THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

COMP3702 Assignment 3

Semester 2 2015

Joshua Rillera (43150621)
Thomas Millward (43236945)

Question 1: This question is based on the following scenario:

UQ Soda is a very small soft drink seller at UQ. It sells two types of soft drinks: Coke and Sprite. It buys its stock each morning, but can only buy at most 2 cans of soda per day and can only store and sell 3 cans of soda per day. You can assume that:

- **Coke and Sprite are the only two types of sodas that the customer may want to buy**
- **The customer will not change its preference and will leave UQSoda without buying anything if his/her choice is not available**
- **The sodas will always be in a good condition**
- **The customers preference depends on the amount of each type of sodas available at the beginning of the day (right after UQSoda buys its stock)**

Despite of its size and lack of information about the behaviour of its consumer, UQSoda wants to stock the sodas such that it can minimize the number of customers who could not get their choice of sodas

a. **The stocking problem of UQ Soda in Question 3 Tutorial 8 is an MDP problem. However, the stocking problem in the above scenario is a reinforcement learning problem, even though the problem is similar to the scenario in Question 3 of Tutorial 8. Please explain the differences and similarity between MDP and Reinforcement Learning. In your explanation, please provide a comparison example based on the UQSoda stocking problems as described above and as described in Tutorial 8.**
b. **Suppose the owner of UQSoda wants to solve the reinforcement learning stocking problem they face using model based Bayesian Reinforcement Learning problem (yes, he knows there's such an approach). The owner knows that this means he has to frame the problem as a POMDP problem, but does not really know what a POMDP is. To help him, your task is to define the POMDP problem that represents the UQSoda stocking problem (as described in this assignment)**

Answer:

Similarities and differences between MDP and Reinforcement Learning

Firstly the similarities. Both have a state and action space, they still need transition and reward functions and they also still need to generate an optimal policy. In both Tutorial 8 and the question above, we are given enough information to create the State and Action space.

The different types of soda we sell, the total number of cans we can purchase and the total number of cans we can store and sell help us generate the State and Action space. These are both available in the question above and the stocking problem given in Tutorial 8.

Now for the differences. In MDP, we have enough information to create the Transition and Reward Function. In Tutorial 8, we are given the probabilities for user consumption for each type of soda. This helps us generate the Transition and Reward Function. Because of this, we can use MDP to solve the problem. In Reinforcement Learning, we do not have information to create the Transition and Reward Function. In this question, we are not given the probabilities for user consumption of each type of soda and we do not know what reward we will get for having a number of sodas left in stock at the end of the day. This meant that we do not know the user's behaviour, so we cannot create these Transition and Reward Functions. Because of this, we need Reinforcement Learning.

POMDP Problem Definition

a. State Space (S)
    a. State Space is a tuple of 3 elements representing the amount of cans for a particular type of soda currently stocked for a day before purchasing for additional stock, the possible transitions and rewards.
    b. Formally: S = $S_{MDP(x,y)}$ * $T_{x,y}$ * $R_{x,y}$, where $S_{MDP(x,y)}$ = <x, y> where x is the number of Coke cans stored, y is the number of Sprite cans stored and x + y <= 3. $T_{x,y}$ and $R_{x,y}$ represent the possible transition and reward functions for the current state. $T_{x,y} \in T$ and $R_{x,y} \in R$
    c. Example
        i. Let s_o = the initial POMDP state, which is the empty vending machine, so s_o = $S_{MDP(0,0)}$ * $T_{0,0}$ * $R_{0,0}$ and $S_{MDP(0,0)}$ = <0, 0>.
        ii. $T_{0,0}$ will be the matrix of the probabilities of ending up in other states when doing a particular action in $S_{MDP(0,0)}$. All possible actions are <0, 0>, <0, 1>, <0, 2>, <1, 1>, <1, 0>, <2, 0> (which we will represent as a^1, a^2, a^3, a^4, a^5 and a^6, respectively. a^1, …, $a^6 \epsilon$ A). All possible states are <0, 0>, <0, 1>, <0, 2>, <0, 3>, <1, 0>, <2, 0>, <3, 0>, <1, 1>, <1, 2>, <2,1>

