CS229 Final Report
Reinforcement Learning to Play Mario

Yizheng Liao
Department of Electrical Engineering
Stanford University
Email: yzliao@stanford.edu

Kun Yi
Department of Electrical Engineering
Stanford University
Email: kunyi@stanford.edu

Zhe Yang
Google Inc.
Email: rabbiest@gmail.com

***Abstract*—Abstract here.**

I. Introduction

Using artificial intelligence (AI) and machine learning (ML) algorithms to play computer games has been widely discussed and investigated. Valuable observations can be made on the ML play pattern vs. that of a human player, and such observation provide knowledge on how to improve the algorithms. Mario AI Competition [1] is such a competition that uses AI and ML techniques to play the classic title Super Mario Bros. The competition provides the evaluation system and benchmarks for the competitors every year.

Reinforcement Learning (RL) [2] is one widely-studied and promising ML method for implementing agents that can simulate the behavior of a player [3]. In this project, we study how to construct a Mario controller agent, which can learn from the game environment using the RL approach. One of the difficulties of using RL is how to define state, action, and reward. In addition, the state space cannot be too large, since playing the game requires real-time response, constraining the time to calculate each step of action. We use a state representation similar to [3], that abstracts the whole 22x22 environment description array into several key attributes. We use the Q-Learning algorithm to evolve the decision strategy that aims to maximize the reward. Our controller agent is trained and tested by the 2012 Mario AI Competition evaluation system.

The rest of this report is organized as follows: Section 2 provides a brief overview of the Mario AI interface and the Q-Learning algorithm; Section 3 explains how we define the state, action, reward to be used in RL; Section 4 provide implementation progress, early evaluation results, and discusses remaining issues and potential improvements.

II. Background

In this section, we briefly introduce Mario AI interfaces and the Q-learning algorithm.

A. Game Mechanics and the Mario AI Interface

The goal of the game is to control Mario to pass the finish line, and gain a high score one the way by collecting items and beating enemies. Mario is controlled by six keys: UP (not used in this interface), DOWN, LEFT, RIGHT, JUMP, and SPEED. In the game, Mario has three statuses: SMALL, BIG, and FIRE. He can upgrade by eating certain items(mushroom and flower), but he will lose the current status if attacked by enemies. Other than Mario, the world consists of different monsters (Goomba, Koopa, Spiky, Bullet Bill, and Piranha Plant), platforms (hill, gap, cannon, and pipe) and items (coin, mushroom, bricks, flower). The game is over once SMALL Mario is attacked, or when Mario falls into a gap. For more specific descriptions, please see [3] and [1].

When performing each step, the Mario AI framework has a call that returns the complete observation of 22 x 22 grids of the current scene. That is, an array containing the positions and types of enemies/items/platforms within this range. This is the whole available information for our agent.

The benchmark runs the game in 24 frames per second. The environment checking functions are called every frame. Therefore, while training we have to train and update the Qtable within 42 milliseconds.

B. Q-Learning

Q-learning treats the learning environment as a state machine, and maintains a reward value, denoted by Q, for each pair of (state, action) in a table. Here, $Q(s, a)$ denotes the Q-value, where s is the state and a is the action. For each action in a particular state, a reward will be given. The Q-value is updated based on reward by following rule:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t)+\alpha_{s,a}(r+\gamma \max(Q(s_{t+1}, a_{t+1}))) \quad (1)$$

In the Q-learning algorithm, there are four main factors: current state, chosen action, reward and future state. In (1), $Q(s_t, a_t)$ denotes the Q-value of current state and $Q(s_{t+1}, a_{t+1})$ denotes the Q-value of future state. $a \in [0, 1]$ is the learning rate, $\gamma\,[0, 1]$ is the discount rate, and r is the reward. (1) shows that for each current state, we update the Q-value based on the maximum Q-value of the next state.

Note we use the decreasing learning rate $\alpha_{s,a}$ [**?**] is different for different (s, a) pairs. Specifically,

$$\alpha_{s_t,a_t} = \frac{1}{\text{\# of times action } s_t, a_t \text{ has been performed}}$$

the decreasing learning rate allows for better convergence.

