“Indoor GPS” for Visually Impaired Individuals

CS282 Final Project Updates

April 21, 2015

Students: Xide Xia, Toby Du

Instructors: Finale Doshi-Velez

Ken Arnold, Allen Schmaltz

Part I. Environment

We will build an environment includes two grid worlds in total based on the Gridworld.py in cs282rl package in order to imitate a two-floor real world building. Besides the agent, there will be several other persons walking around in each grid world. In this project, we will try both online methods and offline methods. We build a mallworld for a visually impaired person to find the destination. Inside this mall, there are walls, other people, an entrance and a destination. The person needs to walk from entrance and go the destination without hitting other people and wall. For example, the following is an example of one floor of a Mallworld.

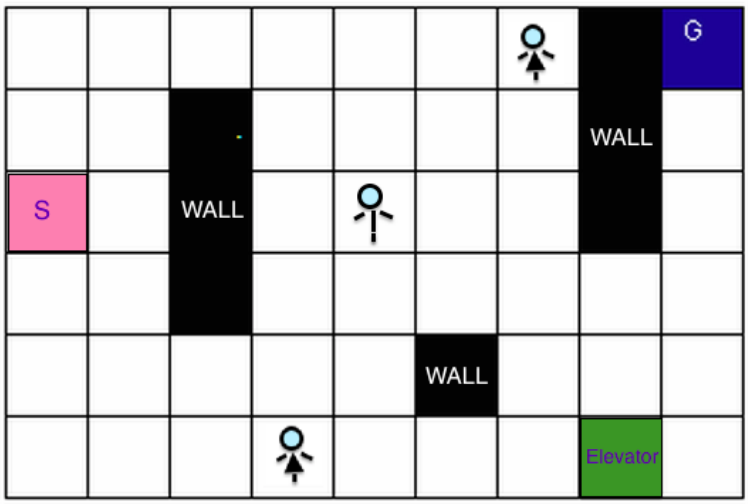

Figure 1. Grid world for a single floor

To implement this mall, we use the gridworld data structure from package cs282rl. However, there are many changes.

1. Mall world

I use 3 dimensional array topology to present our maze. In gridworld, we just have one level of gridworld. Our new grid will have two levels so we use a 3 dimensional array topology to form the maze. Below is an example of the 2 by 8 by 8 maze topology

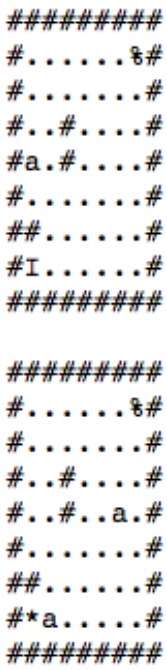

Figure 2. Visualization of a two- floor MallWorld

In the maze, '#' means wall; 'a' means people; '.' means open space; '*' means destination; 'I' means the person to be guided; '%' means stairs. The upper one is the first floor and the below one is the second floor in the MallWorld.

2. Agent and People

Agent will perform some move in the mall so as to get to the destination.

In gridworld, agent's moves are not restricted by people. They are just restricted by wall. Nevertheless, in our mall world, we need to take the position of people into account when agent tries to perform some moves. If the agent crushes into any people, we will punish it.

In addition to the move of agent, people will also move. Now we have two policy of the movement of people. First, they can randomly walk in the mall. Secondly,

they might walk straight most of the time. The first policy is easy to implement but the second policy is much more reasonable for real life. We implement both of the policies and we will see what is the difference between them.

3. Stairs

To make it more real, we also add stairs into the mall. The person can use stairs to go to another floor and find the destination in that floor. As we mentioned above, we use '%' to represent a stair. When agent goes into this position, it will automatically be transported to the stair on the other level.

4. Rewards

In order to define a reinforcement learning problem, now we need to quantify the reward or punishment for each action and state. However, I think these are important variables to tune if we want to use mallworld to simulate a real problem. As a result, we try different value of reward and punishment and we will talk about it later. Now the initial rewards/punishment are {"hit wall": -2, "move":-1, "goal":200, "run into people":-30, "stay": -1}, and we will explore more values later.

