Markov Decision Processes and Reinforcement Learning

Hermann Hans

Department of Computer Science, Georgia Institute of Technology

November 28, 2016

Abstract

In this assignment we'll explore two different Markov Decision Processes (MDPs) and their solutions through the use of value iteration and policy iteration. The MDPs chosen vary greatly in the number of states to reason about in order to highlight differences in convergence, solution, and runtime. Furthermore we'll take a look at Q-Learning, a reinforcement learning approach to find a solution while discarding any knowledge of the mode, rewards, etc.

1 Introduction

1.1 General Approach

All analysis was done using the Brown-UMBC Reinforcement Learning And Planning (`BURLAP` [2]) Java library. This library was extended as necessary to provide any additional output as it was required for the analysis of the MDPs or the algorithms run against them. Runtime, iteration counters, parameter tuning options and more were added, but also additional output options of the graphical user interface such that screenshots of each policy iteration could be made as required. This allowed me to not only reason about the algorithm on a mathematical level, but also visualize the algorithm(s) at each time step and the policies they output.

To make it as easy as possible to make modifications to MDP inputs, a simple image reader was written with the ability of converting images to grid worlds. This allowed for a more iterative approach at the start of this project while trying out different grid worlds and avoided the hassle of having to manipulate large matrices of 1s and 0s directly in the code.

2 Markov Decision Processes

Markov Decision Process (MDP) can be defined as a sequential decision problem for a fully observable (partially observable MDPs are beyond the scope of this assignment) stochastic environment with additive rewards and a Markovian transition model. Briefly, a MDP consists of a set of states S, a set of actions for each state $A(s)$, a reward function $R(s)$ and a transition model $P(s'|s,a)$. Additionally, since we'll be talking about MDPs with infinite horizons, γ is used as a discount factor to reduce the importance of future rewards (and avoid the existential dilemma of immortality).

With this definition in mind, our goal then becomes to find a policy π^* which maximizes the sum of the rewards (discounted by γ):

$$\pi^*(s) = \underset{\pi}{\operatorname{argmax}}\, E\Big[\sum_t \gamma^t R(s_t) \mid \pi\Big]$$

Markov Decision Processes are mainly used for planning and decision making. Some real-life examples of what they've been used for include stock market investment planning, operations and maintenance planning in various sectors, purchase and production planning, etc...

In the subsequent sections, two such processes are described and are used throughout the rest of this paper.

2.1 Tiny Grid World

The first MDP is a small, fairly trivial 5x5 Grid World problem with 20 possible states. The starting position is always at the bottom left of the grid and a terminal position is at the top right. In order to "motivate" the agent to move towards the goal, the terminal position was setup with a reward value of 100, while each "step" (or state change if you will) was set up with a reward value of -1. Two fields marked in red were also added just shy of the starting position which add a penalty of -10 if the agent moves into these fields.

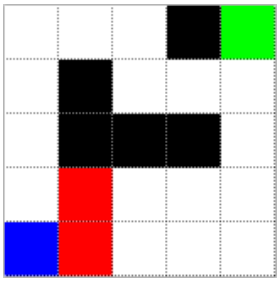


Figure 1: Tiny Grid World. Start state, terminal state, penalty state (Blue, Green, Red)

2.2 Maze

The second MDP used is a 81x81 maze with a total of 3204 states. The maze was produced using a random online maze generator [1] to get an image, which was then loaded into `BURLAP`. For the maze MDP, there are only start and stop states, with no penalty states (red squares from the previous example) added. The reward values for moving and finishing are the same as for the tiny grid world described above.

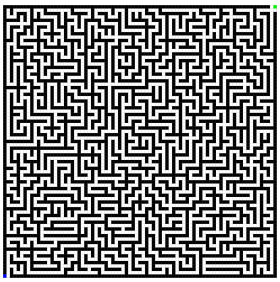


Figure 2: MDPs used. Tiny Grid World [LEFT] and Maze [RIGHT]

For both MDPs, the same stochastic movement function was used. If the agent chooses a direction on the 2D grid world plane (`NORTH`, `SOUTH`, `EAST`, `WEST`), then that action actually only takes place 80% of the time, while the remaining 20% are split between the other 3 directions.

