- **Email**: chrisbrown@utexas.edu
- **EID**: chb595

1 Q-learning

1.1 Enviroment

Figuring out how to quantify the state spaces was the hardest part of the assignment. For the obstacle module, I ended up assigning one label to every combination of adjacent (up/down/left/right) squares that were occupied with obstacles (12, plus 1 for the case where there are no adjacent obstacles).

For the state space of the finish module, there is only one state: the one where you try to get to the end. I thought about having something like a `floor(log(distance-from-end))` state space; which would allow the learner to worry less about speed if the end of the sidewalk was far off. This actually just made the finish module slower, and didn't make sense in this problem, so I did not include this subtlety in the finished product.

I played around with different dimensions of sidewalks and obstacles density, as well as Q-learning algorithm parameters; the results below are each labeled with the parameters.

In each of the plot titles, a means the alpha parameter, g means the discount parameter, and e means the epsilon parameter. For interpretability of results, I have set the reward for finishing equal to the length of the sidewalk. I played around the amount of the penalty. But for most of the runs below, the penalty was around -10.

One problem I ran into pretty quickly was the low epsilon default; at 0.9, we get results that don't seem nearly as good as they ought to be:

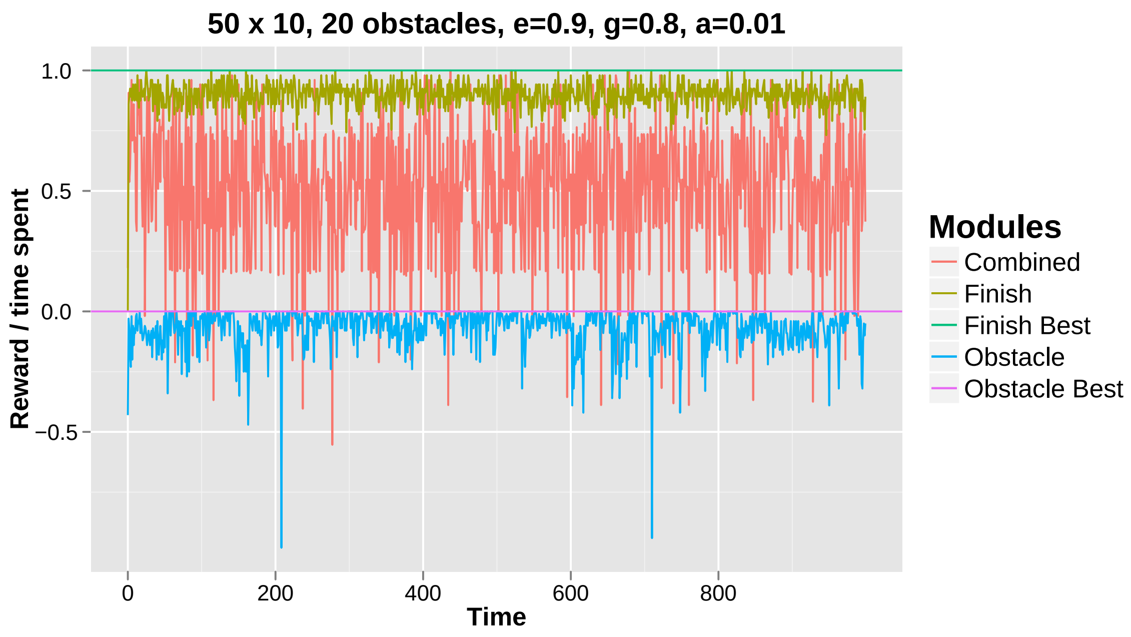


Figure 1: epsilon = 0.9

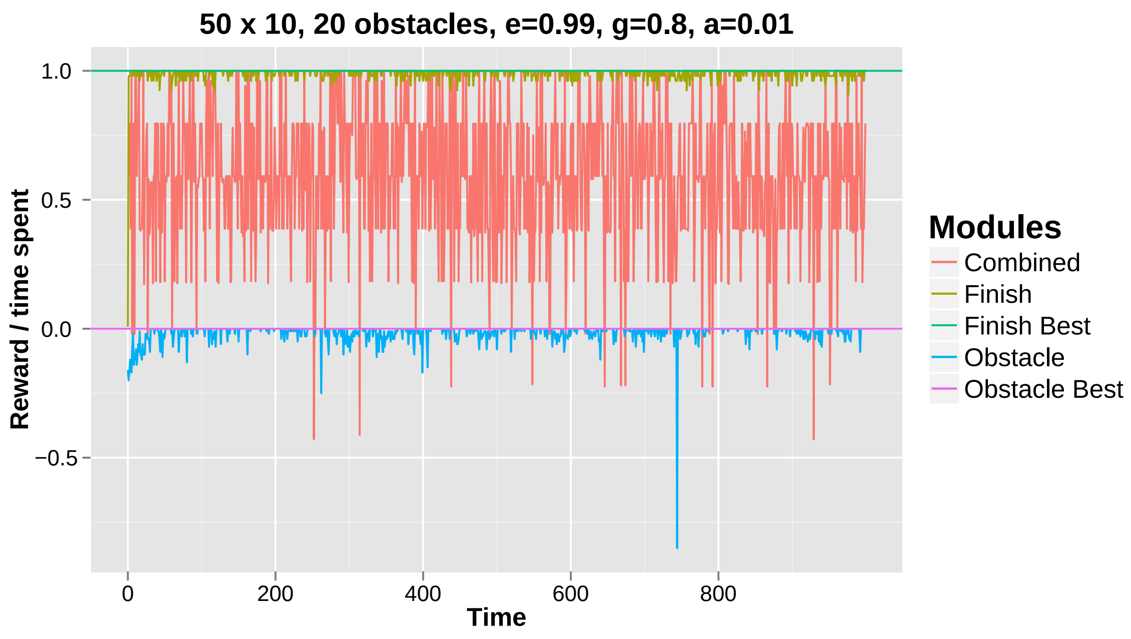


Figure 2: epsilon = 0.99

In Figure 2, for example, we get much better results giving the algorithm high confidence in what it should do at each step, instead of sometimes doing something randomly.

Figuring out appropriate weights for each module's Q was hard, and I may still not have gotten the best weights. This should be part of the learning algorith, I think. The individual episodes should train the Q's, and the merged episodes should train the weights. I did have time to implement this, unfortunately.

In the next two graphs, I show different weightings; the second skews dramatically towards preferring the obstacle-avoidance module.

