Advanced Pac-Man Simulation with Reinforcement Learning

Yuling Liu 504777221

Zilong Zheng 204761783

Hanlin Zhu 604738818

Qi Qu 404888347

Abstract— **Pac-Man is a famous arcade video game in 1980s. Animats, whose virtual simulation modeling animals adaptive behavior, is also an interesting field during the past few years. Combining these two ideas, we implemented a 2D grid based simulation that simulates the behavior of Pac-Man and monsters in an coexist ecosystem. By applying reinforcement learning on both species, the generation will show intelligence growth as iterations grow larger. In addition, we also carefully designed 3 experiments to test certain assumptions, and result shows that the adaptive and learning behavior do happen within either Pac-Man and monsters spices.**

I. INTRODUCTION

The original Pac-Man game is fun to play. However, when the agents difficulty is set to a high level it is usually hard to beat. This also holds for many other games. Thus, one question comes to our mind; Is there a way to automate the process of developing the players intelligent level? Fortunately, the answer is yes. As a result, we used the pygame to simulate the game, and used reinforcement learning to train both the Pac-Man and the monsters brain. Besides, we added some new game objects such as different food type (represented as pac-dots), together with rocks and grass.

A. Background

Our advanced Pac-Man simulation will consist of many Pac-Man in a wide, dynamically changed wrap-around world. In the constructed 2D world, there are monsters wander around. Monsters can eat the Pac-Man and make a one-hit knockout to it. There are also obstacles and grass which serve the functionality of blocking the way and rest shelter for the Pac-Man respectively. There are two kinds of pac-dots now. Besides the normal pac-dots, a new species of pac-dots which are being poisonous begin to grow. At the very beginning, all Pac-Man cant distinguish these two species. After having a bite on the poisonous pac-dot, the Pac-Mans speed will slow down, which makes it easier for monsters to kill.

B. Hypothesis

Pac-Man and monsters that live in this wrap-around world where two kinds of pac-dots, one being poisonous and the other being normal exist, will learn how to adapt to the environment.

1) Pac-Man will learn how to avoid eating poisonous pac-dot.
2) Pac-Man will learn how to rest on the grass and hide under grass to avoid eaten by the monsters.
3) Monsters will learn how to chase after Pac-man to kill them.

C. Goals and problems

To implement this reinforcement simulation, we have some goals to achieve towards the end, and we will also list the problems that need to be solved in this section. For the goals, we want to

1) Simulate Pac-Man and monster model in a wrap-around world with different kinds of objects. Specifically, the simulation should meet the following requirements.
    a) initialized objects and new generated objects should avoid collision with any other environmental objects.
    b) Pac-Man and monsters should move around in a wrap-around world, with or without learning, and avoid colliding with obstacles.
    c) Pac-Man and monsters should properly perform eating action, such that Pac-Man should be eaten as long as they are caught by monsters. The same applies for Pac-Man and pac-dots.
2) Create reinforcement learning model for Pac-Man and monsters to show successful learning outcome in the simulated world, and the following two requirements should meet.
    a) Learned Pac-Man should show completely different action taken compare to Pac-Man without learning when they sensor nearby monsters, pac-dots or grass.
    b) Learning results for both Pac-Man and monsters should improve as number of iterations increase.
3) Make comparisons between different kinds of reinforcement learning algorithms and show the differences. For the problems to solve, we want to figure out
    a) How to randomly generate different kinds of objects in a wrap-around world without collision?
    b) How to setup current state of Pac-Man and monsters to simulate their sensors based on the reinforcement learning?
    c) How to make coordinates of different kinds of objects accessible to both species as well as make environment keep track of status of Pac-Man and monsters at the same time?
4) How to apply brain of both Pac-Man and monsters (learned Q-table) to next generation after death so that new generation can keep learning?
5) How to tune different parameters so that Pac-Man and monsters can learn efficiently, such as parameters in

learning algorithms, number of different objects, and rewards for different actions?