3 Value Iteration

Value Iteration tries to find an optimal solution (policy) for an MDP by iteratively updating the utilities from each state.

$$V_{k+1}(s) = \max_a E\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a\}$$

To get started with value iteration, an initial discount factor γ of 0.99 and maximum δ 0.001 were used. Using the tiny grid world, I first ran some tests on convergence and output the optimal policies and the number of iterations it took to get there. Figure 3 shows the policies at 3 such different timesteps (1, 5, and 10 respectively). What's particularly interesting here is that value iteration immediately recognizes the "penalty" squares, and while it hasn't found the "optimal" movement from the start, it does provide the largest deviation of utility from the other states from the very start. This changes with more iterations as the algorithm converges towards the optimal policy of course.

The same process was applied to the maze MDP, but for the sake of brevity, the plots aren't shown for value iteration. They are shown for policy iteration below though.

Once the optimal policy and the number of iterations using value iterations was known, trials were run against the calculated policy at each iteration. The result of this is shown in figure 4 with a

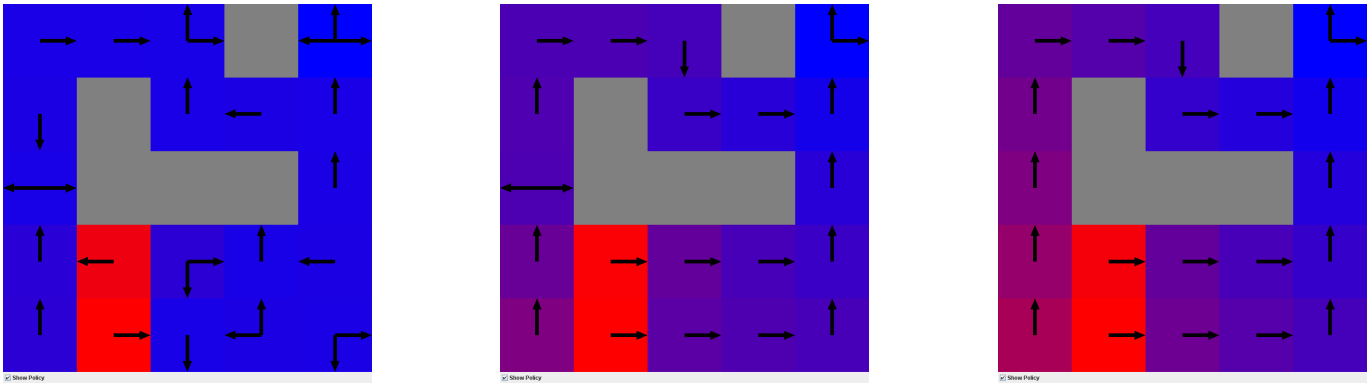


Figure 3: Value Iteration for Tiny Grid World at iteration 1, 5 and 10 (left to right)

logarithmic scale on the y-axis, as the number of required steps to get to the goal drops considerably within the first few iterations. This lines up well with the previous graphics where the biggest policy changes were at the beginning and only minor modifications took place with the last few iterations before converging towards the optimal policy.

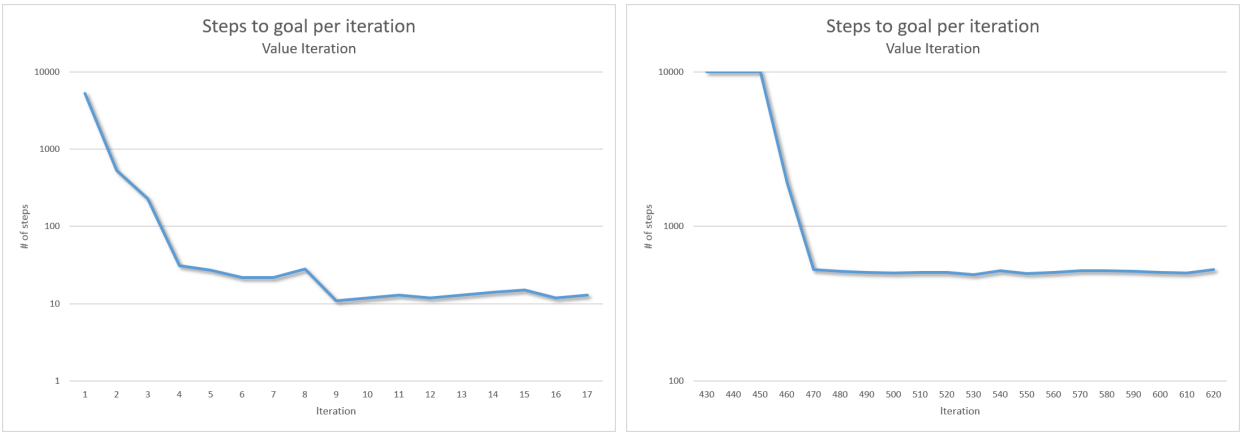


Figure 4: MDPs used. Tiny Grid World [LEFT] and Maze [RIGHT]