1. $T_{0,0} = \begin{pmatrix} a^1_{0,0} & a^2_{0,0} & a^3_{0,0} & a^4_{0,0} & a^5_{0,0} & a^6_{0,0} \\ a^1_{0,1} & a^2_{0,1} & a^3_{0,1} & a^4_{0,1} & a^5_{0,1} & a^6_{0,1} \\ a^1_{0,2} & a^2_{0,2} & a^3_{0,2} & a^4_{0,2} & a^5_{0,2} & a^6_{0,2} \\ a^1_{0,3} & a^2_{0,3} & a^3_{0,3} & a^4_{0,3} & a^5_{0,3} & a^6_{0,3} \\ a^1_{1,0} & a^2_{1,0} & a^3_{1,0} & a^4_{1,0} & a^5_{1,0} & a^6_{1,0} \\ a^1_{2,0} & a^2_{2,0} & a^3_{2,0} & a^4_{2,0} & a^5_{2,0} & a^6_{2,0} \\ a^1_{3,0} & a^2_{3,0} & a^3_{3,0} & a^4_{3,0} & a^5_{3,0} & a^6_{3,0} \\ a^1_{1,1} & a^2_{1,1} & a^3_{1,1} & a^4_{1,1} & a^5_{1,1} & a^6_{1,1} \\ a^1_{1,2} & a^2_{1,2} & a^3_{1,2} & a^4_{1,2} & a^5_{1,2} & a^6_{1,2} \\ a^1_{2,1} & a^2_{2,1} & a^3_{2,1} & a^4_{2,1} & a^5_{2,1} & a^6_{2,1} \end{pmatrix}$
2. $a^x_{y,z}$ is the probability of ending up at MDP state <y, z> after taking action a^x in state <0, 0>.

iii. $R_{0,0}$ will be the matrix of rewards for ending up in other states after taking a particular action in state

1. $R_{0,0}$ =

$$\begin{pmatrix} R_{0,0}((0,0),a^1) & R_{0,0}((0,0),a^2) & R_{0,0}((0,0),a^3) & R_{0,0}((0,0),a^4) & R_{0,0}((0,0),a^5) & R_{0,0}((0,0),a^6) \\ R_{0,0}((0,1),a^1) & R_{0,0}((0,1),a^2) & R_{0,0}((0,1),a^3) & R_{0,0}((0,1),a^4) & R_{0,0}((0,1),a^5) & R_{0,0}((0,1),a^6) \\ R_{0,0}((0,2),a^1) & R_{0,0}((0,2),a^2) & R_{0,0}((0,2),a^3) & R_{0,0}((0,2),a^4) & R_{0,0}((0,2),a^5) & R_{0,0}((0,2),a^6) \\ R_{0,0}((0,3),a^1) & R_{0,0}((0,3),a^2) & R_{0,0}((0,3),a^3) & R_{0,0}((0,3),a^4) & R_{0,0}((0,3),a^5) & R_{0,0}((0,3),a^6) \\ R_{0,0}((1,0),a^1) & R_{0,0}((1,0),a^2) & R_{0,0}((1,0),a^3) & R_{0,0}((1,0),a^4) & R_{0,0}((1,0),a^5) & R_{0,0}((1,0),a^6) \\ R_{0,0}((2,0),a^1) & R_{0,0}((2,0),a^2) & R_{0,0}((2,0),a^3) & R_{0,0}((2,0),a^4) & R_{0,0}((2,0),a^5) & R_{0,0}((2,0),a^6) \\ R_{0,0}((3,0),a^1) & R_{0,0}((3,0),a^2) & R_{0,0}((3,0),a^3) & R_{0,0}((3,0),a^4) & R_{0,0}((3,0),a^5) & R_{0,0}((3,0),a^6) \\ R_{0,0}((1,1),a^1) & R_{0,0}((1,1),a^2) & R_{0,0}((1,1),a^3) & R_{0,0}((1,1),a^4) & R_{0,0}((1,1),a^5) & R_{0,0}((1,1),a^6) \\ R_{0,0}((1,2),a^1) & R_{0,0}((1,2),a^2) & R_{0,0}((1,2),a^3) & R_{0,0}((1,2),a^4) & R_{0,0}((1,2),a^5) & R_{0,0}((1,2),a^6) \\ R_{0,0}((2,1),a^1) & R_{0,0}((2,1),a^2) & R_{0,0}((2,1),a^3) & R_{0,0}((2,1),a^4) & R_{0,0}((2,1),a^5) & R_{0,0}((2,1),a^6) \end{pmatrix}$$

