Reinforcement learning and searching in Pac-man

Artificial Intelligence CS 534 Fall-2017

Qi Wang
Worcester Polytechnic Institute
E-mail: qwang12@wpi.edu

Abudula Aihaitijiang
Worcester Polytechnic Institute
E-mail: aaihaitijiang@wpi.edu

Zeling Lei
Worcester Polytechnic Institute
E-mail: zlei2@wpi.edu

***Abstract*— In this project we implemented several search and learning algorithms in the Pac-man game. In Multi-Agent scenario, use adversarial search algorithms to optimal the game. We implemented adversarial search algorithms (Minimax algorithm, Alpha-beta pruning search, Expectimax algorithm) in Pac-man game. Experimental result shows that the Expectimax algorithm is much better than the minimax algorithm and alpha-beta pruning search (in the medium classic layout). At last, we implemented Q-learning and approximate Q-learning in this game. Pac-man will learn about the value of positions and actions or the weight of feathers. The game will run in two phases. In the training phase, the Pac-man can learn accurate Q-values in quiet mode. After training, it will enter testing mode. We compared the performance of these four algorithms, the Pac-man using the approximate Q learning approach wins the the maximum number of games.**

Keywords—Agent, Q learning, Approximate Q learning, Pac-man, Searching, Minimax, Expectimax, Alpha-Beta Pruning;

I. INTRODUCTION

In many games, there are game characters that interact with users and it is a measure of the quality of the game that how characters' behaviors are similar to realistic ones or whether the skills of characters can be adapted to the skills of the human player. Artificial intelligence is used to generate responsive, adaptive or intelligent behaviors primarily in non-player characters (NPCs) which is similar to human-like intelligence. The techniques typically draw upon existing methods from the field of artificial intelligence[1]. Current generations of computer games offer an amazingly interesting testbed for AI research and new ideas. Such games combine rich and complex environments with expertly developed, stable, physics-based simulation. They are real-time and very dynamic, encouraging fast and intelligent decisions. There are many examples of artificial intelligence being used in game. One of the first examples of AI is the computerised game of Nim made in 1951 and published in 1952 [1]. And also 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 [2]. Reinforcement learning (RL) covers the capability of learning from experience [3-5]. 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. 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. Among them, the arcade video game Pac-man constitutes a very interested test environment. Every year Pac-man competitions are held in the academic world, as they are a good way to show how AI methods can be used to simulate human behaviour.

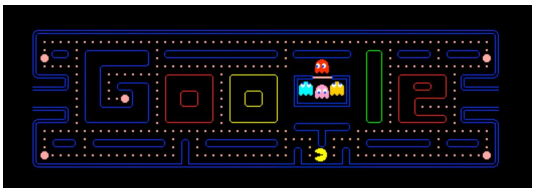

Fig.1: Pac-man game in a typical maze. In this game ghosts played by the computer and chase the Pac-man and try to catch it. Pac-man wins if it has eaten all dots, loses if caught by a ghost. Pac-Man can move (at most) up, down, left and right.

In this project, we investigate the Pac-man game since it offers a real time dynamic environment and it involves sequential decision making. Through this project we can learn and get a better understanding foundational AI concepts. In the Pac-man world, the Pac-man aims at eating more food pellets, while the ghosts want to catch up Pac-man and prevent it from winning. These two kinds of agents form an adversarial situation. We implemented adversarial search algorithms (Minimax, Alpha-beta, and Expectimax) to optimal the game. Our study focuses on the designing of an appropriate state space for building Q-learning (Particularly the Approximate Q-Learning Algorithm) agent to the Pac-man. The environment is difficult to predict, because the ghost behaviour is stochastic and their paths are unpredictable. The reward function can be easily defined by particular game events and score requirements (Fig.1).

We organized the paper as follows: In section 2 we give a brief description of the Pac-man game and background of the reinforcement learning. The proposed structure is presented at section 3 (method). In section 4, we discuss the details of our experiments. Finally, section 5 draws conclusions and discusses issues for future study.

