assignment1

December 8, 2023

[2233]:
```
import numpy as np
import math
```

[2234]:
```
label = np.array(
    [
        ["terminal","terminal","terminal","terminal","terminal"],
        ["unshaded","unshaded","terminal","unshaded","unshaded"],
        ["unshaded","unshaded","terminal","unshaded","unshaded"],
        ["unshaded","unshaded","unshaded","unshaded","goal"],
        ["unshaded","terminal","unshaded","unshaded","unshaded"],
        ["unshaded","terminal","unshaded","unshaded","unshaded"],
        ["terminal","terminal","terminal","terminal","terminal"]
    ]
)
```

[2235]:
```
N_ROW, N_COL = label.shape
```

[2236]:
```
gamma = 0.9
r_g = 5
r_r = -5

s0 = 2
actions = { "rightup" : np.array([-1,1]), "rightdown" : np.array([1,1]) }
```

[2237]:
```
def state_to_coordinates( state ):
    state -= 1
    column = int( state / N_ROW)
    row = state - column * N_ROW
    return row, column
```

[2238]:
```
def coordinates_to_state ( row, column ):
    return (column * N_ROW + row) + 1
```

[2239]:
```
def dynamics ( state_1, action_1 ):
    row_1, col_1 = state_to_coordinates( state_1 )
    label = label[row_1][col_1]
```

```
    if label in [ "terminal", "goal" ]:
        return state_1

    row_2 = row_1 + actions[action_1][0]
    col_2 = col_1 + actions[action_1][1]

    if ( row_2 > N_ROW-1 ) or ( row_2 < 0 ) or ( col_2 > N_COL-1 ) or ( col_2 <␣
 ↪0 ):
        return coordinates_to_state( row_1+1, col_1 )
    else:
        return coordinates_to_state( row_2, col_2 )
```

[2240]:
```
def helper(root, arr, ans):
    arr.append(root)
    left =  dynamics ( root, "rightup" )
    right =  dynamics ( root, "rightdown" )

    if root == left or root == right:
        # This will be only true when the node is leaf node and hence we will␣
 ↪update our ans array by inserting array arr which have one unique path from␣
 ↪root to leaf
        ans.append(arr.copy())
        del arr[-1]
        # after that we will return since we don't want to check after leaf node
        return

    # recursively going left and right until we find the leaf and updating the␣
 ↪arr and ans array simultaneously
    if left != right:
        helper(left, arr, ans)
        helper(right, arr, ans)
    else:
        helper(left, arr, ans)
    del arr[-1]


def Paths(root):
    ans = [] # creating answer in which each element is a array having one␣
 ↪unique path from root to leaf
    arr = [] # arr is a array which will have one unique path from root to leaf␣
 ↪at a time.arr will be updated recursively
    helper(root, arr, ans) # after helper function call our ans array updated␣
 ↪with paths so we will return ans array
    return ans

def printArray(paths, reward):
```

```
optimal_reward = 35*-5
len_optimal = 35
visited_state = set()
goal_optimal_reward = 35*-5
len_goal_optimal = 35
for path in paths:
    state_list = []
    label_list = []
    reward_list = []

    for state in path:
        visited_state.add(state)
        state_list.append(str(state))
        row,col = state_to_coordinates( state )
        label = flappyworld1[row][col]
        label_list.append(label)
        reward_list.append(str(reward[label]))

    print("path : ", end="")
    print(" ---> ".join(state_list), end="")
    print(" : length of shortest path : " + str(len(state_list)))
    print("label : ", end="")
    print(" ---> ".join(label_list))
    print("reward : ", end="")
    print(" + ".join(reward_list), end="")
    total_reward = sum( list(map(int, reward_list)) )
    print( " = " + str(total_reward) + " = total reward")
    if total_reward > optimal_reward:
        optimal_reward = total_reward
        if len_optimal > len(state_list):
            len_optimal = len(state_list)
    lastState = int( state_list[-1] )
    r,c = state_to_coordinates(lastState)
    if flappyworld1[r][c] == "goal" and total_reward > goal_optimal_reward:
        goal_optimal_reward = total_reward
        if len_goal_optimal > len(state_list):
            len_goal_optimal = len(state_list)

    print()
print("="*60)
print(str(len(visited_state)) + " traversed states : ", end=" ")
visited_state = [ str(state) for state in visited_state]
print(visited_state)
print("="*60)
```

0.0.1 Question 1-(a)

- PART1 : briefly explain what the optimal policy would be in Flappy World 1.

- PART2 : is the optimal policy unique ?
- PART3 : does the optimal policy depend on the value of the discount factor $\gamma \in [0, 1]$?

- PART4 : Explain your answer.

0.0.2 Hints

- What is the optimal policy? (the one that has the highest discounted sum of reward.)
- What is the difference between positive vs negative reward values of r_s ?
- Unique or not, list conditions / why you think it is unique.

0.0.3 Overview : WITHOUT considering the discount factor γ

We first get an overview by traverseing all possible paths from the starting state to a terminal state (either RED or GREEN) . - root : starting state 2 - leaf nodes : terminal nodes

```
r_s  = -4
print("="*30+"r_s = " + str(r_s)+"="*30)
reward = { "terminal" : r_r, "goal" : r_g, "unshaded" : r_s }
printArray(Paths(2),reward)
```

```
==============================r_s = -4==============================
path : 2 ---> 8 : length of shortest path : 2
label : unshaded ---> terminal
reward : -4 + -5 = -9 = total reward

path : 2 ---> 10 ---> 16 : length of shortest path : 3
label : unshaded ---> unshaded ---> terminal
reward : -4 + -4 + -5 = -13 = total reward

path : 2 ---> 10 ---> 18 ---> 24 ---> 30 ---> 31 ---> 32 : length of shortest path : 7
label : unshaded ---> unshaded ---> unshaded ---> unshaded ---> unshaded ---> unshaded ---> goa
reward : -4 + -4 + -4 + -4 + -4 + -4 + 5 = -19 = total reward

path : 2 ---> 10 ---> 18 ---> 24 ---> 32 : length of shortest path : 5
label : unshaded ---> unshaded ---> unshaded ---> unshaded ---> goal
reward : -4 + -4 + -4 + -4 + 5 = -11 = total reward

path : 2 ---> 10 ---> 18 ---> 26 ---> 32 : length of shortest path : 5
label : unshaded ---> unshaded ---> unshaded ---> unshaded ---> goal
reward : -4 + -4 + -4 + -4 + 5 = -11 = total reward

path : 2 ---> 10 ---> 18 ---> 26 ---> 34 ---> 35 : length of shortest path : 6
label : unshaded ---> unshaded ---> unshaded ---> unshaded ---> unshaded ---> terminal
reward : -4 + -4 + -4 + -4 + -4 + -5 = -25 = total reward
```

```
============================================================
12 traversed states :  ['32', '2', '34', '35', '8', '10', '16', '18', '24', '26', '30', '31']
============================================================
```

Observation from the route traversal

- there are 12 explored states:
  - staring state : 2
  - RED terminal states : 8, 16, 35
  - GREEN terminal state : 32
  - UNSHADED states : 10, 18, 24, 26, 30, 31, 34
- there are 6 distinct paths from the starting state 2 (root) to a terminal state (leaf node).
  - path A : $2 \to 8$
  - path B : $2 \to 10 \to 16$
  - path C : $2 \to 10 \to 18 \to 24 \to 30 \to 31 \to 32$
  - path D : $2 \to 10 \to 18 \to 24 \to 32$
  - path E : $2 \to 10 \to 18 \to 26 \to 32$
  - path F : $2 \to 10 \to 18 \to 26 \to 34 \to 35$

path	length	R_{acc}	$r_s = -4$	$r_s = -1$	$r_s = 0$	$r_s = 1$	ending state
A	1	$1 * r_s + r_r = 1r_s - 5$	-9 (max)	-6	-5	-4	RED
B	2	$2 * r_s + r_r = 2r_s - 5$	-13	-7	-5	-3	RED
C	6	$6 * r_s + r_g = 6r_s + 5$	-19	-1	5 (max)	11 (max)	GREEN
D	4	$4 * r_s + r_g = 4r_s + 5$	-11	1 (max)	5 (max)	9	GREEN
E	4	$4 * r_s + r_g = 4r_s + 5$	-11	1 (max)	5 (max)	9	GREEN
F	5	$5 * r_s + r_r = 5r_s - 5$	-25	-10	-5	0	RED

Let R_{acc} represents the reward accumulated along the path.