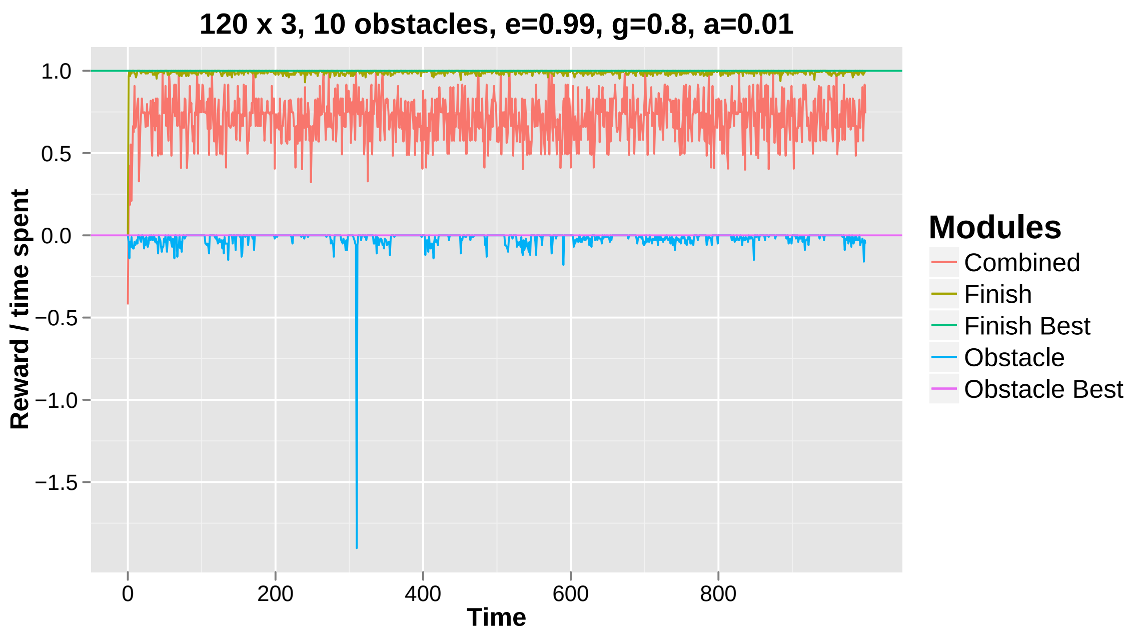


Figure 3: weight = 5/1 obstacle/finish

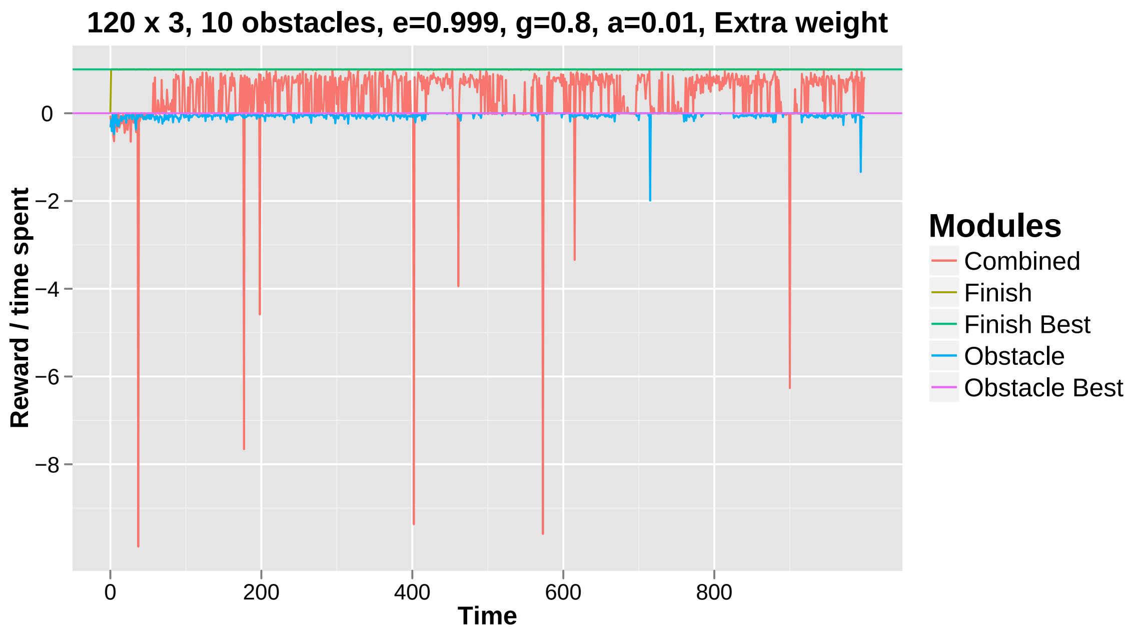


Figure 4: weight = 50/1 obstacle/finish

I made the state space more nuanced for the following test; in this one, instead of looking at just the candidate to the right, the agent could looks at the three blocks one-away, and five blocks two-away, in each direction. Presumably, this would help with being able to keep the epsilon low, since the probability of doing something stupid (random) twice in a row is epsilon squared. It turns out that this does help quite a bit, though it takes longer to learn; see Figure 5.