II. Related Work

Applying reinforcement learning to animat modeling is a popular field in recent years. In this section, we will review some important works in the intersection between reinforcement learning and animat modeling. Those works are related to our work either in the method they use(i.e. reinforcement learning) or the context they apply reinforcement learning.

The most typical application of reinforcement learning in animat modeling is to learn a realistic locomotion policy for artificial life. One work in this stream of research is to learn the control policy for the cart-pole swing-up by using a class of reinforcement learning algorithms called actor-critics. In the work of Dr. Wawrzynski, he combined actor-critics method with a technique called experience delay to train a half-cheeta model to learn the cart-pole swing-up policy. He showed that the control policy is learned after four hours of training. There are also works concerned with learning locomotion of animats without using reinforcement learning. For example, the work from Dr.Terzopoulos in the early years about artificial fish is a representative in this stream.

Another stream of artificial intelligence research concerns how to apply reinforcement learning algorithms to achieve good performance in games. Especially with the recent advance of deep reinforcement learning, games such as DOOM and Pac-man are intensively used as examples of deep reinforcement learning applications. For example, a group of people lead by Dr. Philip Torr augmented deep Q-learning with SLAM technique from robot navigation to train a program for playing DOOM. In our work, we also chose the Pac-man game as the context. The difference between our work and deep reinforcement learning works is that we did not use any deep neural networks for state learning. We showed that a simple Q-learning method is also capable to model our task.

III. Approach Description

The simulation system is developed using Python with pygame Library. To simulate the strategy selection of Pac-Man, we controlled all environment variables but change one to see how the Pac-Man react to the sudden change of Environment variables. Our goal is to find the optimal policy that learned by Pac-Man is beneficial to Pac-Man, i.e. the speed of death will decrease as iteration goes up.

In our experiments, we focus on actions of Pac-Man and Monsters and the decision making for the next action will be based on the Q-table only. In addition, both Pac-Man and Monsters should has learning ability and we expect them to behave more intelligent compare to the first generation. On the other hand, all actions in mating as well as evolution have been simplified, because we do not distinguish between genotype and phenotype for Pac-Man' offspring. Instead, when a Pac-Man dies, a new Pac-Man will inherit the dead Pac-Man's brain and be randomly spawned on the map to keep the amount of Pac-Man unchanged.

A. Model

We used basic reinforcement learning [5] to simulate brain of Pac-Man and Monster. In this case, animats will be able to learn the correct strategy from the positive or negative rewards from the environment. In this project, we simulated both Q-learning and SARSA and compare between the results with learning strategy and results without learning ability.

1) Q-Learning: Q-Learning[6] is a reinforcement learning technique that works by learning an action-value function that ultimately gives the expected utility of taking an given action in a given state and following the policy thereafter.

The prediction of Q-learning can be updated with respect to the predicted return of the next state visited, which is expressed by following equation.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a)) - Q(s_t, a_t)) \tag{1}$$

Learning rate α determines what extent the newly acquired information will override the old information. We set it to be 0.2 to gradually learn from every updates.

Discount factor γ determine the importance of future rewards. We set it to be 0.99, which makes it strive for a long-term high reward.

State variable is defined as a 3x3 matrix around the current coordinates, where each cell denotes type of object at specific position relative to Pac-Man's position. One cell refers to a size of 20x20 pixels in our simulation and we indexed different objects from 1 to 6, which denote Pac-Man, Monster, grass, obstacle, position Pac-dots and normal pac-dots respectively. We use 0 if there is no object in the detecting position.

2) SARSA: State-Action-Reward-State-Action(SARSA) [7] is a reinforcement learning algorithm that will interact with the environment and update the policy based on the actions taken, known as an on-policy learning algorithm.

