# Q learning with function approximation

Cheng-Yuan Yu

April 1, 2018

## 1 Introduction

I am going to use reinforcement learning to deal with pacman through using temporal difference Q-learning and approximate Q function with three main features. After training, my model will help my pacman choose how to move.

## 2 Description

I use only three features to approximate the Q function, instead of using whether there are food or ghosts in each position of the map.

The benefit of that is my method can work with not only a small map, but also bigger maps because too many combination should be learned if not using function approximation. Another benefit is that my model can be learned by small number of trainings because it only has three parameters corresponding to three features which I use.

In this session, I will introduce several things, as following:

1. What kind of features and function I use to approximate Q function
2. How to design of my reinforcement learning
3. How to help my pacman to move after training

### 2.1 Feature extraction

Let us assume the three main features $\boldsymbol{x(s,a)} = [x_0(s,a),\ x_1(s,a),\ x_2(s,a)]^T$.

For the first feature $x_0(s,a)$, bias, it always keeps 1, like an intercept in a common regression model.

Before mentioning the rest of two features, there is an important thing I need to say. The distances of the pacman from food and ghosts are the shortest distances which are calculated by path-searching algorithm, called A* algorithm.

The second feature $x_1(s,\ a)$ means the smallest value of the number of walls surrounding food multiplied by its disance from the pacman if the pacman make the action $a$ at the position (coordinate) $s$.

This feature not only consider the distance between the pacman and food, but also the ways for the pacman to escape from ghosts, which can help the pacman not to eat the food which is closer to it but in a no-escape corner and there is a ghost behind it. If it decides to eat this food, it might be eaten because there is no way for the pacman to escape, shown in Figure 1.

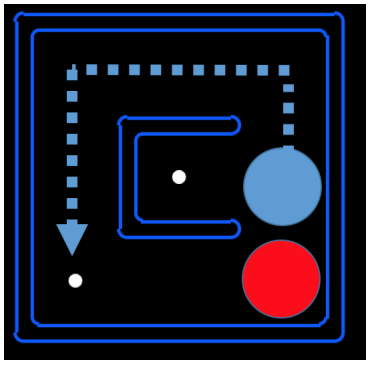

Figure 1: Choose the food far from the pacman

The third feature $x_2(s,\ a)$ roughly means the shortest distance between the pacman and ghost after making the action $a$ at the position (coordinate) $s$, assuming ghosts don't change their positions even if pacman moves. It can be used to describe whether it becomes closer to ghosts if the pacman makes the action $a$ at the position $s$.

However, I do a tricky thing to modify the third feature. I take the minimum of this value and 4, and then minus 3.5. This is the finial value of this feature $x_2(s,\ a)$ because I treat the influences of ghosts who are far from the pacman more than 4 steps are same, and the reason for subtracting it from 3.5 is to make the different sign from the weight corresponding to the second feature, which is easier to train.

## 2.2 Learning process

Before making the whole process, I talk about the definition of reward function as well as approximation function to q function, exploration rate, discount factor, and how to adjust the learning rate for different weights. Let the weight $\boldsymbol{w} = [w_0,\ w_1,\ w_2]^T$ which I am going to train and $a\,(s)$ is a list of legal actions in a position $s$.

### 2.2.1 Definition of reward function

The reward is the function of the state (pacman's position) and action:

$$R(s,\ a) = -1 \times x_1(s,\ a) \times w_1 + 5 \times x_2(s,\ a) \times w_2$$

The reason why I gave more rewards to the third feature is that I want the pacman to give priority to survive rather than get food.

### 2.2.2 Definition of approximation function to Q function

The Q function is approximated by regression model with three features:

$$Q(s,\ a) = \boldsymbol{w}^T \boldsymbol{x}(s,\ a) = w_0 + w_1 \times x_1(s,\ a) + w_2 \times x_2(s,\ a)$$

### 2.2.3 Exploration rate and discount factor

For the exploration rate $\epsilon$, I fix the rate during the training, $\epsilon = 0.1$.

The discount factor $\gamma$ is fixed as well. I will discuss using different $\gamma$ to train my pacman later.