II. Background

A. Pac-man game

Pac-man is an arcade game developed by Namco and first released in Japan on May 22nd, 1980 - it's considered to be a landmark in the history of video games, and is among the most famous arcade games of all time. The player maneuvers Pac-man in a maze that consists of a number of dots (or pills). The goal is to eat all the dots. There are also ghosts in the maze who try to catch Pac-man, and if they succeed, Pac-man loses a life. There is an action space consisting of the four directions in which Pac-man can move (up, down, right, left) at each step. In this project, we are referenced for UC Berkeley's artificial intelligence course, CS 188. And apply an array of AI techniques to playing Pac-Man [6].

B. Q-learning and Approximate Q-learning

In this part, we briefly introduce background of the Q-learning and Approximate Q-learning algorithm. In section 3, we will discuss how we implemented those algorithms in Pac-man game.

1) Q-learning

Q-learning can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP). In this algorithm, a policy is a rule that the agent follows in selecting actions. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state. One of the advantages of this algorithm is that it is able to compare the expected utility of the available actions without requiring a model of the environment. In our implementation Pac-man will play games in two phases. In the first phase, training, Pac-man will begin to learn about the values of positions and actions. Because it takes very long time to learn Q-values. Pac-man is trained in quiet mode, once Pac-man's training is complete, he will enter testing mode.

2) Approximate Q-learning

Q-learning agents that learn weights for features of state, where many states might share the same features. Advantage of the Approximate Q-learning is reducing the size of the Q-table and state will share many features. The approximate Q-learning algorithm is as follows modified algorithm[7]:

- Initialize $Q(s, a)$ arbitrarily
- Repeat (for each episode)
- Initialize s
- Repeat (for each step of episode)
- Choose a from s using policy derived from Q
- Take action a, observe r, s'
- $Q(s,a) = Q(s,a) + \alpha[r + \gamma n[max\ a'\ Q(s',a') - (1-\gamma)r'] - Q(s,a)]$
- $S \leftarrow s'$;

```
GetAction(state):
if state is a terminal state: return the utility of the state
if the next agent is MAX-Agent: return MaxValue(state)
if the next agent is MIN-Agent: return MinValue(state)
```

```
MaxValue(state):
V=-inf
for each successor of the state:
    V=max(V, GetAction(successor))
return V
```

```
MinValue(state):
V=inf
for each successor of the state:
    V=min(V, GetAction(successor))
return V
```

Fig.2: In this table recursive process are shown. Pac-man decides the action that has the max value among the minimum value generated by the ghosts' possible actions.

III. Method

C. Minimax algorithm

A minimax algorithm is a recursive routine to decide the next action in multi-player game. In the multi-level state-space search tree, there is a utility value associated with each node. For Pac-man game, the Pac-man is the max agent which selects the actions that achieve the maximum of utility value. The ghosts are min agents which exert an influence of minimize the utility value. These two kinds of players play the game in turn, that is, the Pac-man decides the action that has the max value among the minimum value generated by the ghosts' possible actions. The recursive process are shows in Fig.2.

D. Alpha–beta pruning

Alpha–beta pruning is also an adversarial algorithm that optimizes the simple minimax algorithm by reducing the large factor number in the minimax search tree. Alpha–beta pruning searches the tree the same as minimax algorithm, but does not need to visit the node with the value that will not influence the final decision of max agent or min agent. The Pac-man is still the max agent and ghosts are the min agents.

In the recursive routine, two values, alpha and beta, are kept. Alpha indicates the max agent's best option on path to root and beta indicates the min agent's best option on path to root. The node with value larger than beta will not influence the decision of min agent and the node with value smaller than alpha will not influence the result of max agent. These nodes can be pruned according to the algorithm.

E. Expectimax algorithm

The Expectimax algorithm is a search algorithm based on minimax search and was first put forward by Donald Michie in 1966. In the Expectimax search tree, the node of min agent in the minimax search tree becomes the chance node, that is, the outcome of the node is not worst but uncertain. The utility value of these chance nodes are not the minimum value but the expected utilities. There is a probability distribution of each outcome of chance node, so the expected utilities are the weighted average.

In this case, Pac-man is still the max agent, however, the algorithm assumes the performance of ghosts is not always the same with min agent. They become expect-agent. Each turn of the game is played by the max agent Pac-man, and expect agent ghosts. For the expect-agent, we assume the probability of each action of ghosts are the same. So, the probability is

$$Pr = \frac{1}{opponent-actions}$$

And we use this probability to calculate the expected utilities.

