Michael Hewner 3/9/2011

My Mario AI uses Q-learning. I was curious to see how far a simple algorithm like Q-learning, which does not attempt to model the "reality" the game could get, insofar as score went. Overall, I was pretty pleased – the trained up agent certainly plays plausibly well even if it does tend to miss obvious wins like coins and powerups (unless you get really lucky).

You can see a video of my AI doing a level 2 difficulty level on my blog:
http://hewner.com/2011/03/08/reforcement-learning-mario-ai/

In a Q-learning system, every possible game state has Q-value (quality value) that combines a reinforcement (R) value (approximate "goodness" of the current state) with the best-known quality of subsequent states. Every time the game enters a state, it updates the Q-value for that state (called S in this equation):

$$updated\ Q(S) = old\ Q(S) + \alpha(R(S) + \gamma\ \times \max Q(S')_{S' is\ a\ state\ reachable\ from\ S})$$

Mario's State (S)

One of the strengths of Q-learning is that the algorithm does not need to represent details of the state explicitly. In the case of my algorithm, the state is explicitly represented as simply Mario's total sequence of moves since the game state. Because this game trains on the same level every time, this unambiguously identifies the state. It does mean, however, that Mario is training on the sequence of moves rather than attempting identify regular general features of the environment.

As a result, no state will ever be visited except by the same path so they rewards are always the same and the quality function simplifies:

$$Q(S) = R(S) + \gamma \times \max Q(S')_{S' is\ a\ state\ reachable\ from\ S}$$

Reward (S)

In my algorithm, the reward of a state is some combination of the score and Mario's bigness. The score is expressed as a fraction of the overall score of the "best" final node…the node that Mario dies in if the path of maximum badness is taken.

In general, the algorithm tends to give a much high value to bigness…otherwise Mario tends to "charge" through groups of enemies getting temporary score boost at the expense of long term success.

Mutation

One common mechanism for mutation (that is, deviation from the known best path in order to search for better ways of doing things) is to have a static "randomization chance" at every node. This is problematical using my representation of Mario because Mario is much more likely to randomize early than persist down the "known good" path for a long time. But as the game continues and good strategies for the beginning of the level exist, it becomes much more worthwhile to explore variations towards the end of the known good path

My solution is the base the randomization likelihood on the fraction of the "best" score Mario has achieved. The graph can be seen here…the x axis is the percentage of the "best" score that has been achieved and the y access is the likehood of randomization at a particular node. I selected parameters so that the probably of mutation at .7 of the best score was .05 while the probably of mutation at .9 of the best score was .5.

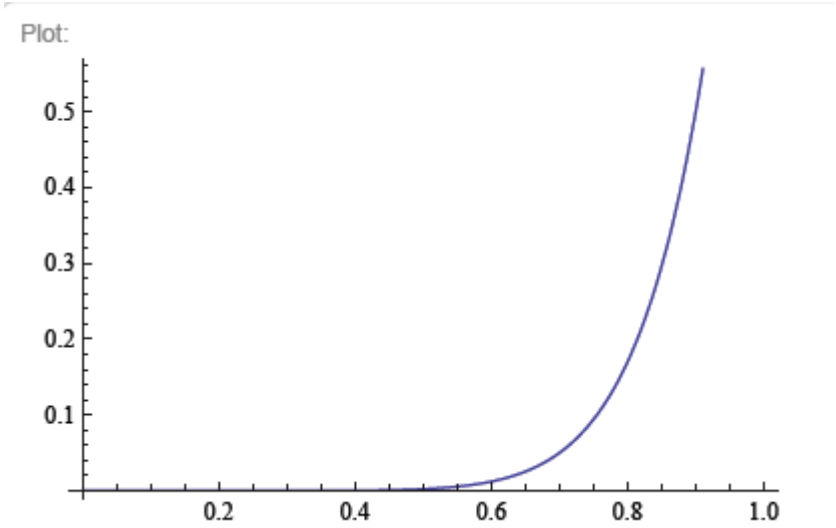

Figure 1 $randomization\ likehood = (score\ percent)^{9.13} \times 1.3$

Future State Prioritization

An important factor is Q-learning algorithm is γ, the extent to which the "goodness" of future states is considered verses the rewards earned in the current state. I have generally found that γ around .7 tends to produce the highest scores in my simulation. If the selection is too low, Mario simply looks for locally good states without regard for the possibility that a locally smart decision might cause trouble in the longer term. If the selection is too high, Mario will prize states that had one long good run, without the considering the fact the that decision may not have a lot to do with the success of those future states. I suspect .7 works because it corresponds well to the way that SMB works – a bad decision (like walking into a pit) may take a few actions to cause problems, but Mario is not a strategic game where a bad move in the early game might secretly cause a major problem very far in the future.

Future Work

The main issue with the current algorithm is that the Mario state space search is essentially random variations from a single "best" path. This ignores the fact that many different paths may be nearly equally promising and that Mario should split his attention between the various options.

Another problem is that because states are representations of move sequences, there is no ability for similar states to be collapsed into a single state. This does not cause a problem in terms of memory, but it does mean that what is learned on a particular path might have to be re-learned on what is a essentially the same state in reality. This turns out not to be much of a problem in practice, because the

moving enemies make states that seem similar pretty different in terms of stragedy (at least, so long as a simple strategy that does not model the enemies explicitly).

The final issue is that Mario has essentially the same likehood of varying on the last run as the first. Towards the end, Mario could definitely score more points by concentrating on varying very high scoring paths, rather than using the “best” path which may have a high future potential but likely will take much more work to beat the current high score.

How to Run this Code

I simply invoke ch.idsia.scenarios.champ.LearningTrack with the parameter “-ag com.hewner.MarioAI.MikeLearningAI” and the jar included in my CLASSPATH.