AI Exercise

FAN YI

2461248Y

1. Introduction:

The motivation of this AI exercise is to design, implement, evaluate and document three virtual agents with different types (a senseless/random agent, a simple agent and a reinforcement agent) which are potentially able to reach a goal in a custom Open AI Gym environment derived from Frozen especially for our 2019-2020 version of the AI course (as *lochlomond_demo.py* defined).

2. Senseless/Random agent

2.1 PEAS analysis:

Performance measure: Arrive at the goal without falling into the hole.
Environment: A virtual simulation environment in a 4x4 or 4x4 layout with four possible states; holes (H), free (F) and a single final goal (reward +1). It is possible to take four virtual actions (up, down, left, right).
Actuators: Virtual discrete commands (up, down, left, right) leading the virtual agent to change position.
Sensors: None, without sensory input.

It is highly stochastic: Since the ice is so slippery that you sometimes do not end up in your desired end state.
Semi-static environment: Since I placed a limit on the number of steps, and the agent is aware of this so time matters and it would objectively change the optimal strategy.
Discrete: In terms of most aspects; states and actions.
Single-agent: It is not competing against anyone.
Partially observable: The agent does not know the full map or what's around it.

2.2 Theory:

Since the agent does not have a sensor to obtain the sensory input from the environment and with no prior knowledge, the actuator just takes random actions to the environment, and end the episode if it falls into a hole or arrive at the goal.

```
action = env.action_space.sample()
observation, reward, done, info = env.step(action)
if(done): ……
```

2.3 Implementation:

1. recieve the argv[] from the terminal command(problem_id, map_name_base);

```
problem_id = int(sys.argv[1])
map_name_base = sys.argv[2]
```

2. initialize the environment with suitable parameters;

env = LochLomondEnv(problem_id=problem_id, is_stochastic=True, map_name_base=map_name_base, reward_hole=0.0)

3. In each episode, take random actions for each state (maximum *max_iter_per_episode* steps per episode);
4. If done (reach the hole/goal, has taken maximum *max_iter_per_episode* steps), end this episode;
5. repeat the *max_episodes* many episodes;
6. record the *success rate, average reward through all episodes, average reward per 100 episodes, steps need to reach the goal, average steps need to reach the goal* of the agent into the *raw_data.xlsx.*

3. Simple agent

3.1 PEAS analysis:

Performance measure: Arrive at the goal without falling into the hole.
Environment: A virtual simulation environment in a 4x4 or 8x8 layout with four possible states; holes (H), free (F) and a single final goal (reward +1). It is possible to take four virtual actions (up, down, left, right).
Actuators: Virtual discrete commands (up, down, left, right) leading the virtual agent to change position and noise free.
Sensors: Oracle, know everything of the environment.

Deterministic: We always end up in desired end state.
Static environment: We have infinite steps until the agent terminates, then we consider it static as it does not change while we are deciding which action(s) to take.
Discrete: In terms of most aspects; states and actions.
Single-agent: It is not competing against anyone.
Fully observable: The agent know the full map.

3.2 Theory

A* (A-star): $f(n)=g(n)+h(n)$
Strategy: combine the cumulative action cost / path cost with the (heuristic) cost/distance of getting from the node to the goal.

Be optimistic - do not overestimate the distance/cost from node to goal:
$h(n) \leq c*(n, n*, a)$
where $c*(n, n', a)$ is the minimum cost of getting from n to n'.

Optimal for tree search (without the possibility for recurring states).
It is efficient since no other optimal algorithm expands fewer nodes than A*

3.3 Implementation

1. recieve the argv[] from the terminal command(problem_id, map_name_base);
problem_id = int(sys.argv[1])
map_name_base = sys.argv[2]
2. initialize the environment with suitable parameters;
env = LochLomondEnv(problem_id=problem_id, is_stochastic=False, map_name_base=map_name_base, reward_hole=0.0)
3. visualize the problem/environment
print(env.desc)
4. extract the state space from the environment *(env2statespace(env))* and define the frozen_lake Graph problem *(using UndirectedGraph, GraphProblem)*
5. find the solution using Astar algorithm *(my_astar_search_graph)* and trace the solution
6. record the *steps number* of Astar search into the *raw_data.xlsx*, and record the *success rate, average reward through all episodes, average reward per 100 episodes* to 1, since this agent will always arrive at the goal with the same number of steps.

