CS229 Milestone Report
Reinforcement Learning to Play Mario

Yizheng Liao, Zhe Yang, Kun Yi

November 16, 2012

1 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 results are compared with the competition benchmarks.

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 and early evaluation results; finally Section 5 discusses remaining issues and potential improvements.

2 Background

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

2.1 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.

2.2 Q-Learning

Q-learning treats the learning environment as a state machine. Q-learning collects several environment variables and define them as some states. The learning process always on some states in Q-learning algorithm. Each state is assigned a unique state number. Q-value is computed for each pair of state and action and is saved in Q-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:

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

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.

3 Mario Controller Design Using Q-Learning

To illustrate why state space size is a problem, consider using the native 22x22 grid representation. There are 19 possibility for each block. Now if we take a 22×22 grid, there are totally

$$19^{22\times 22-1} \tag{2}$$

states, which by no means our algorithm can handle. We therefore use a state abstraction similar to [3]. The 7 attributes our state has are:

- Mario mode: 0 - small, 1 - big, 2-fire
- Enemy present?: a 4-bit field denoting whether there is an enemy in 4 certain directions. From 1st to 4th bit denotes higher front, higher back, ground front, ground back.
- Brick exist: 1 - there exist a brick/question mark within a given range; 0 – there does not exist a brick within a given range
- Item exist: type of most valuable items nearby. 0-no item, 1-coin, 2-mushroom, 3-flower
- Near gap: weather or not there is a gap within a given range
- On gap: weather or not Mario is jumping across a gap or falling in the gap

- Enemy type: represent the enemy type within a given range

As a results, we have a total of $3 \times 2^1 1 = 6144$ states, which is still large, but viable.

We also abstract the actions available by creating ”tasks” for Mario to choose. Each task may consist a series of keystrokes. Once a task is chosen, the corresponding keystrokes are determined from the environment observations using heuristics. In short, we learn the preferences and code the detailed operations. The 4 tasks are:

- Attack monster: attack enemies by stomping, fireball, and shell
- Search block: search bricks and question boxes
- Collect item: collect the coins and upgrade items available
- Pass through task: controlling Mario to cross the landforms and obstacles to move rightward
- Avoidance task: avoiding enemies and not getting hurt

Now, we reduce the number of state-action pairs to $6144 \times 5 = 30720$.

4 Preliminary Results

5 Next Step

For the rest of this project, we are going to train the Q-table based on the algorithms given above. Then we will compare our results with the benchmark. In addition, instead using self-defined reward rule, we will use the reward rule given by the evaluation system. Further, rather than updating Q-value every frame, we will update it every two or five frames. In this case, we relax the constraint on computational complexity. It allows us to implement a more complicated state-action space.

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.