Q-Learning in the snake game

Adam Zaiter

***Abstract*—**

***Index Terms*—**

I. Introduction

This paper will attempt to implement the Q-learning algorithm for a basic game of snake. A hypothesis about the expected learning outcome will be established and reviewed after implementing the Q-Learing algorithm. The implementation will be done in Python and a custom snake game environment will be created. This environment will allow for training of the agent with different hyperparameters as well as visualizing a game played by the trained agent.

Basic knowledge of reinforcement learning is assumed, as this paper is implementing concepts from the Szepesvari paper [1].

II. Snake game

The snake game is programmed in python. It supports square grids of any size that is $>= 3$. The snake starts in the middle of the grid with a head of 1 square in size and attached body with 2 squares in size. There also is food spawned in a random square that is not occupied by the snake. The objective for the snake is to eat as much food as possible. If the food is eaten, the snake gains + 10 score and grows an additional body part of 1 square in size. If the snake collides with the borders of the grid or the head collides with it's own body, the game ends.

III. Episode

An episode contains all of the actions taken in one game of the snake. The episode ends when a terminal state is reached (game ends), described in section II. In the case of the snake game that is either the head of the snake colliding with a border or the head colliding with it's own body.

IV. Step

A step is the performance of one action from the action space in a given state.

V. Q-Learning

Q-Learning finds the optimal policy by learning the best Q values for each state-action pair.

The Q function accepts the current state and an action. It then returns the expected reward for taking said action for the given state.

The agent then plays episodes. For each step in the episode the Q values in the Q table are updated using the Bellman equation. This process goes on until the q function converges to the optimal function q*.

VI. Hypothesis

The goal is to train an agent that will play the snake game with any chosen grid size as best as possible using just Q-Learning. Given the attributes of Q-Learning, it should be possible to train an agent, that plays the game of snake at a very good level. It is not expected to be perfect, but it should play as close to it as possible.

VII. Environment

The environment mimics functionality provided by environments in OpenAI gyms. This made the process of applying the Q-Learning algorithm quite simple, as the OpenAI environments are designed with reinforcement learning in mind.

The environment provides three actions for the agent to take:

1) Turn the snake head left.
2) Turn the snake head right.
3) Continue heading straight.

These values are encoded as integers 0, 1, 2 and aliased as constants LEFT, RIGHT and STRAIGHT in the code.

The environment does not provide the full state of the board, as there are simply too many states to represent for a grid of variable size. Thus, the environment is restricted to this basic information:

- Information about danger. It indicates if after taking an action from the action space in the current state the game will be over. This can be expanded for a lookahead of one or more turns in the future.
- Information about food position. The values are left, right, up and down.
- Information about the snake's current direction. There are four values: left, right, up and down.

After gathering this information, it is encoded in a 2D array. For example, the array for food position is an array of four values. Each position of the array corresponds to a value (left, right, up, down). Thus if the food is left, the array on position left will have the value of 1 (true). Otherwise, the value will be 0 (false). This applies for all positions. In the end, three individual arrays for danger, food and direction are created and are returned concatenated as the current state.

VIII. Training

IX. Observations

The snake seems to learn better in smaller environments.

Even with limited information the snake learns the strategy of creating as much space as possible for it's own body. This results in the snake hugging the borders as much as possible, thus it takes longer paths to get to the reward (food) which might seem strange early in the game. However, when the

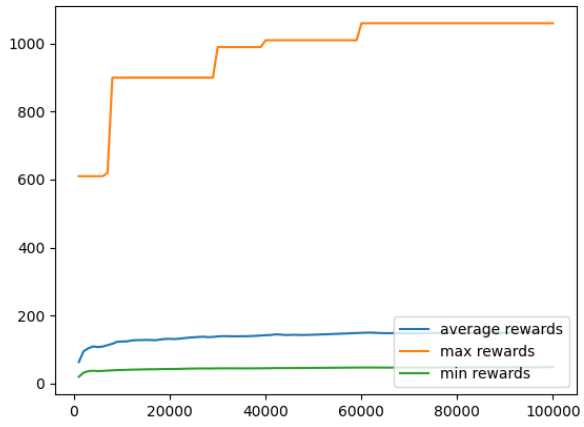

Fig. 1. 3x3 environment 20-size state space

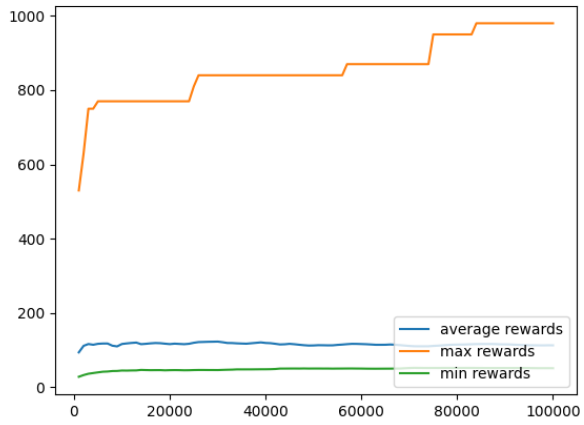

Fig. 2. 3x3 environment 11-size state space

snake's body is longer after collecting a few rewards, the benefits of this strategy are fully visible.

Since the environment is custom, some thought had to be given to what kind of rewards the agent should get as well as their quantity.

X. Extending the state space

XI. Plots

XII. State space

For this project, the computing resources are somewhat limited and as such, these limited resources have to be taken into consideration when designing the state and action spaces. The state space for a snake game with a variable size would be too big, if it were taken as is. That is, because the snake and it's body adds many additional states, as the snake can grow it's body to be the same size as all of the squares in the grid. Also, the head is distinct from the other body parts and would have to be accounted for in all of the states.

The state space is represented as a python dictionary, where the keys are the individual state arrays and keys are their indices. This indices are later used for indexing the Q table. The indices correspond to each state, and hold an array of Q values for each action.

XIII. Training

Best results were seen when training on a 3x3 grid. The training did not take as much time, as fewer moves were made on average.

References

[1] Csaba Szepesvari - Algorithms for Reinforcement Learning [online] Available at: https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf [Accessed 15 January 2022].