligned}$$

In our case the state-transitions probabilities will most of the times be either 0 or 1 since the game will be deterministic in the sense that if the snake is facing north it will always move north, not south or any other direction, hence the next state is known for certain in most situations. The only random element is the placement of the apple so if the action results in the snake eating an apple the next state will not be known for certain.

Policy

A *policy* π is a distribution over actions given states

$$\pi(a|s) = Pr(A_t = a | S_t = s).$$

It fully defines the behavior of an agent. It maps an observed state into an action and it could be stochastic in order to enforce *exploration* which we will come back to later. A policy could be as simple as a lookup table or a complicated search process which further makes this framework very flexible. We have used the former approach.

Reward

The *reward signal* is an essential part to learning that will guide the agent to the target behavior. This implies that it is crucial that during the construction of your reinforcement learning problem the underlying reward function that generates the rewards should reflect what you want accomplished. As a simple example, in the game of chess you would consider winning as good and losing as bad and so the reward should reflect that by for instance give $+1$ reward for winning and -1 reward for losing.

The sole goal of the agent is the maximize the total reward it receives in the long run, so we define the return G_t as

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where $\gamma \in [0, 1]$ is the *discount factor* that determines how much weight the agent should put on immediate rewards versus future rewards. By adding the discount factor $\gamma < 1$ to G_t we bound it so it does not become ∞. The goal of reinforcement learning is to maximize the return G_t but since it is a random variable what we actually do care about is the *state-value function*

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \middle| S_t = s \right] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right],$$

and the *action-value function*

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \middle| S_t = s, A_t = a \right] = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right].$$

In words these functions expresses the expected return starting from state s and then following policy π, and the expected return starting from state s and performing action a and then following policy π respectively.

These recursive equations are known as the Bellman equations discovered by Richard Bellman in the 1950s [1]. What we are interested in is finding the Bellman optimality equations

$$\begin{aligned}
v_*(s) &= \max_\pi v_\pi(s) \\
&= \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_*(s') \right], \\
\forall_s v_*(s) &\geq \forall_{s,\pi} v_\pi(s), \\
q_*(s, a) &= \max_\pi q_\pi(s, a) \\
&= \sum_{s',r} p(s', r|s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right], \\
\forall_s q_*(s, a) &\geq \forall_{s,\pi} q_\pi(s, a),
\end{aligned}$$

and having those makes it trivial to find the optimal policy that maximizes the expected return which is the goal of the agent. It turns out that there always exists such a policy, possibly multiple such policies.

The optimal policy π_* is

$$\begin{aligned}\pi_*(a|s) &= \begin{cases} 1, & \text{if } a = \underset{a}{\operatorname{argmax}}\, q_*(s,a) \\ 0, & \text{otherwise} \end{cases} \\ &= \begin{cases} 1, & \text{if } a = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right] \\ 0, & \text{otherwise,} \end{cases}\end{aligned}$$

and we can clearly see that is it easier to acquire the optimal policy if we have access to the optimal action-value function. The problem is that there are, in general, no closed form solution to finding these optimal value functions and therein lies the difficulty of reinforcement learning. After a summary of the Snake's MDP and an explanation of exploration we will present a few iterative solution methods for approximating these optimal value functions, the action-value function in particular, that have been used in this report.

Snake MDP

We mentioned the term *episodic task* and it refers to problems that have a natural notion of final time step which Snake naturally does, that is when the snake collides with something and dies. We denote such a state as the *terminal state*.

To summarize the Snake's MDP we have that the $\mathcal{A} = \{\text{North}, \text{South}, \text{East}, \text{West}\}$ and the state-transition probabilities are mostly either 0 or 1 with the exception of when the action makes the snake eat an apple which the environment will react to by placing a new apple on the grid uniformly at random. We have not yet defined $\mathcal{S}$ nor the reward function which we will come back to in the method.

Exploration versus Exploitation

An important concept in reinforcement learning is the *exploration-exploitation* trade-off. It refers to the problem that the agent does not know how it should behave to maximize the expected return and thus have to interact with the environment to learn by trial-and-error. This means that it has to explore the environment by interactions and observe their consequences and the reward signals. This information does not contain what the "right" action was supposed to be but the agent can still construct a policy from it. Exploiting means that the agent is using its current policy to determine the best action, i.e., it is greedy with respect to the state-value function (or action-value function). It can happen that the current policy is stuck in a local optima because the agent has not explored the search space thorough enough to build a good estimate of the performance of other actions.

A simple technique to enforcing exploration is by training using ϵ-greedy policy, see algorithm 1, which takes the greedy action with probability $1 - \epsilon$ and a uniformly random action with probability ϵ. This is the approach taken in this report but there are more sophisticated techniques to balance these two acts.