III. Mario Controller Design Using Q-Learning

A. Mario State

The state consists of the following fields:

- Mario Mode: 0 - small, 1 - big, 2-fire
- Direction: 8 directions + stay. Total of 9 possible values;
- If stuck: 0 or 1. Set true if Mario doesn't make movement over several frames.
- If on group: 0 or 1.
- If can jump: 0 or 1.
- If collided with creatures: 0 or 1.
- Nearby enemies: denoting whether there is an enemy in 8 certain directions in 3x3 window (or 4x3 window in large/fire Mario mode)
- Midrange enemies: denoting whether there is an enemy in 8 certain directions in 7x7 window (or 8x7 window in large/fire Mario mode).
- Far enemies: denoting whether there is an enemy in 8 certain directions in 11x11 window (or 12x11 window in large/fire Mario mode).
- If enemies killed by stomp: 0 or 1.
- If enemies killed by fire: 0 or 1.
- Obstacles: 4-bit boolean indicating whether there exist obstacles in front of Mario.

Note the nearby, midrange, and far enemies attributes are exclusive.

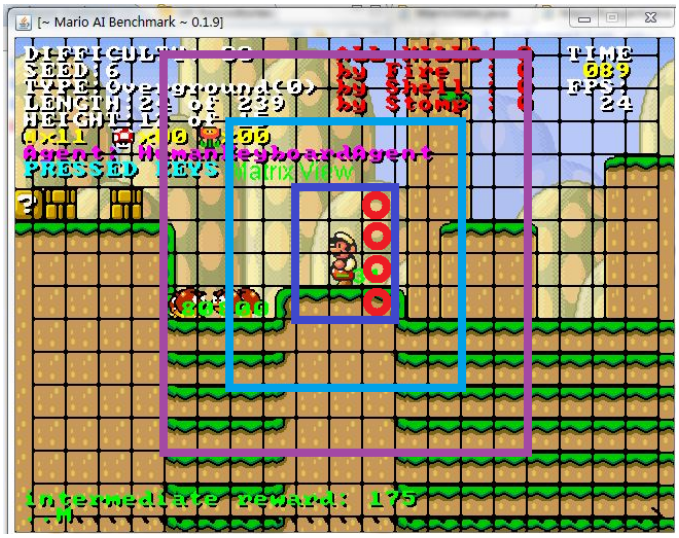

Fig. 1. Mario Scene

The figure 1 shows a typical scene of the Mario environment consisting of the platform and enemies. The area within dark blue box indicates the range of nearby enemies. The area between dark blue and lighter blue boxes indicates the mid-range enemies. The area between purple and lighter blue enemies indicates the far enemies. For instance, in the figure we have both mid-range and far enemies. The four red circles denote the obstacles array.

B. Actions

The Mario agent performs one of 12 actions. The combination {LEFT, RIGHT, STAY} x {JUMP, NOTJUMP} x {SPEED(FIRE), NOSPEED}.

C. Rewards

Our reward function is a combination of weighted state attribute values and the delta distance/elevation it performs from the last frame. We also let the reward of moving forward decrease when nearby enemies are present. Note our reward function is "approximately" only a function of our state only. Had us included the magnitude of speed, it will be completely determined by the state.

IV. Evaluation Results

-we train each Mario mode for 500 iterations, for example under 10 different random seeds. -After every 100 training iterations, we test the agent by running 50 episodes and use the average metrics as the performance indicator. -we have two tests: one chooses one of the training seeds, the other chooses a new random seed. -we expect performances of the two tests to be similar.

V. Conclusion

References

[1] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 mario ai competition," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Citeseer, 2010.

[2] R. Sutton and A. Barto, *Reinforcement learning: An introduction*, vol. 1. Cambridge Univ Press, 1998.

[3] J. Tsay, C. Chen, and J. Hsu, "Evolving intelligent mario controller by reinforcement learning," in *Technologies and Applications of Artificial Intelligence (TAAI), 2011 International Conference on*, pp. 266–272, IEEE, 2011.