Reinforcement Learning in Pygame

Gavin Thomas
Denison University
thomas_g8@denison.edu

I. Introduction

Reinforcement Learning has been used to optimize ad placement, train robots to do new tasks, and even to beat grand masters in chess. Many of the bots that play video games can be trained using Reinforcement Learning! Reinforcement Learning is a machine learning technique distinct from supervised and unsupervised learning. It is a general framework for learning how to decisions on sequential tasks, where an agent learns in an interactive environment. To solve these problems, we apply a mathematical framework to the problems and calculate the optimal policy.

Reinforcement learning is useful for problems where there is an agent that is able to get feedback from the environment, in which there are delayed rewards as well as possible immediate rewards. In addition, the problem should be able to be modeled as a Markov Decision Process, a formalization of sequential decision making.

As a student, I found that although reinforcement learning methods are widely theorized about, there are few resources that provide a simple, visual example with human-intuitive games. For this reason, I created my own. In this paper, I will briefly explain the basics of reinforcement learning. Then, I will describe a homemade, turn-based game to apply the theory and analyze the results to determine the success of reinforcement learning.

A. A Brief Overview of Reinforcement Learning

Reinforcement learning is a sub-field of machine learning that learns from an agent interacting with an environment, attaining rewards for each action, and maximizing this numerical reward. These actions can affect not only the next subsequent reward, but all subsequent rewards. Reinforcement learning is different than supervised learning, as it does rely on a training set, with labeled data. It is also different than unsupervised learning, as we are modeling for the expected reward.

The agent and the environment are the 2 main elements in a reinforcement learning system. The agent must be able to sense the state of the environment and take actions that affect that state. In addition, it must have goals relating to that state. The environment is the space in which the agent inhabits.

There are an additional 3, with an optional 4th, subelements in a reinforcement learning system. These are: policy, reward, value function, and model (optional). The policy is what defines the agent's behavior at a given state. The policy can be a mapping, from state to action, or a more complicated function. A numerical reward is received by the agent at each step. The reward is to be considered in the short run, as it determines what can be considered a good or bad event. Following, the value function determines what is good in the long run, as it is the expected reward the agent can accumulate in the future, starting from that state. Finally, the optional sub-element, the model, mimics the behavior of the environment to allow the agent to make inferences about how the environment will behave.

But how do we solve for the optimal policy π^* which determines the proper action at a given state? There are two methods to determine the optimal policy that I will define. These are value iteration and policy iteration.

Value iteration recursively calculates the value function until it converges to the optimal value function. From there, it is easy to derive the optimal policy. Policy iteration, on the other hand, iterates over different policies, starting from a random policy. At each step, you improve and then evaluate the policy until the evaluation of the policy changes by less than a pre-determined value. Each policy evaluation is started with the value function for the previous policy to increase the speed of convergence. For my analysis, I will use policy iteration, because Sutton and Barto suggested that it often converges faster than value iteration (Sutton & Barto, 2018).

II. The Alien-Meteorite Game

To apply this theory, I created a turn-based game in python using pygame. This game features an alien agent with the simple goal of staying alive as meteorites fall from space. The environment is fully-observable with dimensions $nxnxb$.

The game is initialized with the alien agent randomly placed into a position in the bottom row. The alien agent may not leave the bottom row, but may move to adjacent columns. Simultaneously, b meteorites are randomly generated to be in the top row, in which meteorites can not be in the same position. From this initial state, and all subsequent states, the alien agent has three possible