Algorithm 1 ϵ-Greedy Policy

Require: State s, Action-value function q, Exploration rate $\epsilon \in [0, 1]$

```
1: function EpsilonGreedyPolicy($s, q, \epsilon$)
2:     sample $u \sim U(0, 1)$
3:     if $u < \epsilon$ then
4:         sample $i \sim U(\{1, 2, \ldots, |\mathcal{A}|\})$
5:         $a = \mathcal{A}_i$
6:     else
7:         $a = \underset{a}{\operatorname{argmax}}\, q(s, a)$
8:     return $a$
```

A term that was mentioned previously was *episodic task* which refers to problems that have a natural notion of final time step which Snake does, that is when the snake collides with something. We denote this state as the *terminal state* which will follow with a reset of the game in the same starting state independently of how it ended.

Temporal Difference Learning

Temporal Difference (TD) learning is a learning methodology that learns directly from raw experience without a model of the environment's dynamics. It updates new estimates partially on previous estimates without waiting for the final outcome. The algorithms, or learning methods, that have been used in this paper are all based on TD learning and will be described below. Algorithm 2 shows the general outline of how it works and what will differ between the algorithms is line 9, the update rule of $Q(S, A)$. We differentiate choosing the action, determined by our policy, from actually performing it so it becomes a two step process to take an action. We have used a tabular representation of the Q-function, i.e., we map a state-action pair to an action-value using a hash table.

Algorithm 2 TD Learning Algorithm

Require: Learning rate $\alpha \in [0,1]$, Discount factor $\gamma \in [0,1]$

```
1: function TDLearning($\alpha, \gamma$)
2:     Initialize $\forall_{a \in \mathcal{A}, s \in \mathcal{S}} Q(a,s)$ arbitrarily and $Q(\text{terminal state}, \cdot) = 0$
3:     for each episode do
4:         Initialize $S$
5:         Choose $A$ from $S$ using policy $\pi$ ($\epsilon$-greedy) derived from $Q$
6:         while $S$ is non-terminal state do
7:             Perform action $A$, observe state $S'$ and reward $R$
8:             Choose $A'$ from $S'$ using policy $\pi$ ($\epsilon$-greedy) derived from $Q$
9:             Update $Q(S,A)$ based on $(S, A, R, S', A', \pi, \alpha, \gamma)$
10:            $S = S'$
11:            $A = A'$
12:     return Q
```

Q-Learning

Q-learning is a widely used algorithm that was developed by Watkins [10] and variations of it is heavily used today combined with the emergence of neural networks [4], [9]. The update rule is defined as

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right].$$

It is a so called *off-policy* method which is indicated by the max operation. That means it uses a greedy algorithm to estimate the return at the next step rather than following its current policy, therefore is off its own policy.

Sarsa

Sarsa on the other hand is an on-policy method for estimating the action-value function and its name refers to that we use the quintuple of events $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, and it has the update rule

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right].$$

The difference betwen Sarsa and Q-learning is that it does not use the max operator and is rather using its current policy to estimate the next action-value. There is also a natural extension to Sarsa called *Expected Sarsa* which takes the expected action-value estimate under the current policy. The update rule is defined as

$$\begin{aligned} Q(S_t, A_t) &= Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E} \left[Q(S_{t+1}, A_{t+1}) | S_{t+1} \right] - Q(S_t, A_t) \right] \\ &= Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \end{aligned}$$

Other Learning Methods

There are many other learning methods that are not used in this report that have interesting properties. Methods such as those based on *dynamic programming* which is an important optimization method, *Monte Carlo*, and also extensions to those as well as to the algorithms presented above. We highly recommend the book written by Sutton and Barto [8] for more information.

Method

In order the answer the research questions presented in the introduction, we have divided the experiments into two parts. One experiments in which we look at the correlation between the reward function and the goal of the game and another where we investigate the performance of different state representations.

Reward Experiment

Our hypothesis is that it is essential that the reward function reflects what the goal of the problem is and in our case that is to gather as many apples as possible, preferably in fewer time steps. To perform this experiments we have constructed 9 reward functions with all the combinations of the following characteristics

Reward at the end of the episode
± score or 0
Reward after each time step
±1 or 0

The reasoning for the construction of these particular reward functions is the answer the research questions.

Is it important for the reward function to reinforce behavior we want the agent to learn? If this is true we expect the reward functions that gives positive score rewards regardless of reward per time step to have the best performances.

Is punishing bad behavior equivalent to reinforcing good behavior? If this is true we expect the reward function that gives 0 rewards at the end of the episode and -1 reward each time step has the similar performance to the ones hypothesised as the best in the previous question.

Is it better if the reward function combines reinforcement and punishment? If this is true we expect the reward function that gives positive score rewards at the end of the episode and -1 reward at each time step has the best performance.

We also added two additional reward functions

NegTravelNegBorderCollisionPosScore: This function penalize -1 each time step and an extra big penalty (-10000) when the snake collides with the game border. At the end of the episode it gets a positive reward equal to the game score.

NegDistanceNegBorderCollisionPosBodySize: This function is similar in that it penalize significantly when the snake is colliding with the game border. However, instead of giving a fixed penalty each time step it is based on the Manhattan distance from the head of the snake to the apple. At the end of the episode it gets a positive reward equal to 100 × the length of the snake which is equivalent to the game score + 100.

