CS 4180/5180: Reinforcement Learning and Sequential Decision Making (Fall 2020)– Lawson Wong

Name: [Virender Singh]
Collaborators: [Sunny Shukla]

To run the codes please use main.py in the src folder. All the plots can also be seen in the plt folder.

Question 1.

Response:

(1a) No, the multi-step bootstrapping method won't be able to perform as well as the Dyna method. Because Dyna has following advantages:
i. It updates a set of values instead of just one value like without planning case.
ii. It exploits both the real and simulation experience for learning.
With n-step bootstrapping method though, first advantage is also provided by any multi-step bootstrapping method. But DynaQ has an edge in the second advantage because multi-step methods do not exploit this property.

(1b)
Advantages of using n-step returns instead of one-step returns in tabular Dyna-Q:
It will be able to look at all the previously visited states and compute the optimal q values for each state-action pair.
Disadvantages of using n-step returns would be :
Speed and efficiency of the algorithm to converge to optimal q values. As compared to one step it will now take into account the future n steps. Also, it might happen in nstep update process that it is unable to find one of the next n states in the model and this will lead to not optimal changes in the q values.

Question 2.

Response:

(2a) Dyna-Q+ tends to execute two kinds of actions:
(i) actions which has high values according to previous experience
(ii) actions that haven't been chosen for a long time.
For this reason, Dyna-Q+ outperforms Dyna in both phases. In the first phase, initially, DynaQ simply randomly choose actions, but Dyna-Q+ tend to try new actions or the effective old actions. This makes Dyna-Q+ learn the values more effectively. In the second phase, Dyna-Q+ is more exploratory, and therefore performs better than Dyna.

(2b) Let's assume the environment is fixed, when the algorithm learns the environment reasonably well, it will keep trying exploratory actions as in Dyna-Q+ is not helpful anymore, or even wasteful. Since both algorithm converge to the optimal policy, the difference of cumulative reward diminishes.

Question 3.

Response:

(3a) DynaQ+ pseudocode:

- Initialize Q(s,a) and Model(s,a) for all s∈S, and a∈A(s)
- Loop forever:
  - S ← current state
  - A ← ϵ -greedy(S,Q)
  - Take action A, observe resultant reward R, and state, S'
  - Q(S,A) ← $Q(S,A) + \alpha \times$ (R + $\gamma max_a Q(S',a) - Q(S,A)$)
  - Model(S,A) ← R,S',t
  - Loop repeat n times
    - S ← random previously observed state
    - A ← random action taken previously at S
    - R, S, t' ← Model(S,a)
    - $\tau \leftarrow t - t'$
    - update time of (s,a) pair to be t
    - R ← $R + \kappa\sqrt{\tau}$
    - Q(S,A) ← $Q(S,A) + \alpha \times$ (R + $\gamma max_a Q(S',a) - Q(S,A)$)
  - t←t+1

(3b) Figure 1 shows the plots for the figures 8.4 and 8.5 from the textbook which are plotted in a deterministic environment with different set of obstacles in both the plots. Curves represent DynaQ and DynaQ+ for both the environments.

The different set of parameters used are $\epsilon = 0.1$, $\kappa = 0.01$, n = 250. I tried the plots for different set of parameter values and eventually used the above optimal values. I have tried it for 10 number of trials and the resulting plots are shown for the average and standard deviation of these 10 trials.

If we look at both the plots, we can see that the DynaQ+ outperforms the DynaQ in both the cases. In first case, after the environment is changed, we see a stationary curve for some time i.e. agent finds it hard to reach the goal for the changed environment. But it learns to reach the target and eventually the cumulative reward starts growing.