### 2.2.4 Learning rate

There is a point about how to adjust the learning rate $\alpha$ for different weights.

Firstly, I use the original learning rate $\alpha\,(t) = \frac{1}{100+t}$, where $t$ is the t times of updating. Then,

I introduce the adjusted learning rate, called $\alpha^*(x_2)$.

In each iteration, if the third feature $x_2$ is not equal to 0.5 which means the shortest distance between the pacman and ghosts is less than 4, the adjusted learning rate $\alpha^*(x_2)$ will be set as 10. Otherwise, it is set as 1.

This is because the pacman has less chance to be closer to ghosts than food, which causes the pacman not to have the same opportunity of learning how to avoid ghosts than eat food. This problem appears in a bigger map, although it is not noticeable in a small map. If we ignore this problem, the pacman will go straightforward to eat food, even if there is a ghost in front of it.

Hence, when pacman is closer to be ghost, I would like to update the third feature more. The learning rate for the third feature $x_2$ will be the original learning rate $\alpha(t)$ multiplied by the adjusted learning rate $\alpha^*(x_2)$. For $x_0$ and $x_1$, they only use $\alpha(t)$ as their learning rate.

### 2.2.5 Updating

I am going to demonstrate how to update the weights if having the features of the last step, $\boldsymbol{x}(s^{last}, a^{last})$, and the position of the current step, $s^{now}$.

Firstly, I need to find the maximum Q value for the current step $s^{now}$ by selecting the legal action in the current position $s^{now}$,

$$\max_{a \in a(s^{now})} Q(s^{now},\ a) = \max_{a \in a(s^{now})} \boldsymbol{w}^T \boldsymbol{x}(s^{now},\ a)$$

Secondly, I calculate the Q value and reward for the last step, $Q(s^{last},\ a^{last})$ and $R(s^{last}, a^{last})$,

$$\begin{aligned} Q(s^{last}, a^{last}) &= \boldsymbol{w}^T \boldsymbol{x}(s^{last},\ a^{last}) \\ &= w_0 + w_1 \times x_1(s^{last},\ a^{last}) + w_2 \times x_2(s^{last},\ a^{last}) \end{aligned}$$

$$R(s^{last}, a^{last}) = -1 \times x_1(s^{last},\ a^{last}) \times w_1 + 3 \times x_2(s^{last},\ a^{last}) \times w_2$$

Thirdly, I update three weights at the $t^{\text{th}}$ time,

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha(t) \times \left( R(s^{last}, a^{last}) + \gamma \times \max_{a \in a(s^{now})} Q(s^{now},\ a) - Q(s^{last}, a^{last}) \right) \\ w_1 &\leftarrow w_1 + \alpha(t) \times \left( R(s^{last}, a^{last}) + \gamma \times \max_{a \in a(s^{now})} Q(s^{now},\ a) - Q(s^{last}, a^{last}) \right) \times x_1^{last} \\ w_2 &\leftarrow w_2 + \alpha(t) \times \alpha^*\left(x_2^{last}\right) \left( R(s^{last}, a^{last}) + \gamma \times \max_{a \in a(s^{now})} Q(s^{now},\ a) - Q(s^{last}, a^{last}) \right) \times x_2^{last} \end{aligned}$$

where

$$\begin{aligned} x_i^{last} &= x_i\left(s^{last},\ a^{last}\right) \text{ i} = 1,\ 2 \\ &\text{and} \\ \alpha^*\left(x_2^{last}\right) &= 10 \quad if\ x_2^{last} \neq 0.5 \\ &\quad\ 1 \quad if\ x_2^{last} = 0.5 \end{aligned}$$

## 2.3 Making decision

After updating the weight by using the features of the last step, $\boldsymbol{x}(s^{last}, a^{last})$, I need to decide what action the pacman is going to take in the current position $s^{now}$. In this part, we need to consider there is a chance for it to explore according to the exploration rate $\epsilon$.

Firstly, I generate a random variable $u$ from the uniform distribution, $U\,(0,\ 1)$.