In order to do this experiments we also need to choose state representations and we choose to use two very different ones. The board state with the game score and the directional state with the game score, as described below.

State Experiment

In order to determine the importance of the the state representation we have decided upon 4 different representations. Additionally, we have decided to test whether it is important for the state to contain the board dimensions to help the agent to infer deadly state-action pairs. And also if the current game score is important for the agent to be able to predict the future rewards more accurately.

Intuitively, it would be reasonable to assume that if the agent knows the dimensions of the board it could more easily infer state-action pairs that result in a terminal state and thus end the game with potentially less total reward.

By the same reasoning is it probably a good idea to give the agent access to its current score for it to improve the estimation of future rewards and thus making better decisions.

The 4 state representations are the following

- **Board**: Represents the complete board state as the game engine represents it.
- **SnakeFood**: Represents coordinates of the snake's head/body and the coordinate of the food source.
- **Directional**: Represents the direction the snake should travel to get to the food source without following the dynamics of the game. That means it might not be possible to change to that direction with a single action, e.g., the snake travels east but should travel west.
- **ShortestPath**: Represents the shortest path from the head of the snake to the apple.

and each state is split up into 4 different states considering with/without board dimensions and with/without score, which gives 16 states in total.

To perform this experiments we also require a reward function which was decided based on early experiments. It turned out to be the reward function that gave 0 reward each time step and positive score as reward at the end of the episode was the most promising.

Hyperparameters

For all the experiments we have used the following hyperparameters based on trial-and-error experimentation

- ϵ-greedy policy with exploration rate $\epsilon = 0.15$
- Training for 1 million episodes
- Initialized the Q-function with 0s

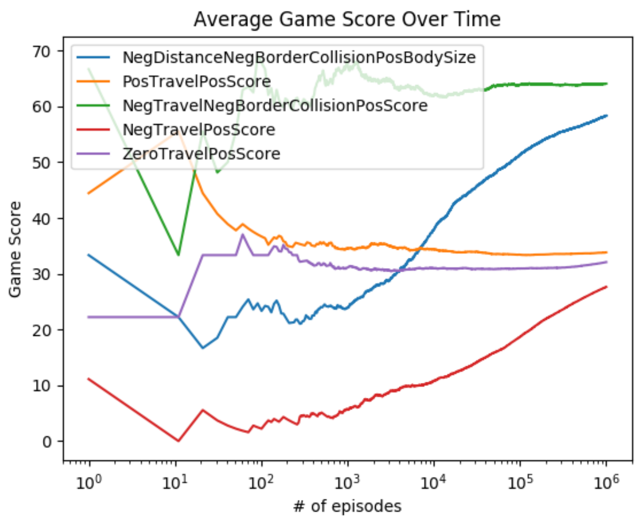
Learning rate α	Discount factor γ
Q-Learning	
0.85	0.85
Sarsa	
0.15	0.95
Expected Sarsa	
0.15	0.95

Result

In this part of the report we will present the result that have been gathered from our two experiments on reward functions and state representations. It will be presented in the order aforementioned.

Reward Experiment

In this section we will present the results given by the reward experiment. The upper plots have used the board state representation and the lower plots have used the directional state. All the results have been averaged over 3 different experiments.



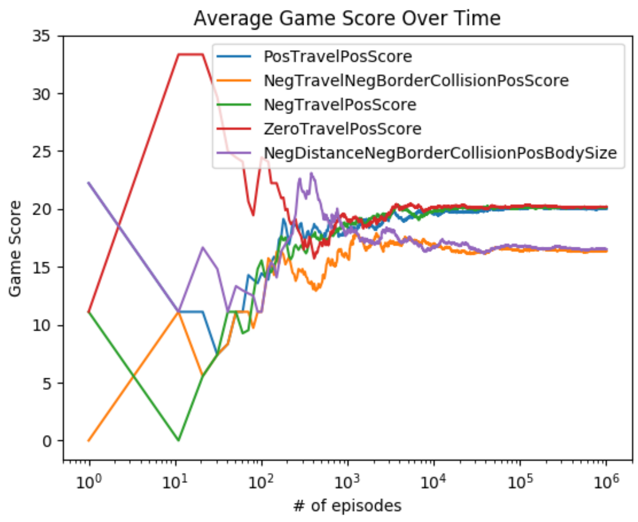
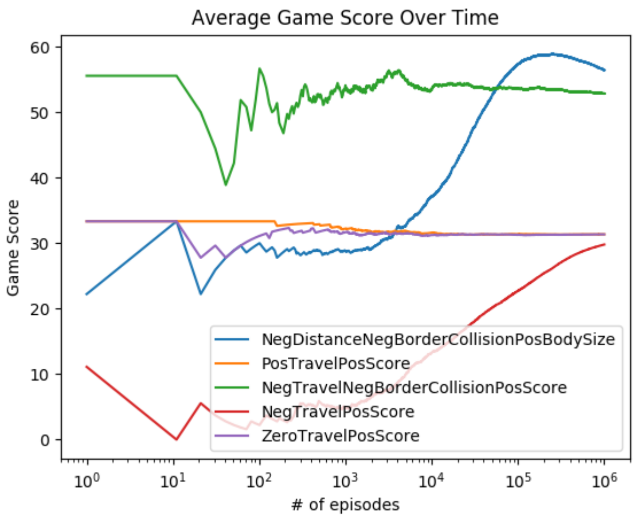


Figure 3: Results generated by Q-Learning. Upper: Board state. Lower: Directional State.



