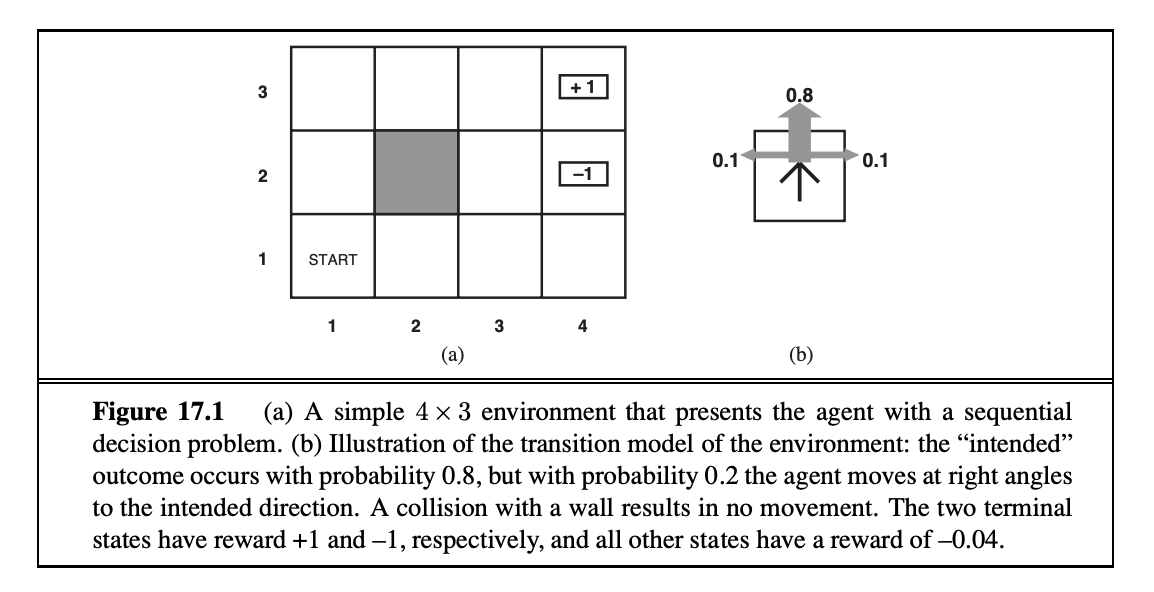
# Question 1



1. **R&N Problem 17.1: (8 points)**  
   Consider the **4x3** world shown in Figure 17.1(a). There is an impassable wall/obstacle at state (2,2). The state **(4,3)** is a terminal state with a reward of **+1**, and state **(4,1)** is a terminal state with a reward of **-1**. All other states yield a reward of **-0.04** upon exiting them.

The transition model is stochastic, as illustrated in Figure 17.1(b):

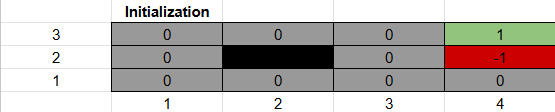
* There is an 80% chance the agent moves in the intended direction.
* There is a 10% chance the agent moves 90 degrees left relative to the intended direction.
* There is a 10% chance the agent moves 90 degrees right relative to the intended direction.

If the agent attempts to move into an external boundary or the internal wall at (2,2), it remains in its current state.

a. What are the optimal values (V∗) of the states at (1,1), (2,1), and (1,2)?

b. What is the optimal policy (π∗) for the states at (1,1), (2,1), and (1,2)?

Using value iteration, after approximately 20 iterations the values for each of the states converged by use of the equaton: V(s)=amax​s′ ∑​P(s′∣s,a)⋅[R(s,a,s′)+γV(s′)]. Values are computed from the states closest to the terminal state and back propagated to the initial start position. As we iterate, eventually these values would converge to some final acceptable solution.

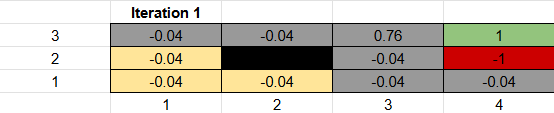
All values are first initialized to 0 with the exception of the two terminal states.