A policy is a function that maps S to A.
Let $\searrow$ represents the action "right and down".
Let $\nearrow$ represents the action "right and up".
We represent the function as a dictionary, where the key is the state, and the value is the action.

path	policy	states
A	{2:$\nearrow$}	$2 \to 8$
C	{2:$\searrow$,10:$\searrow$,18:$\nearrow$,24:$\nearrow$}	$2 \to 10 \to 18 \to 24 \to 30 \to 31 \to 32$
D	{2:$\searrow$,10:$\searrow$,18:$\nearrow$,24:$\searrow$}	$2 \to 10 \to 18 \to 24 \to 32$
E	{2:$\searrow$,10:$\searrow$,18:$\searrow$,26:$\nearrow$}	$2 \to 10 \to 18 \to 26 \to 32$

Without considering the discount factor γ, the optimal policy(ies) corresponds(x) to the path(s) that renders(x) the max R_{acc}.

r_s	optimal path	unique?
-4	A	unique

r_s	optimal path	unique?
-1	D & E	not unique
0	C & D & E	not unique
1	C	unique

0.0.4 Types of Paths (Policies)

We can categorize paths into 2 types. - $Type_G$: Paths end at a **GREEN** terminal state.
- $Type_R$: Paths end at a **RED** terminal state.

Define $L_{Gmax} = \max_{P \in Type_G} |P|$, the longest length among all paths that are in $Type_G$
Define $L_{Gmin} = \min_{P \in Type_G} |P|$, the shortest length among all paths that are in $Type_G$
Define $L_{Rmax} = \max_{P \in Type_R} |P|$, the longest length among all paths that are in $Type_G$
Define $L_{Rmin} = \min_{P \in Type_R} |P|$, the shortest length among all paths that are in $Type_G$

Here, the length of path equals to the number of actions in a policy excluding action taken at the terminal state.

path	length	R_{acc}	$r_s = -4$	$r_s = -1$	$r_s = 0$	$r_s = 1$	ending state	type
A	1	$1 * r_s + r_r = 1r_s - 5$	-9 (max)	-6	-5	-4	RED	type R
B	2	$2 * r_s + r_r = 2r_s - 5$	-13	-7	-5	-3	RED	type R
C	6	$6 * r_s + r_g = 6r_s + 5$	-19	-1	5 (max)	11 (max)	GREEN	type G
D	4	$4 * r_s + r_g = 4r_s + 5$	-11	1 (max)	5 (max)	9	GREEN	type G
E	4	$4 * r_s + r_g = 4r_s + 5$	-11	1 (max)	5 (max)	9	GREEN	type G
F	5	$5 * r_s + r_r = 5r_s - 5$	-25	-10	-5	0	RED	type R

$L_{Gmax} = \max_{P \in Type_G} |P| = \max\{|A|, |B|, |F|\} = \max\{1, 2, 5\} = 5$
$L_{Gmin} = \min_{P \in Type_G} |P| = \min\{|A|, |B|, |F|\} = \min\{1, 2, 5\} = 1$
$L_{Rmax} = \max_{P \in Type_R} |P| = \max\{|C|, |D|, |E|\} = \max\{6, 4, 4\} = 6$
$L_{Rmin} = \min_{P \in Type_R} |P| = \min\{|C|, |D|, |E|\} = \min\{6, 4, 4\} = 4$

if $r_s > 0, R_{acc} = \max\{L_{Rmax} * r_s + r_r, L_{Gmax} * r_s + r_g\} = \max\{5r_s - 5, 6r_s + 5\} = \max\{0, r_s + 10\} - 5 + 5r_s$
if $r_s = 0, R_{acc} = \max\{r_r, r_g\} = \max\{-5, 5\}$
if $r_s < 0, R_{acc} = \max\{L_{Rmin} * r_s + r_r, L_{Gmin} * r_s + r_g\} = \max\{1r_s - 5, 4r_s + 5\} = \max\{0, 3r_s + 10\} - 5 + r_s$

if $r_s > 0, r_s + 10$ is larger \$ $\rightarrow$\$ longest Type_G path(s) is(are) optimal
if $r_s = 0, r_g$ is larger \$ $\rightarrow$\$ all Type_G path(s) is(are) optimal
if $r_s < 0$,
- if $3r_s + 10 > 0$: $3r_s + 10$ is larger \$ $\rightarrow$\$ shortest Type_G path(s) is(are) optimal
- if $3r_s + 10 = 0$: \$ $\rightarrow$\$ shortest path(s) is(are) optimal
- if $3r_s + 10 < 0$: 0 is larger \$ $\rightarrow$\$ shortest Type_R path(s) is(are) optimal

if $r_s > 0$, longest path(s) that ends(x) at the GREEN state is(are) optimal
if $r_s = 0$, all path(s) that ends(x) at the GREEN state is(are) optimal
if $r_s < 0$,
- if $r_s > -10/3$: shortest path(s) is(are) that ends(x) at the GREEN state is optimal

- if $r_s = -10/3$: shortest path(s) is(are) optimal
- if $r_s < -10/3$: shortest path(s) that ends(x) at a RED state is(are) optimal

$r_s = 1 > 0$, path C is the longest path that ends at the GREEN state \$ →\$ path C is optimal.
$r_s = 0$, path C,D,E end at the GREEN state \$ →\$ path C,D,E are optimal.
$r_s = -1 > -10/3$: path D,E are the shortest paths that end at the GREEN state \$ →\$ path D,E, are optimal.
$r_s = -4 < -10/3$: path A is the shortest path that ends at a RED state \$ →\$ path A is optimal.

