

An Approach to Infinite Mario

Paul Ringstad

June 18, 2009

Overview

Environment

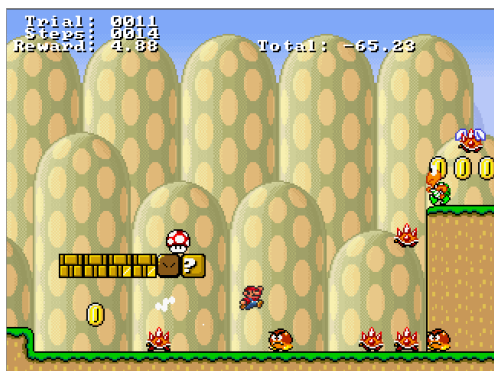
State Space

Learning

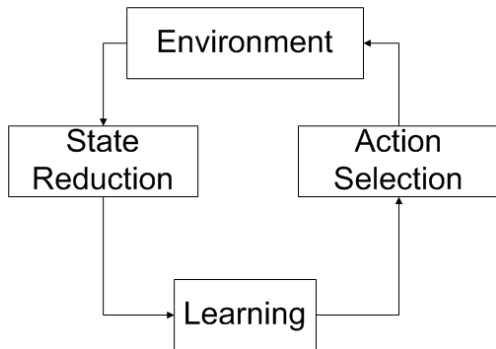
Results

Demo

Infinite Mario



Overview of system components



Domain description

A description of the domain is as follows:

- ▶ reminiscent of Super Mario Bros.
- ▶ control of Mario is 3-directional: left, right, or none
- ▶ mario can jump
- ▶ mario can run fast or slow
- ▶ environment is composed of elements, of type tile or monster, for which metadata is exposed (i.e. absolute location)
- ▶ recognizable structures such as pits and staircases in the levels
- ▶ elements can be accessed only by visiting the screen (no lookahead)

Domain description cont.

Additional features of the domain are:

- ▶ the game resets if Mario dies or if the number of steps in an episode exceeds some limit
- ▶ previous actions may affect the outcome of a current action (i.e. running then jumping)
- ▶ rewards from the environment are the result of stepping, collecting coins, smashing blocks, gathering mushrooms, and dying.
- ▶ rewards vary depending on the parameterization and may include some noise
- ▶ parameterization of a level does not change during a trial

Notes on the domain

The following aspects of the domain were useful in the designing of the agent:

- ▶ Many "acceptable" paths between two locations in the level
- ▶ Few, if any, entirely wrong paths
- ▶ Mario is highly maneuverable
- ▶ Actions mid-jump can be used to control the trajectory
- ▶ Access to sprites makes it easy to identify elements in the screen
- ▶ Levels contain locales with high monster density and stretches with few or no monsters

Challenges

Some of the challenges faced in the Mario domain include:

- ▶ High dimensionality of the state
- ▶ Dynamic nature of the environment
- ▶ Time constraints during learning
- ▶ Unknown reward distributions

Intuitions about the state space

Taking into account the aforementioned characteristics of the domain, the process by which the raw state is transformed follows from the hypotheses:

- ▶ There are non-salient features of the state that can be ignored
- ▶ Salient elements of the state that are closer in proximity to Mario are more critical in determining a policy at that state
- ▶ Discretizing the state should not prevent the agent from learning an effective policy

In the end, the goal of transforming the state space is to decrease its size

Discretization

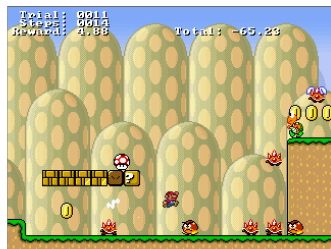
For a given screen, parameters for each of the tiles and monsters are discretized. An element from the state becomes:

- ▶ Elem: $\text{offX} \times \text{offY} \times \text{tile} \times \text{monster} \times \text{monsterType} \times \text{isPit} \times \text{atWallRight} + \text{null}$
- ▶ offX : $\{-10, -9, \dots, 10\}$
- ▶ offY : $\{-7, -7, \dots, 7\}$
- ▶ tile : $\{ \text{'null'}, \$, \text{b}, \text{?}, \text{—}, \text{!}, 1..7 \}$
- ▶ monster : $\{0,1\}$
- ▶ monsterType : $\{0,1\}$ ("*stomp-able*" or not)
- ▶ isPit : $\{0,1\}$
- ▶ atWallRight : $\{0,1\}$

Reducing the state space

Problem: After discretizing, the resulting state space is still really large!

Solution: Only keep those elements from the discretized state that are most relevant to the task of staying alive



Ranking

The ranking process works as follows:

- ▶ Define element comparisons in terms of offset distance, block type, whether the element is a monster, and whether it is a pit
- ▶ Rank the elements of the discretized state using the comparisons above
- ▶ Use the *top k* ranked elements in our final representation (in our agent $k = 3$)

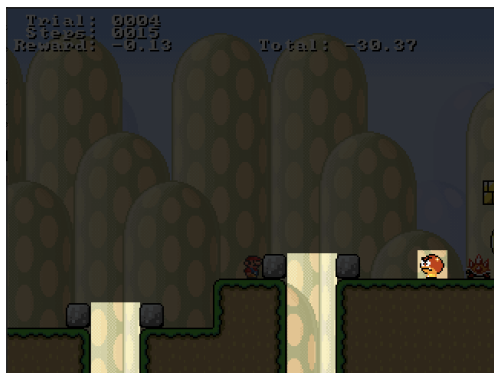
Example 1: After state reduction



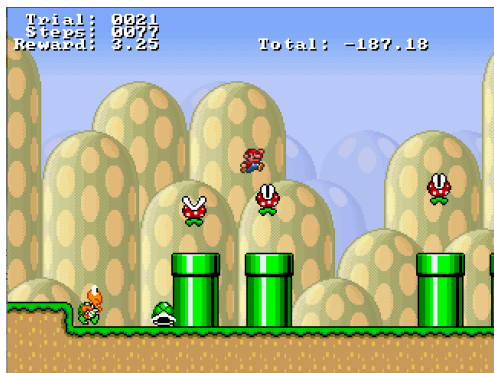
Example 2: Before state reduction



Example 2: After state reduction



Example 3: Before state reduction



Example 3: After state reduction



Complete state representation

Using the k elements from ranking ("RankedElems"), our complete state representation is:

- ▶ State: Action \times RankedElems
- ▶ Action: PrevAction + null
- ▶ PrevAction: jumping \times speed \times direction
- ▶ jumping: $\{0,1\}$
- ▶ speed: $\{0,1\}$
- ▶ direction: $\{-1,0,1\}$

Notes on the state reduction

- ▶ Deterministic
- ▶ Easy to determine reduced state for a screen: good for debugging
- ▶ Ranking allows embedding of implicit domain knowledge
- ▶ Size of resulting state space changes with k

Intuitions about learning in the Mario domain

Our intuitions concerning learning and performance for an agent in this domain are as follows:

Performance (cumulative reward) will benefit if exploration is limited to the end of the previous (failed) trajectory

Aspects of the domain that support this hypothesis are:

- ▶ Many "acceptable" paths between locations in level
- ▶ Few, if any, entirely wrong paths
- ▶ If the agent dies and the level restarts, the previously used path is usually acceptable up to a place near the previous point of termination
- ▶ Mario is highly maneuverable

Learning Algorithm

On every step, *reward* is returned by the environment and the following is executed:

```

 $\gamma \leftarrow .75$ 
 $MLB \leftarrow 100$ 
 $idx \leftarrow size(states)$ 
for  $i = idx$  to  $i = idx - MLB$  do
   $s \leftarrow states(i)$ 
   $a \leftarrow actions(i)$ 
   $Q(s, a) \leftarrow Q(s, a) + \gamma^{idx-i} * reward$ 
   $i \leftarrow i - 1$ 
end for
  
```

Notes on the learning algorithm

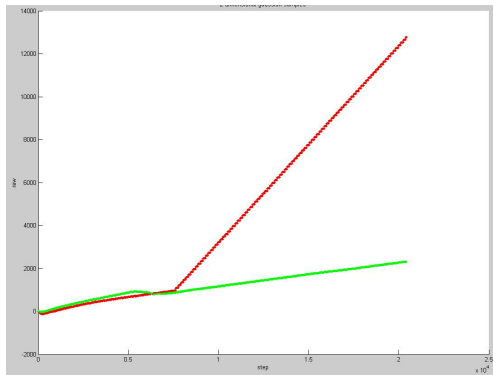
The following parameterization is also employed:

- ▶ Rewards arriving from the environment are biased: $r_{step} = .01$, $r_{death} = -10$, $r_{finish} = +100$
- ▶ Additional reward $+1.5$ for traversing intervals of 10 tiles away from the start
- ▶ Experience-replay in which the reward at each step is distributed to state-action pairs in the trajectory history
- ▶ Actions are chosen greedily

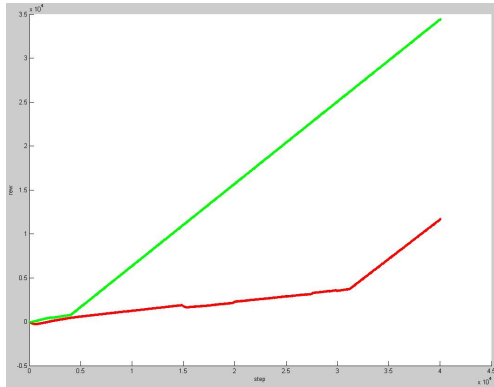
Known issues with the learning algorithm

- ▶ Learned values do not converge
- ▶ Partially observable state and non-convergence property could irreparably set the wrong policy
- ▶ Learned policy does not generalize well between levels
- ▶ Highly dependent on reward distributions

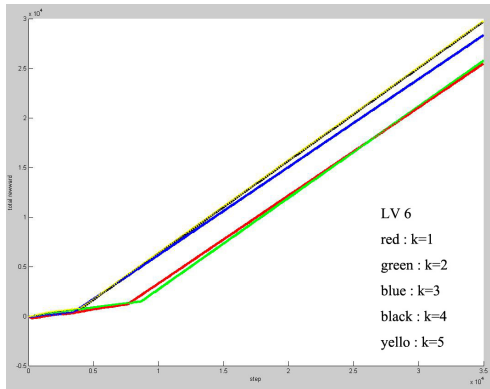
Comparing with John's test agent



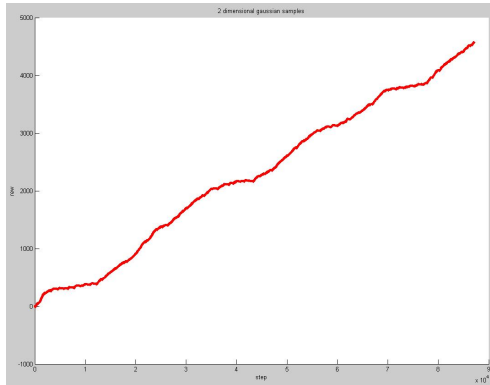
Comparing with SARSA



Varying k



Without interval reward



Demo time!

Thanks to teammates Saehoon Yi and Fengming Wang.

Thank you!