Secondly, according to this random variable $u$, the pacman can decide to make the action $a^{now}$ according to the Q function or random choice.

$$\begin{aligned} a^{now} &= \arg\max_{a \in a(s^{now})} Q(s^{now},\ a) \text{ if u } \geq \epsilon \\ &= \text{ randomly select one action from a } (s^{now}) \text{ if u } < \epsilon \end{aligned}$$

For the next iteration, I replace $s^{last}$ and $a^{last}$ with $s^{now}$ and $a^{now}$,

$$\begin{aligned} s^{last} &\leftarrow s^{now} \\ a^{last} &\leftarrow a^{last} \end{aligned}$$

### 2.4 Whole process of Reinforcement learning

In this part, I just demonstrate the process of training, as following:

- Step 1 Initialize the weights, $w_0 = 0,\ w_1 = 0,\ w_2 = 0$
- Step 2 Get the current position $s^{now}$ and randomly select one action $a^{now}$ from the list of legal actions, $a\,(s^{now})$
- Step 3 Save the current position and action to $s^{last}$ and $a^{last}$, and then, make the action $a^{now}$
- Step 4 Get the current position $s^{now}$
- Step 5 Calculate $R\left(s^{last}, a^{last}\right)$, $Q\left(s^{last}, a^{last}\right)$ and $\max_{a \in a(s^{now})} Q\left(s^{now}, a\right)$
- Step 6 Update the weight $w_0,\ w_1,\ w_2$
- Step 7 Using the updated weight to get the current action $a^{now}$ with consideration of exploration
- Step 8 Save the current position and action to $s^{last}$ and $a^{last}$, and then, make the action $a^{now}$

Step 1 to Step 3 are only for the first step, and then, repeat Step 4 to Step 8.

## 3 Result

Firstly, I use 9 different discount factors from 0.1 to 0.9 to train my pacman in a small map for observing how weights change over iterations, illustrated in Figure 2. For these three weights, all of them can converge more quickly with a low discount factor than with a high discount factor. For instance, in the case $\gamma$ less than 0.5, the three weights become stable after 7000 iterations, whereas all of the weights need at least 25000 iterations with $\gamma$ more than 0.7. Especially, $w_0$ with $\gamma = 0.9$ need more than 35000 to converge.

This is because when the pacman with a high discount factor, it tends to consider the future steps more, which causes more historical values go into the current training process and makes weights change more easily. Hence, they need more iterations to become stable. However, no matter which discount factor $\gamma$ I use, the pacman have the 100% winning rate after trained 2000 games and playing 50 games in a small map because the environment of this map is simple.

Next, I am going to discuss the winning rate in the different maps and changing of parameters during training by using the discount factor $\gamma = 0.8$. Also, it will be demonstrated that the

result of pacman who has been trained by a small map or by a medium map plays in a small, medium, and large map.

I show that my weights which are trained in a small map (Figure 5) and a medium map (Figure 6) respectively. The weight has been updated 37,615 times in a small map and been updated 192,708 times in a medium map, and then, I use these weights to play the pacman 100 games in each map. I get 100% winning rate in a small map and 91% winning rate in a medium map, shown in Table 1.

Because I use stochastic gradient descent to train my pacman, the weights are not stable at the beginning. They become stable after 25,000 times of updating in the small map and after 125,000 times of training in the medium map, shown in Figure 3 and Figure 4. In fact, I only have 3 weights. Hence, they don't really need a lot of training.

Moreover, because I use function approximation to describe my Q function, I can use the weights to play in the different map. Here, I use the weights trained in the small map to play 30 times in the medium map and large map (Figure 7), and use the weights trained in the medium map to play 30 times in the small and large map, shown in Table 2. Both of two set of weights get 100% winning rates in the small map, round 90% winning rates in the medium map, and about 70% winning rates in the large map. Though it doesn't perform in the large as well as in the small and medium, it is still great.

As the result shows, it is able to work well in the different map although we get different weights by using different maps. Both of two maps train the pacman to know the general strategy according to the three features I used.