The same thing was applied to the much larger maze MDP, but a maximum number of 10,000 steps was used to stop the search at each iteration if it couldn't be found within that timespan. This cap was hit at every iteration until ca. 460 iterations at which point a policy was found that enabled the maze to be solved in under 1000 steps right away.

4 Policy Iteration

Policy Iteration, as opposed to value iteration, uses the policies directly. Starting out with some initial policy, the algorithm attempts to improve the policy at each iteration until it converges. This is actually a two-step process, divided up into policy evaluation and policy improvement.

In `BURLAP` there's parameters not only to set the maximum number of iterations of policies, but also for the policy evaluation step which is executed as an inner loop on each iteration. Capping the

number of iterations for this inner loop had the effect of decreasing the runtime of the algorithm. Additionally, moving this number of iterations below the convergence of value iteration for the same MDP had the effect of increasing the number of iterations required by policy iteration to converge. Playing around with these settings has the potential to tune the algorithm's runtime while still finding an optimal solution to an MDP.

Just like previously, the first analysis was done on convergence with the obvious result being that policy iteration takes way less iterations to find an optimal policy. In the case of the tiny grid world example, it only took 3 iterations, while the maze grid world took 19 iterations to converge.

From the policy output at each iteration we can also see the more granular step size of policy iteration in respect to updating the policy. In figure 5 we can see the policies at iterations 1, 9 and 19 (converged).

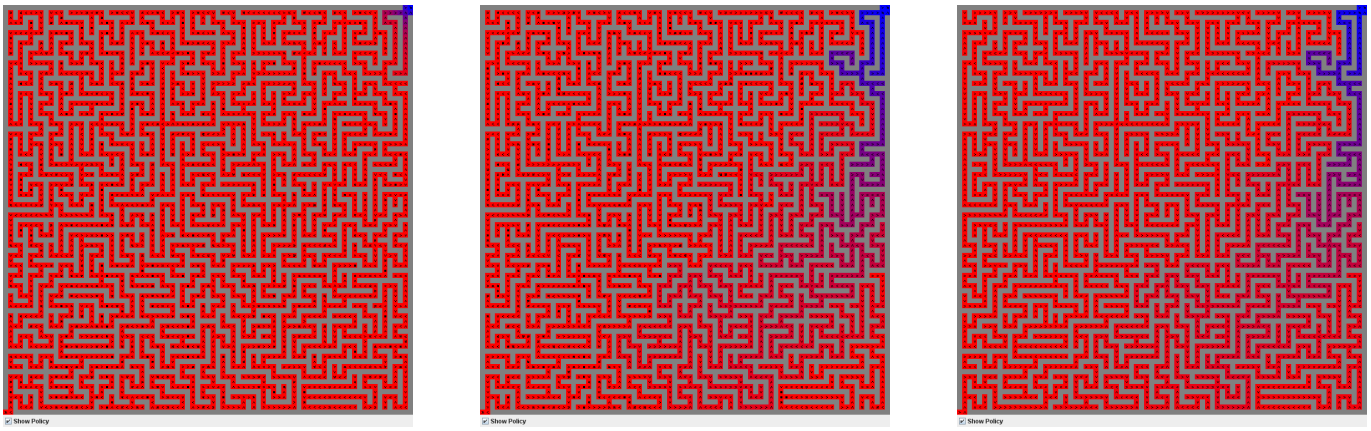


Figure 5: Policy Iteration at iteration 1, 9, and 19 (converged)

Like with value iteration, running an agent through the maze at each policy iteration shows that there's a significant drop in the steps required as we get closer to converging towards the optimal policy. Figure 6 shows this drop when going from iteration 12 to 13, after which we've basically converged anyway. The remaining updates done to the policy are mostly to states that are already cut-off from the main path connecting the start and stop state, explaining why we can practically see the convergence in the number of steps required, while theoretically the optimal policy still hasn't been found.