5. State

Intuitively, the state spaces are the combination of the position of agent and all positions of people, but the spaces will increase exponentially as the number of people increases. If the spaces are too large, we don't have enough memory to store every situation. To simplify, we will just consider the people's position that is close enough. For example, if the people are in the other floor or 10 meters away from the agent, we don't need to know the precise positions of those people. In the algorithm we are running now, to narrow down the space, we use a 5 by 5 window and put the agent in the center of this window and see whether people are inside this window. After this modification, we can end up with 2*8*8*25^{number of people} and now we have 3 people in our mallworld. Totally, there are 2535462 states.

Part II. Method

Model-Free Approaches:

1. SARSA

In this project, we built a SARSAAgent class for this algorithm, its inputs included num_states, num_actions, epsilon, discount_factor, and Q_initial_value. When initializing, it built a value_table to record all the Q(s,a) values. It also had a reset function to handle the start of a trial. Beside the agent class, I wrote a run_trial function to run each trail, its inputs included sarsa_agent, rewards_by_iteration, rewards_by_episode, and learning_rate. In the run_trial function, it called the SARSAAgent class's interact() to get the next action for next state and upate the

Q value table as well. To get the next action, we used the epsilon-greedy method. In SARSA, since it was on-policy algorithm, it learnt action values relative to the policy it follows.

2. Q Learning

The abstraction of Q-Learning was similar to SARSA's except the part of updating the Q value table. Since it is off-policy algorithm, Q Learning agent learnt action values relative to the greedy policy. (Notice: we use the same run trial function for both SARSA and Q-Learning agents, called run_trail_ss_or_ql.)

Model-Based Approaches:

3. RMAX

We also built a class for RMAX, the inputs included num_states, num_actions, discount_factor, and min_visit count. During initialization, it built a transition_obervation matrix that recorded, during the whole experiment, the visiting counts for each (s,a,s') triplet. The Reward matrix recorded R(s,a,s') and the mdp matrix presented Prob(s'|s,a). In this class, there was a update_MDP() function that updated the mdp matrix by computing the new Prob(s'|s,a) according to the transition_obervation matrix. And then, the value_iteration function computed the new policy according to the new mdp we got. In this class, I used the value iteration to get the new policy. The policy was updated every T=50 iterations. Besides the RMAXAgent class, there was also a run_trail function to run each trail using RMAX method.

Part III. Result

3.1 MallWorld A: People Walking Around Randomly

In this MallWorld A, people (except our agent) are walking around in randomly. In other word, they choose a direction randomly for each step.

Model-free: SARSA

In this part, we test our SARSA agent on the MallWorld A. We run 5 trials for both 40000 iterations of experience and 600 episodes. For the parameters, we set them as:
learning rate = 0.1
epsilon = 0.1
discount_factor = 0.95

And the results are as following:
a. plot of cumulative rewards respect to iterations
b. plot of total rewards per episodes

c. plot of totals times of hitting people per episodes
d. plot of Q-value of the first floor
e. plot of Q-value of the second floor