We may preview question 1-(b)

```
What value of r_s from 1-(a) would cause the optimal policy to return the shortest path to the
```

```
Answer : r_s = -1
```

0.0.5 Reduction of the search space

There are 12 explored states: - staring state : 2 - RED terminal states : 8, 16, 35 - GREEN terminal state : 32 - UNSHADED states : 10, 18, 24, 26, 30, 31, 34

For a terminal state $s_{terminal}$, since taking an action on $s_{terminal}$ will end the episode, there is NO next state s' to transition to.
Consequently, formula $V_k^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s) V_{k-1}^\pi(s')$ can be reduced to $V_k^\pi(s_{terminal}) = R(s_{terminal})$
This implies that $V_k^\pi(s_{terminal})$ will remain invariant after the 1st iteration.
- RED : $V_k^\pi(s_{red}) = R(s_{red}) = r_r = -5, \forall k > 1$
- GREEN : $V_k^\pi(s_{green}) = R(s_{green}) = r_g = 5, \forall k > 1$

So, there is no need to take terminal states into the calculation of convergence as their values/utilities remain invariant $V^\pi(s_{terminal}) = R(s_{terminal})$.
- RED : $V^\pi(s_{red}) = R(s_{red}) = r_r = -5$
- GREEN : $V^\pi(s_{green}) = R(s_{green}) = r_g = 5$

Also, we know that, 1. taking any action on state 31 will result in a transition to state 32
$V_1(31) = R(31) + \gamma * P(32|31) * V_0(32)$
$V_1(31) = R(31) + \gamma * P(32|31) * r_g$
$V_1(31) = R(31) + \gamma * r_g$
$V_1(31) = r_s + \gamma * r_g$
$V(31) = r_s + \gamma * r_g$ is a constant

2. taking any action on state 30 will result in a transition to state 31
   $V_1(30) = R(30) + \gamma * P(32|31) * V_0(31)$
   $V_1(30) = R(31) + \gamma * P(32|31) * (r_s + \gamma * r_g)$
   $V_1(30) = R(31) + \gamma * (r_s + \gamma * r_g)$
   $V_1(30) = r_s + \gamma * (r_s + \gamma * r_g)$
   $V_1(30) = r_s * (1 + \gamma) + \gamma^2 * r_g$
   $V(30) = r_s * (1 + \gamma) + \gamma^2 * r_g$ is a constant

3. taking any action on state 34 will result in a transition to state 35
   $V_1(34) = R(34) + \gamma * P(35|34) * V_0(35)$
   $V_1(34) = R(31) + \gamma * P(32|31) * r_r$
   $V_1(34) = R(31) + \gamma * r_r$

$V_1(34) = r_s + \gamma * r_r$
$V(34) = r_s + \gamma * r_r$ is a constant

Therefore, the only states we need to consider in the Policy Iteration Algorithm are : [2, 10, 18, 24, 26]

[2241]:
```
policy = np.zeros((N_ROW,N_COL), dtype='U1')
value = np.zeros((N_ROW,N_COL))
```

[2242]:
```
for row in range(N_ROW):
    for col in range(N_COL):
        if label[row][col] == "terminal":
            value[row][col] = -5
            policy[row][col] = "T"
        elif label[row][col] == "goal":
            value[row][col] = 5
            policy[row][col] = "G"
value
```

[2242]:
```
array([[-5., -5., -5., -5., -5.],
       [ 0.,  0., -5.,  0.,  0.],
       [ 0.,  0., -5.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  5.],
       [ 0., -5.,  0.,  0.,  0.],
       [ 0., -5.,  0.,  0.,  0.],
       [-5., -5., -5., -5., -5.]])
```

0.0.6 Table Filling

[2243]:
```
def dp(gamma,rs):
    # For state that will run into a wall after taking an action.
    for row in range(N_ROW-1,-1,-1):
        isTerminal = (label[row][N_COL-1] == "terminal")
        isGoal = (label[row][N_COL-1] == "goal")
        if not (isTerminal or isGoal):
            value[row][N_COL-1] = rs + gamma * value[row+1][N_COL-1]
            policy[row][N_COL-1] = "W"

    # For other states.
    for col in range(N_COL-2,-1,-1):
        for row in range(N_ROW-1,-1,-1):
            label_ = label[row][col]
            isTerminal = (label_ == "terminal")
            isGoal = (label_ == "goal")
            if not (isTerminal or isGoal):
                valueUp = value[row-1][col+1]
                valueDown = value[row+1][col+1]
                if valueUp > valueDown:
```

```
                    policy[row][col] = "U" #going "right&Uown" is optimal
                elif valueUp < valueDown:
                    policy[row][col] = "D" #going "right&Down" is optimal
                else:
                    policy[row][col] = "E" #Either going "right&up" or going␣
↪"right&down" is optimal
                value[row][col] = rs + gamma * max(valueUp,valueDown)
```

0.0.7 Print the path under the policy starting from the start state.

- if there is a state when "E" occurs, meaning that either going R&U or R&D is optimal, then, the policy is NOT unique.

[2244]:
```
def printPath(startstate,policy,tiebreaker):
    isUnique = True
    path = [str(startstate)]
    row, col = state_to_coordinates(startstate)
    while True:
        if policy[row][col] == "U":
            row-=1
            col+=1
        elif policy[row][col] == "D":
            row+=1
            col+=1
        elif policy[row][col] == "E":
            row+=tiebreaker
            col+=1
            isUnique = False
        elif policy[row][col] == "W":
            row+=1
        path.append( str( coordinates_to_state ( row, col ) ) )
        if label[row][col] == "terminal" or label[row][col] == "goal":
            break
    if isUnique:
        print("the path is unique.")
    else:
        print("the path is NOT unique.")
    print ("--->".join(path))
```

0.0.8 1-(a)-1. Optimal Policy under $r_s = -4$

- "T" means a terminal state, any action will end the episode.
- "G" means the goal state, any action will end the episode.
- "E" means either going "Right and Up" or "Right and Down" is optimal
- "U" means either going "Right and Up" is optimal
- "D" means either going "Right and Down" is optimal
- "W" means that tacking an action from this state will result in hitting a wall

[2245]:
```
rs = -4
gamma = 0.9
dp(gamma,rs)
policy
```

[2245]:
```
array([['T', 'T', 'T', 'T', 'T'],
       ['U', 'E', 'T', 'D', 'W'],
       ['E', 'D', 'T', 'D', 'W'],
       ['D', 'U', 'E', 'U', 'G'],
       ['D', 'T', 'U', 'U', 'W'],
       ['E', 'T', 'U', 'D', 'W'],
       ['T', 'T', 'T', 'T', 'T']], dtype='<U1')
```

0.0.9 1-(a)-1. Path (sequence of states) from start state s_2 to a terminal state under the optimal policy under $r_s = -4$.

[2246]:
```
printPath(2,policy,1)
```

```
the path is unique.
2--->8
```

0.0.10 1-(a)-2. Optimal Policy under $r_s = -1$

- "T" means a terminal state, any action will end the episode.
- "G" means the goal state, any action will end the episode.
- "E" means either going "Right and Up" or "Right and Down" is optimal
- "U" means either going "Right and Up" is optimal
- "D" means either going "Right and Down" is optimal
- "W" means that tacking an action from this state will result in hitting a wall

[2247]:
```
rs = -1
gamma = 0.9
dp(gamma,rs)
policy
```

[2247]:
```
array([['T', 'T', 'T', 'T', 'T'],
       ['D', 'E', 'T', 'D', 'W'],
       ['D', 'D', 'T', 'D', 'W'],
       ['U', 'D', 'E', 'U', 'G'],
       ['U', 'T', 'U', 'U', 'W'],
       ['E', 'T', 'U', 'D', 'W'],
       ['T', 'T', 'T', 'T', 'T']], dtype='<U1')
```

0.0.11 1-(a)-2. Path (sequence of states) from start state s_2 to a terminal state under the optimal policy under $r_s = -1$.