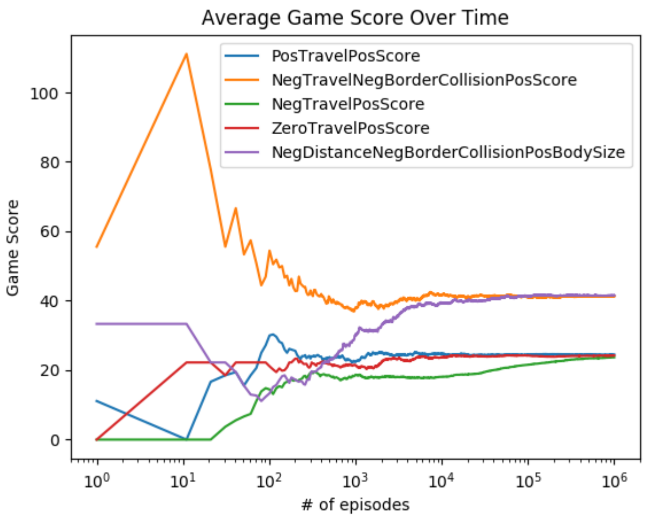


Figure 4: Results generated by Sarsa. Upper: Board state. Lower: Directional State.

The plots show the average game score the agent learns to score on average which is the goal to maximize. What we mean by "average over time" is that we take the average of all data points up to (inclusive) the time we are interested in, i.e., a moving average with an infinite tail. In our case time refers to the number of episodes we have trained the model.

We also looked at the correlation between the rewards over time (not shown) and the game score curves above and can be read off in table 1.

	Avg. Correlation
	Q-Learning
Board State	0.95
Directional State	0.84
	Sarsa
Board State	0.97
Directional State	0.91

Table 1: The average correlation between the reward functions and game score.

State Experiment

In this section we present the results from the state experiment. For each algorithm we have looked at the 16 states presented in the method. Since the goal is to look at the objective of the problem we plot the average game score each state gets over 1 million episodes of training. Same as before we have averaged the result over 3 experiments.

The first thing we looked at was augmenting the state representations with board dimensions and game score. Figure 5 shows the results using the board representation and trained with Q-learning. We saw similar results for other states and algorithms.

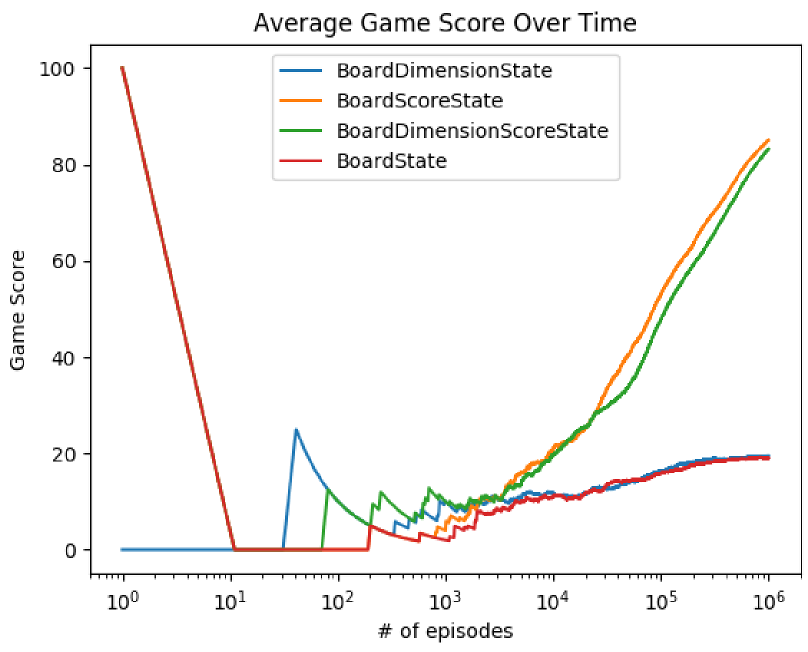


Figure 5: Board state with different augmented information.

We trained the models on a lot of different states as stated in the method and in figure 6 shows the most promising states trained with Q-learning. We have included test performance in table 2 by averaging scores over 10000 episodes (games) using the greedy policy without exploration ($\epsilon = 0$). It also shows the performance of an agent that chooses action uniformly at random as a comparison of what we would expect from a terrible model.