For the first iteration we inspect the state just left of the terminal state with reward 1 i.e. state (3,3). Computing the optimal values for this state yields:

* With an intention of moving right:
  + V((3,3), right) = 0.8 (-0.04 +(1 x 1)) = -0.768
  + V((3,3), up) = 0.1(-0.04 + (1x 0)) = -0.004
  + V((3,3), down) = 0.1(-0.04 + (1x 0)) = -0.004
  + Summing these gives: **0.76**

All other states will give a reward of 0 so we can compute their optimal values at this iteration in one pass eg for state (0,0):

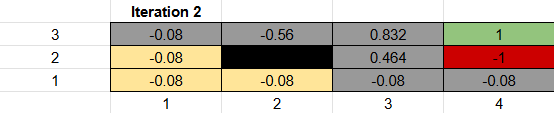
* V((0,0), right) = 0.8 (-0.04 +(1 x 0)) = -0.032
* V((0,0), down) = 0.1 (-0.04 +(1 x 0)) = -0.004
* V((0,0), up) = 0.8 (-0.04 +(1 x 0)) = -0.004
* Summing these yield: -0.04 and can be used to represent each remaining state at this iteration.

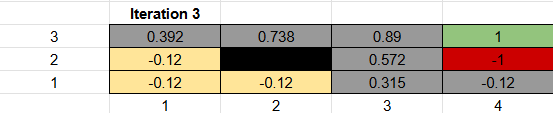


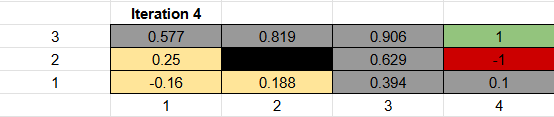
For the next iteration we again start with the state (3,3) again and manually compute the optimal state values:

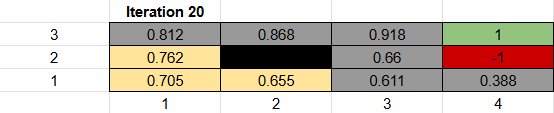
* With an intention of moving right:
  + V((3,3), right) = 0.8 (-0.04 +(1 x 1)) = -0.768
  + V((3,3), up but stay) = 0.1(-0.04 + (1x 0.76)) = -0.032
  + V((3,3), down) = 0.1(-0.04 + (1x-0.04)) = -0.008
  + Summing these gives: **0.832**
* With an intention of moving down:
  + V((3,3), down) = 0.8 (-0.04 +(1 x -0.04)) = -0.064
  + V((3,3), right, terminal) = 0.1 (-0.04 +(1 x -1)) = -0.104
  + V((3,3), left) = 0.1 (-0.04 +(1 x -0,04)) = -0.008
  + This sums to: -0.176
* With intention of moving left:
  + V((3,3), left) = 0.8 (-0.04 +(1 x -0.04)) = -0.064
  + V((3,3), down) = 0.1 (-0.04 +(1 x -0,04)) = -0.008
  + V((3,3), up, stay) = 0.1 (-0.04 +(1 x 0.76)) = 0.072
  + This sums to: 0.128
* With intention of moving up:
  + V((3,3), up, stay) = 0.8 (-0.04 +(1 x 0.76)) = 0.576
  + V((3,3), left) = 0.1 (-0.04 +(1 x -0.04)) = -0.008
  + V((3,3), right) = 0.1 (-0.04 +(1 x 1)) = 0.096
  + This sums to: 0.664

The max of these summation is taken as: max[0.832, -0.176, 0.128,0.664] = 0.832 moving right. The optimal policy is to move right with optimal value state as 0.832. This process is repeated for all states and the results of the first four iterations are illustrated below. The values were found to converge around 16 to 20 iterations:









Based on the values obtained from the iteration process, the optimal values and policies for states: (1,1), (2,1), and (1,2) are: [1,1] = 0.705 moving up, [2,1] = 0.655 moving left and [1,2] = 0.762 moving up.

# Question 2 R&N problem

# 

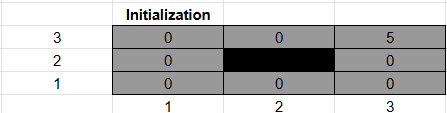
A is the starting state and B is the terminal state with reward +5. The middle square contains a wall. The agent receives a reward of –0.04 in all other states. The discount factor γ is 0.9. The agent can choose from four actions: Up, Down, Left, and Right. When the agent bumps into a wall, it stays in the same state.

a. What is the optimal value of each state (excluding B)?

b. What is the optimal policy from each state?