4. RL agent

4.1 PEAS analysis:

Performance measure: Obtain the maximum reward (arrive at the goal) without falling into the hole.
Environment: A virtual simulation environment in a 4x4 or 8x8 layout with four possible states; holes (H), free (F) and a single final goal (reward +1). It is possible to take four virtual actions (up, down, left, right).
Actuators: Virtual discrete commands (up, down, left, right) leading the virtual agent to change position.
Sensors: Percept perfect information about the current state and thus available actions in that state; no prior knowledge about the state-space in general.

It is highly stochastic: Since you only end up in your desired end state with a certain probability.
Semi-static environment: Since I placed a limit on the number of steps, and the agent is aware of this so time matters and it would objectively change the optimal strategy.
Discrete: In terms of most aspects; states and actions.
Single-agent: It is not competing against anyone.
Partially observable: The agent knows the information about the current state and thus available actions in that state; no prior knowledge about the state-space in general.

4.2 Theory:

Q learning using action-utility representation instead of learning utilities, aim directly at mapping states to actions without the transition model.

$U(s) = max_a Q(s, a)$

The $Q(s,a)$ value represent the long term value of taking action a in state s (we store it as a numpy array).

And we use the observed ***transitions*** to adjust the Q(s,a)-values of the observed states so that they agree with the Bellman equations.

$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma max_{a'} Q(s', a') - Q(s,a))$
α: *learning rate*
γ: *discount factor*

The formal algorithm is as follows:

1. Initialise Q for all states and actions.
2. For *max_episodes* episodes, follow steps from 3–9:
3. Initialise state, S
4. For each step of the episode follow steps from 5–8:
5. Choose an action A, from state S, with some policy derived from Q (using greedy to balance exploration/exploitation).
6. Now take action, A and get new state S’ and reward R.
7. Update the Q value for S and A using the above equation.
8. Set S as the current state (S = S’).
9. Terminate if the state S is the terminal.

4.3 Implementation:

1. recieve the argv[] from the terminal command(problem_id, map_name_base);
problem_id = int(sys.argv[1])
map_name_base = sys.argv[2]
2. initialize the environment with suitable parameters;
env = LochLomondEnv(problem_id=problem_id, is_stochastic=True, map_name_base=map_name_base, reward_hole=-0.4)
3. Initialize Q-table with states many rows and actions many columns:
Q = np.zeros([env.observation_space.n,env.action_space.n])
4. In each episodes, choose an action a in the current world state (s) with Greedy:
*action = np.argmax(Q[state,:] + np.random.randn(1, env.action_space.n) * (1./(i+1)))*
(At the beginning of the learning, with less reliable experience, we prefer to explore the world, then with the episode i increase, the agent rely more and more on exploitation (what we know by repeating the episodes) rather than on exploration to find the goal.)
Get new state and reward from environment:
state1, reward, done, _ = env.step(action)
Update Q-Table with new knowledge:
Q[state,action] = Q[state,action] + lr(reward + y*np.max(Q[state1,:]) - Q[state,action])*

5. repeat the *max_episodes* many episodes;
6. get the final after training Q-table
7. choose the action according to the final Q-table(action of state = max(Q-value) and plot it with arrows
8. record the *success rate, average reward through all episodes, average reward per 100 episodes, steps need to reach the goal, average steps need to reach the goal* of the agent into the *raw_data.xlsx.*

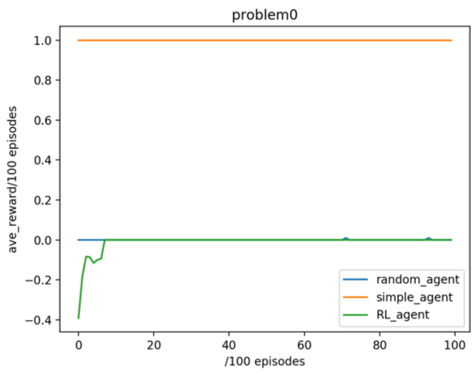
5. Evaluation:

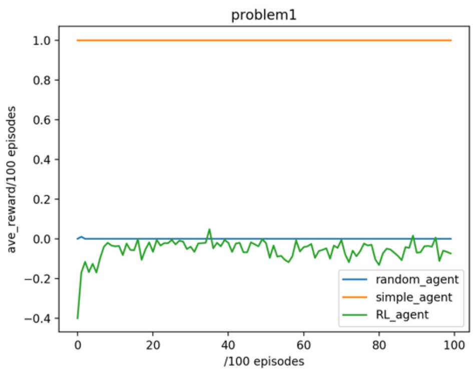
When executing the code of the three agents above, I record the relevant metrics into the *raw_data.xlsx* file, which is initialized by *Excel.py.* So in *run_eval.py* script, I read the relevant data from the *raw_data.xlsx* file and write it into a new *results_summarising.xlsx* file as well as plot the required figure according to the data in *results_summarising.xlsx* file.

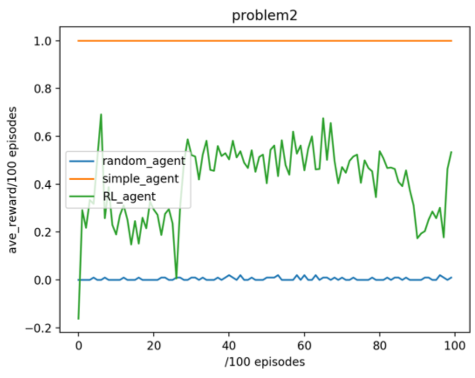
The results summarizing over the 8 instances, learning behavior and convergence of the 3 agents are as follows:

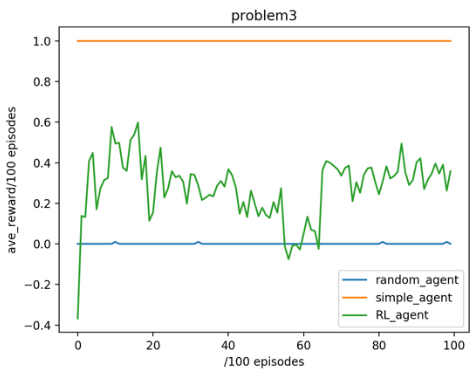
- 8x8-base:

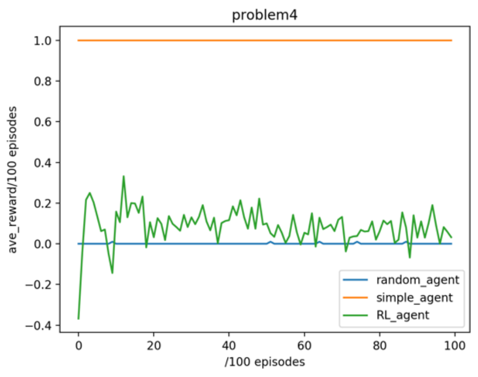
Compare the learning behavior of each problem among three agents:

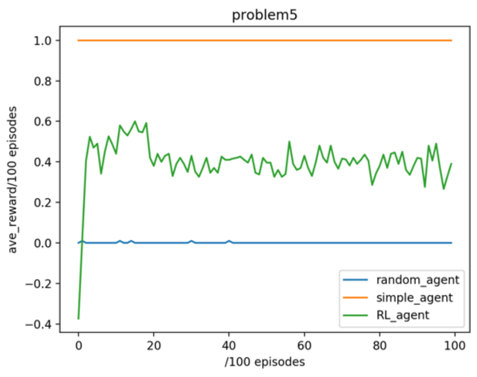


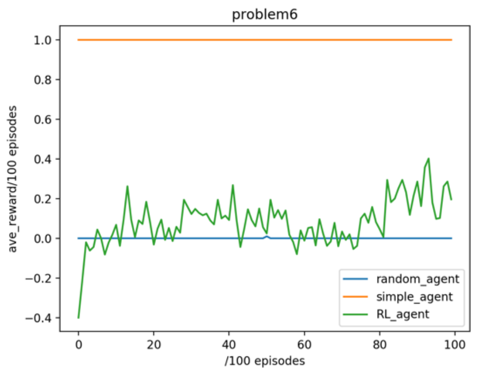


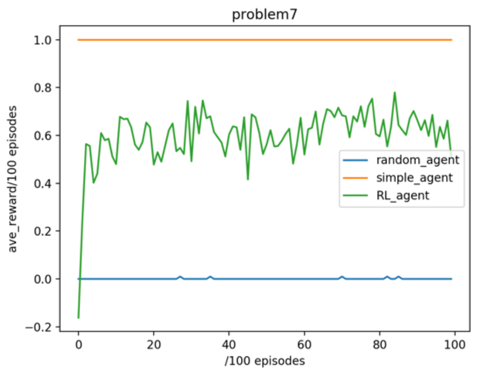




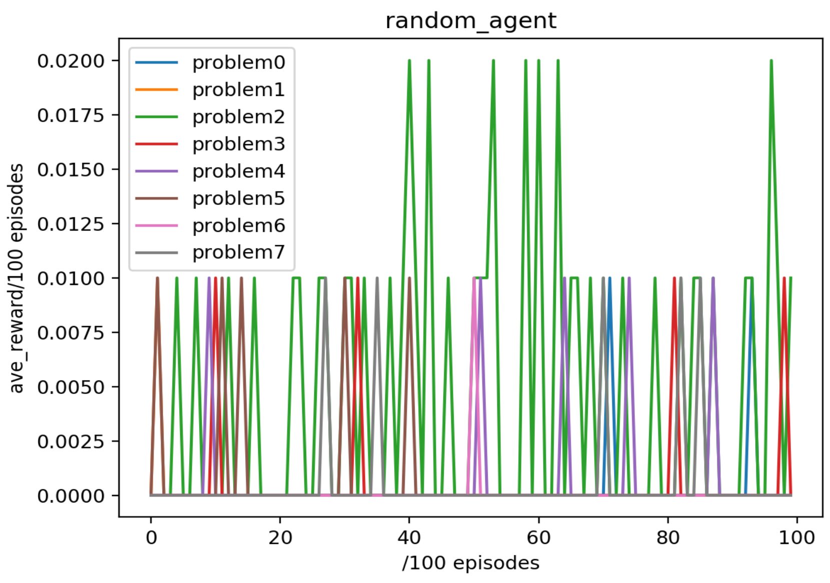




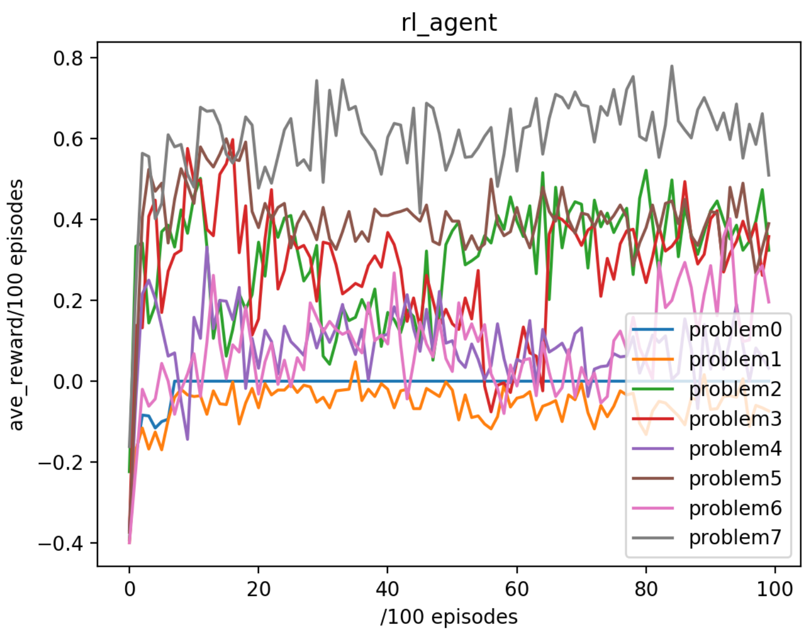




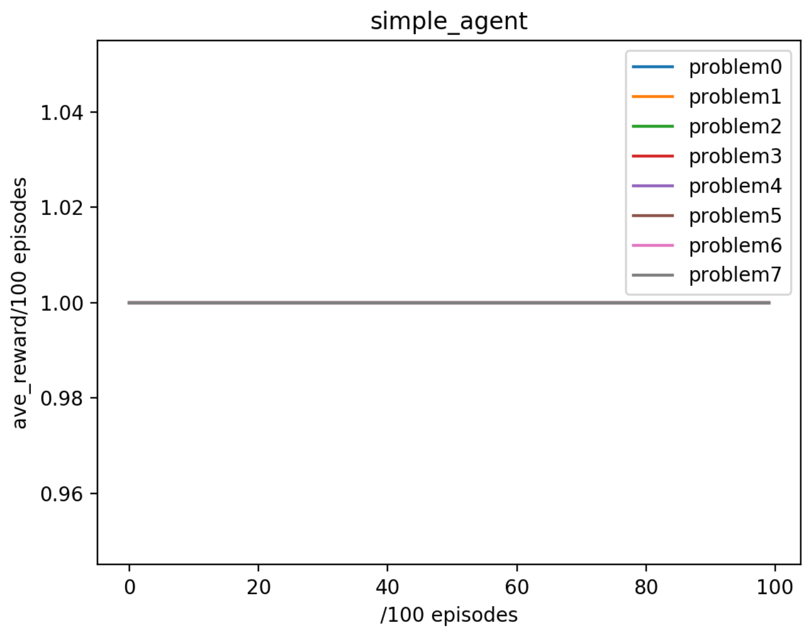
The average reward of random agent among 8 problems:



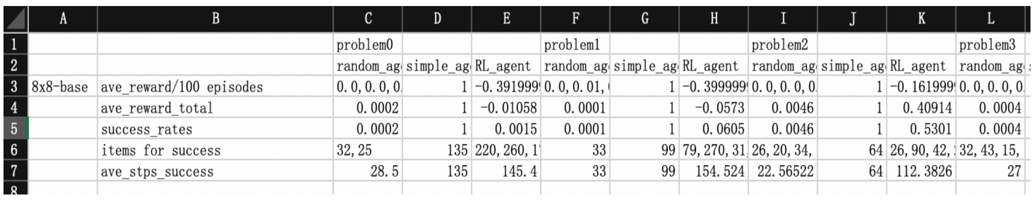
The average reward of RL agent among 8 problems:



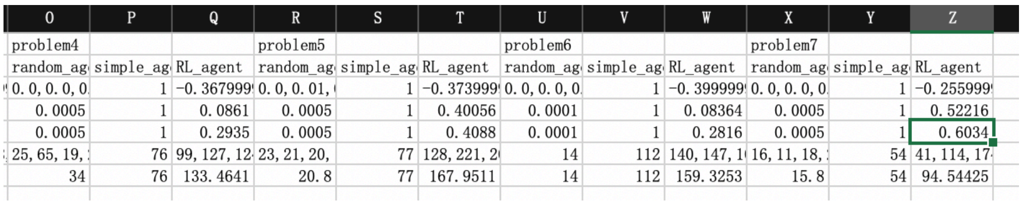
The average reward of simple agent among 8 problems:



The summarizing results over the 8 instances of the problem:



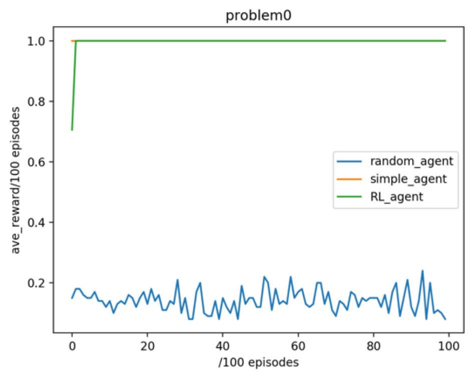
	A	B	C	D	E	F	G	H	I	J	K	L
1			problem0			problem1			problem2			problem3
2			random_ag	simple_ag	RL_agent	random_ag	simple_ag	RL_agent	random_ag	simple_ag	RL_agent	random_ag
3	8x8-base	ave_reward/100 episodes	0.0,0.0,0.	1	-0.391999	0.0,0.01,	1	-0.399999	0.0,0.0,0.	1	-0.161999	0.0,0.0,0.
4		ave_reward_total	0.0002	1	-0.01058	0.0001	1	-0.0573	0.0046	1	0.40914	0.0004
5		success_rates	0.0002	1	0.0015	0.0001	1	0.0605	0.0046	1	0.5301	0.0004
6		items for success	32,25	135	220,260,1	33	99	79,270,31	26,20,34,	64	26,90,42,	32,43,15,
7		ave_stps_success	28.5	135	145.4	33	99	154.524	22.56522	64	112.3826	27
8												