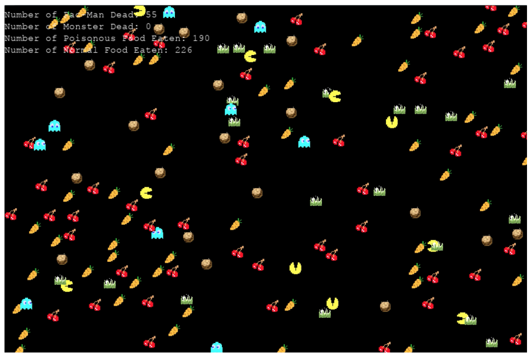
Different from Q-Learning, SARSA learns Q values associated with taking the policy it follows itself, which is denoted by the following equation.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_{t+1} + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2}$$

B. Environment

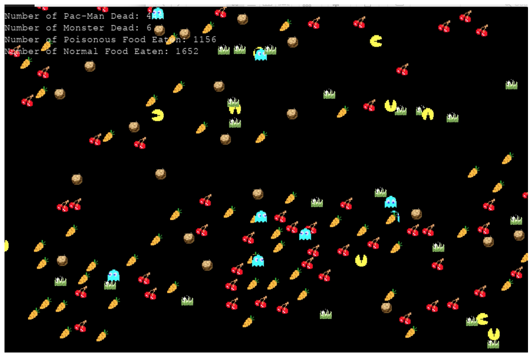
Figure 1 shows the simulation of the world. We used wrap-around unbounded 2D plane of size 800 by 800 pixels. All objects are in size of 20x20 pixels and are initialized randomly without collisions. In this case, we can measure the position of every object in cell and the wrap-around world simulates the infinite world which is better for the learning process to converge.

To ensure that poisonous and non-poisonous pac-dots have positive and negative effect to Pac-Man. Poisonous pac-dots



```
Number of Pac-Man Dead: 55
Number of Monster Dead: 0
Number of Poisonous Food Eaten: 190
Number of Normal Food Eaten: 226
```

Fig. 1. Initial Stage of Pac-Man Simulation



```
Number of Pac-Man Dead: 4
Number of Monster Dead: 6
Number of Poisonous Food Eaten: 1156
Number of Normal Food Eaten: 1652
```

Fig. 2. Intermediate Stage of Pac-Man Simulation

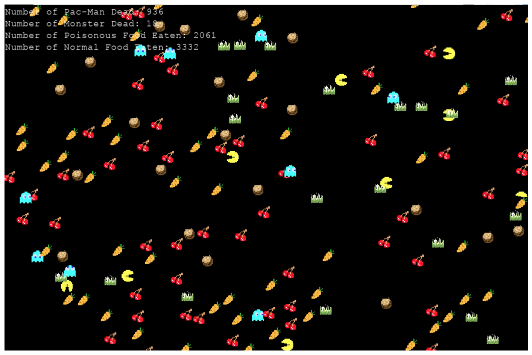
will return a negative reward (-50) when eaten by Pac-Man and the speed of Pac-man will be lowered by half. In contrast, non-poisonous pac-dots will return a postive reward (+50) to Pac-Man. With this setting, we expect Pac-Man prefer to eat more non-poisonous food after learning.

Three screenshots are taken during real simulation execution with Q-learning enabled. Figure 1 represents the initial stage, the amount of normal pac-dots eaten and poisonous pac-dots eaten are at the same level. Figure 2 is an intermediate stage where iteration goes larger, and the difference between two kinds of pac-dots eaten begin to vary. Figure 3 represents a point that towards final stage, and it is obvious to see that the number of normal pac-dots eaten far exceed the poisonous pac-dots, which reflects the learning ability and the increased intelligence of the Pac-Man species.

C. Agent

Similar to all other objects, the size of angents (Pac-Man and Monster) is fixed to be 20x20 pixels. To make it closer to a real world simulation, we decreased energy level of agents while walking and set the reward for walk one step to be -1 and all agents would die if the energy level is decreased to be zero.