Figure 6: Steps required at each iteration of PI in maze

5 Q-Learning

Unlike the previous two algorithms, Q-Learning (QL) is actually a model-free, reinforcement learning method.

For QL, γ was also set to 0.99 to compare it to the previous algorithms. I setup a learner that would iterate through 500 learning steps (more than enough). For each learning step, I stepped through the resulting policy ten times and took the average of those runs. The result is shown in figure 7, plotting the number of steps required at each iteration of QL.

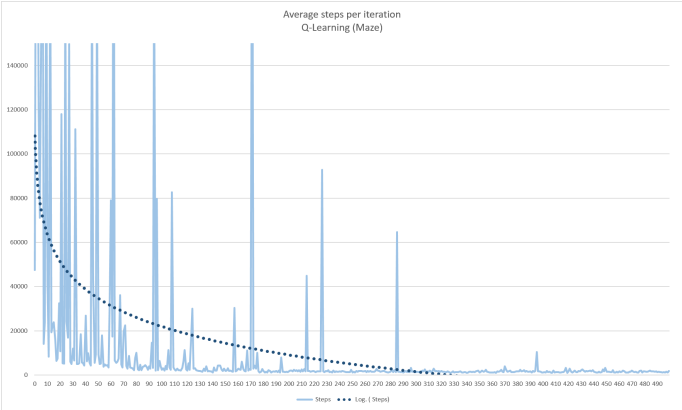


Figure 7: Average number of steps at each iteration of QL for maze

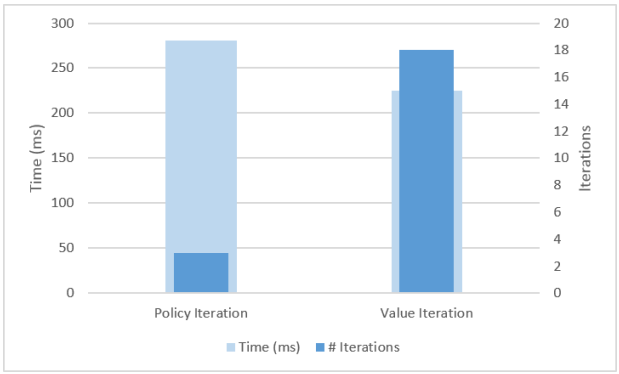
What may be hard to read out of the graph is that the algorithm converges toward around 1000 average steps required for the policy it finds. While this is more steps than what was required for the other, model based algorithms, this is still a very impressive result considering the runtime is close to an order of magnitude faster and is doing it without the extra information that the other two algorithms can use.

The next section goes into a bit more detail with regards to runtime and steps required in comparison the model based algorithms.

6 Further comparisons

Comparing value and policy iteration in figure 8, even using the tiny grid world with its 20 states, we can already see the fundamental difference between these two algorithms in the number of iterations required to converge. The stark contrast is highlighted even further when using the maze grid world, where value iteration takes some 623 iterations to converge while policy iteration only takes 19.

Like previously already mentioned, for policy iteration we can tune the inner value iteration loop. For the graphs displayed I had actually used a much larger value than required. Moving it closer, but not below the value of the value iteration convergence found previously yielded much better run times. Furthermore, using values smaller than the value iteration convergence, increased the number of iterations that policy iteration required to converge. Intuitively this makes sense as we are