[2248]:
```
startState = 2
printPath(startState, policy,1)
```

```
the path is NOT unique.
2--->10--->18--->26--->32
```

0.0.12 1-(a)-3. Optimal Policy under $r_s = 0$

- "T" means a terminal state, any action will end the episode.
- "G" means the goal state, any action will end the episode.
- "E" means either going "Right and Up" or "Right and Down" is optimal
- "U" means either going "Right and Up" is optimal
- "D" means either going "Right and Down" is optimal
- "W" means that tacking an action from this state will result in hitting a wall

[2249]:
```
rs = 0
gamma = 0.9
dp(gamma,rs)
policy
```

[2249]:
```
array([['T', 'T', 'T', 'T', 'T'],
       ['D', 'E', 'T', 'D', 'W'],
       ['D', 'D', 'T', 'D', 'W'],
       ['U', 'D', 'E', 'U', 'G'],
       ['U', 'T', 'U', 'U', 'W'],
       ['E', 'T', 'U', 'U', 'W'],
       ['T', 'T', 'T', 'T', 'T']], dtype='<U1')
```

0.0.13 1-(a)-3. Path (sequence of states) from start state s_2 to a terminal state under the optimal policy under $r_s = 0$.

[2250]:
```
startState = 2
printPath(startState, policy,1)
```

```
the path is NOT unique.
2--->10--->18--->26--->32
```

0.0.14 1-(a)-4. Optimal Policy under $r_s = 1$

- "T" means a terminal state, any action will end the episode.
- "G" means the goal state, any action will end the episode.
- "E" means either going "Right and Up" or "Right and Down" is optimal
- "U" means either going "Right and Up" is optimal
- "D" means either going "Right and Down" is optimal
- "W" means that tacking an action from this state will result in hitting a wall

[2251]:
```
rs = 1
gamma = 0.9
dp(gamma,rs)
policy
```

[2251]:
```
array([['T', 'T', 'T', 'T', 'T'],
       ['D', 'E', 'T', 'D', 'W'],
       ['D', 'D', 'T', 'U', 'W'],
       ['U', 'D', 'U', 'U', 'G'],
       ['U', 'T', 'U', 'U', 'W'],
       ['E', 'T', 'U', 'U', 'W'],
       ['T', 'T', 'T', 'T', 'T']], dtype='<U1')
```

0.0.15 1-(a)-4. Path (sequence of states) from start state s_2 to a terminal state under the optimal policy under $r_s = 1$.

[2252]:
```
startState = 2
printPath(startState, policy,1)
```

```
the path is unique.
2--->10--->18--->24--->30--->31--->32
```

0.0.16 1-(b)

Consider different possible grids and grid shading (with walls at the border similar to Flappy World 1) in which the green target square is reacheable from the starting square.

1-(b)-1. What value of rs from part (a) would cause the optimal policy to return the shortest path to the green target square in all cases?
$r_s = -1$

1-(b)-2. Find the optimal value function for each square in Flappy World 1 using this value of rs? i.e. show the value functions for each square.

[2253]:
```
rs = -1
gamma = 0.9
dp(gamma,rs)
value
```

[2253]:
```
array([[-5.     , -5.     , -5.     , -5.     , -5.     ],
       [-0.1585 , -5.5    , -5.     ,  2.15   ,  2.15   ],
       [-1.14265,  0.935  , -5.     ,  3.5    ,  3.5    ],
       [-0.1585 , -0.1585 ,  2.15   ,  2.15   ,  5.     ],
       [-1.14265, -5.     ,  0.935  ,  3.5    , -5.95   ],
       [-5.5    , -5.     ,  2.15   , -5.5    , -5.5    ],
       [-5.     , -5.     , -5.     , -5.     , -5.     ]])
```

1-(a)

1. For all terminal states s_{term}, since taking an action on s_{term} will end the episode,

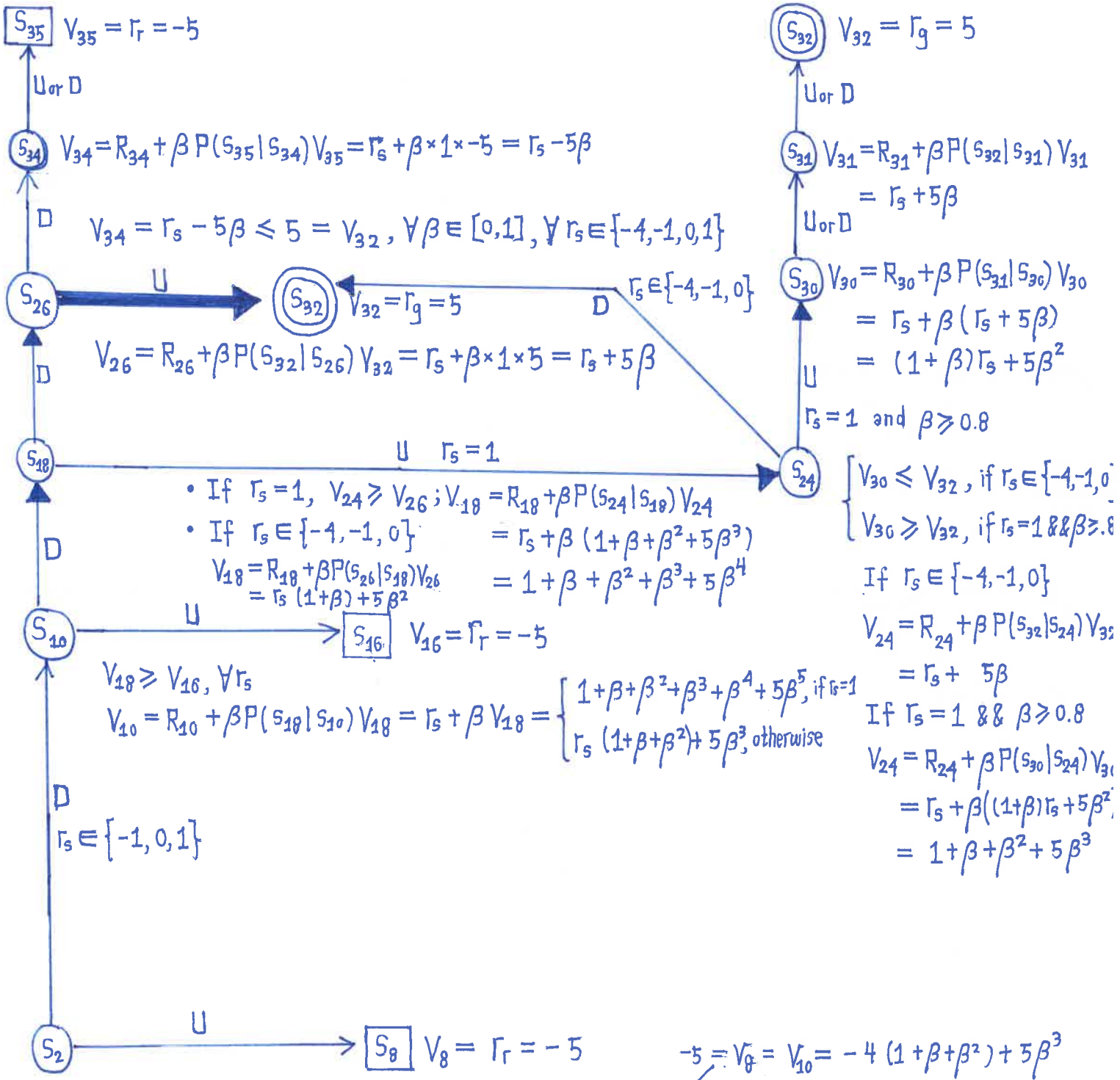
$$V^{\pi^*}(s_{term}) = R(s_{term}) + \sum_{s'} P(s'|s_{term}) V^{\pi^*}(s') = R(s_{term}) = \begin{cases} r_g, & \text{if } s_{term} \text{ is the GREEN termal state} \\ r_r, & \text{if } s_{term} \text{ is a REN terminal state} \end{cases}$$

2. denote moving Up and Right as U

   denote moving Down and Right as D

3. denote the discount factor gamma γ as β to avoid being confused with the reward r_s, r_g, r_r

4. in the following discussion, we will not write the superscript R^{π^*}, $P^{\pi^*}(s'|s)$, ... for simplicity.