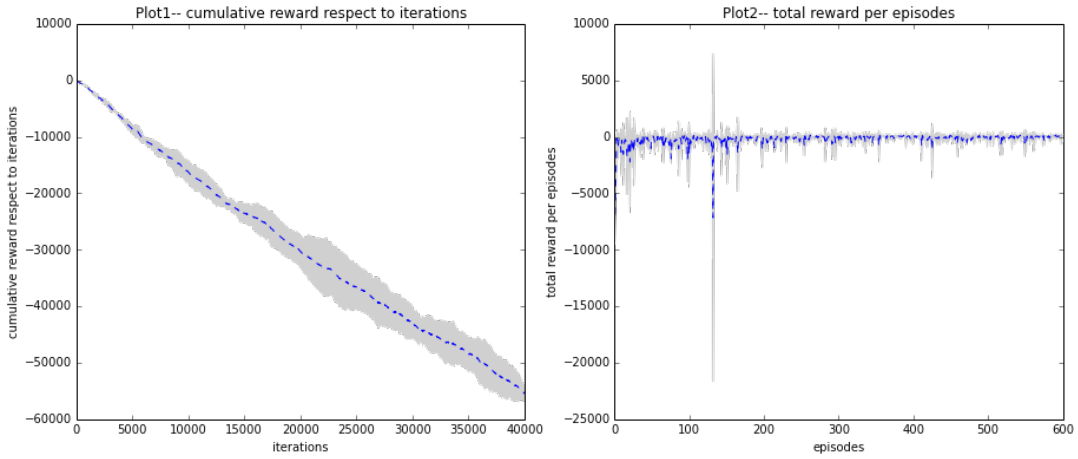

Figure 3: Left:plot of cumulative rewards respect to iterations, Right:plot of total rewards per episodes

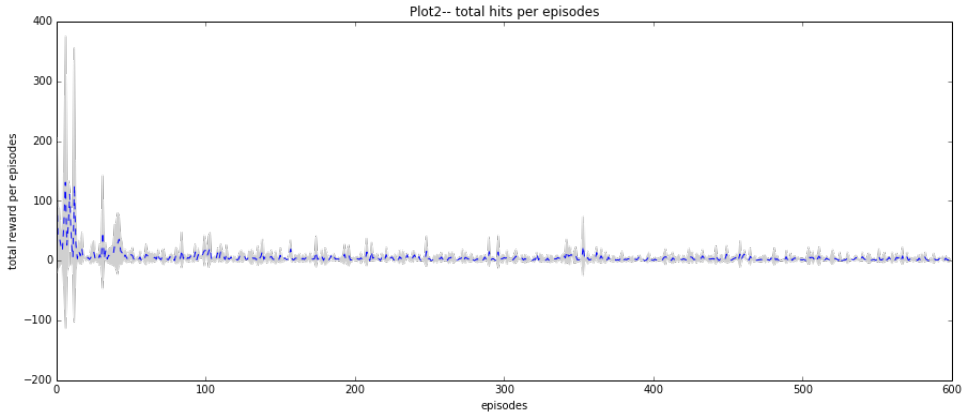

Figure 4: plot of totals times of hitting people per episodes

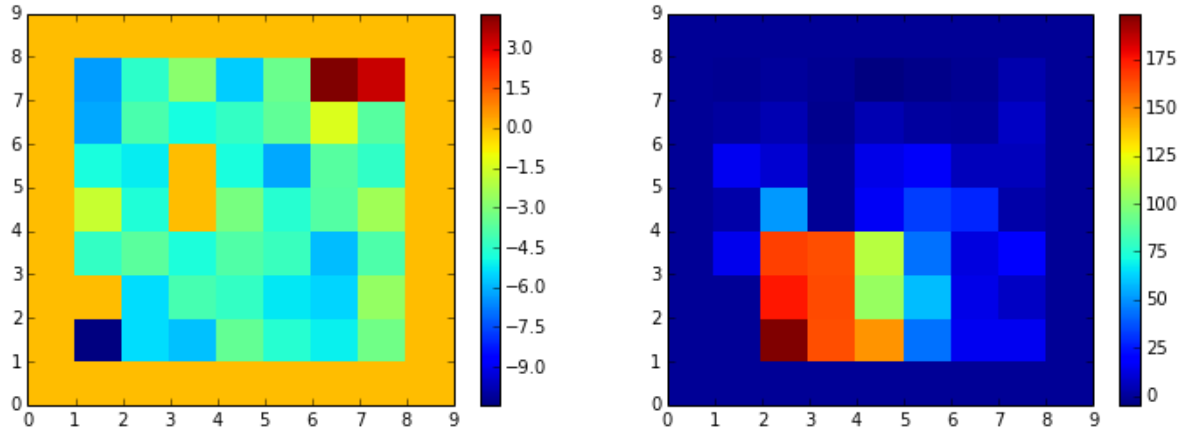

Figure 5: Left: plot of Q-value of the first floor, Right: plot of Q-value of the second floor

Model-based: RMAX

In this part, we implemented our RMAX agent on the MallWorld A. However, it turned out the MoemoryError as following. We think the reason of this kind of error is that, as a model-based reinforcement learning approach, RMAX method asks for a transition_observations matrix with size of num_states+1 * num_actions* num_states+1. However, in the MallWorld, there are about 2.5 million states and 5 actions in total. Thus, it requires a table with size = (5 * 2.5 million^2) which is hard to realize because of the memory limitation.