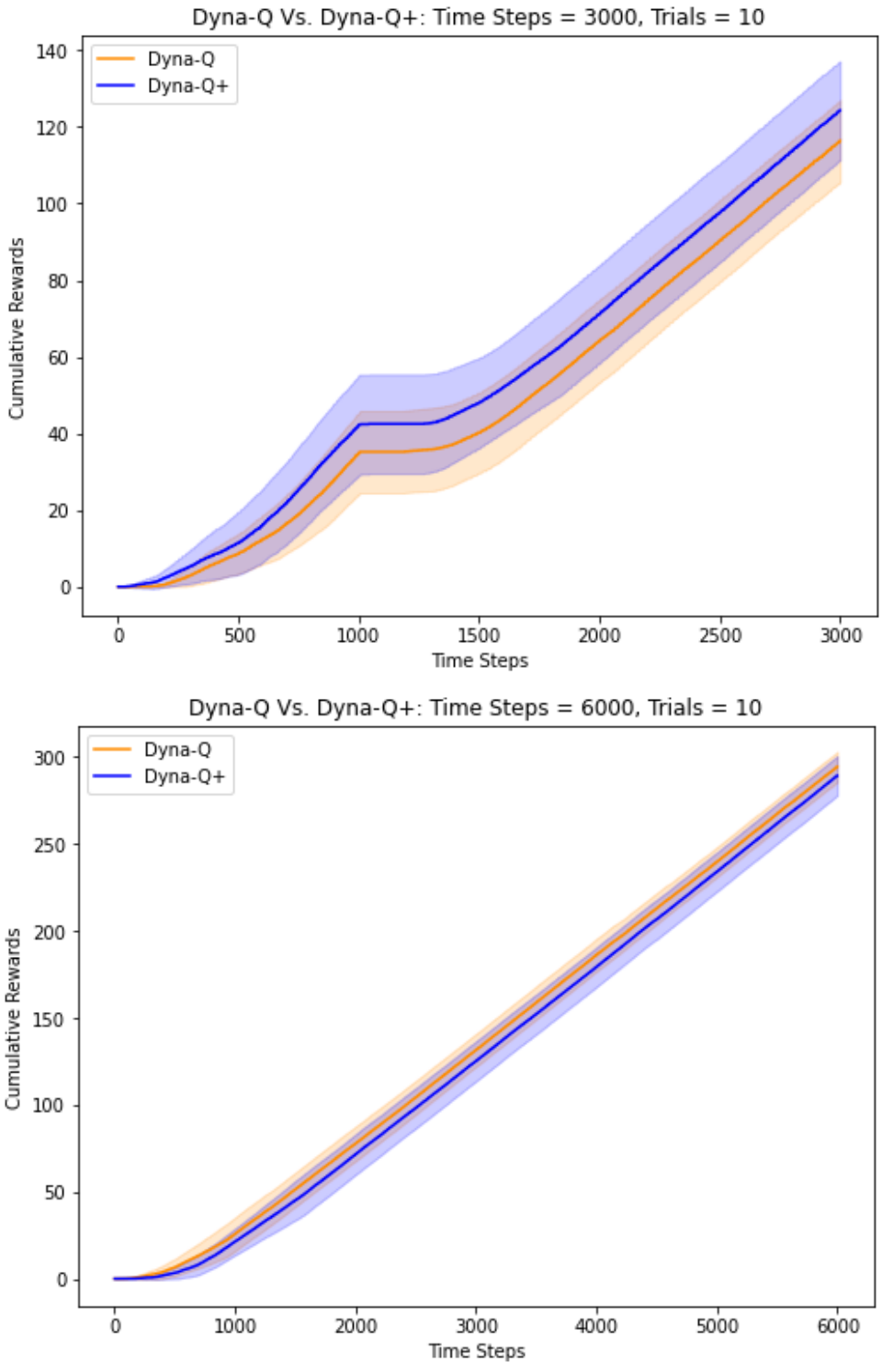

Figure 1: Q3b: Figures 8.4 and 8.5 for changing environment with DynaQ and DynaQ+

