Project 3

Mouni Thangudu

November 16, 2018

1 Loading

In the folder, you will find 3 files. `N_a.json`, `Q.json`, `Qagent.py`. Here, the first two files are data files for the trained agent which store the python dictionaries in `json` format which I can load using `json` library. I have two dictionary objects in my program, one for storing Q - values for each state-action pair and another to store N(s,a) that keeps track of how often we've been in state s, performing action a.

For the final learned model, I only need the Q - values to decide the action. I included N(s,a) nonetheless since it assisted me during the training.

To run the file, you need to make sure all the three files are in the same folder, then you need to execute the following command. Loading the model is taken care internally.

```
$ python Qagent.py
```

2 Performance

On an average, 3 frogs cross reach the home. Sometimes, I observed even 4 frogs make it. Sometimes it will be less. Accounting for the negative reward, the average reward per game is around 1.4. My agent can avoid turtles before they sink by coming back to the midway and taking an alternate route(more on it in the next section), my agent will also avoid crocodiles, moving cars, and steps only on moving logs and turtles and homes. I have included a screen shot of an arbitrary instance during the training where the frog is trying to reach to the leftmost home. The agent will have the complete picture of the entire game at every point of time and will make the appropriate choice. For now, the agent cannot differentiate between an empty home and a home with a fly, this is a choice I had to make to reduce the state space. A general instance of the game played by the agent is shown in Figure 1

3 State design

My state space includes several properties about the environment which is extracted from `obs`. Now I describe different parts of the state.

3.1 Frog peripherals

The agent is always aware of the objects in all 8 directions using 8 Boolean variables. The presence of an car/block makes this variable true. Absence of object makes it false. In this way, we will have 2^8 possible states for the frog peripherals. A general transformation of any arbitrary instance is shown in Figure 2