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-115-1d04a0dc20b9> in <module>()
----> 1 agent_rmax = RMAXAgent(2535462, num_actions=task.num_actions, discount_factor = 0.95, min_visit=5)

<ipython-input-113-c8a101d98e8d> in __init__(self, num_states, num_actions, discount_factor, min_visit)
      4         self.num_actions = num_actions
      5         self.discount_factor = discount_factor
----> 6         self.transition_observations = np.zeros((num_states+1, num_actions, num_states+1))
      7         self.min_visit = min_visit
      8         self.update_flag = 0

MemoryError:
```

Figure 6: Memory Error

3.2 MallWorld B: People Walking Along their way

In this MallWorld B, people (except our agent) are walking in their original direction with 20% chance that they may turn left, right, around. In other word, they keep the original action with 80% probability and randomly choose another action with 20% probability. When hitting a wall, they turn around.

Model-free: SARSA

In this part, we test our SARSA agent on the MallWorld B with same configuation: We run 5 trials for both 40000iterations of experience and 600 episodes. For the parameters, we set them as:
learning rate = 0.1
epsilon = 0.1
discount_factor = 0.95

And the results are as following:
a. plot of cumulative rewards respect to iterations
b. plot of total rewards per episodes
c. plot of totals times of hitting people per episodes
d.plot of Q-value of the first floor
e.plot of Q-value of the second floor