Table 1: Training result

| Map | Times | $w_0$ | $w_1$ | $w_2$ | winning rate |
|---|---|---|---|---|---|
| small map | 37,615 | -11.9102414288 | -3.16231479177 | 20.7739307569 | 100% |
| medium map | 192,708 | -5.70973904088 | -3.56826244675 | 18.4421243465 | 91% |

Table 2: Training result

| Which map used to train | Small map | Medium map | Large map |
|---|---|---|---|
| small map | 100% | 90% | 73% |
| medium map | 100% | 91% | 67% |

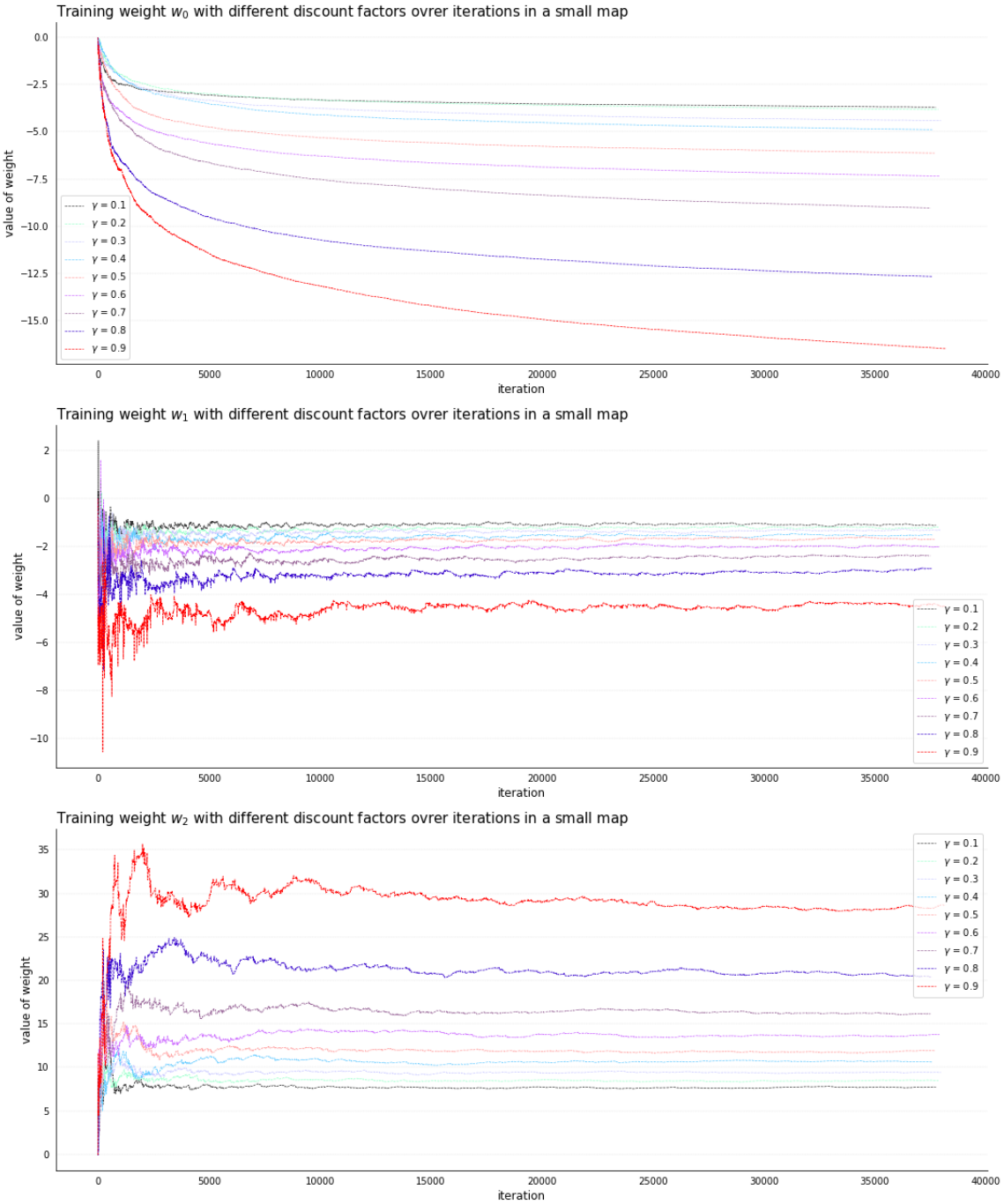


Figure 2: Training result in a small map by different discount factors

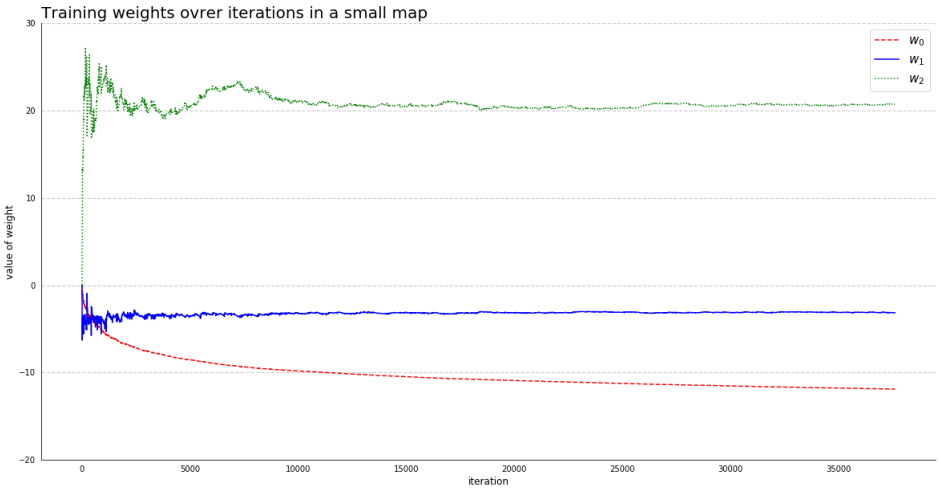