A similar approach to solving question one is taken where the following equation is employed:

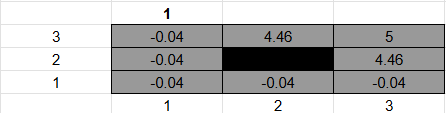
V(s)=amax​s′ ∑​P(s′∣s,a)⋅[R(s,a,s′)+γV(s′)]. First all states excluding the terminal state are initialized to 0. Computation is done on the states closest to the terminal state and then values are backpropagated throughout the girdworld.

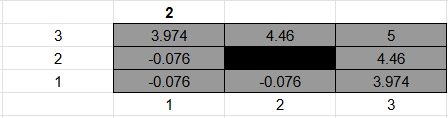


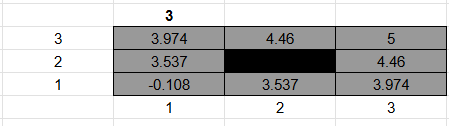
For the first iteration the state (2,3) is analyzed.

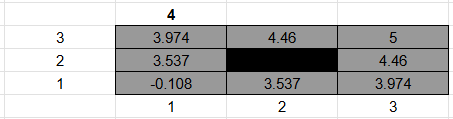
* With intention of moving right:
  + V(3,3 right) = 1 (-0.04 + (0.9 x5) = 4.46
* With intention of moving left:
  + V(1,3 left) = 1 (-0.04 + (0.9 x 0) = -0.004
* With intention of moving down:
  + V(2,2 down, stay) = 1 (-0.04 + (0.9 x 0) = -0.004
* With intention of moving up:
  + Same as moving down

Taking the max of these values gives 4.46 moving right as the optimal value and policy for this state. This approach is repeated for 5 iterations until the values converged to give the optimal value and policy for all states. The values for these iterations are illustarated below:









For the fifth iteration and state (2,3) analyzed.

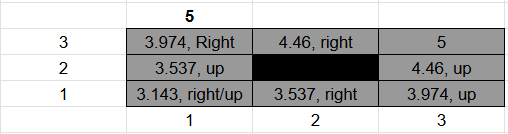
* With intention of moving right:
  + V(3,3 right) = 1 (-0.04 + (0.9 x5) = 4.46
* With intention of moving left:
  + V(1,3 left) = 1 (-0.04 + (0.9 x 3.974) = 3.5
* With intention of moving down:
  + V(2,2 down, stay) = 1 (-0.04 + (0.9 x 4.46) = 3.974
* With intention of moving up:
  + Same as moving down
* Taking max of these = 4.46 moving right.

For the fifth iteration and state (1,3) analyzed:

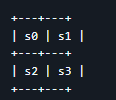
* With intention of moving right:
  + V(2,3 right) = 1 (-0.04 + (0.9 x4.46) = 3.9741
* With intention of moving left:
  + V(1,3 left, stay) = 1 (-0.04 + (0.9 x 3.974) = 3.9740
* With intention of moving down:
  + V(2,2 down) = 1 (-0.04 + (0.9 x 3.537) = 3.143
* With intention of moving up:
  + Same as moving left
* Taking max of these = 3.974 moving right.

For the fifth iteration and state (1,2) analyzed:

* With intention of moving right:
  + V(2,2 right, stay) = 1 (-0.04 + (0.9 x 3.537) = 3.143
* With intention of moving left:
  + Same as moving right
* With intention of moving down:
  + V (1,1 down) = 1 (-0.04 + (0.9 x -0.108) = -0.1372
* With intention of moving up:
  + V (1,3 up) = 1 (-0.04 + (0.9 x 3.974) = 3.5366
* Taking max of these = 3.537 moving up.