The optimal policy is a function that maps S to A. Since the policy is Deterministic and for simplicity, we represent the optimal policy by the sequence of states if we follow the optimal policy under each r_s 1-(a)

$r_s = 1$: 2 —D→ 10 —D→ 18 —U→ 24 —U→ 30 —U or D→ 31 —U or D→ 32

$r_s = 0$: 2 —D→ 10 —D→ 18 —U→ 24 —D→ 32; 18 —D→ 26 —U→ 32

$r_s = -4$ && $\beta \leq \beta_{sol} \approx 0.2$

$r_s = -1$

for $r_s = 0$, $\beta = 0$ the following one path is also optimal

2 —D→ 10 —D→ 18 —U→ 24 —U→ 30 —U or D→ 31 —U or D→ 32

$r_s = -4$ && $\beta \geq \beta_{sol}$: 2 —U→ 8

Only for $r_s = -4$, the optimal policy is dependent of the value of the discount factor β

For $\begin{cases} (r_s = 0 \text{ and } \beta = 0), \\ r_s = 1 \\ (r_s = -4 \text{ and } \beta \geq \beta_{sol}) \end{cases}$ the optimal policy is Unique

For $\begin{cases} r_s = 0 \\ (r_s = -4 \text{ and } \beta \leq \beta_{sol}), \\ r_s = \end{cases}$ the optimal policy is NOT Unique since there are two paths that lead to the same $V^{\pi^*}(s)$

From 1-(a), when we discussed the special case $\gamma = 0$,

we the **Optimal Policy** for each value of r_s.

$r_s > 0$: LONGEST path that ends at the GREEN terminal state

$r_s = 0$: ALL path that ends at the GREEN terminal state.

$\frac{-10}{3} < r_s < 0$: SHORTEST path that ends at the GREEN terminal state.

$r_s = \frac{-10}{3}$ SHORTEST path

$r_s < \frac{-10}{3}$ SHORTEST path that ends at a RED terminal state

---

in the last part of 1-(a) we validate the similar idea for $\gamma = 0.9$

length = 6 $\boxed{r_s = 1}$ (2) → (10) → (18) → (24) → (30) → (31) → (32)

length = 4 $\boxed{r_s = 0}$ $\boxed{r_s = -1}$ (2) → (10) → (18) → (24) → (32), (18) → (26) → (32)

length = 1 $\boxed{r_s = -1}$ (2) → (8)

⇒ $\boxed{r_s = -1}$ would cause the optimal policy return the shortest path to the GREEN target square in all cases

---

1-(b)-3. What is the optimal action from square 27?
GO RIGHT AND DOWN to S_{35}

[2254]:
```
policy
```

[2254]:
```
array([['T', 'T', 'T', 'T', 'T'],
       ['D', 'E', 'T', 'D', 'W'],
       ['D', 'D', 'T', 'D', 'W'],
       ['U', 'D', 'E', 'U', 'G'],
       ['U', 'T', 'U', 'U', 'W'],
       ['E', 'T', 'U', 'D', 'W'],
       ['T', 'T', 'T', 'T', 'T']], dtype='<U1')
```

[2255]:
```
startState = 27
printPath(startState, policy,1)
```

```
the path is unique.
27--->35
```

0.0.17 1-(c) [5 points (Written)]

Now consider Flappy World 2.
It is the same as Flappy World 1,
except there are no walls on the right and left sides.
Going past the right end of Flappy World 2 simply loops you to left hand side.
Take a look at Figure 2b for a successful run by Karel in Flappy World 2.

Let $r_s \in \{-4, -1, 0, 1\}$.

```
["1","8","15","22","29"],
["2","9","16","23","30"],
["3","10","17","24","31"],
["4","11","18","25","32"],
["5","12","19","26","33"],
["6","13","20","27","34"],
["7","14","21","28","35"]
```

- starting from s_2, we can see that s_{20} and s_{23} will never be visited; therefore, we exclude them from the discussion.

- $V(s_{terminal}) = R(s_{terminal}) = r_r = -5$

- $V(s_{goal}) = R(s_{goal}) = r_g = 5$

- $V(s_9) = R(s_9) + \gamma * R(s_{15}) = r_s - 5 * \gamma$
- $V(s_6) = R(s_{12}) + \gamma * R(s_{12}) = r_s - 5 * \gamma$

0.0.18 Annotations

[2256]:
```
label_array = np.array(
    [
        ["terminal","terminal","terminal","terminal","terminal"],
        ["unshaded","invariant","terminal","unvisited","unshaded"],
        ["unshaded","unshaded","terminal","unshaded","unshaded"],
        ["unshaded","unshaded","unshaded","unshaded","goal"],
        ["unshaded","terminal","unshaded","unshaded","unshaded"],
        ["invariant","terminal","unvisited","unshaded","unshaded"],
        ["terminal","terminal","terminal","terminal","terminal"]
    ]
)
# shape of Flappy World 1.
N_ROW = label_array.shape[0] #7
N_COL = label_array.shape[1] #5

# state_array : state space
S = []
for row in range(N_ROW):
    for col in range(N_COL):
        if label_array[row][col] == "unshaded":
            S.append((row,col))
S = np.asarray(S)
nS = len(S) # size = 16

# action_space : action space
A = [False,True] #False means DOWN; True means Up;
nA = len(A) # size = 2

# hyper-parameters for the policy iteration algorithm
gamma, rs, rg, rr = 0.9, -4, 5,-5

value_array = np.zeros( ( N_ROW, N_COL ) )
for row in range(N_ROW):
    for col in range(N_COL):
        label = label_array[row][col]
        if label == "goal":
            value_array[row][col] = rg
        elif  label == "terminal":
            value_array[row][col] = rr
        elif  label == "invariant":
            value_array[row][col] = rs + gamma * rr
value_array
```

[2256]:
```
array([[-5. , -5. , -5. , -5. , -5. ],
       [ 0. , -8.5, -5. ,  0. ,  0. ],
```

```
[ 0. ,  0. , -5. ,  0. ,  0. ],
[ 0. ,  0. ,  0. ,  0. ,  5. ],
[ 0. , -5. ,  0. ,  0. ,  0. ],
[-8.5, -5. ,  0. ,  0. ,  0. ],
[-5. , -5. , -5. , -5. , -5. ]])
```

0.0.19 Dynamics/Transitions for NON-TERMINAL states.

Simulate the transition for NON-TERMINAL states in Flappy World 3.

Args:
state (int) : starting state (from-state) of the transition
action (bool) : action taken on the starting state : False means DOWN; True means UP

Returns:
(int) : row index of ending state (to-state) of the transition
(int) : column index of ending state (to-state) of the transition

[2257]:
```
def transition ( row_1, col_1, action ):
    # A = [False,True] #False means DOWN; True means Up;
    if action:
        return row_1 - 1, ((col_1 + 1) % N_COL)
    else:
        return row_1 + 1, ((col_1 + 1) % N_COL)
```

0.0.20 Policy Evaluation

Evaluate the value function from a given policy.