Figure 3: Training result in a small map

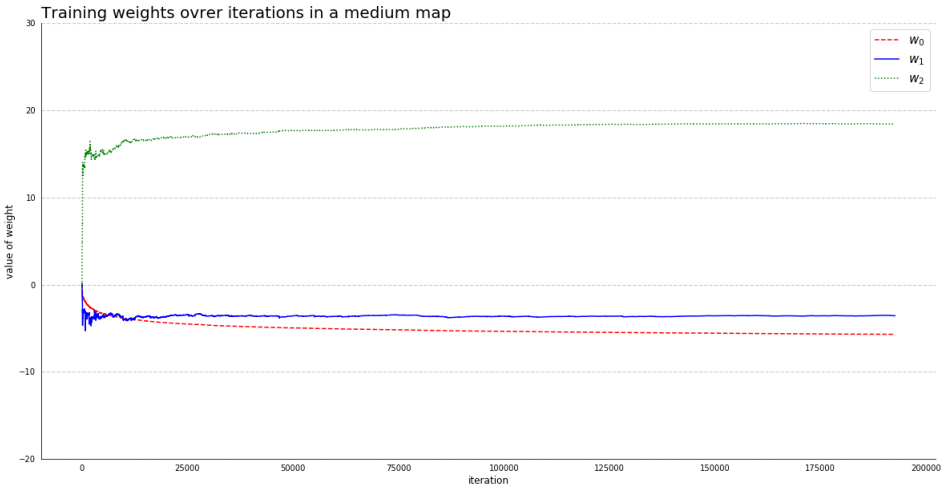


Figure 4: Training result in a medium map

Figure 5: Small Map

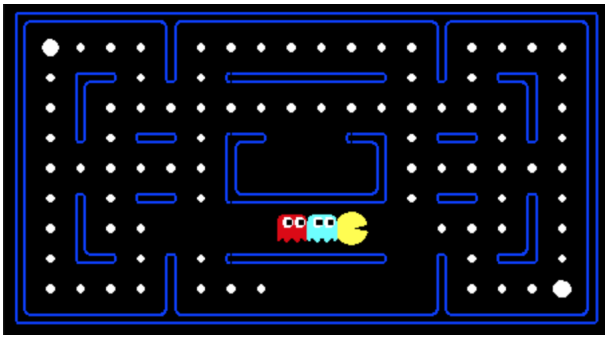

Figure 6: Medium Map

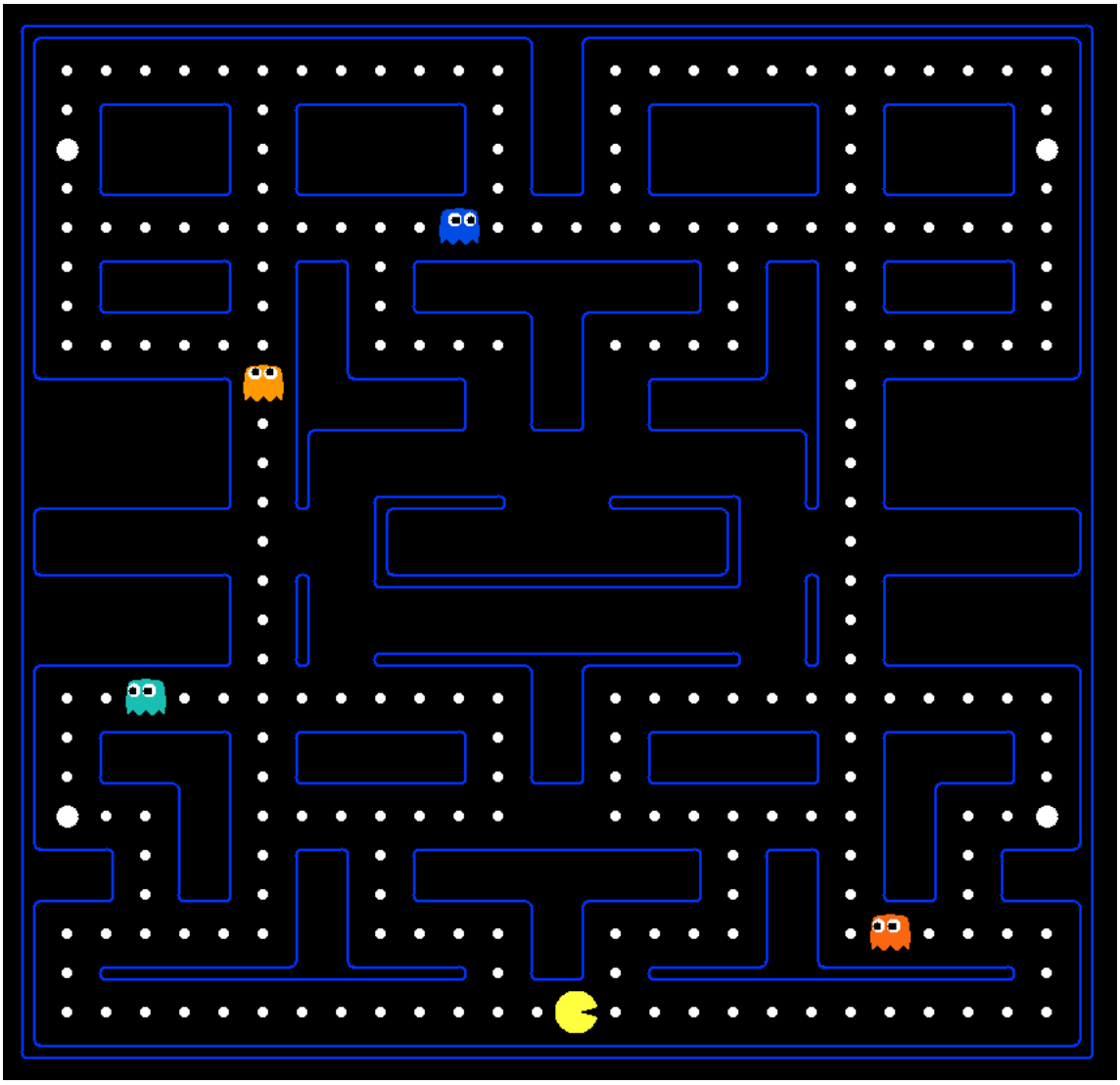

Figure 7: Large Map