The same approach is used for the remaining states to give the following value/policy output:

# Question 3



State s3 is a terminal state with reward +1. All other transitions have reward 0. From each non-terminal state, the agent can move in four directions: Up, Down, Left, Right. If the agent would move off the grid, it stays in the same state. The discount factor γ is 0.8 and the learning rate α is 0.5.

a. All Q-values are initialized to 0. The agent performs the following experiences (state, action, reward, next state): (s0, Right, 0, s1), (s1, Down, 0, s3), (s3, NA, 1, Terminal), (s0, Down, 0, s2), (s2, Right, 0, s3), (s3, NA, 1, Terminal). Manually perform Q-learning updates for each of these experiences and show your calculations.

b. After these updates, what action would an ε-greedy agent with ε = 0.2 choose in state s0? Show your work.

c. What are the pros and cons of using a higher value of α (e.g., 0.9) versus a lower value (e.g., 0.1) in Q-learning?

# Question 4

**Feature-Based Q-Learning: (8 points)**  
Suppose you are implementing an approximate Q-learning agent for Pacman using the following features:

* + f₁(s,a) = 1 if action a moves Pacman closer to the nearest food, 0 otherwise
  + f₂(s,a) = 1 if action a moves Pacman away from the nearest ghost, 0 otherwise
  + f₃(s,a) = number of food pellets remaining in state s

**a. If the weights are w₁ = 2.0, w₂ = 3.0, and w₃ = -0.5, calculate Q(s,a) for a state where action a moves Pacman closer to food, away from a ghost, and there are 4 food pellets remaining.**

The Q(s,a) = w1f1(s,a) + w2f2(s,a) + w3f3(s,a).

Given that f1(s,a) = 1 ie. moving towards a food pellet, f2(s,a) = 1 ie. moving away from a ghost and finally f3(s,a) = 4 ie. the number of food pellets remaining and w₁ = 2.0, w₂ = 3.0, and w₃ = -0.5 then the Q(s,a) can be computed as:

Q(s,a) = 2.0(1) + 3.0(1) - 0.5(4)

Q(s,a) = 3

**b. Suppose Pacman takes this action and arrives in a state where there are now 3 food pellets remaining. The reward for this transition is 1. Assuming γ = 0.9, calculate the temporal difference (TD) error for this transition.**

The TD(temporal difference) error, δ can be calculated using the formula:

δ=[r+γ⋅a′max​Q(s′,a′)]−Q(s,a)

where the reward, r = 1, discount factor γ = 0.9 the new state has 3 food pellets remaining. An assumption is made that the new action state pair is a movement towards a food pellet and away from a ghost. Based on this then: f1 = 1, f2 = 1 and f3 = 3. Then:

Q(s′,a′)=2(1)+3(1)−0.5(3)=2+3−1.5=3.5

Then the TD can be calculated as:

δ= [1+0.9(3.5)] −3.0 = 1+3.15−3.0

δ= 1+3.15−3.0

δ=1.15

**c. Using the TD error from part b and a learning rate α = 0.2, calculate the updated weights after this experience.**

Given that α = 0.2 and δ = 1.15:

1. w1 = 2.0 + (0.2 x 1.15) = 2.23
2. w2 = 3.0 + (0.2 x 1.15) = 3.23
3. w3 = -0.5 + (0.2 x 1.15 x 4) = 0.42

**d. What might be a better feature than simply counting the number of food pellets? Briefly explain why.**

A better feature would be to check the actual distance to the food pellet rather than just checking the amount of food pellets. An effective method that can be used is one that was utilized in previous assignments where we made use of the Manhattan / Euclidean distance. This is overall a better feature as it gives spatial information to determine how far away a reward as the current feature of just counting the remaining food pellets doesn’t tell pacman if the food pellet is one step away or one hundred. This approach also creates a gradient that guides the pacman towards food more effectively as opposed to ‘randomly’ stumbling across a food pellet. The use of the distance to food pellet can be expanded upon and modified eg. use of the inverse of the distance value to smoothen the gradient. Another interesting use case is that the distance variable can enable development of a food density map which the pacman can use to determine the areas that are worth exploring as opposed to ‘barren’ areas.

# Question 5

**Exploration-Exploitation: (6 points)**  
In ε-greedy Q-learning, we've discussed how the exploration rate ε affects learning.

**a. What problems might occur if ε is set too high? What if it's set too low?**

**b. Many implementations decrease ε over time. Why is this a good strategy? How might you implement a schedule for decreasing ε?**

**c. Besides ε-greedy, describe one other approach to balancing exploration and exploitation in reinforcement learning.**