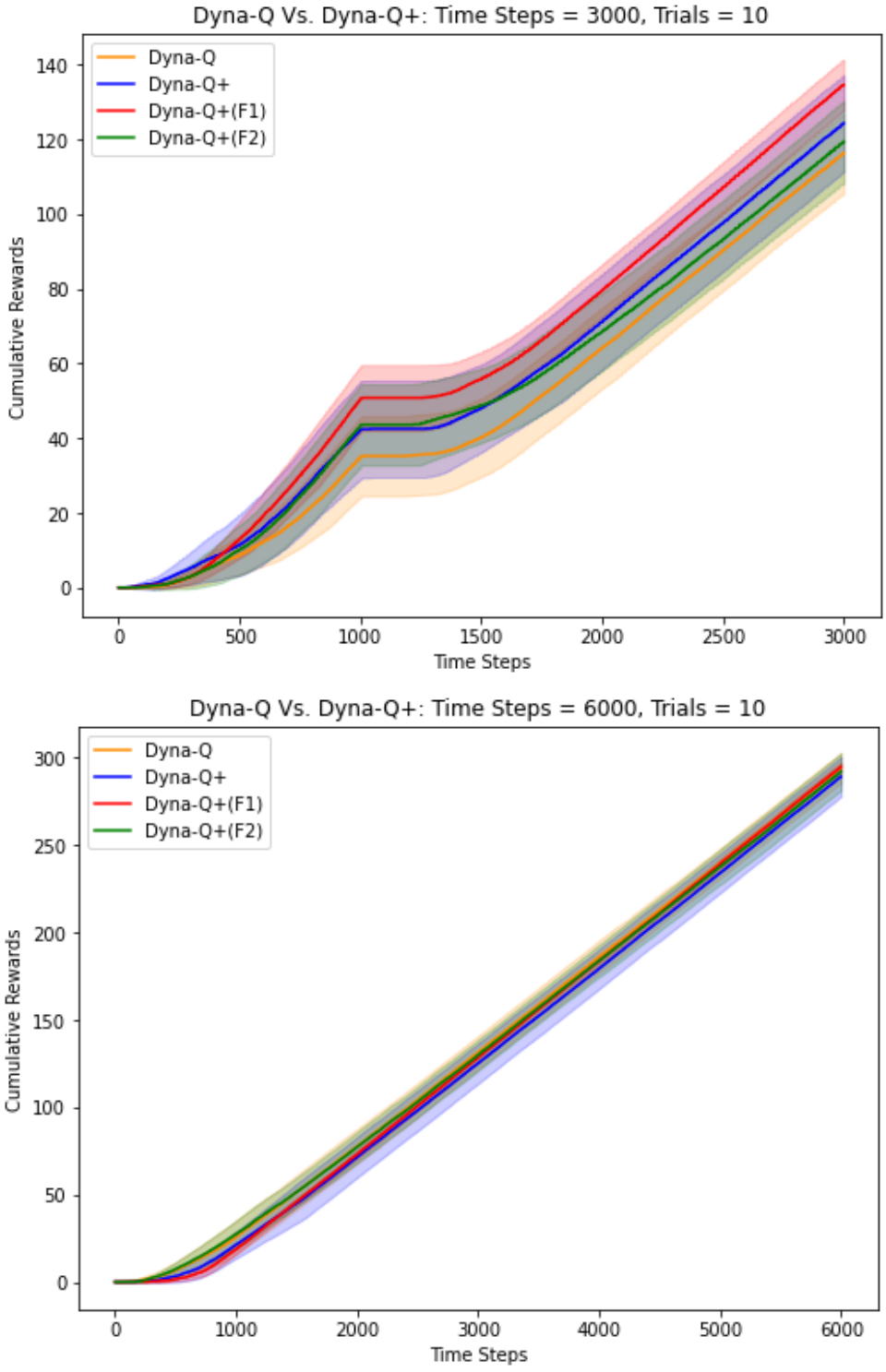

Figure 2: Q3c: Figures 8.4 and 8.5 for changing environment with DynaQ+ and its different footnote version

(3c) Figure 2 shows the plots for the figures 8.4 and 8.5 changing environments for DynaQ+ and their footnote versions i.e.
version1: actions that have never been selected can also be selected and
version2: initial model for such actions is that they would lead back to the same state with a reward of zero.
The figure suggests that the footnote versions of the DynaQ+ perform similar to the DynaQ+. Maybe that's one of the reason these variations were kept in the footnote. Although I would say even if the algorithms are equally good, they are important enough to be the part of the main text.

Question 4.

Response:

(4a)

In the DynaQ+ version with the action selection, states which have not been tried for very long time are given higher priority. However, in the version of reward function, we are giving higher priority to the state-action pairs which have not been tried for a long time.

The first version action-selection improves exploration in the learning phase of the agent whereas the second version reward selection improves the exploration in the planning phase of the agent.

The action-selection version is not limited to exploring the regions which have been selected till now. It is able to explore even the states which have not been explored yet. Its performance is not reduced with time because its exploration domain is the whole state space.

The reward selection tries to explore out of states already visited. This method is computationally less expensive as the domain of exploration is smaller. The performance however can be affected by the increasing states in the visited state set.