F. Markov decision processes

Before implementing q-learning in Pac-man game, we need to first understand the logic of Markov decision processes (MDP).

A MDP consists of five elements which is the usual setting of Reinforcement learning (RL). These elements are as follows:

S is a set of states.

A is a set of actions.

$T(s,a)$ are the state transition probabilities. For each state $s \in S$ and action $a \in A$, $T(s,a)$ is a distribution over the state space. $T(s,a,s')$ gives the distribution over what states we will transition to if we take action a in state s.

$\gamma \in [0,1]$ is called the discount factor.

R(s,a) is the reward function when taking action starting from s. [8]

Suppose the agent does the following actions:

Start from $s0$, take action $a0$ to $s1$,

Start from $s1$, take action $a1$ to $s2$

Start from $s2$, take action $a2$ to $s3$

…

Start from $sn-1$, take action $an-1$ to sn

The goal of reinforcement learning to to maximum the expected utility

$$U(s) = E(R(s0) + \gamma R(s1) + \gamma 2R(s2) + .. + \gamma(n-1)R(sn-1) + \gamma nR(sn)) \quad [8]$$

The reward is discounted by the parameter γ.

From the function, it follows that there is a direct relationship between the utility of a state and the utility of its neighbors: the utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action. So the Bellman function is given:

$$U(s) = Max(a)(\Sigma(s')T(s,a,s')(R(s,a,s') + \gamma U(s'))) \quad [8]$$

s' here is the possible next state by taking action a.

Based on the value iteration algorithm, we could find the best utility for each state.

Value iteration algorithm [5]:

1. For each state s, initialize $V(s) = 0$
2. Repeat until convergence {
   For every state, update

$$U(s) = Max(a)(\Sigma(s')T(s,a,s')(R(s,A,S') + \gamma U(s'))$$

}

G. Q - learning

In Q-learning, we know the set of states, and the set of actions as in MDP, but we do not know the reward function and the transition function in the beginning. Rather than evaluation the MDP model first and then find the value function in the policy, we directly estimate the value function without worrying about transition function and reward function.

For a fixed action, which is what we looked at first, we could learn the value with temporal difference learning. i.e.

new utility's $sample = R(s,a,s')\gamma max(a')Q(s',a')$,

Incorporate the new sample into a running average.

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(sample)$$

Then we do the Q value iteration and in Q value iteration the update is a weighted average so we can approximate it from samples and we can learn the optimal value function Q function and policy through Q learning. we get a sequence of experiences state, update the estimates at each transition. The Q value iteration is as follows, quite similar to the structure of value iteration algorithm [8]:

1. For each state s, initialize $U(s) = 0$.
2. Repeat until convergence {
   For every state, update

$$Q(s,a) = Max(a)(\Sigma(s')T(s,a,s')(R(s,a,s') + \gamma Q(s',a'))$$

}

In the process of learning, it does not matter how you select action. We use simple baseline way of exploring, is ε-greedy, which means you flip a coin before taking action, then weighted by ε and 1 - ε for each of the sides, with Probability = ε, take random action, otherwise act on the current policy.

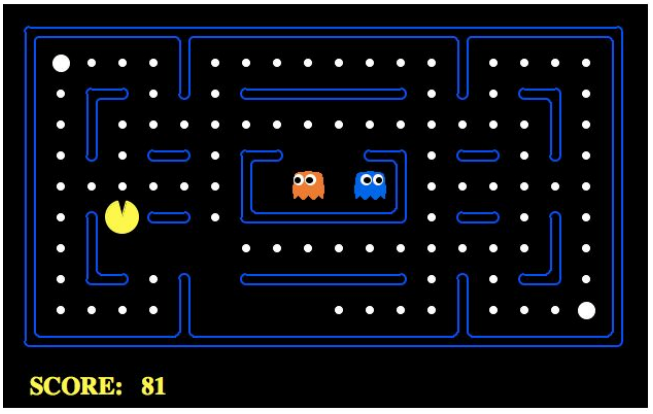

Fig3: Layout Medium Classic. In this project, Pac-Man agent will find paths through his maze world, both to reach a particular location and to collect food efficiently.