2. Where $R_{0,0}((y, z), a^x)$ is the reward for ending up at MDP state <y, z> after taking action a^x in state <0, 0>

b. Action Space (A)

a. Action Space is a tuple of 2 elements representing the amount of cans for a particular type of soda that was purchased for stock in a day.

b. Formally: A = <P_0, P_1> where P_0 is the number of Coke cans purchased and P_1 is the number of Sprite cans purchased and $P_0 + P_1 <= 2$

c. Example

i. All the possible actions are as follows: <0, 0>, <0, 1>, <0, 2>, <1, 1>, <1, 0> and <2, 0>

c. Observation Space (Ω)

a. The observation space is the amount of cans left in the vending machine, the estimated probability for ending up in the current state and the reward for being in the state.

- b. Formally: O = <C_0, C_1, T, R> where C_0 is the number of Coke cans left and C_1 is the number of Sprite cans left and $C_0 + C_1 <= 3$. T is the estimated probability for ending up in the current state and R is the reward for being in the current state. This is assuming that the number of cans users can consume in a day is equal to the number of cans we can purchase
- c. Example
  - i. Let O_1 be an observation made.
  - ii. O_1 = <2, 1, 0.5, 10>, which means that we are out of stock (0,0), 2 cans of Coke and 1 can of Sprite are left. We estimate that there's a probability of 0.5 of making it into this state and we received a reward of 10.

d. Transition Function (T)

- a. Assuming that the MDP model is as described by POMDP state s ∈ S
- b. Given the previous state s_t , the action taken a_t and the new state s_{t+1}, where t = current time slice, s_t, $s_{t+1} \in S$ and $a_t \in A$ the transition function will return the probability of ending up at s_{t+1} after doing a_t at state s_t.
- c. Informally, it will be the probability of ending up with a particular number of cans for each soda type after purchasing soda.
- d. Formally, $T(s_t, a_t, s_{t+1}) = T((<x, y>, T_t, R_t), <m, n>, (<a, b>, T_{t+1}, R_{t+1})) = T_C(x, a, m) * T_S(y, n, b)$, where T_C and T_S are the individual transition functions for Coke and Sprite, respectively. **Take note that this assumes that the transitions for Coke and Sprite are independent of each other.** We have not been given information about whether or not these transitions are independent of each other but we can initially assume that these are independent. If, through observation, it is found that these transitions are dependent on each other, then the transition function needs to be redefined.
  - i. Regardless of the type of soda, we can calculate the probabilities in 3 different ways. Firstly, if we suddenly end up with more cans of coke than when we started (e.g. for T_C, m > x + a), then the probability is 0 because users will not be able to restock the vending machine themselves.
  - ii. Secondly, if the amount of cans left at the end of the day is > 0 but <= the number of cans we had at the beginning of the day (e.g. for T_C, 0 < m <= x + a), then we need to refer to a matrix of probabilities for that type of soda. Let's denote this as P_{soda} where soda is either Coke or Sprite.
    1. $P_{soda} = \begin{pmatrix} p_{0,0} & \cdots & p_{0,j} \\ \cdots & \cdots & \cdots \\ p_{i,0} & \cdots & p_{i,j} \end{pmatrix}$
    2. Where $p_{a,b}$ = the probability of the user consuming j cans of soda type x when there is currently i cans in stock. Take note that i <= 3 and j >= 3.

3. If we say $P_{soda}[i, j]$ refers to the probability at row i and column j of the matrix, then for this possibility, we return $P_{soda}[c_{start} + a_{purchase}, c_{start} + a_{purchase} – c_{end}]$. (e.g. $T_C = P_C[x + a, x + a – m]$)

iii. Finally, if the amount of cans left at the end of the day is == 0, then the user probably consumed (or wanted to consume) a quantity of soda greater than or equal to the quantity of soda we started with (e.g. for T_C, user probably consumed >= x + a cans of coke).