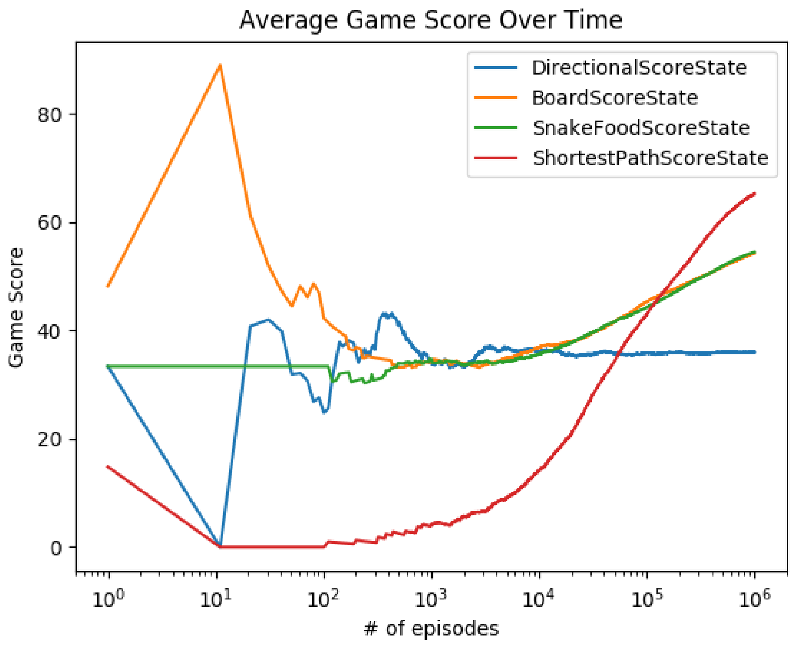


Figure 6: Performance of state representations trained using Q-learning.

State	Avg. Score	Std. Deviation
ShortestPathScoreState	129.50	51.11
BoardScoreState	103.12	17.73
DirectionalScoreState	66.12	64.328
SnakeFoodScoreState	104.12	20.08
Random Agent	20.13	46.47

Table 2: Test performance over 10000 episodes.

Similar results as previously are shown in figure 7 and table 3 but trained with Sarsa. We have not included any results for Expected Sarsa since its performance and behavior was very similar to Sarsa. The appendix at the end includes more results on other states that was less promising in terms of game score performance.

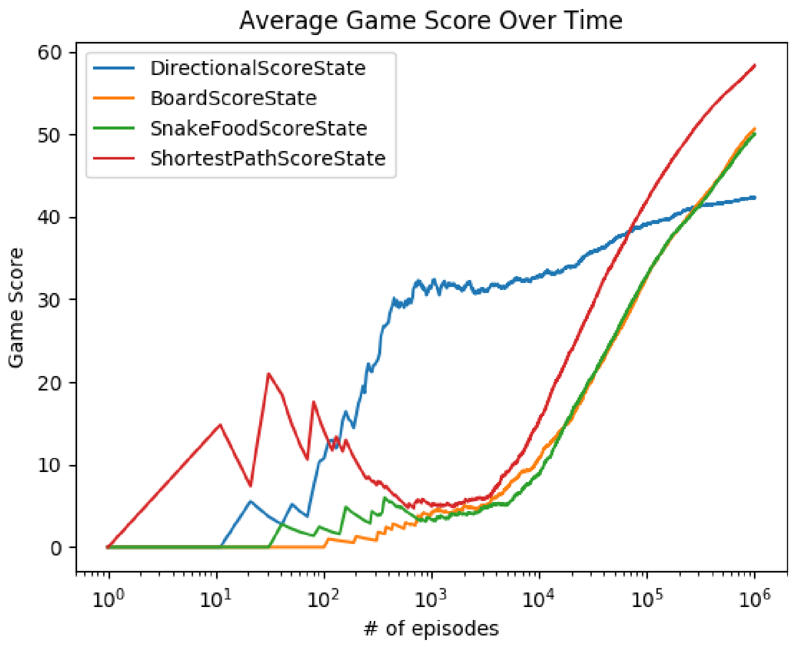


Figure 7: Performance of state representations trained using Sarsa.

State	Avg. Score	Std. Deviation
ShortestPathScoreState	118.59	38.95
BoardScoreState	101.02	10.05
DirectionalScoreState	72.16	71.60
SnakeFoodScoreState	101.02	10.05
Random Agent	20.13	46.47

Table 3: Test performance over 10000 episodes.

Discussion

In this part we discuss the results that we have gotten from our two experiments and point out what we have learned. We discuss them in the order they were presented previously.

Reward Results

The first thing we noticed from the reward experiment is that the reward functions create two clusters where those function within a cluster have similar performance. We can see that those that give a big negative penalty for border collision create one cluser and the other three functions create another cluster. For Q-learning the state representation seems to be important in determining which of the clusters perform better while that is not the case for Sarsa. That indicate that they do have different behavior. Already here we can see that the board state perform in general better than the directional state, we will look at it further in the analysis of the state experiment.

Since the average game score is not even 100 this means that the agent does not eat a single apple every episode which may indicate that it has not actually learnt anything and is just randomly moving around and gets lucky. By watching the agent play the game we have seen that it usually navigates to the first food source but then commits suicide by colliding with the game border. On rare occasions when the next food gets located nearby it eats two of them before colliding with the game border. That was the main motivation for creating the reward functions that penalize that kind of behavior. For those we saw it learns to get food until it was long enough to kill itself by self-colliding so it does not really learn to play but rather how to kill itself as soon as possible.