In the case of Pac-man, the state spaces are the combination of possible locations for Pac-man and possible location for ghost. With at most four actions per state. The food works as a reward. In sum, the paceman knows the location of the ghost, it knows the location of the food. In sum, it has full state information, just don't know how the game works. Overtime it learns to play.

H. Approximate Q - learning

Q learning has drawbacks in larger layout of in the game. You have to explore every state action pair infinitely. Then as you update the Q value estimates, one way to solve the problem is generalize the state, and we could turn to use feather based representation. In the case of Pac-man, we use "the number of ghosts 1-step away", "eats-food" added when there is no ghost, "distance to closest food" as the feathers. Using this feather representation, we can write a Q function for any state using a few weights:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a) \quad (9)$$

Next we need to calculate the weight value w_1, w_2, … w_n in the learning process. The algorithm following is the way of calculating the weighting parameters:

for $transition = (s,a,r,s')$,

$$difference = [R(s,a,s') + \gamma max(s',a')] - Q(s,a)$$

for $i \in [1,n]$ update w_i:

$$w_i := w_i + \alpha \cdot difference \cdot f_i(s,a) \quad (9)$$

This method is similar to linear regression, the difference is that we use the newly learned value $R(s,a,s') + \gamma max(s',a')$ instead of real value to to the calculation.

I. Assessment Protocol

The running time of three adversarial search algorithms (Minimax, Alpha-beta pruning and, Expectimax) suggests the performance of each algorithm in the same layout of the game. We know that the time complexity of the Minimax search is $O(b^m)$. With Alpha-beta pruning optimization, the optimal time complexity drops to $O(b^{m/2})$. And Expectimax is pretty like Minimax. The timer will start when the Pac-man starts the game, and when Pac-man wins or loses the game, the timer ends. The shorter time the algorithm costs, the higher efficiency the algorithm has.

It is obvious that the adversarial search algorithms are tree search methods and they all run in a bottom-up manner. However, the decision of Pac-man is influenced by different utility functions in these different algorithms. The number of movements of Pac-man can vividly indicate the variance of strategy taken by Pac-man. In the ideal world, an intelligent algorithm makes Pac-man go less steps and win more times.

For Q-learning and approximate Q-learning, we assess the Pac-man's performance in different layouts. In a certain layout, we calculate how long it takes for Pac-man to execute one training and how many times of trainings are needed for Pac-man to perform well in this layout. Most importantly, we evaluate the result by first see if Pac-man can win the game (i.e. eating all the food pallets without being hunted by ghost) with limited times of training. Then we calculate the percentage the Pac-man could win in a fixed amount of games. The algorithm that lead to the most winning games is the best for the layout. To evaluate the performance even in more detail, we could calculate the score everytime one game finishes. Then the average score is counted as the performance of the algorithm in a certain layout. The one that having the most average score is the best algorithm for the scenario.

To compare the performance of the five algorithms, we need to find out if a certain algorithm is feasible in a given layout. Find all that could be used, then calculate the percentage of winning games in the this layout. we do not compute score to judge the performance in this project.

IV. Result

When running the game, we use different algorithms on the default Medium Classic layout with two ghosts defined by the game framework. Each time we keep a record of the results, including the total score of Pac-man, the running time of the algorithm, the number of moves Pac-man has made and, whether the Pac-man has lost the game (Fig.4 and Fig.5).

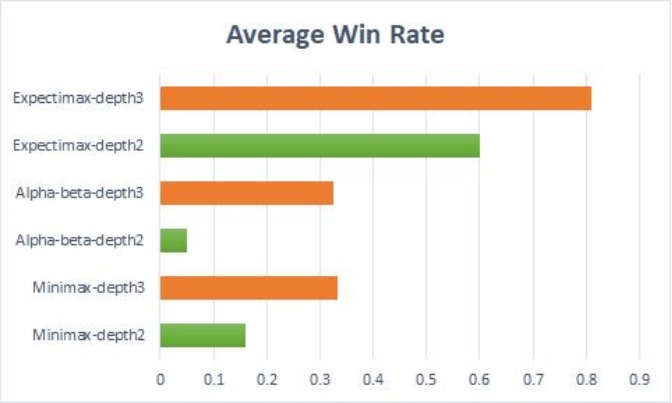

Fig.4: Average win rate of algorithms. In this figure shows result of the win rate between different algorithms(Expectimax, Alpha-beta, Expectimax)

