Autonomous Agents (COMP513)

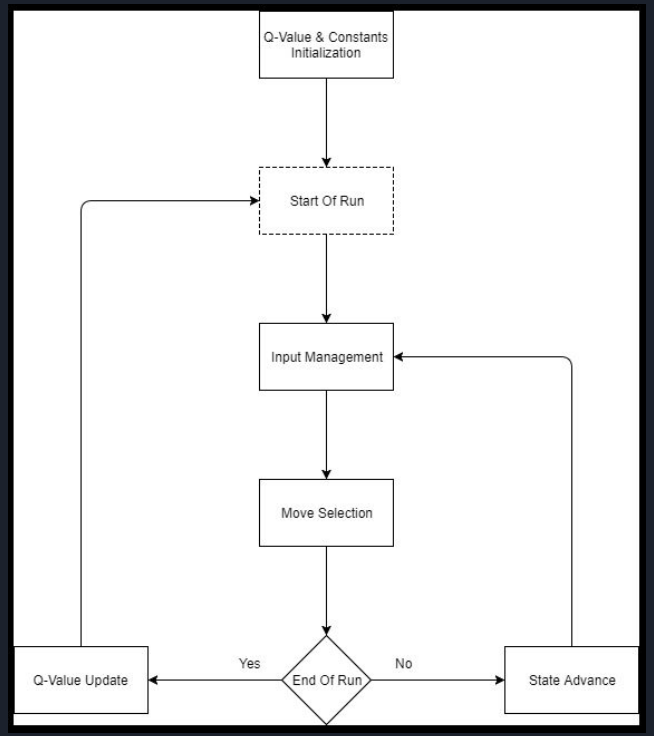
Q-Learning Super Mario World

Papathanasiou Theodoros 2011030058

Implementation Model

Four Foundational Parts:

1. Q-value and Constant Initialization
2. Input Management
3. Move Selection
4. Q-Value & State Update OR State Update



Q-value and Constant Initialization

After extensive experimental runs we concluded on the following values for the Q-Value update function constants:

learning rate: $\alpha = 0.5$

discount factor: $\gamma = 0.8$

We also concluded on the value and annealing rate of the temperature parameter in Boltzmann exploration:

temperature: $T = 4$

annealing rate: 0.001 / run

Input Management

On this part of our implementation, we use MarI/O's infrastructure to detect mario's movement to the right.

We keep track of the rightmost mario has traveled through the change of pixels in the emulated rom, and compute an analogous reward.

Move Selection

After each state advance, our agent considers what it’s next move should be.

The first step is to compute each action’s probability based on an exploitive exploration technique, Boltzmann exploration.

Then, a pseudo-random number is generated and we choose an action based on their probabilities.

The largest the Q-Value of an action, the largest its probability.

Q-Value & State Update OR State Update

At each iteration of our game algorithm we check if mario has been stationary for too long.

We check if our rightmost position has changed and if not we update a timeout counter.

If the timeout counter reaches 60 we end the run and proceed to the Q-Value update, else we just proceed to the next state

Q-Value & State Update OR State Update cont.

When the run comes to an end, the agent updates it's Q-Values. We iterate from the first state to the last and update the values based on the result we received.

$$Q(s,a) = Q(s,a)+a(r+maxQ(s',a')-Q(s,a))$$

If s' is a terminal state then:

$$Q(s,a) = Q(s,a)+a(r-Q(s,a))$$

The reward is given as:

$$r = \text{Rightmost Pixel on X Axis} / 8$$

Results

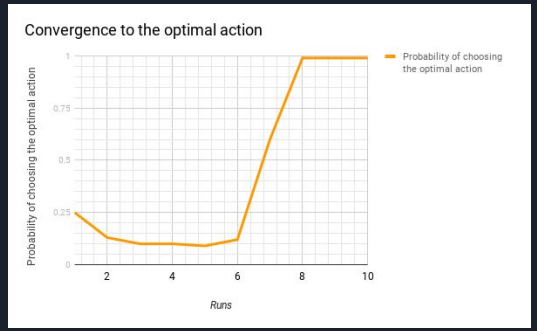
With our current configuration we get the following statistics

Avg. runs per obstacle

1.47

Avg. runs to finish

32



Future Work

For future work, we suggest the following:

- Improvement in recognising when a run has ended e.g. through the death animation.
- Improvement in the design of the state space and its representation either through neural networks or a more efficient lua based architecture
- Introduction of machine vision(pattern recognition) to create a more observable game and a knowledge based on causes rather than only the effects of events

Questions?

Thank you for your time

Thodoris Papathanasiou