The results suggest that it is important that the reward function to reinforce desirable behavior. This is more obvious when looking in the appendix and compare states that do not give any reinforcement to those that do. This is further supported by the fact that the reward function and the game score are highly correlated as shown in table 1 which indicate that is it important that the reward function reflect the goal that we want the agent to achieve. This is further indicated by the example above about the agent that started killing itself by self-colliding when given large negative rewards for colliding with the game border. So penalizing or rewarding sub-goals, such as not colliding with the game border, may influence the agent's behavior in an undesirable manner.

State Results

The first thing we looked at was to compare the regular states with the augmented ones. From figure 5 it is obvious that the state should contain the game score for significant increase in performance. The board dimension do not seem to be important at all and perhaps the reason is that it is a static value, thus do not add anything in the long run. One problem may arise when including the game score variable given our tabular representation which is that as soon as the game score changes all information learned from another game score is wiped. This can also be true without the score if the state contain information about the length of the snake or positioning of the body.

In figure 6 and 7 we can see that the algorithms behave slightly differently, especially with regards to the directional state. That state seems to have converged with poor performance for Q-learning but still show some potential in Sarsa. The reason behind this difference is perhaps because Sarsa is an on-policy algorithm and is able to take advantage of the directional information better than an off-policy algorithm such as Q-learning.

The shortest path state has the best performance both for Q-learning and Sarsa. By watching its behavior in a simulator we could see that it actually performed decent in picking up 1-2 apples. Since this state do not change fundamentally whenever picking an apple except the game score part it may be that is suffers from the problem mentioned above. We can also see that it has not converged so longer training time is necessary to increase its performance.

What is particular interesting is that the board state and the snake-food state have almost identical performance. Important to note is that the snake-food state is basically a sparse representation of the board state so they do contain the same information and the algorithms do not seem to be bothered by that at all. This indicates that it can be beneficial to find a more computational efficient representation that contain the same information without big losses in performance.

Tables 2 and 3 give us some insight into the performance of the different states. They provide the same conclusions about the ranking in performance but interestingly enough, according to the standard deviation, the board and snake-food states are less variable. This may be because they contain the whole game state rather than some simplified representation. We can see that the random agent gets roughly one apple every 5th game and our models gets roughly one apple per game, except the directional state, which is not particular impressive.

We can say, from the results, that the state representation is also very important for learning and their performance can vary depending on your choice of algorithm. It seems that these methods are stable enough to not be disturbed by constant information such as the board dimensions and that such information can be considered essentially useless. However, given the results we cannot claim that the models have learnt to play Snake.

Conclusion

We have looked at two fundamental building blocks of reinforcement learning, the reward function and the state representation, to investigate their importance to learning. We used a simple variation of the game Snake as our problem. We have shown that the reward function and the state representation are important in order for the agent to learn the objective. The reward function should reflect the object and the state requires that it provides useful information. The characteristics of the learning algorithms do seem to play a part in which reward functions and state representations that are most useful which should be taking into consideration during the design phase of your reinforcement learning problem.

We used the algorithms as they were first discovered, but there have been many extensions to improve upon them. It would be interesting to see if the experiments give similar outcome with extensions such as double Q-learning [3], experience replay [5], and function approximations rather than a tabular representation of the Q-function.

References

[1] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

[2] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: An overview. In *Proceedings of the 34th Institute of Electrical and Electronics Engineers (IEEE) Conference on Decision and Control, 1995.*, volume 1, pages 560–564. IEEE, 1995.

[3] Hado V Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[4] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

[5] Long-H Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3/4):69–97, 1992.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[8] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction, volume 1. MIT press Cambridge, 1998.

[9] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 2094–2100, 2016.

[10] Christopher J C H Watkins. Learning from delayed rewards. PhD thesis, King's College, Cambridge, 1989.

A Reward Experiment

Here are some additional results from the reward experiment.