Fig.5: Average running time of algorithms. In this figure shows result of the running time between different algorithms(Expectimax, Alpha-beta, Expectimax)

The Minimax algorithm was executed using default depth 3 over 40 times. Minimax algorithm leads the winning of 33% of the games. . The average of moves of Pac-man is 1151. The average score is -32.95. And the average running time is about 112.47s. In this case, the Pac-man often thrashes around even there is a dot next to him and this thrashing costs a lot of time. In the same situation, the alpha-beta algorithm performs better. Pac-man won about 33% of the games using alpha-beta algorithm. Pac-man moves around 1177 steps on average. The mean of scores is 59.19. The average running time of the algorithm is about 102.30s. When the search depth was decreased to 2, the running time got less but the win rate became very low.

The expectimax algorithm was also executed in the same situation using default depth 3 over 40 times. The result shows that the Pac-man won 81% of the game. The average of moves the Pac-man made is 330. The average score is 1345. And the average running time is 16.47s. While when the search depth decreased to 2, the running time has reduced to 12s, but the win rate decreased to 60% .

For Q-learning, the Pac-man works well in small layout, i.e. small grid (Fig.6). It takes time for him to learn the state, which should combine the Pac-man's position, the ghost's position, the two food pellets' and its positions, so the total amount of state is 12 * 12 * 4 = 576. And from each position, the Pac-man or the ghost could at most choose four actions, i.e. North, East, West or South. Thus the value Q(s, a) is what the Pac-man need to learn in the training process. After playing 2000 training games, the Pac-man wins 50 games for 50 testing games in the Small Grid layout.

However, in the larger layout Medium Classic as in Fig.3, the Pac-man loses 50 games after playing 2000 training games. So we resort to approximate Q learning, which generalizes large amount of states and compresses them to a few number of feathers. In the same layout, the Pac-man performs very well winning 48 out of 50 games after playing 50 training games. To our surprise, the agent could win 44 out of 50 games with mere 25 training. Also the Pac-man wins 46 out of 50 games with 100 times of training. Out of curiosity, we use a much bigger layout called Original Classic (Fig.7), which has 4 ghosts, many more food pellet, and thus much more complicated states for Q learning.

Fig.6: Layout Small Grid. We testing Pac-man implement Q-learning algorithm, the Pac-man works well in small layout.

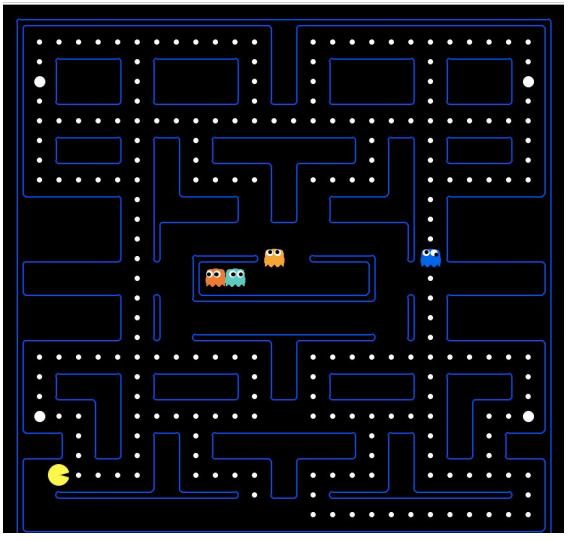

Fig. 7: Layout Original Classic. we use a much bigger layout called Original Classic , which has 4 ghosts, many more food pellet, and thus much more complicated states for Q learning.