(4b) (4c)

Figure 3 shows the curves for the DynaQ, DynaQ+(reward selection) and the DynaQ+ with updated action selection for both the changing environments presented in figure 8.4 and 8.5 where action is selected on the basis of the following formula:

$$Q(S_t, a) = Q(S_t, a) + \kappa\sqrt{\tau(S_t, a)} \tag{1}$$

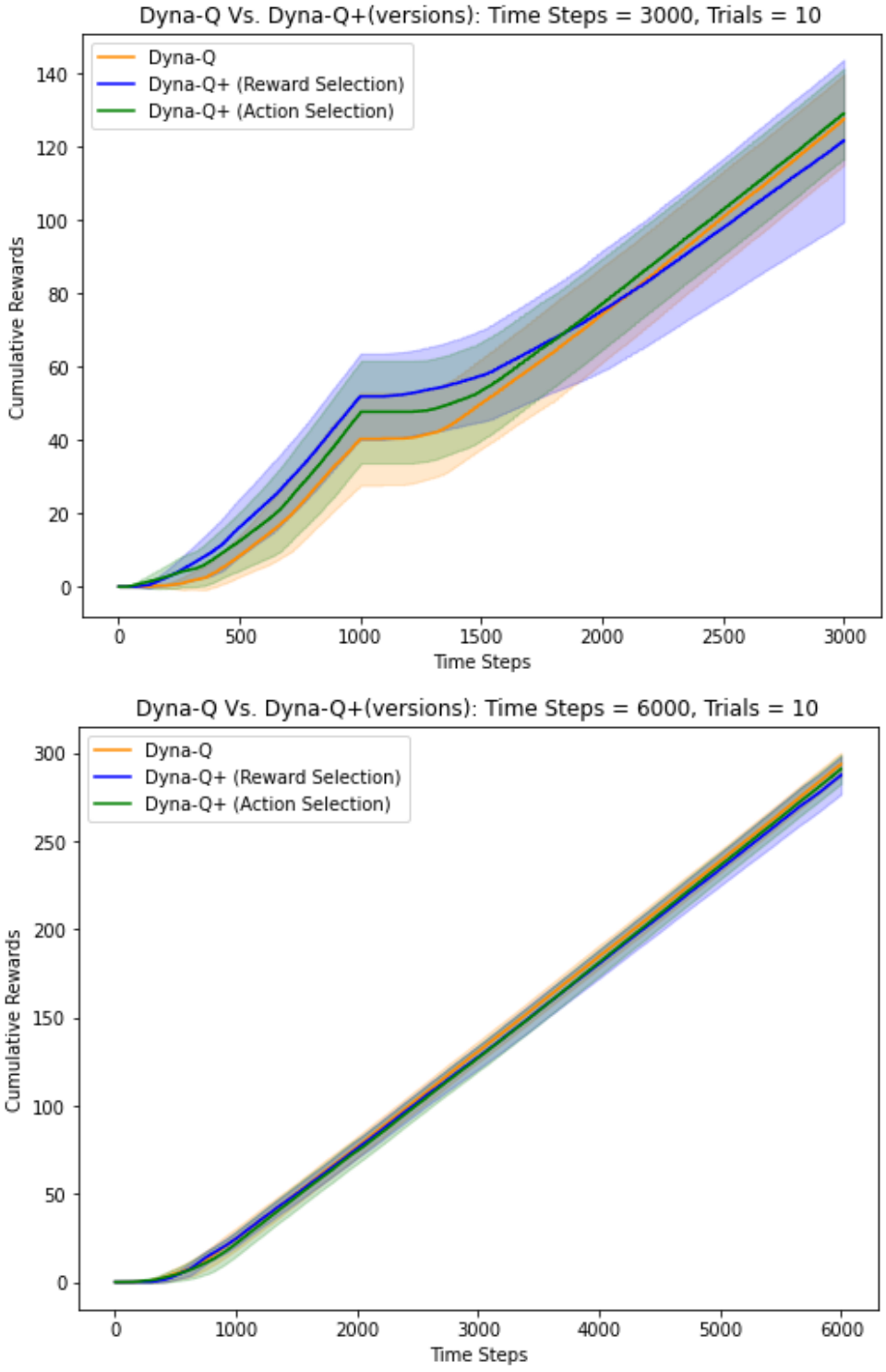

Figure 3: Q4b: Domains for 8.4 and 8.5 for changing environment with DynaQ+ and its different footnote version

Question 5.

Response:

(5a) DynaQ for stochastic environment pseudocode:

- Initialize Q(s,a) and Model(s,a) for all s∈S, and a∈A(s)
- Loop forever:
  - S ← current state
  - A ← ϵ -greedy(S,Q)
  - Take action A, observe resultant reward R, and state, S'
  - Observe set of possible state-probability tuples i.e. T = $\{(s_1, p_1),...,(s_n, p_n)\}$
  - Q(S,A) ← $Q(S,A) + \alpha \times$ (R + $\gamma max_a Q(S',a) - Q(S,A)$)
  - Model(S,A) ← T
  - Loop repeat n times
    - S ← random previously observed state
    - A ← random action taken previously at S
    - R, S ← Model(S,A) sampled according to the probabilities
    - Q(S,A) ← $Q(S,A) + \alpha \times$ (R + $\gamma max_a Q(S',a) - Q(S,A)$)

(5b) Figure 4 shows the two different plots for two different domains of 8.4 and 8.5. For each plot, we have two curves, one for stochastic environment and one for stochastic and changing environment. We can see that curves in the stochastic environments also follow the same trend as the deterministic environment.