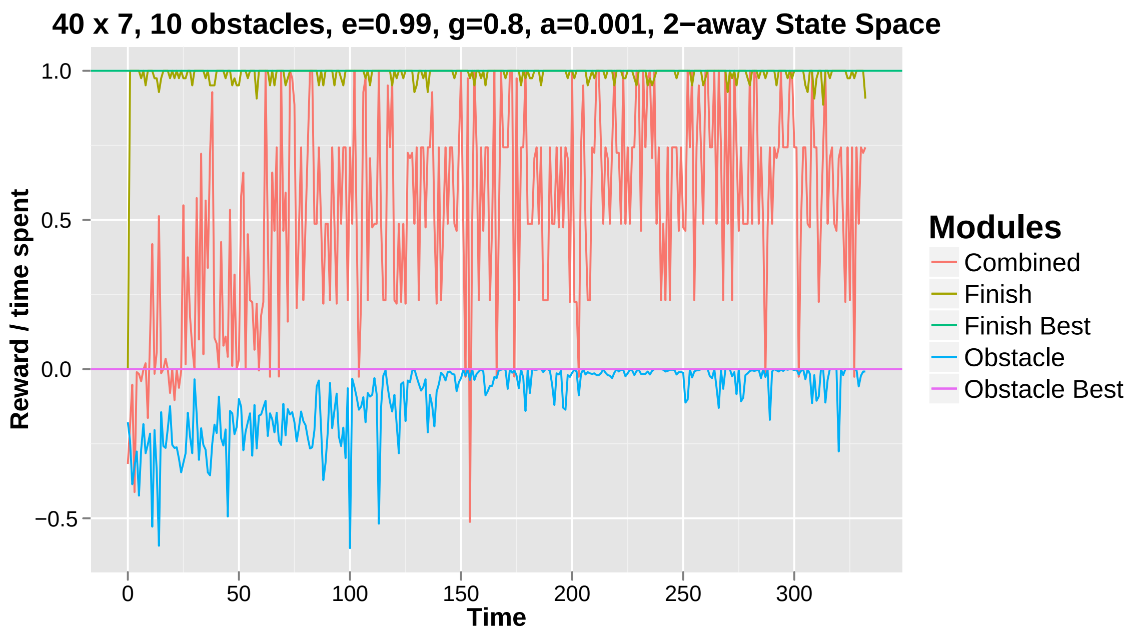


Figure 5: Two-away obstacle state space visibility

Certainly, it takes longer to train, but the result seems reassuring.

Here's another case, with much higher penalties; not a whole lot changes, and the reason why the Combined trend is so far underneath the Obstacle trend is because the Combined module doesn't account for time; if it was smart, if it accidentally hit more than one obstacle, it would hover around the end until the time was almost up, and then finish. But it's not smart (it doesn't know about the *reward*/*time* metric), so it finishes as quickly as possible.