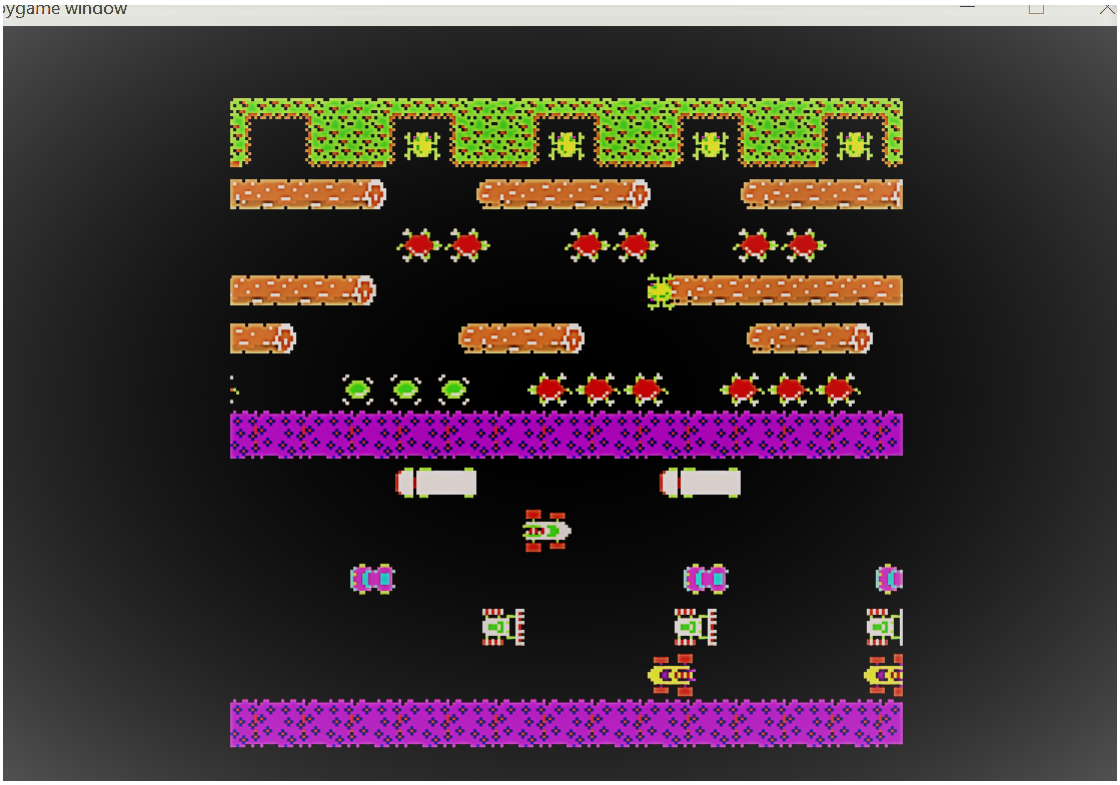

Figure 1: An instance of the game

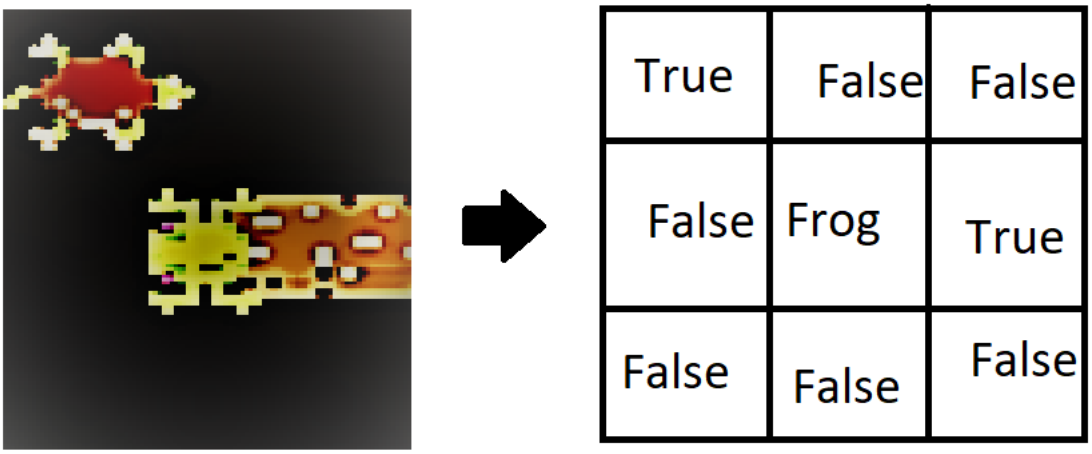

Figure 2: Transformation of frog peripherals into a state

3.2 Frog position

This is very simple, which is basically the x and y position of the frog. Having no restrictions on the values taken by them makes the state space unbounded. So, I made the entire environment into a grid,

where each cell is the same size as the frog. So, depending on where the frog is in the grid, we get a different x and a y for the frog. I have a grid of size 13×13 over the entire game play area. So there are 169 possibilities for the frog position.

3.3 State of the home

The frog needs to be able to see the occupied homes from the beginning so that it can make informed decisions from the the ground level itself. So, for that I have 5 Boolean variables for the 5 homes. They will be true if it is occupied by the crocodile or the frog. They will be false if they are free, or there is a fly. So, with this, the frog knows about the empty homes even when it is in the land or anywhere in the play area. There are $2^5 = 3232$ such possibilities. An arbitrary instance of homes and the conversion is shown in Figure 3