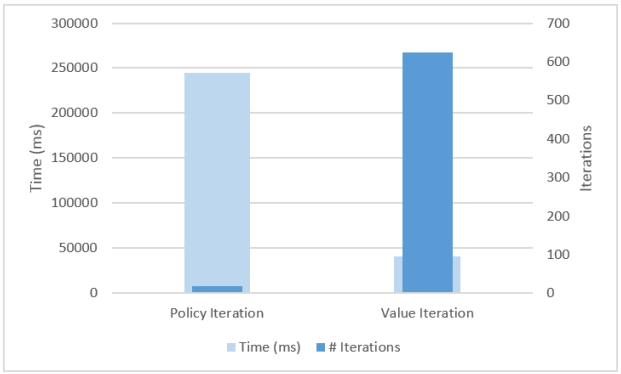
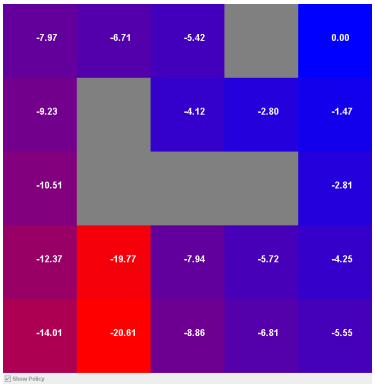
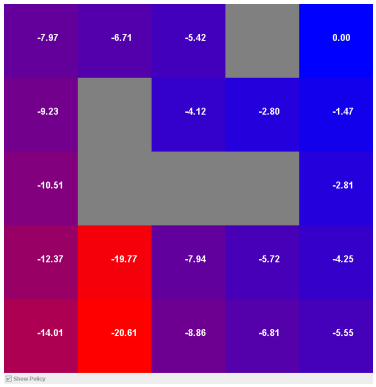


Figure 8: Runtime and iteration comparisons between VI and PI of tiny [LEFT] and maze [RIGHT]

now moving below the threshold where a single policy iteration run can already start to converge, therefore it will require more outer loop iterations to get to the same point.

In figure 9, the optimal policies for the tiny grid world are shown. For VI and PI, they both converge to the same policy, while the "optimal" policy for QL has learned different values.





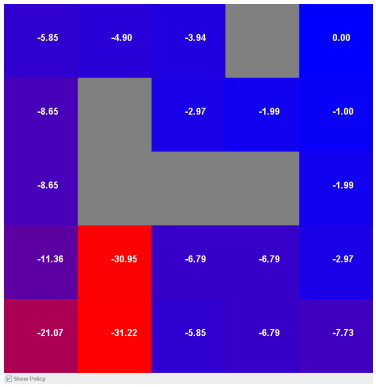
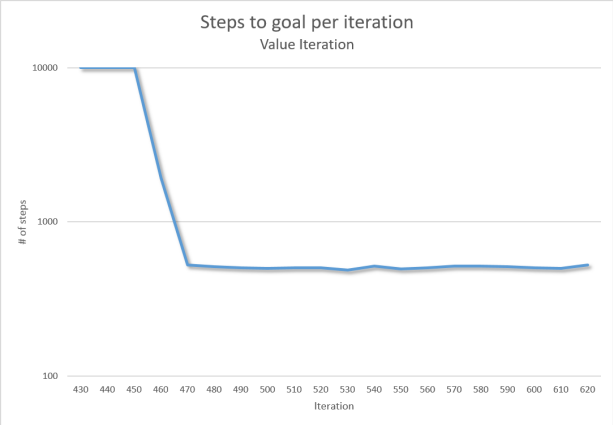


Figure 9: Optimal policies (VI & PI) and learned policy from Q-Learning

Another interesting analysis point was the number of steps actually required to run through the maze grid world for each iteration. Value iteration and policy iteration were both set up to run up to the point of convergence found earlier, but after each iteration a "run-through" was performed with the provided policy. For value iteration the step size for each iteration was increased from 1 to 10 and a cap of 10,000 steps maximum was used before terminating and moving to the next iteration. Additionally, the run-through was done 3 times and the average of those runs was taken.

In both cases we can see that there's a certain point at which the number of steps required abruptly drops down and immediately converges. For value iteration this happened between setting the maximum iterations from 450 to 460, while for policy iteration this event occurs at iteration 12 to 13.

To get more insight as to the cause of this, the actual policies were output graphically and compared, and in both cases we can see that the policies no longer change in the upper right of the maze (close to goal state), but only minor changes are happening towards the bottom left and