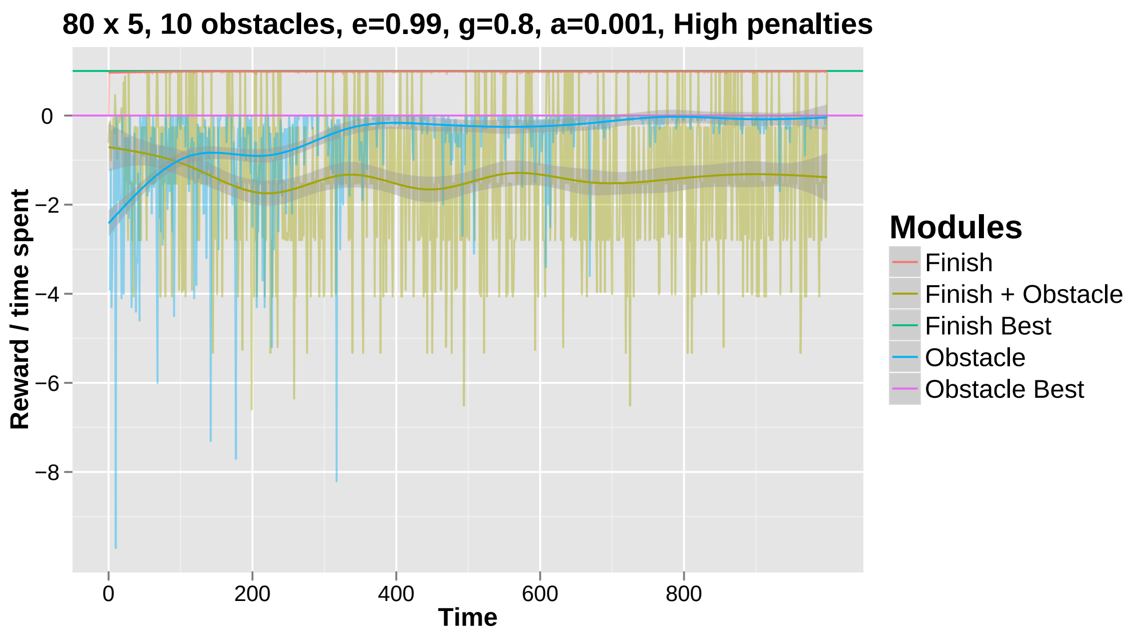


Figure 6: High penalties of -100

If I set the penalty lower, the results suddenly *look* a lot better:

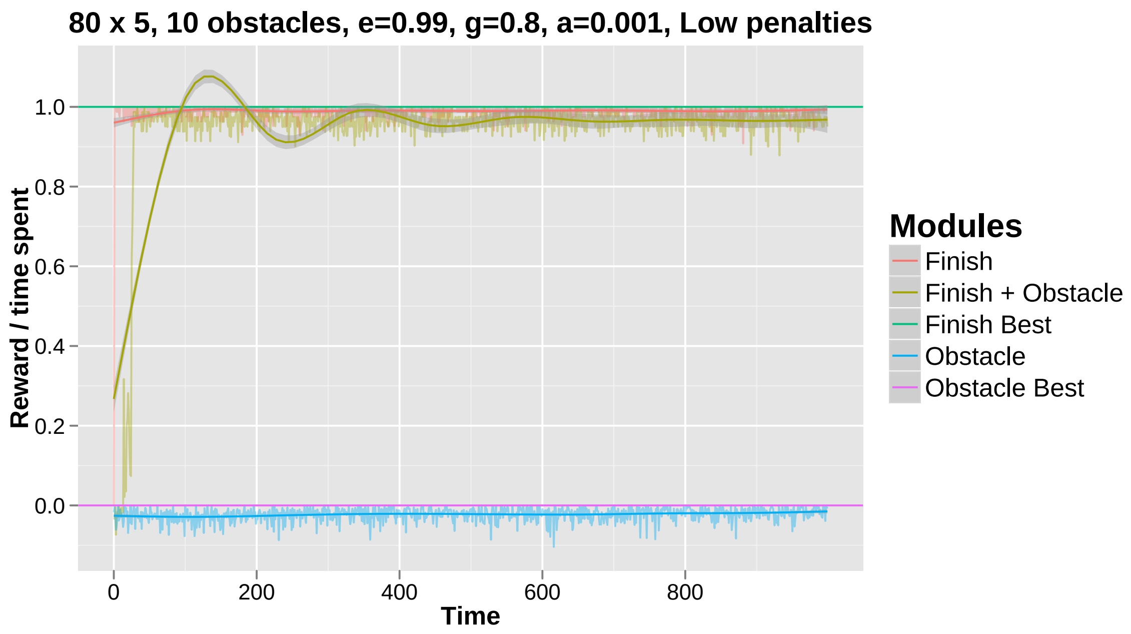


Figure 7: Low penalties of -1

2 Pitfalls

There were a couple of hacks that I had to build into my algorithm:

a. Often in the very first runs of the `finish` module, the agent would jitter left and right while running the width of the sidewalk several times, not making any forward progress. As far as I can tell, this was due to tiny differences between the score for going left as opposed to right, in the range of < 0.0001. Of course, sorting will register tiny differences just as significant as more reasonable (0.5) differences, so my hack was to pick a random action if the advantage between choosing the best and the second best actions was < 0.001.