Since we want to see the change of strategy by generation, the brain of Pac-Man, i.e. the learned Q-table, will be inherited to next generation after death, while speed, energy level and all other physical attributes will be reset. In our design, Pac-Man has a probability of ϵ to randomly choose next action regardless of current status and learned Q-table.



```
Number of Pac-Man Dead: 936
Number of Monster Dead: 18
Number of Poisonous Food Eaten: 2061
Number of Normal Food Eaten: 3332
```

Fig. 3. Final Stage of Pac-Man Simulation

This is because there is always a probability that creatures don't follow the learned strategy in real world.

IV. Evaluation

In order to investigate the learning outcome exhibited by Both Pac-Man and monster species, we carefully designed three experiments in this section by controlling either their brain or external game objects that can potentially affect the learning curve. Besides, we also designed an extra experiment which compares Q-learning and SARSA so that a better learning algorithm can be chosen to simulate our environment.

There are many variables in our simulation. To make it simple, we use the following shorthands to represent the main configurations: Pac-Man is abbreviated as PM with two associated fields. The first is the Pac-Man's amount while the second is the learning enabling field. For example, PM(30, Q-Learning) means the environment is initialized with 30 Pac-Man, Q-learning is applied to their brain. Monster is abbreviated as MS, and it has two exact fields as Pac-Man; The first represents the Monsters' amount while the second represents the learning method. Thus, MS(10, No learning) means after initialization there are 10 monsters without learning ability in the map. Pac-dots also has two fields associated with the abbreviation PD. The first field indicates the number of poisonous pac-dot while the second indicates the number of normal pac-dot, so PD(30, 40) means there are 30 poisonous pac-dots and 40 normal pac-dots in the map. For other configurations' shorthand, Grass amount is abbreviated as GA and Obstacles amount is shortened as OA, both with one field indicating the amount.

A. Learn to avoid poisonous pac-dots

In the first experiment, we want to test the learning ability of Pac-Man in distinguishing poisonous pac-dots and normal pac-dots. We fixed the initial environment setting with configurations of GA(20), OA(20) PD(50, 50).

Figure 4 shows the number of different pac-dots eaten by Pac-Man without learning ability, i.e. PM(20, no learning) and MS(20, no learning). Figure 5 has PM(20, Q-learning) and MS(20, Q-learning) to see how Pac-Man with learning ability react to the same environment settings in order to live longer.