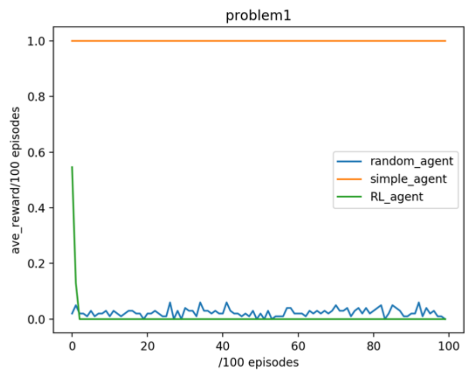


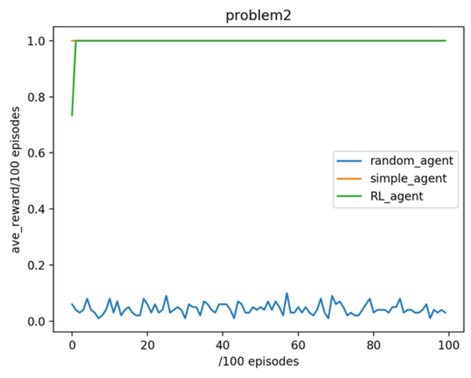
O	P	Q	R	S	T	U	V	W	X	Y	Z
problem4			problem5			problem6			problem7		
random_ag	simple_ag	RL_agent	random_ag	simple_ag	RL_agent	random_ag	simple_ag	RL_agent	random_ag	simple_ag	RL_agent
0.0,0.0,0.	1	-0.367999	0.0,0.01,	1	-0.373999	0.0,0.0,0.	1	-0.399999	0.0,0.0,0.	1	-0.255999
0.0005	1	0.0861	0.0005	1	0.40056	0.0001	1	0.08364	0.0005	1	0.52216
0.0005	1	0.2935	0.0005	1	0.4088	0.0001	1	0.2816	0.0005	1	0.6034
25,65,19,	76	99,127,12	23,21,20,	77	128,221,2	14	112	140,147,1	16,11,18,	54	41,114,17
34	76	133.4641	20.8	77	167.9511	14	112	159.3253	15.8	54	94.54425

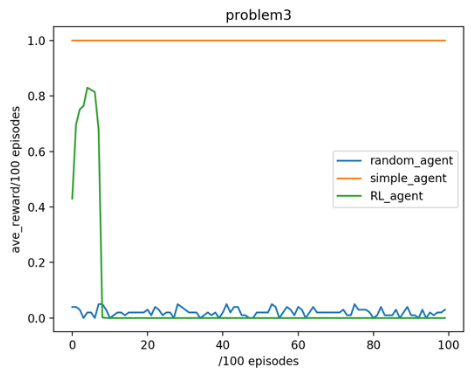
- 4x4-base:

Compare the learning behavior of each problem among three agents:

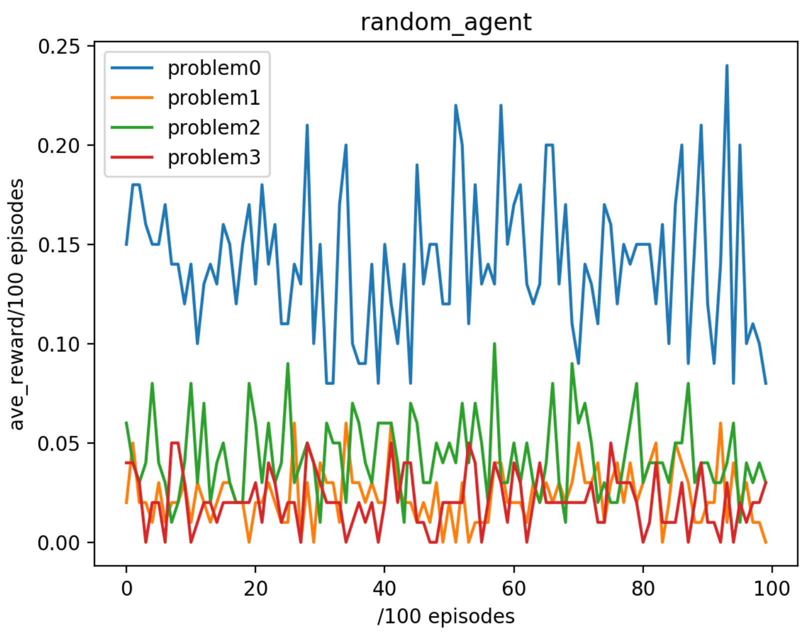




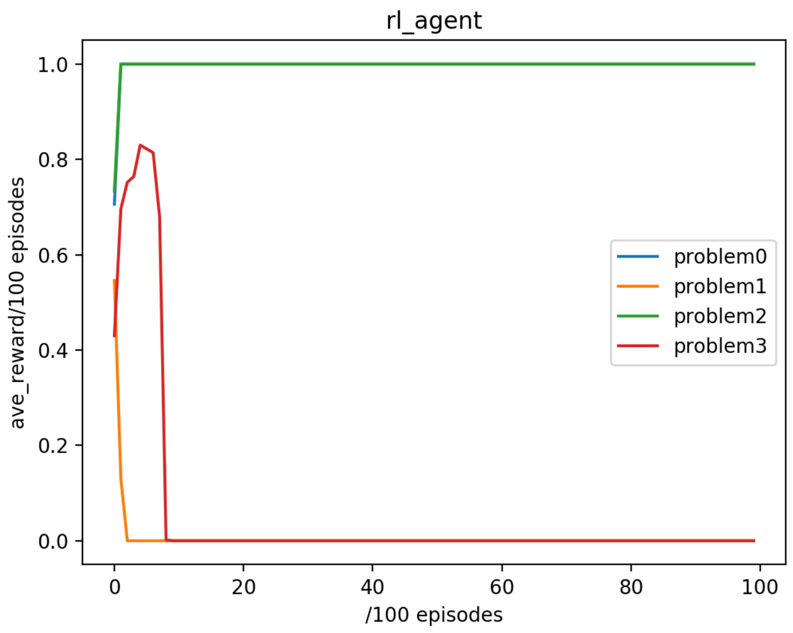




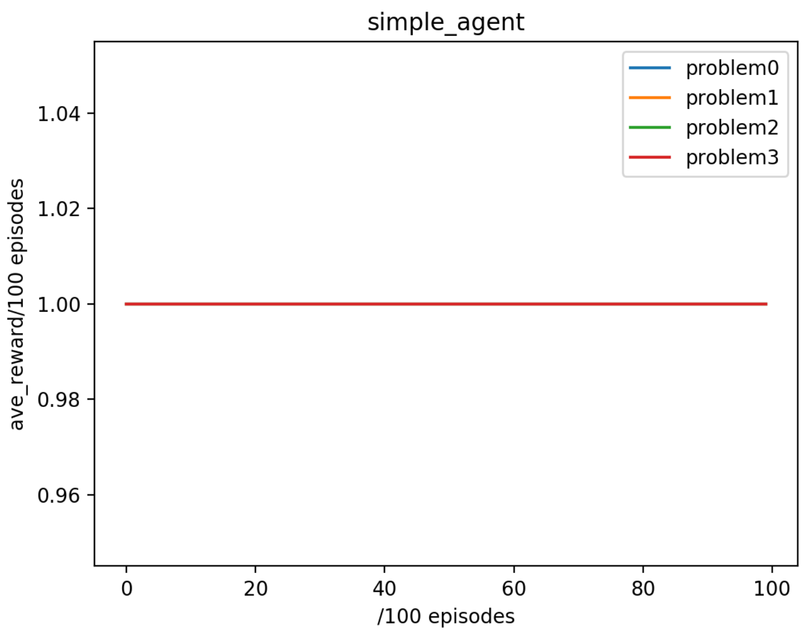
The average reward of random agent among 4 problems:



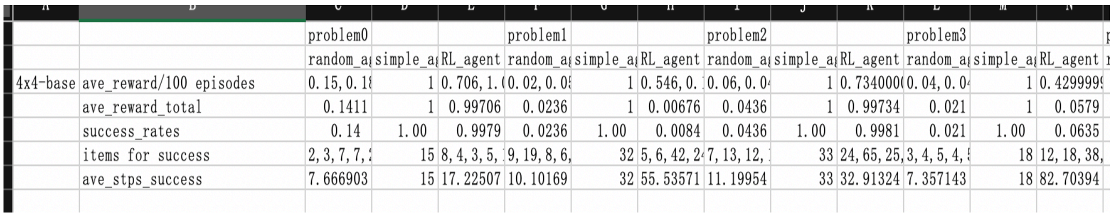
The average reward of RL agent among 4 problems:



The average reward of simple agent among 4 problems:



The summarizing results over the 8 instances of the problem:



		problem0			problem1			problem2			problem3		
		random_a	simple_a	RL_agent	random_a	simple_a	RL_agent	random_a	simple_a	RL_agent	random_a	simple_a	RL_agent
4x4-base	ave_reward/100 episodes	0. 15, 0. 1	1	0. 706, 1.	0. 02, 0. 0	1	0. 546, 0.	0. 06, 0. 0	1	0. 734000	0. 04, 0. 0	1	0. 429999
	ave_reward_total	0. 1411	1	0. 99706	0. 0236	1	0. 00676	0. 0436	1	0. 99734	0. 021	1	0. 0579
	success_rates	0. 14	1. 00	0. 9979	0. 0236	1. 00	0. 0084	0. 0436	1. 00	0. 9981	0. 021	1. 00	0. 0635
	items for success	2, 3, 7, 7,	15	8, 4, 3, 5,	9, 19, 8, 6,	32	5, 6, 42, 2	7, 13, 12,	33	24, 65, 25,	3, 4, 5, 4,	18	12, 18, 38,
	ave_stps_success	7. 666903	15	17. 22507	10. 10169	32	55. 53571	11. 19954	33	32. 91324	7. 357143	18	82. 70394

As for the 8x8-base as well as 4x4-base, we can see that the average reward of RL agent is convergent, while that of the random and simple agent are not convergent:
The average reward of RL agent will fluctuate within a certain range after a short-term study; Normally it needs 1000 iterations to learn before convergence.
The convergence of the RL agent results due to the balance between exploration and exploitation.
The average reward of random agent is always a random;
The average reward of simple agent is always a constant 1.

As for the simple agent, since it uses A* to search the goal each time, it will always arrive at the goal with a certain number of steps no matter how many times you repeat, therefore the success rate and average reward of simple agent is constant 1.
But since the simple agent arrive at the goal by searching the state one by one using A* algorithm, the average steps it need for success are more than that of random success, but normally less than that of RL agent.

With the balance of exploration and exploitation as well as the experience accumulated, the RL agent reaches the goal with a significantly high probability than the random agent, even for difficult problems, like problem 0, but with more average steps needed to succeed.

Normally this RL agent is suitable for the whole problem instances, but it is obvious that the performance of RL agent for some difficult problems, for example, problem 0, are worse than that of easy ones.
(The problem 0 has the lowest successful rate for RL agent as well as random agent, and it takes much more steps than other problems for the simple agent to find the solution.)
Therefore, to some extent, we can say that the harder the problem is, the more steps are needed for A* to search for the solution, the worse the RL agent learning performance is.

Besides, the RL agent is not robust because the Q-table will change a little between each experiment (some states have similar Q-value among different actions, so the Action $_{state}$=Max(Q-value) is not totally same for different experiment), the single best policy of each experiment will change slightly.

By comparison between the results of the simple agent and RL agent, most of the state in RL agent belonging to the simple agent optimal way could find the optimal actions.

Since the random agent does not have the sensor to percept the information of the environment, the actuator just takes random action to change the position, therefore, it totally arrives at the goal by ‘luck’ (with an extremely small probability).

6. Conclusion

According to the comparison among the three agents about all of the problems, we know that the simple agent using A* search will always find the optimal way from the start state to the goal. The random agent without sensor hardly reaches the goal by luck. And the RL agent could arrive at the goal with obviously high probability after a short-term training.

Besides, difficult problems prefer to need more steps for A* to find the solution, and the learning performance of RL agent for difficult problems is worse than that of easy ones.