Args:
policy (np.array[nS]): The policy to evaluate. Maps states to actions.
tol (float): Terminate policy evaluation when max |value_function(s) - prev_value_funct

Returns:
value_function (np.ndarray[nS]):
The value function of the given policy,
where value_function[s] is the value of state s.

[2258]:
```
def policy_evaluation( value_function, state_space, policy, rs, gamma=0.9,␣
↪tol=1e-3 ):
    prev_value_function = np.copy( value_function )
    while True:
        for fromStateIndex, action in enumerate(policy):
            fromRow, fromCol = state_space[ fromStateIndex ]
            value_function[ fromRow, fromCol ] = rs + gamma *␣
↪prev_value_function[ transition ( fromRow, fromCol, action ) ]
```

```
        # Terminate policy evaluation when max |value_function(s) -␣
↪prev_value_function(s)| < tol
        if np.max( np.abs(value_function - prev_value_function)) < tol:
            break

        prev_value_function = np.copy( value_function )

    return value_function
```

0.0.21 Policy Improvement

Given the value function from policy improve the policy.

```
    Args:
            value_from_policy (np.ndarray): The value calculated from the policy
            policy (np.array): The previous policy

    Returns:
            new_policy (np.ndarray[nS]): An array of integers. Each integer is the optimal
            action to take in that state according to the environment dynamics and the
            given value function.
```

[2259]:
```
def policy_improvement( valueFunction, stateSpace, actionSpace, policy, rs,␣
↪gamma=0.9):
    Q_ = np.zeros( ( 2, len(stateSpace) ) )
    # Q_[0] : for DOWN Q-Value
    # Q_[1] : for UP Q-value

    for action in actionSpace:
        for fromStateIndex, fromState in enumerate(stateSpace):
            fromRow, fromCol = fromState
            Q_[int(action)][fromStateIndex] = rs + gamma * valueFunction[␣
↪transition ( fromRow, fromCol, action ) ]
    return Q_[0]<Q_[1]
```

[2260]:
```
def completePolicy(S,p_pi):
    policy = np.zeros((N_ROW,N_COL), dtype='U1')
    for i in range(N_ROW):
        for j in range(N_COL):
            policy[i,j] = "_"
    for index, action in enumerate(p_pi):
        fromRow, fromCol = S[index]
        policy[fromRow, fromCol] = "U" if action else "D"
    print(policy)
    return policy
```

0.0.22 Policy Iteration

- (deterministic) Bellman backup for a particular policy π, this is a part of policy evaluation because this is trying to figure out how good is a particular policy in a decision process.)
  - Iteration of Policy Evaluation : $V_k^\pi(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s'|s) V_{k-1}^\pi(s')$
- optimal policy : $\pi^*(s) = \arg\max_\pi V^{\pi(s)}$
  - There exists a unique optimal value function, but the optimal policy MAY NOT be unique.
- Policy Search :
  - Number of deterministic policies $= |A|^{|S|} = 2^{12} = 4096$
- Policy Iteration :
  - set i = 0
  - initialize $\pi_0(s)$ randomly for all states s.
  - while i == 0 or $\|\pi_i - \pi_{i-1}\|_1 > 0$
    - $V^{\pi_i} \leftarrow$ MDP V function policy **evaluatio** of π_i
    - $ $_{i+1} \leftarrow$ $ policy **improvement**
    - i $\leftarrow$ i + 1
- State-action value of a policy π
  - $Q^\pi(s,a) = R(s,a) + \gamma * \sum_{s \in S} P(s'|s) V^\pi(s')$
- Compute State-action value of a policy π_i
  - For $s S, a A: $ $Q^{\pi_i}(s,a) = R(s,a) + \gamma * \sum_{s \in S} P(s'|s) V^{\pi_i}(s')$
- Compute new policy $\pi_{i+1}, \forall s \in S$:
  - $\pi_{i+1}(s) = argmax_a Q^{\pi_i}(s,a), \forall s \in S$

[2261]:
```
def policy_iteration( value, stateSpace, actionSpace, rs, gamma=0.9, tol=1e-3):
    policy = np.zeros( len(stateSpace), dtype=bool )
    iteration = 0
    while True:
        value = policy_evaluation( value, stateSpace, policy, rs, gamma, tol )
        policy_improved = policy_improvement( value, stateSpace, actionSpace,␣
↪policy, rs, gamma )

        if np.sum(policy_improved != policy) == 0:
            break

        policy = policy_improved

    return value, policy
```

Starting in square 3,
for each of the possible values of rs:
1-(c)-1. Briefly explain what the optimal policy would be in Flappy World 2?

[2262]:
```
def show_policy(rs,startRow,startCol):
    print( "The value function is" )
    print("="*30)
    V_pi, p_pi = policy_iteration( value_array, S, A, rs, gamma=0.9, tol=1e-3)
```

```
    print(V_pi)
    print("="*30)
    print( "The policy is" )
    print("="*30)
    p_pi = completePolicy(S,p_pi)
    print("="*30)

    if (rs<0):
        row, col = startRow, startCol
        path = [ str(coordinates_to_state(row,col) ) ]
        reward = V_pi[row,col]

        while True:
            action = p_pi[row][col]
            if action == "U":
                row -= 1
                col = (col+1)% N_COL
                reward += V_pi[row,col]
                path.append( str(coordinates_to_state(row,col) ) )
            elif action == "D":
                row += 1
                col = (col+1)% N_COL
                reward += V_pi[row,col]
                path.append( str(coordinates_to_state(row,col) ) )
            else:
                break
        print( " ---> ".join(path) )
        print("total reward = ")
        print(reward)
```

0.0.23 case 1 : $r_s = -4$

The policy is to ends through a RED terminal state.

```
show_policy(-4,1,0)
```

```
 - The value function is
==========================================
[[ -5.     -5.     -5.     -5.     -5.   ]
 [ -8.5    -8.5    -5.      0.     -8.5  ]
 [-11.65   -7.195  -5.      0.5   -11.65 ]
 [ -8.5    -8.5    -3.55  -14.485   5.   ]
 [ -8.5    -5.    -11.65    0.5   -11.65 ]
 [ -8.5    -5.      0.     -8.5    -8.5  ]
 [ -5.     -5.     -5.     -5.     -5.   ]]
==========================================

 - The policy is
```

```
========================
[['_' '_' '_' '_' '_']
 ['U' '_' '_' '_' 'U']
 ['D' 'D' '_' 'D' 'D']
 ['D' 'U' 'D' 'D' '_']
 ['D' '_' 'D' 'U' 'D']
 ['_' '_' '_' 'D' 'D']
 ['_' '_' '_' '_' '_']]
========================
one of the paths is : 3 ---> 11 ---> 17
total reward = -25.15
```

0.0.24 case 2 : $r_s = -1$

```
show_policy(-1,1,0)
```

```
The policy is to ends through a GREEN terminal state.

The value function is
================================================================
[[-5.         -5.         -5.         -5.         -5.        ]
 [-0.1585     -8.5        -5.          0.         -4.7698234 ]
 [-4.18869267  0.935      -5.          3.5        -1.14265   ]
 [-0.1585     -3.54299185  2.15       -2.028385    5.        ]
 [-4.18869267 -5.         -2.8255465   3.5        -1.14265   ]
 [-8.5        -5.          0.         -2.028385   -4.7698234 ]
 [-5.         -5.         -5.         -5.         -5.        ]]
================================================================

The policy is
========================
[['_' '_' '_' '_' '_']
 ['D' '_' '_' '_' 'D']
 ['D' 'D' '_' 'D' 'D']
 ['U' 'D' 'D' 'D' '_']
 ['U' '_' 'D' 'U' 'U']
 ['_' '_' '_' 'U' 'U']
 ['_' '_' '_' '_' '_']]
========================
one of the paths is : 3 ---> 11 ---> 19 ---> 27 ---> 33 ---> 4 ---> 10 ---> 18 ---> 26 ---> 32
total reward = -2.301766015
```