A.1 Q-Learning

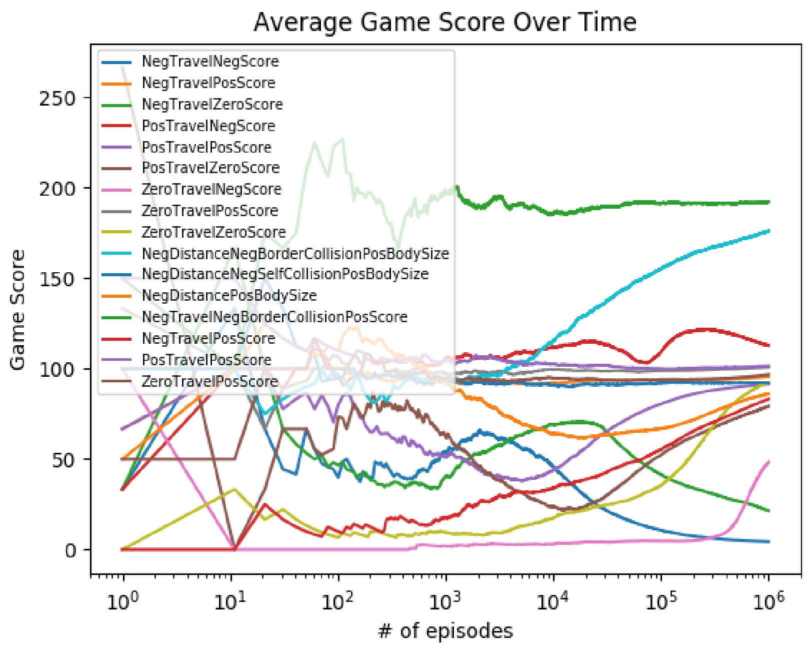


Figure 8: Performance of reward functions trained using Q-learning with the board state representation.

A.2 Sarsa

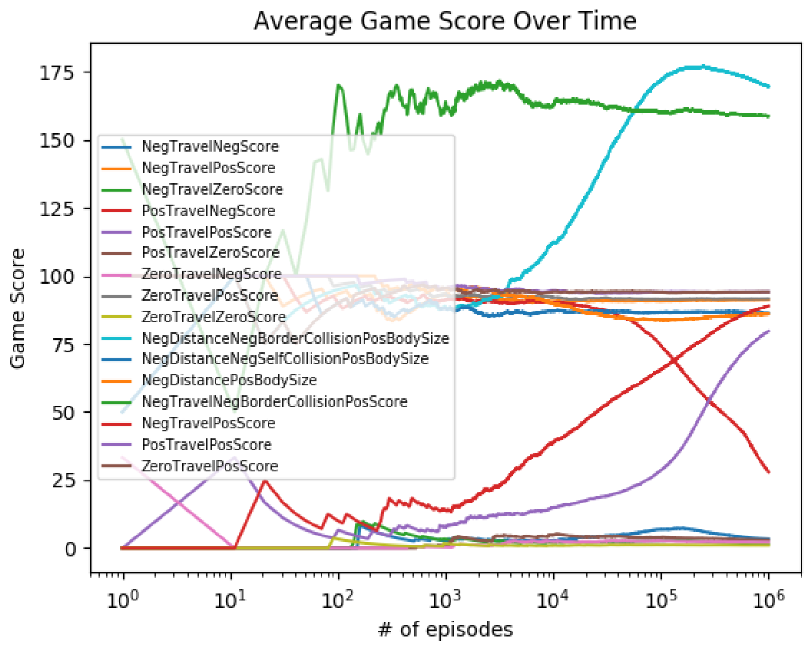


Figure 9: Performance of reward functions trained using Sarsa with the board state representation.

A.3 Expected Sarsa

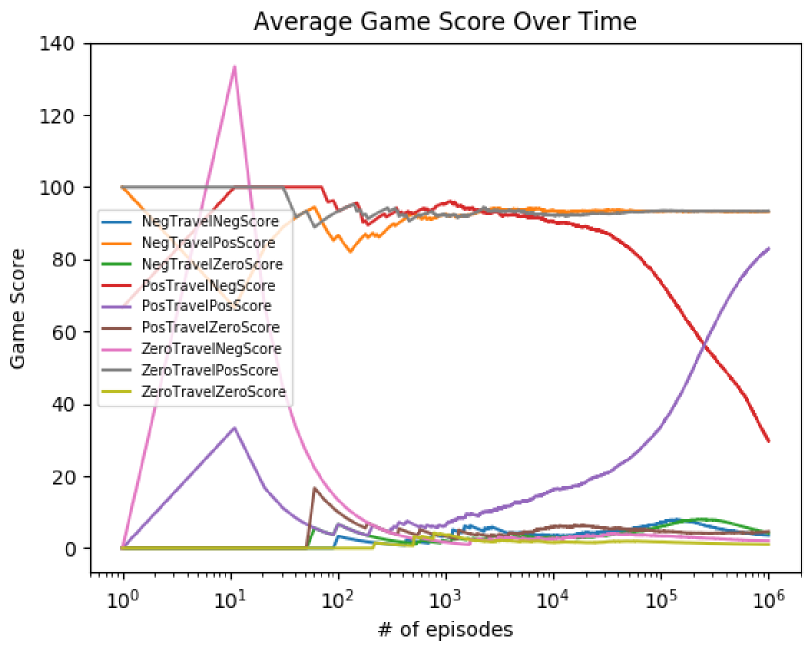


Figure 10: Performance of reward functions trained using Expected Sarsa with the board state representation.

B State Experiment

Here are some additional results from the state experiment.