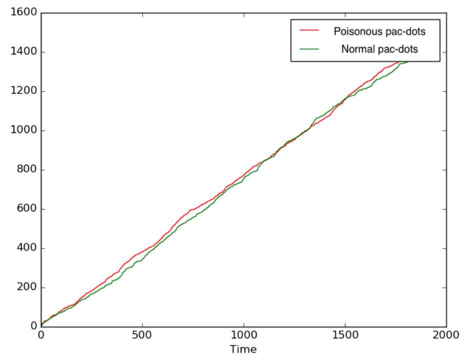


Fig. 4. Disable Q-learning

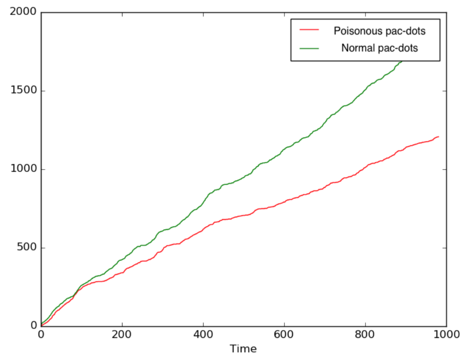


Fig. 5. Enable Q-learning

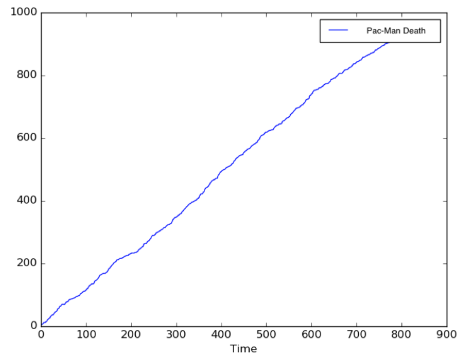


Fig. 6. Pac-Man death without grass

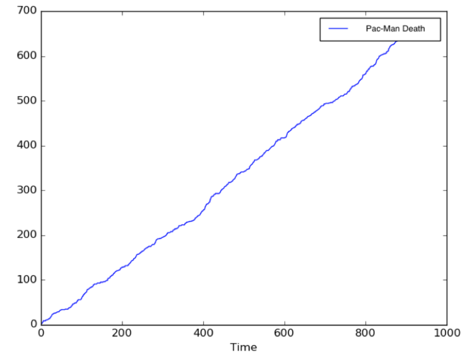


Fig. 7. Pac-Man death with 50 units of grass

Based on the figures, it is clear that disable Q-learning and enable Q-learning yield completely different results. Figure 4 shows as time goes by, the poisonous pac-dots and normal pac-dots eaten by the Pac-Man and monsters are about the same, and this is because without learning the Pac-Man will eat all kinds pac-dots blindly. On the other hand, Figure 5 shows as time goes by the difference between normal pac-dots and poisonous pac-dots eaten by Pac-Man will increase, which is obvious since this time Pac-Man will learn how to avoid poisonous pac-dots after the bite because eating poisonous pac-dots again will slow down its speed and make it easier for monsters to kill.

B. Learn to Take Advantage of Grass

In the second experiment, we want to test if Pac-Man can learn to rest on the grass or hide under the grass in order to avoid killing by the monsters. The two figures have four same configurations which include PM(20, Q-learning), MS(20, Q-learning) OA(20) and PD(20, 20). They only differs in Grass amount, where figure 6 is set to no grass with GA(0) and figure 7 is set to GA(50).

Figure 6 and figure 7 both show the death rate of the Pac-Man species as time iteration goes larger. However, figure 6 is simulated without grass while figure 7 is given 50 units of grass. It is easy to observe that the death amount of Pac-Man in figure 6 far exceed the death amount of Pac-Man in figure 7 if we set the same iteration for both of them. As a result, we can say Pac-Man gains the learning ability of resting on the grass since in figure 6 the Pac-Man group will have no place to hide.

C. Learning ability of Monsters

In the third experiment, we want to test the learning outcome of monsters assuming all Pac-Man can learn. We control GA(20), OA(20), PD(20, 20) and PM(20, Q-learning) the same for both testing groups. But for figure 8, we set MS(20, No learning) while enable Q-learning in figure 9.

Figure 8 shows the death rate of the Pac-Man will decrease and eventually converge since Pac-Man can learn how to avoid Monsters while Monsters can't learn how to catch the Pac-Man. In Figure 9, when we re-enable Q-learning for Monsters, we see a very similar figure as we see in the previous experiment. The death rate is about constant due to learning of both Pac-Man and Monsters, only the Pac-Man death amount is greater compare to Figure 7 since Figure 7 has more grass in the environment.

D. Comparison between Q-learning and SARSA

This is an extra experiment we did to evaluate the performance between two reinforcement learning algorithms. GA(20), OA(20) and PD(20, 20) are set to both testing groups. The difference is we set PM(20, Q-learning), MS(20,