1. Using the probability matrix we defined earlier, we will return:

$$T_{soda} = \sum_{i=start+purchase}^{max} P_{soda}[start + purchase, i]$$

2. Where max = the maximum amount of soda a user can consume in a day (needs to be observed), start = the amount of cans we start with and purchase = amount of cans we purchased.

e. To conclude, T_{soda}(start, purchase, end) will give the following values:

i. 0 *(if end > start + purchase)*

ii. P_{soda}[start + purchase, start + purchase – end] *(if 0 < end <= start + purchase)* and

iii. $\sum_{i=start+purchase}^{max} P_{soda}[start + purchase, i]$ *(if end = 0)*

f. For example,

i. Let's say we started with 0 cans of Coke, 2 cans of Sprite, purchased 1 can of Coke and ended up with 0 cans for both Coke and Sprite at the end of the day

ii. Then, our transition function would look like:

1. T(<0, 2>, <1, 0>, <0, 0>) = T_C(0, 1, 0) * T_S(2, 0, 0)

iii. Let's say max = 3 and we had probability matrices:

$$P_C = \begin{pmatrix} 0.4 & 0.2 & 0.3 & 0.1 \\ 0.3 & 0.4 & 0.1 & 0.2 \\ 0.1 & 0.6 & 0.2 & 0.2 \\ 0.1 & 0.2 & 0.6 & 0.1 \end{pmatrix} \text{ and } P_S = \begin{pmatrix} 0.3 & 0.2 & 0.4 & 0.1 \\ 0.6 & 0.2 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.4 & 0.4 \end{pmatrix}$$

iv. Then, $T_C = P_C[1, 1] + P_C[1, 2] + P_C[1, 3]$ and T_S = P[2, 2] + P[2, 3]

1. T_C = 0.4 + 0.1 + 0.2 = 0.7

2. T_P = 0.6 + 0.1 = 0.7

v. So T = 0.7 * 0.7 = 0.49

e. Observation Function (Z)

a. Given the new state s_{t+1}, the action taken a_t and the observation o_t, where t = current time slice, $s_{t+1} \in S$ and $o_t \in O$, the observation function ($Z(s_{t+1}, a^x, o^t)$) will give the expected next state and reward

b. Formally: $Z(s_{t+1}, a^x, o_t) = (S_{x,y}, R)$ where $S_{x,y} \in S$ and R is the expected reward.

c. Example

i. Let s = (0, 0, $T_{0,0}$, $R_{0,0}$) where $T_{0,0}$ and $R_{0,0}$ are the transition and reward functions for this state, respectively.
ii. Let a^1 = <0, 0>
iii. Let o = <0, 0, 0.5, -5>
iv. Suppose we gave these values to our observation function. Then, we receive:
    1. Z(s, a^1, o) = ([<0, 0>, $T_{0,0}$, $R_{0,0}$], -10) where $T_{0,0}$ and $R_{0,0}$ are the possible transition and reward functions for having no cans left. We have received a reward of -10.

f. Reward Function (R)
    a. Assuming that the MDP model is as described by POMDP state s ∈ S
    b. Given the new state s and the action taken a, where s ∈ S and a ∈ A, the reward function will give us the reward for making it to s after doing a.
    c. Informally, the reward function will return the penalty for failing in providing for the user requests
    d. Formally, R(s, a) = x where x = the penalty
    e. Like the transition function, we need to split the reward function because we have two types of soda. So:
        i. R(s, a) = R((<x, y>, T_t, R_t), <m, n>) = R_C(x, m) + R_S(y, n), where R_C and R_s represent the rewards for Coke and Sprite respectively. We can use R_{soda} to represent these two components generally
    f. We also use the probability matrices in the transition function for this. This is because we need to go through all possible rates of consumption to find out the exact penalty. So:

$$R_{soda}(start, purchase) = -1 * \sum_{i=start+purchase+1}^{max} (i - start - purchase) * P_{soda}[start + purchase, i]$$

        i. Where start = the starting cans for soda, purchase = the amount of cans purchased for soda and P_{soda} representing the probability matrix for soda.
    g. A few notes about this:
        i. We multiply by -1 because we are representing the reward as a penalty.
        ii. If we started with and purchased 0 cans of soda, we don't consider the event when a user doesn't want to consume a particular soda because if we did, we would just end up with 0 to sum with other values.
        iii. max (like the transition function) needs to be observed because we do not know this yet. However, we can say that max >= 3 because that is the maximum amount of cans we can store. So initially, we only assume that users would consume at most 3 cans of soda. If users suddenly want to consume more, we take note of that and change our probability matrices
    h. For example,

i. Let's say we started with 0 cans of Coke, 2 cans of Sprite and purchased 1 can of Coke

ii. Then, our reward function would look like:

   1. T(<0, 2>, <1, 0>) = R_C(0, 1) * R_S(2, 0)

iii. Let's say max = 3 and we had probability matrices:

$$P_C = \begin{pmatrix} 0.4 & 0.2 & 0.3 & 0.1 \\ 0.3 & 0.4 & 0.1 & 0.2 \\ 0.1 & 0.6 & 0.2 & 0.2 \\ 0.1 & 0.2 & 0.6 & 0.1 \end{pmatrix} \text{ and } P_S = \begin{pmatrix} 0.3 & 0.2 & 0.4 & 0.1 \\ 0.6 & 0.2 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.4 & 0.4 \end{pmatrix}$$

iv. Then,

   1. R_C = -1 * (((2 – 1) * P_C[1, 2]) + ((3 – 1) * P_C[1, 3])) = -1 * (0.1 + (2 * 0.3)) = -1 * (0.1 + 0.6) = -0.7
   2. R_S = -1 * ((3 – 2) * P_S[2, 3]) = -1 * 0.1 = -0.1

v. Hence, R = -0.7 + -0.1 = -0.8

Question 2:

Let’s consider Q-learning and SARSA with ϵ greedy algorithm for action selection. Please explain the effect of different ϵ values on the performance of the learning methods. In particular, please explain the effect when ϵ = 0, ϵ = 1 and as ϵ increases from 0 to 1

Answer:

The epsilon-greedy algorithm involves assigning weight to each action. Changing the value of ϵ determines the probability of selecting an action to generate the Q value for each episode. Setting ϵ = 0 means that the action with the highest weight will be selected with probability 1 while all other actions are selected with probability 0. Setting ϵ = 1 means that the action with the highest weight will be selected with probability 0 while all other actions are selected with probability 1. Hence, setting ϵ at extremes determines the priority meant by the weight. (If high weight, best action if ϵ = 0, worst action if ϵ = 1. This means that as ϵ increases from 0 to 1, we start placing priority on the other possible actions. Starting at ϵ = 0, we have only selected the best action, but as we increase ϵ, each action gets a higher and higher chance of being evaluated.

Knowing this, we can check the effect of the values of ϵ on Q-Learning and SARSA

<u>*Q-Learning*</u>

i. ϵ = 0

   Only observe the reward and the next state after taking the most favoured action. Base the Q-value solely on the best action (exploitation). Hence, we are choosing to do the most favoured action for every single state. When calculating the new Q-value, we pick the action with the largest Q-value (a * [r + discount factor * max$^{a'}$ * Q(s’, a’) – Q(s, a)]), but since we only pick the most favoured action when changing the Q-value, other actions may never be selected, depending on their initialized value

ii. ϵ = 1

   Observe every action except the most favoured action (exploration). Base the Q-value on a range of actions. When calculating the new Q-value, we have a range of actions to choose from, in contrast to when ϵ = 0

iii. ϵ increases from 0 to 1

   If we start at ϵ = 0, the first few episodes (or the first few steps of the first episode) will have a Q-value dependent on the most favoured action. But as we increase ϵ, we start taking into account other actions. As a result, the other episodes have a wider variety of actions to take. Once ϵ = 1, the most favoured action is no longer taken into account in calculating the Q-value.