B.1 Q-Learning

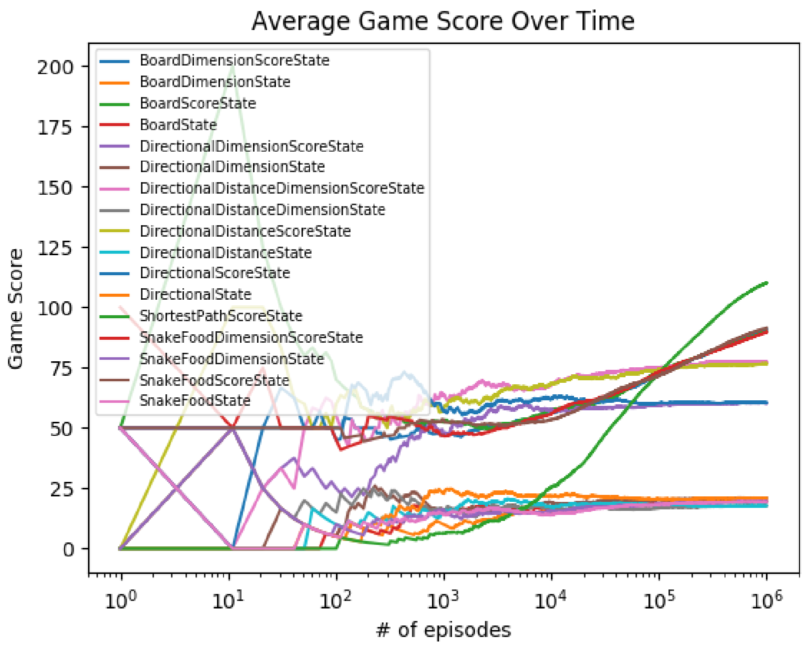


Figure 11: Performance of state representations trained using Q-learning.

B.2 Sarsa

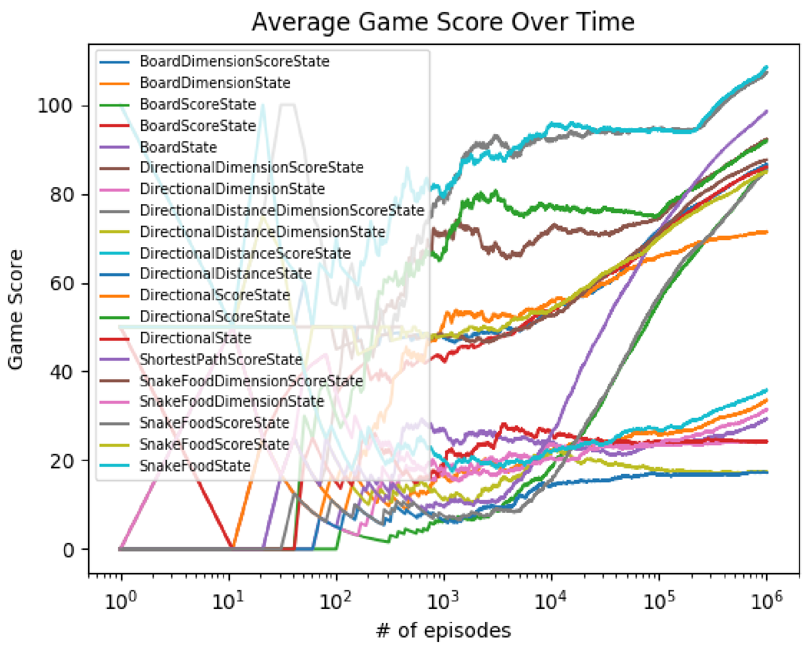


Figure 12: Performance of state representations trained using Sarsa.

B.3 Expected Sarsa

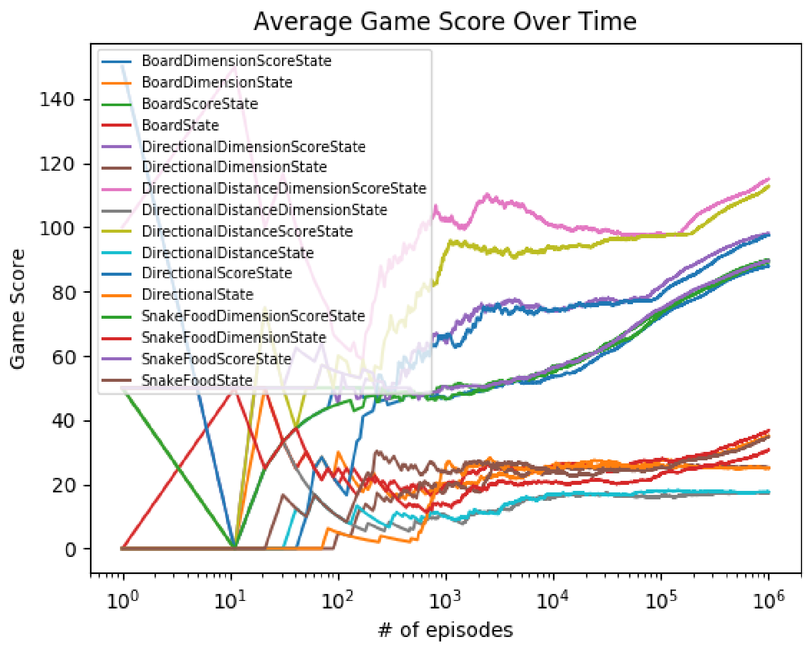


Figure 13: Performance of state representations trained using Expected Sarsa.