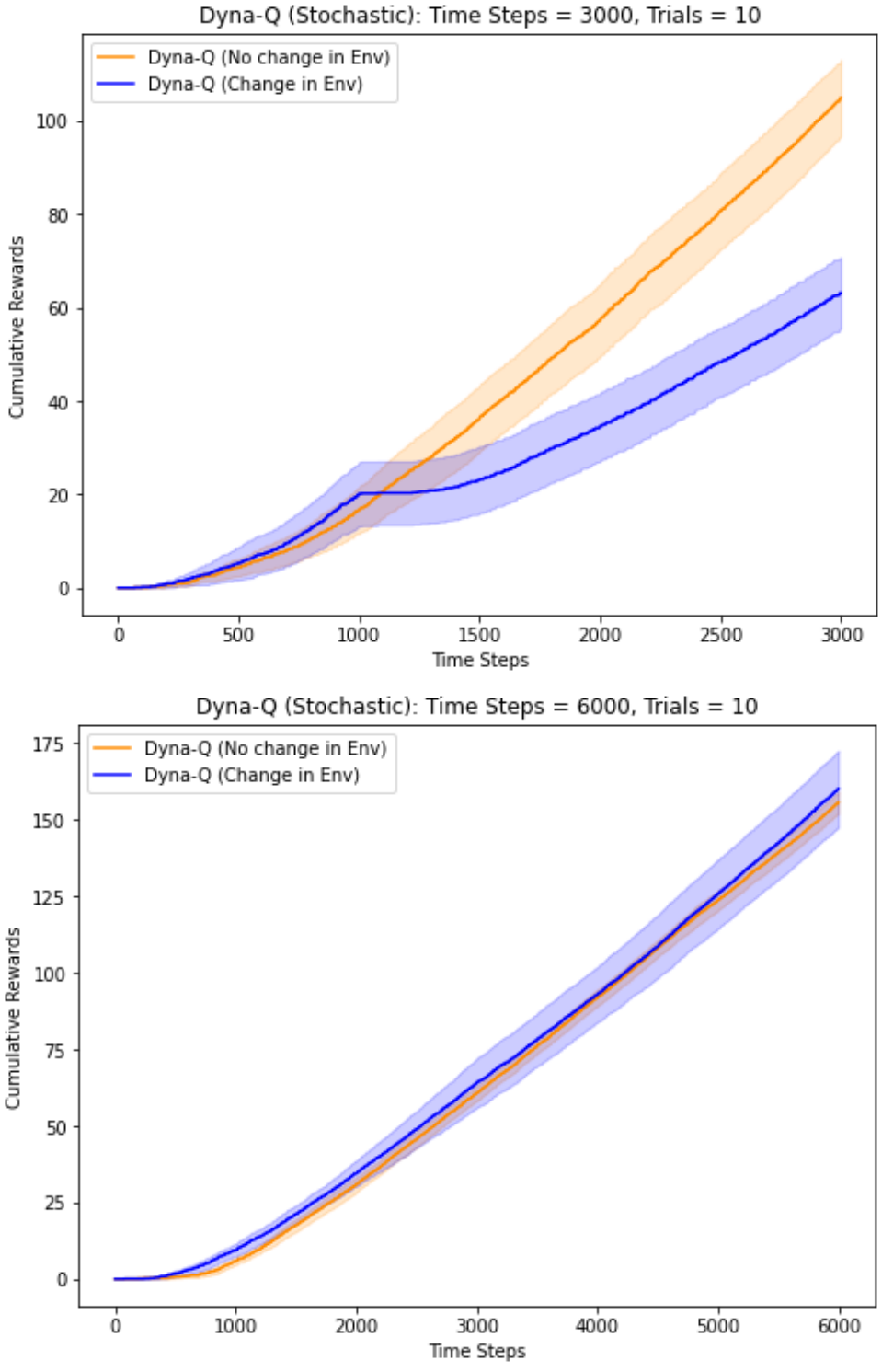

Figure 4: Q5: Stochastic environments and stochastic+changing environment for the maze domains presented in figure 8.4 and 8.5