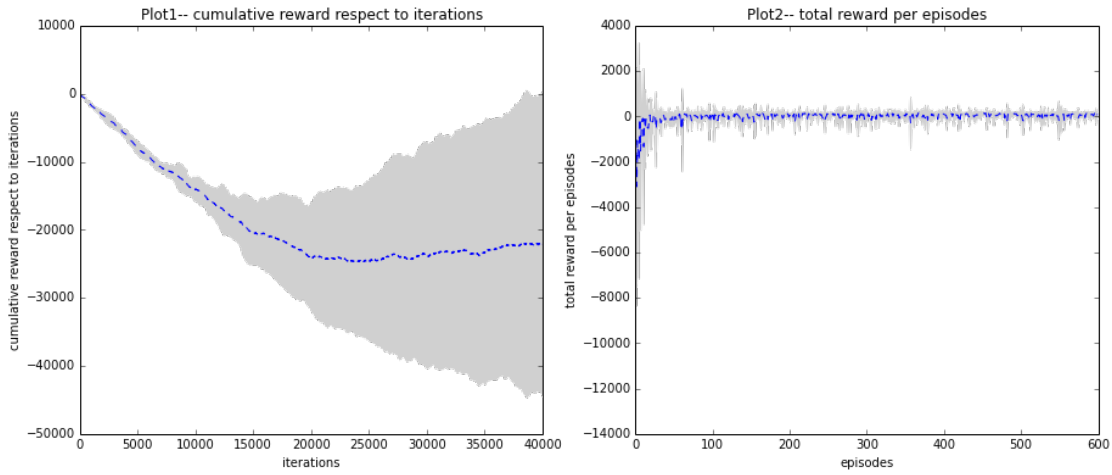

Figure 7: Left:plot of cumulative rewards respect to iterations, Right:plot of total rewards per episodes

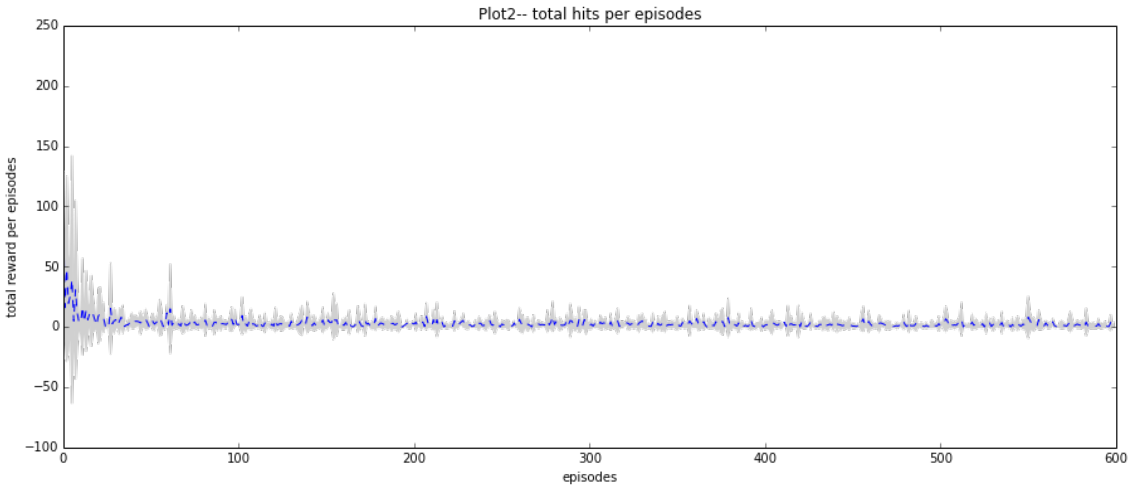

Figure 8: plot of totals times of hitting people per episodes

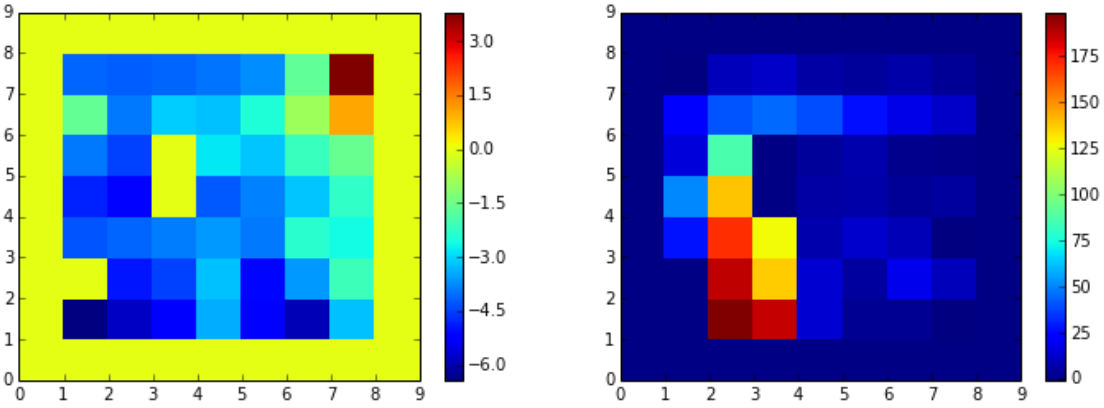

Figure 9: Left: plot of Q-value of the first floor, Right: plot of Q-value of the second floor

Part IV. To-Do List

1. Try more parameters sets {gamma, alpha, epsilon} and compare the results.
2. Try more model-free method such as Q-learning on both Mallworld A and Mallworld B and compare the results with SARSA.
3. Add Option method such as “go to the stair”
4. Build a more complicated Mallword, such as add more floors or some other elements. And then test reinforcement learning approaches on it.

Reference

1. Sutton R S, Barto A G. Introduction to reinforcement learning[M]. MIT Press.

2. Brafman R I, Tennenholtz M. R-max-a general polynomial time algorithm for near-optimal reinforcement learning[J]. The Journal of Machine Learning Research, 2003, 3: 213-231.

3. Asmuth J, Li L, Littman M L, et al. A Bayesian sampling approach to exploration in reinforcement learning[C]//Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence. AUAI Press, 2009: 19-26.

4. Konidaris G, Barreto A S. Skill discovery in continuous reinforcement learning domains using skill chaining[C]//Advances in Neural Information Processing Systems. 2009: 1015-1023.

5. Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.