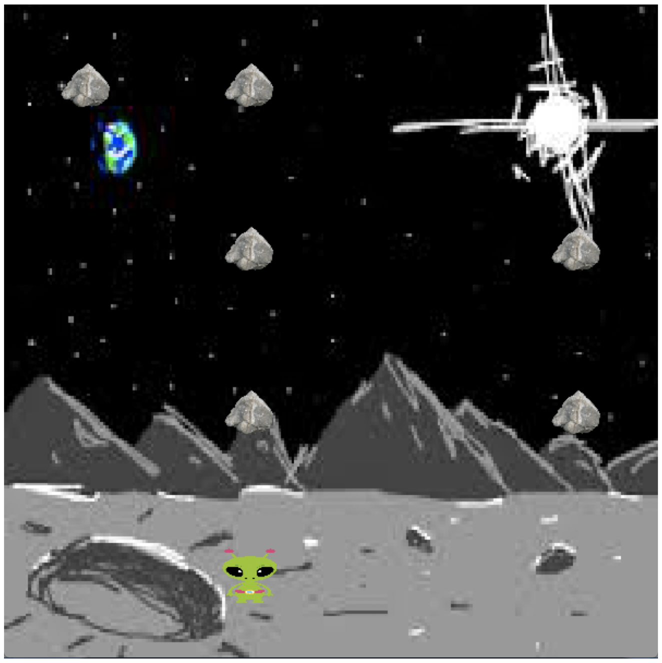

FIG. 1. Example State

actions of set {Right, Left, None[1]}. If an agent is in the leftmost position, and moves left, the alien agent bounces off the wall and remains in the same position, making up and left equivalent moves in the leftmost column. After an action is selected via a movement policy, the meteorite move down 1 row (retain column position), and a new top row of b meteorites is generated. Then, a corresponding reward is given. If the alien did not collide with a meteorite, it receives a reward of 1, while a collision yields a reward of -1000. Following, a turn is comprised of an action and a corresponding reward. This combination of meteorites and alien agent is defined to be a single state with dimensions in a game with dimensions $nxnxb$. An example state is shown in Figure 1.

Score is defined to be the number of actions taken before a collision, a state in which the alien agent and a meteorite occupy the same position in the environment. The game is said to be over when after a collision occurs, at which the game returns the score.

In a given game, there are $\binom{n}{1}$ or n alien states and $\sum_{i=0}^{n} \binom{n}{b}^i$ meteorite states, giving,

$$TransientStates = n * \sum_{i=0}^{n} \binom{n}{b}^i$$

And,

$$RecurrentStates = n * \binom{n}{b}^n$$

[1] Also defined as Up.

I have written 3 policies to play this game:

- **Random:** Randomly selects action from the action set.
- **Greedy:** The agent can 'see' one row ahead. Takes action to maximize the next state reward. Ties are broken randomly.
- **RL:** The agent can 'see' the entire environment. Takes action determined by policy iteration to maximize future reward.

For each, the policy chooses an action based on a given state until a collision occurs. This game is not infinite because, with $b > 1$, there exists states in which the alien agent must die in t turns.

A. Expected Score

The expected score of the random policy can be calculated to assert that the system is working properly. Under a random policy, the alien's transition matrix is as follows:

$$P = \begin{bmatrix} 2/3 & 1/3 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 2/3 \end{bmatrix}$$

If we let π be the steady state distribution, by definition, $\pi = \pi P$. Where, $\pi = [.25, .25, .25, .25]$ for a uniform distribution. Upon calculation, the $\pi = \pi P$ assertion holds true using the transition matrix P and π defined. Therefore, we can assert that the alien states follows a uniform distribution and that the alien is equally likely to be in every state.

In a 4x4x2 environment, there is a .5 chance of collision every turn, following a geometric distribution. The meteorite takes 3 turns to reach the bottom row, which means the 4th turn is the first turn in which the alien can collide.[2]

If X is the score after the third turn and k is the probability of a collision after the second turn for k=0,1,2,3,..., then,

$$P(X = k) = 0.5^k * 0.5$$

$$E(X) = \sum_{i=0}^{\infty} k * P(X = k)$$

[2] If the alien collides on the 4th turn, it will receive a score of 3.

$$E(X) = 0.5 \sum_{i=0}^{\infty} k * 0.5^k = 1$$

$$E(Score) = 3 + E(X) = 3 + 1 = 4$$

Therefore, if the random policy yields a result of 4, I can assert that the reinforcement learning system is functioning properly.

B. Results

To test my policies, I ran 10,000 simulations in a 4x4x2 environment for each of the three policies I have defined. The summary statistics of the scores can be found in Table I.

Category	**Mean**	**Median**	**StDev**	**Min**	**Max**	**n**
Random	3.98	4.0	1.39	3	17	10,000
Greedy	16.55	12.0	14.18	3	152	10,000
RL	45.86	33.0	40.93	5	382	10,000
RL - outliers	39	32.0	29.39	5	131	9,549

TABLE I. Scores by Policy

From these results, I can assert that my score for the Random Policy aligns with the expected results. Therefore, the system is working properly. Also, under the random policy, the alien did spend equal time in each state, which validates my assertion.

As expected, the RL policy significantly outperformed the Greedy policy, which in turn significantly outperformed the Random policy. Each of the score distributions are skewed right, which indicates that the maximum values are potential outliers.

III. Key

- n: number of rows/columns
- b: number of meteorites per row
- TS: total states