Apply Reinforcement learning to SNAKE

CS440 Artificial intelligence term project

Zheng Li, *Civl and Environmental Engineering Department, University of Illinois at Champaign-Urbana*

***Abstract*—SNAKE is a classic video game that originated from the concept where the player maneuvers a line which grows in length, with the line itself being a primary obstacle. In the game, the trail grows longer as the player acquire more food in the plane, the game becomes progressively more difficult because the player has to avoid both plane boundary and snake itself. This term project aims to develop an artificial agent that plays SNAKE using the Q-learning algorithm. This project adapts relative quadrant to represent the whole game state which significantly improves the computational cost. The final result shows that the algorithm converges at 1000 learning games with reduced state-action domain and the trained agent is able to capture more than 20 food items in a 10x10 plane board. It is very crucial to choose efficient state space when performs reinforce learning as the size of state space directly affects the efficiency of the learning process.**

***Keywords*—Artificial intelligence, Q-learning, SNAKE game.**

I. Introduction

SNAKE (Figure 1) is a popular video game that has been popularized by Nokia Mobile phones in 1998. It is originated from the concept where the player maneuvers a line which grows in length, with the line itself being an obstacle as well. In the game, the player starts with a dot on a two-dimensional plane and it moves forward to collect food item, it leaves a trail behind as it moves. The trail gets longer as player acquire more items on the plane. Consequently, the game is progressively more difficult as the snake grows longer and it will eventually end in two scenarios: 1) player hit the border of the plane. 2) player hit its trail. The score is counted as how many food items the player has collected. The reinforcement learning is one of the most intriguing technology in artificial intelligence, it learns the optimal strategy in any states by trial and error using certain reward function. It could be very useful in the problem that cannot be represented as explicit mathematical form. The goal of this project is to develop an AI agent to play SNAKE game using reinforcement learning algorithm and discuss the agent's performance.

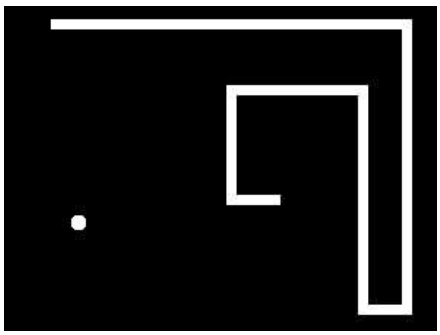

Figure 1. SNAKE gameplay schematics

II. Methods

A. Mathematical Analysis of SNAKE game

Since the whole goal of SNAKE is to collect as many food items as possible without hitting border as well as its own body. each snake movement can be considered as a self-avoiding walk (SAW) of a lattice in $\Re^2$. The most simple and straightforward way to play the game is to find the shortest way to the collectible item (food). However, Viglietta(2013) [1] proved that any game involving collectible item, location traversal, and a one-way path is NP-hard. In this project, Snake is a game that involving traversing a one-way path, i.e., the snake cannot cross itself. Besides, the new collectible item is randomly generated once snake collected one, So picking every single shortest SAW for each step in an episode is NP-hard.

B. Reduction of State Space for Reinforcement Learning

One of the most fundamental elements of conducting reinforcement learning algorithm is to ensure a finite state space. If we consider the state space as the location of snake and food, the size of state space is enormous due to the randomness of collectible item and length of the snake. Thus, using the exact location of food and snake could result in a large size of state space. Therefore, a different view of state space characterization is used in our algorithm. Instead of considering the explicit location of the snake, food. The snake location can be parameterized by one of the three conditions:

$$\{h_{straight}, h_{left}, h_{right}\}$$

$h_{straight}$,h_{left}, and h_{right} is a three-class variable indicating whether there is a food, border or snake body in its corresponding direction relative to snake head. It can be written as:

$$h_{i\in\{straight,left,right\}} = \begin{cases} 1 & \text{contains food} \\ \text{-1} & \text{hits body\&border} \\ 0 & \text{nothing} \end{cases}$$

Similarly, let's assume (v_x, v_y) is the vector point from snake head to food dot in the game broad. the food location can be characterized by two binary variables $\{f_x, f_y\}$. Since the food item will never appears on the snake head, so the (v_x, v_y) will never be (0,0). A mathematical representation is:

$$f_{i\in\{x,y\}} = \begin{cases} 1 & \text{v}_i > 0 \\ \text{-1} & \text{v}_i < 0 \end{cases}$$

Using $\{h_{straight}, h_{left}, h_{right}, f_x, f_y\}$ to represent game state, the total size of state space is therefore reduced to $3^3 \times 2^2 = 108$, and the training time will be improved prominently.

C. Reinforcement Learning - Q learning

In Q-learning, an agent tries to learn the optimal policy from its history of interaction with the environment. A history is defined as a sequence of state-action-rewards:

$$\{s_0, a_0, r_0, s_1, a_1, r_1 ... s_n, a_i, r_i\}$$

For the Snake game, the game itself can be treated as a Markov Decision Process, and the agent only needs the information from the previous state. Given a state s_i, the agent is able to take the optimal action a_i to reach state s_{i+1}. A total Payoff table $Q(s, a)$ is the discount reward and the aim is to maximize the value of $Q(s, a)$ using following equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \textbf{argmax}\{Q(s^{'}, a^{'})\} - Q(s, a))$$