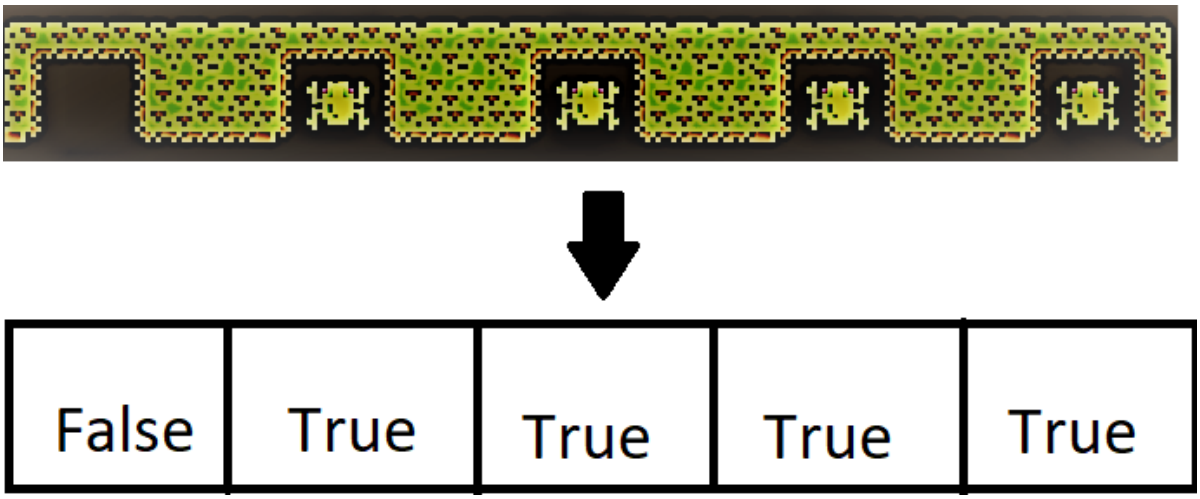

Figure 3: Transformation of homes into state

3.4 Sinking turtles

Since the turtles that sink is always the same turtle group(Knowledge gained through observation of the game arena), I choose to include this turtle group's position in the state representation. So, the x coordinate of this turtle group will be part of the state. I constrain this value in the same way as I constrained x of the frog. I will have 13 different positions for this turtle group based on the where this group is in the grid as mentioned in the previous section. There are two such sinking groups. recording the state for one group is enough because both of them travel with the same speed. This state was not promising and also increased my state space so much. I included it anyways since it is working. In hindsight, I shouldn't have included it.

3.5 Frog action

Since the Q value is calculated for the state - action pair. I have also included this action taken to be the part of the state. So there are 5 possibilities for this action. Which are up, down, left, right, no action.

3.6 Representation of the entire state

So, having mentioned all the different properties of the state, I will have 89,989,120 unique states. Since, all the states need not be explored, the training is not as time-consuming as it might appear looking at the number. The better states will get reinforced and the other states will get cut-down. This is why I was able to achieve the trained model in 30 hrs and I believe it can be improved even further by training it more. I have a string representation which represents each unique state. I used this object so that the look up and hashing will be faster. An arbitrary instance of my state is "1000000008389000003None". The None at the suffix represents the action None.

4 Math

There are several design choices I need to make in choosing the appropriate values for learning rate α and discount factor γ. I have chosen α to be $\frac{1}{\sqrt{N(s,a)}}$, where $N(s,a)$ is the count of the times the frog visited this state s and choose action a and γ to be 0.9. The reason for choosing α because I don't want the reward to have less effect because the frog dies several times even before it reaches its first home. So, by having this α, we ensure the reward when obtained will have significant effect when compared with an α value of $\frac{1}{N(s,a)}$ This choice is obtained by trial and error. The value of γ is my first choice and it worked fine. I didn't have to fine tune it.

5 Exploration vs Exploitation

The exploration vs exploitation I used is 90% exploitation and 10% exploration from the beginning and it remains that way. I have tried with multiple variations of this strategy by increasing the exploration in the initial stahes and also stopping exploration whenever the frog explored a particular state 500 times already. Although the later two didn't have much effect on the overall learning, a constant percentage of exploration always seems to be good. I have also tried different ratios of these two but 10% seems to work better than the rest.

6 Observations

Initially, keeping in mind the fact that this kind of table based approach will blowup my state space and it will take really long to learn. So, I initially started with linear regression approach, but I have learned that, for a game like frogger with multiple different objects with different properties and which act in close vicinity, the fine tune required for the features in linear regression is insane and need multiple iterations to figure out the best features. And still we might not be sure if this model will work for a new environment with a simple change of the environment. So, I switched to state based approach. It took around 48 hours to train the frog. I have tried with multiple different versions of the states, for example by removing the homes, by including the fly and crocodile, but I have not pursued them further because of the increased state space. If I had more time, I would figure out how to tune the frog more efficiently so that it won't die ever. The frog is exactly behaving like how a human would play. This is because I have incorporated all the elements of the game which the human thinks before he makes a move. A more shallow view with less number of states might be implemented which might work and train faster too, but it won't appear human like when the frog plays.