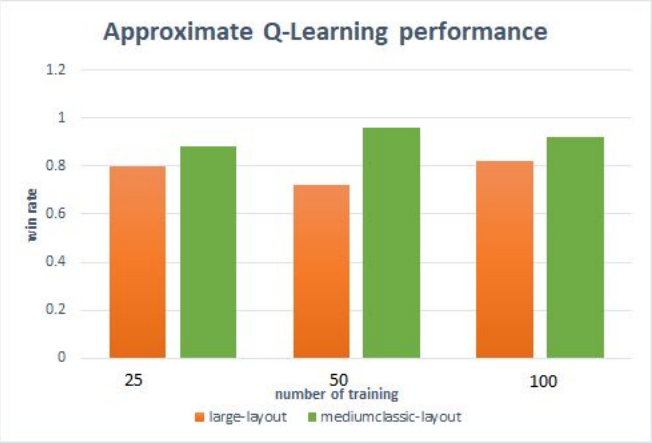

Fig.8: Approximate Q-Learning Performance. In this figure shows performance of the approximate Q-Learning in larger layout and medium class layout.

But it is not a problem for the modified Q learning. With the same number of feathers, the Pac-man played games in the largest layout. The result is impressive. He got trained with 25 games and win 40 out of 50 games, trained with 50 games and win 36 out of 50 games, trained with 100 games and win 41 out of 50 games. The results are shown in Fig.8.Though we could expect that the more the Pac-man get trained, the more games he would win. But the reality may not be the case. We conclude that a proper time of training could be enough for Pac-man to win 80% of games. So even if we do not train the Pac-man with 1000 games, and see how he could perform. the Pac-man would excel in playing this kind of game, With the correct choices of feathers and a certain time of training.

V. Conclusion

In accordance with the above results, we can draw some conclusions. Clearly, the Expectimax algorithm is much better than the Minimax algorithm and Alpha-beta pruning search. When playing with a random opponent, this algorithm performs better since the random adverse does not always leads to the worst outcome and the probability has been considered. So the Pac-man did not waste a lot of time thrashing around. Although the Alpha-beta search costs less time than Minimax algorithm, the Pac-man did not win more games and they both only designed for worst outcome. Moreover, the branch factor and search depth also influence the complexity of the games, which in turn forced the implementation of a reinforcement learning algorithm, Q-learning algorithm, which decides the actions of Pac-man not merely based on the current game state, but can intelligently have an insight into the surrounding environment.

Compared the performance of these four algorithms, the Pac-man using the approximate Q learning approach win the most of games. This feature based learning algorithm also saves times for training. In the Small Grid scenario, it takes a Pac-man 2000 times of training in order to learn the best policy at each state. But for approximate Q learning, we use feature instead of state action to express to value of Q value. After the states are generalized, a mere 25 times of training would make Pac-man win 80% in 50 games. In the Q learning scenario, Pac-man would never get a chance to win even one game. But there is a tradeoff between learning fast and state representation. The summary of states could lead to the cases that the Q value is the same calculated by weight and feather, but in reality, the Q value could be totally different even though the feather value is the same.

For future study, we could combine the Deep Neural network with Reinforcement learning, and see how the Pac-man would perform in this scenario. In reality, some data such as time series data may not be expressed in MDP, so no state and action pair could be found. Deep learning could creating architecture of hierarchical feature representations [10], features, both learned and hidden, as well as weight could be generated during the process.

References

[1] Wikipedia:https://en.wikipedia.org/wiki/Artificial_intelligence_(video_games)#cite_note-5.
[2] L. Galway, D. Charles, and M. Black. Machine learning in digital games: A survey. Artificial Intelligence Review, 29:123–161, 2008.
[3] 2. R. Sutton. Learning to predict by the method of temporal differences. Machine Learning, 3(1):9–44, 1988.
[4] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 4:237–285, 1996.
[5] R.S. Sutton and A.G. Barto. Reinforcement Learning: An Introduction. MIT Press Cambridge, USA, 1998.
[6] http://ai.berkeley.edu/project_overview.html.
[7] P.Deepshikha and P.Punit. Approximate Q-Learning: An Introduction : 2010 Second International Conference on Machine Learning and Computing.317-320. 2010.
[8] Stanford machine learning courses notes http://cs229.stanford.edu/notes/cs229-notes12.ps.
[9] UC Berkeley Intro to AI courses slides reinforcement learning II http://ai.berkeley.edu/lecture_slides.html.
[10] Deep learning for reinforcement learning in Pac-man. http://www.ias.tu-darmstadt.de/uploads/Site/EditPublication/Hochlaender_BScThesis_2014.pdf.