In the above, α is the learning rate, γ is the discount factor. r is the reward function in the game. Above iterative update has been proven to be converged to the local-optimal Q-function. The snake movement includes three possible move:

$$\{Go\ Straight,\ Move\ Left,\ Move\ Right\}$$

A ϵ-greedy approach is implemented to allow the agent to explore more possibilities during the learning process. The exploration probability is ϵ is set to be 0.01 during training. This enables the agent to explore a little more possibilities by taking random actions thus narrow down the optimal actions. A pseudocode of Q-learning algorithm is shown below:

Algorithm 1 Q learning with ϵ greedy policy

```
Initial State s ← MDP(θ)
while n < number of learning games do
    Initialize Q(s, a) ← 0.
    Choose action a from s using ϵ-greedy policy
    r ← s.Rewards()
    while snake not hit wall/itself do
        s_t ← s.doAction(a)
        Choose action a from s using ϵ-greedy policy.
        Q(s_t, a_t) ← Q(s, a)+α(r+γ argmax{Q(s', a')}−Q(s, a))
        r ← s.Rewards()
        s ← s
        a ← a
    Update game parameter
Output final Q-Table
```

III. Results

A. Tuning parameters

The discount factor γ was set to be 0.90, a decay learning rate α was used during the learning process:

$$\alpha = \frac{20}{100 + t}$$

The rewards were shown in Table 1

Table 1: Rewards Table

Scenario	eat food	hit wall	hit itself	else
Reward	+100	-50	-50	-10

The reason of setting -10 at $'else'$ is to ensure snake to find the food as quick as possible, otherwise, the snake might fall into infinity moving cycle in the board without heading for food.

B. Learning Curve

SNAKE agent was trained with 1000 games and 5000 games respectively. The learning curve of each case is shown in Figure 2 and Figure 3.

Figure 2. Learning Curve with 1000 iterations

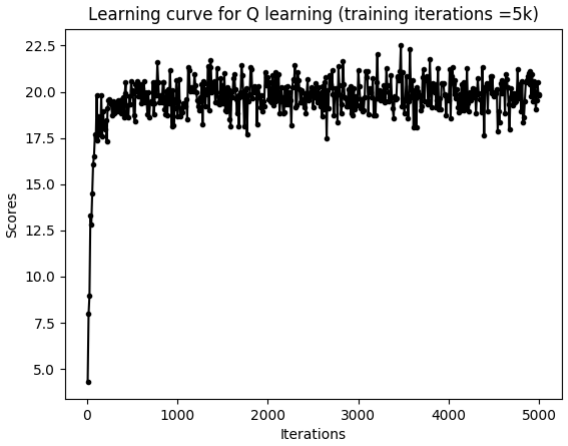

Figure 3. Learning Curve with 5000 iterations

The score shown in the learning curve is the average score over 100 test games. Figure 2 shows that the agent is able to collect more than 20 food items in a 10×10 game board when it has been learning 1000 games. This is because the size of state space is only 108 and the entire Q-table is 108×3 matrix. In comparison, In the previous assignment, the Pong game's state space has the size of 10369, which is much larger than

this project. The pong game requires more than 100K games to be able to learn a good agent, while SNAKE shows a good performance only at 1000 games. Figure 3 shows a learning curve with 5000 learning games. It indicates that after 1000 games, the agent has reached its optimal performance as the scores will not increase but exhibit behaviors of fluctuations. A video clip of agent play SNAKE is also included in the code package.

IV. Conclusion

In this term project, I tried to apply the similar concept learned in Pong game and develop an agent to play SNAKE. The most important lesson learned from this term project is that having a right view of how to formalize the problem plays a significant role in reinforcement learning. For instance, the size state space would be more than 10^5 if I use the exact location of snake and food items as the representation of the game state. That ended up with maximizing a large Q-table as well as learning a tremendous amount of learning games. Using reduced state space, the agent is able to learn how to play much more quickly (10^3 games) with good performance. The learning curve also confirms that 10^3 games are enough for an agent to be trained.

V. FUTURE WORK

Due to the limitation of computation resources and project time. This project only focus on developing a AI agent that plays SNAKE well. However, there are many potential work to be done in the future:

1. A sensitivity analysis can be conducted by changing parameters. One of the potential attempts could be to see how the rewards settings will affect the agent performance. In fact, this has already studied during the parameter tuning process. It is found that the value of rewards when snake hit nothing affects the process of convergence. For instance, if set rewards of hitting nothing as 10, then the agent will spend more time moving on the board and do nothing since there is no penalty for doing nothing. However, giving a small penalty of doing nothing will ensure snake to collect item as quick as possible. But setting different penalties for hitting walls and snake body is yet to be done.

2. Different algorithms are suggested to be implemented as a comparative study. One of the possibilities of to derive a deterministic heuristic algorithm for SNAKE and compare its performance with AI agent.

3. Instead of developing one agent trying to collect food items as many as possible, an alternative is to develop two agents at the same time and let them compete on the same game board. Because in the original settings, the primary obstacles are board boundary and snake itself. Two snakes in the same board would introduce more dynamics in the game since it has two snakes as obstacles for both of them to avoid. The first change would be increasing the state space to let agent differentiate its own body and opponents when is about to hitting a snake body.

References

[1] Brian Sallans and Geoffrey E. Hinton, *Reinforcement learning with factored states and actions*, 5:10631088. Journal of Machine Learning Research, 2004.

[2] Giovanni Viglietta, *Gaming is a hard job, but someone has to do it!*, 2013.

[3] Jun Zhang, Bowei Ma and Meng Tang, *Exploration of Reinforcement Learning to SNAKE*, 2015.

[4] Risto Miikkulainen, Bobby Bryant, Ryan Cornelius, Igor Karpov, Kenneth Stanley, and Chern Han Yong., *Computational Intelligence in Games*,