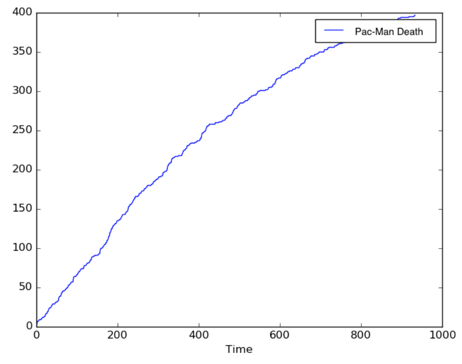


Fig. 8. No learning for Monsters

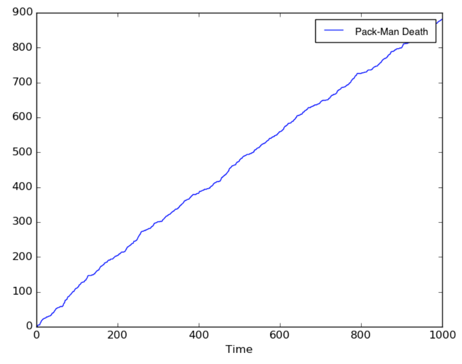


Fig. 9. Enable Q-Learning for Monsters

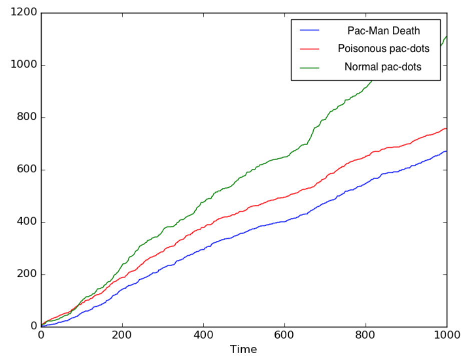


Fig. 10. Q-Learning

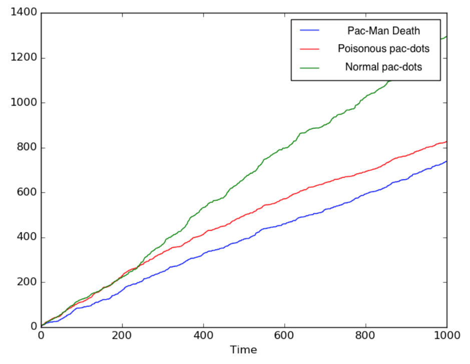


Fig. 11. SARSA

Q-learning) for figure 10 while we set PM(20, SARSA) and MS(20, SARSA) for figure 11.

As figure 10 and figure 11 display Q-learning and SARSA for the same environment setting respectively, we do not observe big difference for the number of poisonous pac-dots eaten by the Pac-Man and Pac-Man's dead amount. However, by using SARSA, it gives a higher number of poisonous pac-dots eaten by Pac-Man. Thus, we make an assumption that SARSA may have slightly better performance than Q-learning.

V. CONCLUSIONS

To conclude, in this report, we present an advanced Pac-Man simulation that models the real-time generation evolution of certain spices. Though the simulation is conducted in a virtual environment, from the evaluation section, it is clear that both agents (Pac-Man and monster) exhibit learning ability if the Q-learning or SARSA is enabled. We control the variables in terms of learning ability and objects amount to observe the different behavior for both Pac-Man and monsters. Fortunately, they do react to the change and make adapation accordingly. Another advantage of our simulation is that it allows user input, where user can specify the the number of Pac-Man and monsters in the environment, also for their speed to see the behavior difference due to the colony size. Last but not least, we compared the performance between two different learning algorithms and discussed the resultant plot. Based on the result, we conclude that SARSA has slightly better performance. However, it still remains a mystery on why SARSA performs better. It can result from different configurations in the initialization stage, different learning rate affected by random walk position of both species and etc. , so we will leave it as future work.

References

[1] Wawrzyski, P. (2009). Real-time reinforcement learning by sequential ActorCritics and experience replay. Neural Networks, 22(10), pp.1484-1497.

[2] Terzopoulos, D., Tu, X. and Grzeszczuk, R. (1994). Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World. Artificial Life, 1(4), pp.327-351.

[3] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. Torr. Playing doom with slamaugmented deep reinforcement learning. arXiv preprint arXiv:1612.00380, 2016.

[4] Tziortziotis N., Tziortziotis K., Blekas K. (2014) Play Ms. Pac-Man Using an Advanced Reinforcement Learning Agent. In: Likas A., Blekas K., Kalles D. (eds) Artificial Intelligence: Methods and Applications. SETN 2014. Lecture Notes in Computer Science, vol 8445. Springer, Cham

[5] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.

[6] Watkins, Christopher John Cornish Hellaby. Learning from delayed rewards. Diss. University of Cambridge, 1989.

[7] Rummery, Gavin A., and Mahesan Niranjan. On-line Q-learning using connectionist systems. University of Cambridge, Department of Engineering, 1994.