SARSA

i. $\epsilon = 0$

Similar to Q-Learning. The difference is that we are setting the entire policy to be the most favoured action, where in Q-Learning we are simply setting the Q-value. This is regardless of what state we are in.

ii. $\epsilon = 1$

Similar to Q-Learning again, except now the policy will be any one of the less favoured actions. The most favoured action will not be a part of the policy at all

iii. ϵ increases from 0 to 1

Once again, similar to Q-Learning, but now, every single action now has a possibility of being a part of the policy. We start at $\epsilon = 0$. Initially, the policy will state that we use the most favoured action. As we increase ϵ, the policy will tell us to take the less favoured actions. Once $\epsilon = 1$, the most favoured actions is no longer taken into account in calculating the Q-value and we will only select the less-favoured actions

Question 3:

This question is based on the following simple navigation scenario.

A robot is navigating in an environment that has been discretised into 4 grid cells, as shown in Figure 1. We know the robot's action space is {left, right} and that effect of performing an action is non-deterministic. However, we do not have a stochastic model of this non-deterministic behaviour of the robot. Worse, we do not know the cost of each action that the robot takes nor the reward for reaching the goal. Despite this lack of information, we want to find a policy for the robot to move from cell-1 to cell-4 with minimum cost.

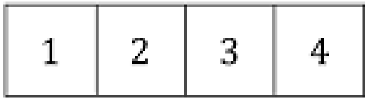

Figure 1: Simple navigation problem.

Suppose we want to solve this problem using model-free reinforcement learning approaches. We set the initial values of all states to be zero, generated an initial policy π and used this policy to generate the following two episodes.

<u>Episode-1:</u> cell1-right- -1, cell1-right- -1, cell2-right- -1, cell2-right - -1, cell3-right-9, cell4

<u>Episode-2:</u> cell1-right- -1, cell2-right- -1, cell2-right- -1, cell3-right-9, cell4

Now we want to evaluate the policy π. Please write down the value functions of the first two iterations of the following model-free reinforcement learning methods:

a. **TD(0)**
b. **TD(2)**
c. **Monte Carlo**

Answers:

a. Before starting, we need to set/know the following values
    a. Discount Factor (γ)
        i. No discount factor given so arbitrarily set the discount factor to be 1 to simplify calculations
    b. States
        i. We only have 4 states. These are Cell1, Cell2, Cell3 or Cell4
    c. α value
        i. The α value is not given, so let's arbitrarily set it to be 1 to simplify calculations

- d. Reward
  - i. Each reward is given after each action
- e. Initial values for all states
  - i. All states start with a value of 0

b. TD(0) Answer

- a. Episode 1
  - i. 1st Step (Cell1 – right - -1 -> Cell1)
    1. V(Cell1) = 0 + 1(-1 + 0 – 0) = -1
  - ii. 2nd Step (Cell1 – right - -1 -> Cell2)
    1. V(Cell1) = -1 + 1(-1 + 0 - -1) = -1
  - iii. 3rd Step (Cell2 – right - -1 -> Cell2)
    1. V(Cell2) = 0 + 1(-1 + 0 – 0) = -1
  - iv. 4th Step (Cell2 – right - -1 -> Cell3)
    1. V(Cell2) = -1 + 1(-1 + 0 - -1) = -1
  - v. 5th Step (Cell3 – right – 9 -> Cell4)
    1. V(Cell3) = 0 + 1(9 + 0 – 0) = 9
  - vi. Values For States
    1. V(Cell1) = -1
    2. V(Cell2) = -1
    3. V(Cell3) = 9
    4. V(Cell4) = 0
- b. Episode 2
  - i. 1st Step (Cell1 – right - -1 -> Cell2)
    1. V(Cell1) = -1 + 1(-1 + -1 - -1) = -2
  - ii. 2nd Step (Cell2 – right - -1 -> Cell2)
    1. V(Cell2) = -1 + 1(-1 + -1 - -1) = -2
  - iii. 3rd Step (Cell2 – right - -1 -> Cell3)
    1. V(Cell2) = -2 + 1(-1 + 9 - -2) = 8
  - iv. 4th Step (Cell3 – right – 9 -> Cell4)
    1. V(Cell3) = 9 + 1(9 + 0 – 9) = 9
  - v. Values For States
    1. V(Cell1) = -2
    2. V(Cell2) = 8
    3. V(Cell3) = 9
    4. V(Cell4) = 0
- c. Final Values
  - i. V(Cell1) = -2
  - ii. V(Cell2) = 8
  - iii. V(Cell3) = 9
  - iv. V(Cell4) = 0

c. TD(2) Answer

- a. Episode 1
  - i. 1st Step (Cell1 – right - -1 -> Cell1)
    1. $V(Cell1) = 0 + 1(-1 + 1(-1) + 1^2(-1) + 1^3(0) - 0) = -3$

ii. 2nd Step (Cell1 – right - -1 -> Cell2)