0.0.25 case 3 : $r_s = 0$

The policy is to ends through a GREEN terminal state.

```
show_policy(0,1,0)
The value function is
================================================================
```

```
[[-5.         -5.         -5.         -5.         -5.        ]
 [ 3.2805     -8.5        -5.          0.          1.7433922 ]
 [ 1.93710245  3.645      -5.          4.5         2.95245   ]
 [ 3.2805      2.15233605  4.05        2.657205    5.        ]
 [ 1.93710245 -5.          2.3914845   4.5         2.95245   ]
 [-8.5        -5.          0.          2.657205    1.7433922 ]
 [-5.         -5.         -5.         -5.         -5.        ]]
================================================================

The policy is
======================
[['_' '_' '_' '_' '_']
 ['D' '_' '_' '_' 'D']
 ['D' 'D' '_' 'D' 'D']
 ['U' 'D' 'D' 'D' '_']
 ['U' '_' 'D' 'U' 'U']
 ['_' '_' '_' 'U' 'U']
 ['_' '_' '_' '_' '_']]
======================
one of the path is : 3 ---> 11 ---> 19 ---> 27 ---> 33 ---> 4 ---> 10 ---> 18 ---> 26 ---> 32
total reward = 32.566077995
```

0.0.26 case 4 : $r_s = 1$

The policy is to not ever terminate

```
show_policy(1,1,0)
```

```
The value function is
================================================================
[[-5.         -5.         -5.         -5.         -5.        ]
 [ 9.99928329 -8.5        -5.          0.          9.99890763]
 [ 9.99878625  9.99920366 -5.          9.99901686  9.99935496]
 [ 9.99928329  9.99865139  9.99911518  9.99941947  5.        ]
 [ 9.99878625 -5.          9.99947752  9.99901686  9.99935496]
 [-8.5        -5.          0.          9.99941947  9.99890763]
 [-5.         -5.         -5.         -5.         -5.        ]]
================================================================

The policy is
======================
[['_' '_' '_' '_' '_']
 ['D' '_' '_' '_' 'D']
 ['D' 'D' '_' 'U' 'D']
 ['U' 'D' 'D' 'D' '_']
 ['U' '_' 'D' 'D' 'U']
 ['_' '_' '_' 'U' 'U']
```

```
 ['_' '_' '_' '_' '_']]
======================
```

```
total reward will become infinity.
```

0.0.27 Response to 1-(c)-1

- **$r_s = -4 :$** The optimal policy is to terminate through a red square
- **$r_s = -1 :$** The optimal policy is to terminate through a green square
- **$r_s = 0 :$** The optimal policy is to terminate through a green square
- **$r_s = 1 :$** The optimal policy is to not ever terminate

Once again consider different grids and grid shading
(without walls at either end similar to Flappy World 2)
in which the green target square is reacheable from the starting square.

1-(c)-2. What is the value of rs that would cause the optimal policy to return the shortest path to the green target square for all cases? $r_s = -1$
Explanation :
Though both the policie for $r_s = -1$ and "r_s = 0" are to terminate through a green square.
In $r_s = -1$, the value functions for non-terminal states are mostly NEGATIVE, leading to potential shorter path to reach the green state.
In $r_s = 0$, the value functions for non-terminal states are mostly POSITIVE, leading to potential longer path to reach the green state.

1-(c)-3. Find the optimal value for each square in Flappy World 2 using the value of rs,that would cause the optimal policy to return the shortest path to the green target square for all cases? For $r_s = -1$

```
show_policy(-1,5,3)
```
```
The value function is
```
```
=====================================================================
[[-5.         -5.         -5.         -5.         -5.        ]
 [-0.1585     -8.5        -5.          0.         -4.7698234 ]
 [-4.18869267  0.935      -5.          3.5        -1.14265   ]
 [-0.1585     -3.54299185  2.15       -2.028385    5.        ]
 [-4.18869267 -5.         -2.8255465   3.5        -1.14265   ]
 [-8.5        -5.          0.         -2.028385   -4.7698234 ]
 [-5.         -5.         -5.         -5.         -5.        ]]
=====================================================================
```

```
The policy is
```
```
======================
[['_' '_' '_' '_' '_']
 ['D' '_' '_' '_' 'D']
 ['D' 'D' '_' 'D' 'D']
 ['U' 'D' 'D' 'D' '_']
 ['U' '_' 'D' 'U' 'U']
 ['_' '_' '_' 'U' 'U']
```

```
 ['_' '_' '_' '_' '_']]
=======================
```

1-(c)-4. What is the optimal action from square 27? Hint:
There are three possible long-term behaviours,
1. terminate through a red square
2. terminate through a green square
3. do not ever terminate

Consider these cases when formulating the optimal policy for each value of rs.

```
show_policy(-1,5,3)
The value function is
==============================
[[-5.         -5.         -5.         -5.         -5.        ]
 [-0.1585     -8.5        -5.          0.         -4.7698234 ]
 [-4.18869267  0.935      -5.          3.5        -1.14265   ]
 [-0.1585     -3.54299185  2.15       -2.028385    5.        ]
 [-4.18869267 -5.         -2.8255465   3.5        -1.14265   ]
 [-8.5        -5.          0.         -2.028385   -4.7698234 ]
 [-5.         -5.         -5.         -5.         -5.        ]]
==============================
The policy is
==============================
[['_' '_' '_' '_' '_']
 ['D' '_' '_' '_' 'D']
 ['D' 'D' '_' 'D' 'D']
 ['U' 'D' 'D' 'D' '_']
 ['U' '_' 'D' 'U' 'U']
 ['_' '_' '_' 'U' 'U']
 ['_' '_' '_' '_' '_']]
==============================
the optimal action is to go "right and up" and follows the following path to reach a GREEN ter
27 ---> 33 ---> 4 ---> 10 ---> 18 ---> 26 ---> 32
total reward = 8.255465000000001
```

1-(d) [5 points (Written)]

Consider a general MDP < S , A, P , R, γ >, and in this case assume that the horizon H is infinite (so there is no termination).

A policy π in this MDP induces a value function V^{π} (lets refer to this as V^{π}_{old}).

Now suppose we have the same MDP where all rewards have a constant c added to them and then have been scaled by a (i.e. $r_{new} = a * (c + r_{old})$).

Can you come up with an expression for the new value function V^{π}_{new} induced by \pi in this second MDP in terms of V^{π}_{old}, c, a, and γ?$

You can start this question with the expression for the original value function and consider how this expression would change for scaled rewards:

$V^{\pi}_{old}(s) = \mathbb{E}_{\pi}[G_{old,t}|x_t = s]$
Where the return is defined as the discounted sum of rewards :
$G_{old,t} = \sum_{k=0}^{k=\infty} \gamma^k * r_{t+k+1}$

Now consider how we would rewrite the following expression in terms of the original.
$V^{\pi}_{new}(s) = \mathbb{E}_{\pi}[G_{new,t}|x_t = s]$
Where $G_{new,t}$ is comprised of the rewards which have been translated and scaled.