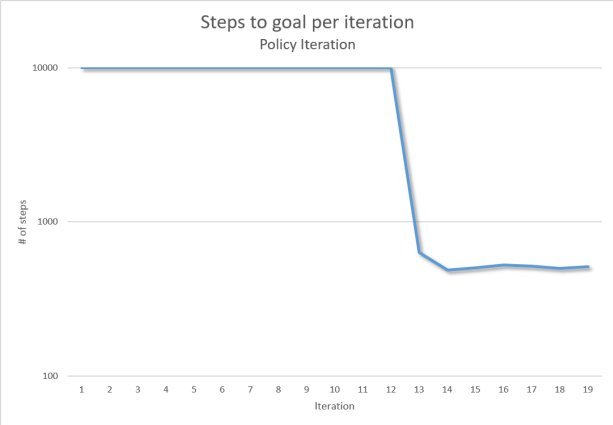
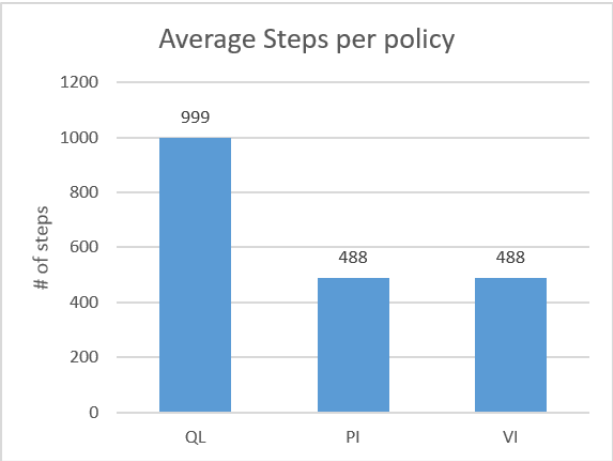


Figure 10: Comparison of steps required for PI and VI

some other remote parts of the map. These changes, subtle as they may be, play the largest towards actually reducing the number of steps required in the maze grid world. Additional, long-running tests with an increased maximum allowed steps (or unlimited steps), would've been interesting as well to visualize the rate of change as we get closer to the optimum policy.

Finally, a comparison was done between the number of steps required for an agent using the optimal policy as well as the runtime required to get to said optimal policy. Figure 11 shows this comparison graphically. Sine PI and VI converge to the same optimal solution it should then come as no surprise that they require the same number of average steps to move through the maze. While QL takes more steps relative to the other algorithms, the runtime I recorded (conservatively) is much lower than that of PI and VI. Additional analysis could be done here, especially with regard to "convergence" in QL, as this wasn't fully explored by me for this assignment.



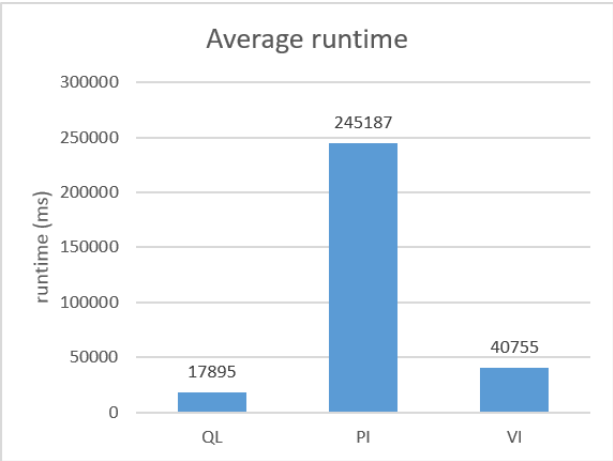


Figure 11: Comparison of QI, PI, and VI in average steps and runtime

7 Conclusion

While moving through some standard analysis functions for this assignment, the maze grid world in particular highlighted some interesting issues. Particularly the sharp drop from the number of required steps in both value and policy iteration as the algorithms move towards convergence. Once a proper connection from start to finish was made in those cases, the remaining policy updates, while still moving towards an "optimal" policy in the sense of updating all other states in the grid world, weren't necessarily relevant anymore. This suggests to me, that there must be other optimizations that can be made to these algorithms (shortcuts if you will), which would return a policy that's just "good enough".

With Q-Learning, while the number of steps required by an agent didn't come down to the "optimal" solution of a model based algorithm that's aware of the mode and reward functions, in the case of the maze grid world it came up with a policy an order of magnitude faster than the other two algorithms.

There's a lot of potential for further exploration here. I missed out on a lot of parameter tuning in this assignment which would've made for interesting discussions as well. Tuning and comparing parameters such as the discount factor, maximum deltas, modifying the transition probabilities or step penalities in the same MDP, all of these could be explored in more detail in an extension of this assignment.

References

[1] Maze generator - http://www.delorie.com/game-room/mazes/genmaze.cgi.

[2] James MacGlashan. BURLAP brown-umbc reinforcement learning and planning.