Artificial Intelligence to play Mario

Harshvardhan Kalra | Praveen Kumar Jhanwar

Indraprastha Institute of Information Technology, Delhi

Progress so far

We have tried two different methods to make our agent learn to play in Mario:

1. A basic approach:
   We keep on performing random actions until game terminates and store some number of most recently performed actions, say X. Then on the next trial run, we perform X actions and then again try random actions and repeat the process again and again.
2. Basic Q-learning approach:
   We applied a standard Q-learning algorithm with experience replay for preliminary results. Since Mario can have only 4*2=8 possible moves at any point(four from direction and 2 from A/B), we train a small neural net with mse error to train the agent. Training was done for 2000 episodes, with older episodes being replayed every time Mario dies, with a batch size of 16. To prevent stagnation, random steps are taken in between.

Further Work

- We will improve upon the basic Q-learning approach by defining a proper state description to include:
  1. Mario mode (small, big, fire)
  2. Direction of velocity of Mario
  3. Direction of location of enemy in nearby regions
  4. Whether it killed an enemy and how (stomp/fire)
  5. Whether it collided with other creatures.

  And also, to try different parameters (alpha, gamma etc), vary the reward function to obtain the best possible performance.
- For the current implementation, we have considered a simple reward function of the distance covered horizontally.

Advanced RL Techniques

If time permits,

- Since Q-learning may limit in this case as it observes only the current frame and thus may not be able to estimate the delayed rewards. To address this, we will modify and implement Deepmind's work on Atari. In this, we will use Deep Q network, which is a multi-layered convolutional neural network that outputs a vector of action values given state s and network parameter.
  This basic approach may introduce some bias of exploring only states that result from recent actions that were expected to maximize the reward. So, we can use double DQN to try to eliminate this bias, which basically uses two Q functions that are independently learned, one to determine reward-maximizing action and other to estimate its values. We can also use experience replay to use and benefit from the <state, action,...> sequences we explored earlier.
  This approach is computationally very expensive.
- We will implement SARSA(State Action Reward State Action), which is just an online version of Q-learning and compare it with our Q-learning approach.

Handling Adversaries

Since we are using Reinforcement Learning techniques, we don't need to program anything in the agent to handle adversaries separately, The agent will itself try out different actions and explore the state space, while the reward function will penalize the agent on colliding will any adversary, and will reward the agent for destroying/avoiding the adversary. As for the current implementation, it has not yet encountered adversaries which need a specific way of destruction, like only stomping or only firing bullets. We intend to handle these accordingly.