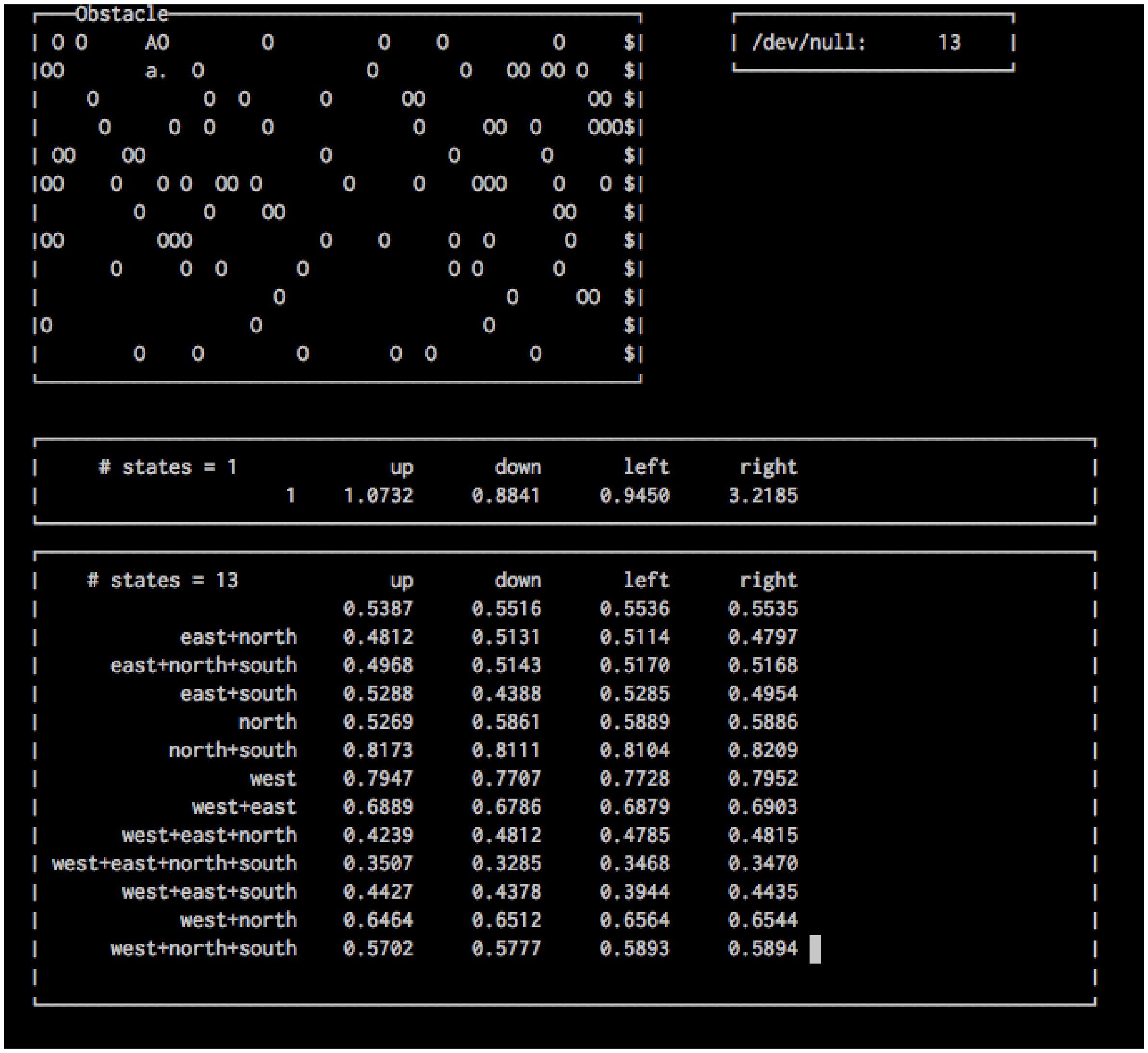
   I could have accomplished the same thing by adding a very amount of noise, e.g. $\mathcal{N}(0, 0.001)$, to the Q function, but the threshold solution was easier and faster.