1. V(Cell1) = -3 + 1(-1 + -1 + -1 + 0 - -3) = -3

iii. 3rd Step (Cell2 – right - -1 -> Cell2)

1. V(Cell2) = 0 + 1(-1 + -1 + 9 + 0 – 0) = 7

iv. 4th Step (Cell2 – right - -1 -> Cell3)

1. V(Cell2) = 7 + 1(-1 + 9 + 0 + 0 – 7) = 8

v. 5th Step (Cell3 – right – 9 -> Cell4)

1. V(Cell3) = 0 + 1(9 + 0 + 0 + 0 – 0) = 9

vi. Values For States

1. V(Cell1) = -3
2. V(Cell2) = 8
3. V(Cell3) = 9
4. V(Cell4) = 0

b. Episode 2

i. 1st Step (Cell1 – right - -1 -> Cell2)

1. V(Cell1) = -3 + 1(-1 + -1 + -1 + 8 - -3) = 5

ii. 2nd Step (Cell2 – right - -1 -> Cell2)

1. V(Cell2) = 8 + 1(-1 + -1 + 9 + 8 – 8) = 15

iii. 3rd Step (Cell2 – right - -1 -> Cell3)

1. V(Cell2) = 15 + 1(-1 + 9 + 0 + 9 – 15) = 17

iv. 4th Step (Cell3 – right – 9 -> Cell4)

1. V(Cell3) = 9 + 1(9 + 0 + 0 + 0 – 9) = 9

v. Values For States

1. V(Cell1) = 5
2. V(Cell2) = 17
3. V(Cell3) = 9
4. V(Cell4) = 0

c. Final Values

i. V(Cell1) = 5

ii. V(Cell2) = 17

iii. V(Cell3) = 9

iv. V(Cell4) = 0

d. Monte Carlo Answer

a. Episode 1

i. 1st Step (Cell1 – right - -1 -> Cell1)

1. V(Cell1) = 0 + 1(-1 + -1 + -1 + -1 + 9 – 0) = 5

ii. 2nd Step (Cell1 – right - -1 -> Cell2)

1. V(Cell1) = 5 + 1(-1 + -1 + -1 + 9 - 5) = 6

iii. 3rd Step (Cell2 – right - -1 -> Cell2)

1. V(Cell2) = 0 + 1(-1 + -1 + 9 – 0) = 7

iv. 4th Step (Cell2 – right - -1 -> Cell3)

1. V(Cell2) = 7 + 1(-1 + 9 – 7) = 8

v. 5th Step (Cell3 – right – 9 -> Cell4)

1. V(Cell3) = 0 + 1(9 – 0) = 9

- vi. Values For States
  1. V(Cell1) = 6
  2. V(Cell2) = 8
  3. V(Cell3) = 9
  4. V(Cell4) = 0

- b. Episode 2
  - i. 1st Step (Cell1 – right - -1 -> Cell2)
    1. V(Cell1) = 6 + 1(-1 + -1 + -1 + 9 – 6) = 6
  - ii. 2nd Step (Cell2 – right - -1 -> Cell2)
    1. V(Cell2) = 8 + 1(-1 + -1 + 9 – 8) = 7
  - iii. 3rd Step (Cell2 – right - -1 -> Cell3)
    1. V(Cell2) = 7 + 1(-1 + 9 – 7) = 8
  - iv. 4th Step (Cell3 – right – 9 -> Cell4)
    1. V(Cell3) = 9 + 1(9 - 9) = 9
  - v. Values For States
    1. V(Cell1) = 6
    2. V(Cell2) = 8
    3. V(Cell3) = 9
    4. V(Cell4) = 0
- c. Final Values
  - i. V(Cell1) = 6
  - ii. V(Cell2) = 8
  - iii. V(Cell3) = 9
  - iv. V(Cell4) = 0