b. Somewhat relatedly, when the `obstacle` module was running, the agent would sometimes hug the starting edge of the sidewalk. Because there was no penalty for running in that direction, and there could be a penalty for hitting an obstacle, that's a reasonable strategy. I tried handing out small penalties for running off the edges, but this didn't noticeably help performanace overall, and this bad habit was rare, so I didn't include these penalties in the analyses above.

3 Code

I coded the homework in Python. Python is awesome because it lets you have multiple functions, each able to call the others, in *one* file! This feature, unfortunately, is not shared by all languages used in this class.

The code is nothing clever. However, there is a fun curses-powered visualization; simply run the script with 'draw' as one of the arguments: `python qlearn.py draw`. The program is much faster without this viz functionality, though. Also, you can easily configure the environment parameters via a filename: `python qlearn.py s20x5-o20-e0.9-a0.01-g0.8-out.csv` will create a 20-long by 5-wide sidewalk with 20 obstacles, an epsilon parameter of 0.9, etc. (same abbreviations as above). In the output, I have created a trail for the agent, 'a' and '.', which correspond to his previous two positions.



```
┌──Obstacle─────────────────────────────────────────┐     ┌────────────────────────┐
| O O     AO        O         O    O         O     $|     | /dev/null:      13     |
|OO      a.  O              O       O   OO OO O   $|     └────────────────────────┘
|    O         O  O      O      OO              OO $|
|     O     O  O    O            O     OO  O    OOO$|
| OO    OO               O          O       O      $|
|OO    O   O O  OO O       O     O    OOO    O   O $|
|        O     O    OO                       OO    $|
|OO        OOO           O    O     O  O      O    $|
|      O     O  O      O            O O      O     $|
|                    O                   O     OO  $|
|O                 O                   O           $|
|        O    O        O       O  O        O       $|
└───────────────────────────────────────────────────┘


┌───────────────────────────────────────────────────────────────────────────────────┐
|     # states = 1             up       down       left      right                  |
|                    1    1.0732     0.8841     0.9450     3.2185                  |
└───────────────────────────────────────────────────────────────────────────────────┘

┌───────────────────────────────────────────────────────────────────────────────────┐
|    # states = 13             up       down       left      right                  |
|                         0.5387     0.5516     0.5536     0.5535                  |
|           east+north    0.4812     0.5131     0.5114     0.4797                  |
|     east+north+south    0.4968     0.5143     0.5170     0.5168                  |
|           east+south    0.5288     0.4388     0.5285     0.4954                  |
|                north    0.5269     0.5861     0.5889     0.5886                  |
|          north+south    0.8173     0.8111     0.8104     0.8209                  |
|                 west    0.7947     0.7707     0.7728     0.7952                  |
|            west+east    0.6889     0.6786     0.6879     0.6903                  |
|      west+east+north    0.4239     0.4812     0.4785     0.4815                  |
| west+east+north+south    0.3507     0.3285     0.3468     0.3470                  |
|      west+east+south    0.4427     0.4378     0.3944     0.4435                  |
|           west+north    0.6464     0.6512     0.6564     0.6544                  |
|     west+north+south    0.5702     0.5777     0.5893     0.5894                  |
|                                                                                   |
└───────────────────────────────────────────────────────────────────────────────────┘
```

Figure 8: Screenshot of visualization.