Response

$V^{\pi}_{old}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{t=\infty} \gamma^t r_t | s_0 = s]$
$V^{\pi}_{new}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{t=\infty} \gamma^t a(r_t + c) | s_0 = s]$
$V^{\pi}_{new}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{t=\infty} \gamma^t a r_t | s_0 = s] + \mathbb{E}_{\pi}[\sum_{t=0}^{t=\infty} \gamma^t ac | s_0 = s]$
$V^{\pi}_{new}(s) = a\mathbb{E}_{\pi}[\sum_{t=0}^{t=\infty} \gamma^t r_t | s_0 = s] + ac\mathbb{E}_{\pi}[\sum_{t=0}^{t=\infty} \gamma^t | s_0 = s]$
$V^{\pi}_{new}(s) = a\mathbb{E}_{\pi}[\sum_{t=0}^{t=\infty} \gamma^t r_t | s_0 = s] + ac\mathbb{E}_{\pi}[\frac{1}{1-\gamma} | s_0 = s]$
$V^{\pi}_{new}(s) = a\mathbb{E}_{\pi}[\sum_{t=0}^{t=\infty} \gamma^t r_t | s_0 = s] + ac\frac{1}{1-\gamma}$
$V^{\pi}_{new}(s) = aV^{\pi}_{old}(s) + ac\frac{1}{1-\gamma}$

$$V_1^{\pi^+}(s_{start}) - V_1^{\pi}(s_{start}) \overset{\text{performance difference lemma}}{=} \sum_{t=1}^{t=H} \mathbb{E}_{x_t \sim \pi} \Big(\underbrace{Q_t^{\pi^+}(x_t, \pi^+(x_t,t))}_{\text{(L)}} - \underbrace{Q_t^{\pi^+}(x_t, \pi(x_t,t))}_{\text{(R)}} \Big)$$

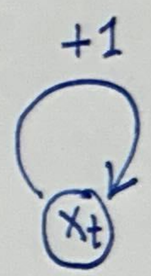
For each timestamp t,

case I. if $x_t \in \overline{S^+}$, then $\pi^+(x_t,t) = \pi(x_t,t) \Rightarrow (\quad) = 0$

case II if $x_t \in S^+$, $\boxed{\pi^+(x_t,t) \overset{\text{def}}{=} a_t^+}$

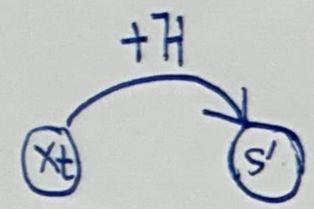
case II−1. if $\pi(x_t,t) = a_t^+ \Rightarrow (\quad) = 0$

case II−2. if $\pi(x_t,t) = b_t \neq a_t^+$



(L) $Q_t^{\pi^+}(x_t, a_t^+) = r_t(x_t, a_t^+) + p(s_{t+1} = x_t \mid s_t = x_t, a_t = a_t^+) V_t^{\pi^+}(x_t)$

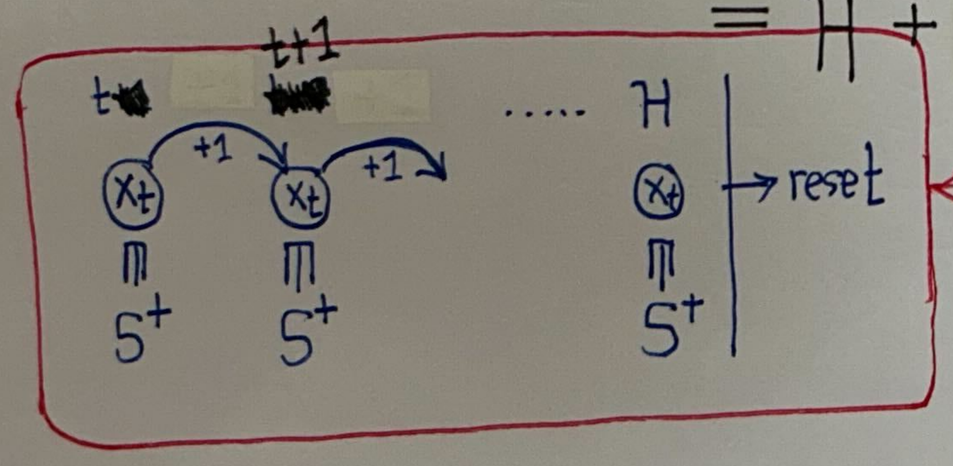
$= 1 + V_t^{\pi^+}(x_t)$



(R) $Q_t^{\pi^+}(x_t, b_t) = r_t(x_t, a_t) + p(s_{t+1} = s' \mid s_t = x_t, a_t = b_t) V_t^{\pi^+}(s')$

$= H + V_t^{\pi^+}(s')$

$V_t^{\pi^+}(x_t)$



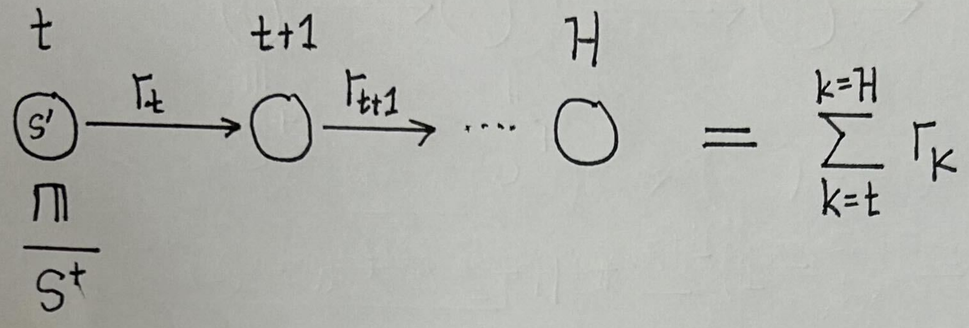
$= H - (t$ [illegible] $)$

II-2-1 $V_t^{\pi^+}(s')$ if $s' \in S^+$ then this is the exactly the same as $\Rightarrow$ (L) − (R) $= 1 - H < 0$

II-2-2 if $s' \in \overline{S^+}$, we will discuss in the page.

Ⅱ-2-2 $\overline{S'} \in \overline{S'}$

$V_t^{\pi^+}(S')$:



$$= \sum_{k=t}^{k=H} r_k$$

$$(L) - (A) = 1 - H + (H-t) - \sum_{k=t}^{k=H} r_k$$

$$(L) - (R) = 1 - H + (H-t) - \sum_{k=t}^{k=H} r_k$$

$$\leq 1 - t \leqslant 0, \quad \forall t = 1, 2, 3, \cdots H$$

3. Nonstationary Discount Factor γ

K iterations of value iterations

V_K, V_K' : initial value functions at timestamp.

time-dependent Bellman backup operator B_k for an arbitrary timestamp k

$$V_{k-1} \overset{\text{def}}{=} B_k V_k = \max_a \left[R(s,a) + \gamma_k \sum_{s'} P(s'|s,a) V_k(s') \right]$$

$$= \max_a \left[R(s,a) + \left(1 - \frac{1}{k+1}\right) \sum_{s'} P(s'|s,a) V_k(s') \right]$$

3-(a)

$$\|B_k V_k - B_k V_k'\| = \| \max_a \left[R(s,a) + \gamma_k \sum_{s'} P(s'|s,a) V_k(s') \right]$$

$$- \max_{a'} \left[R(s,a') + \gamma_k \sum_{s'} P(s'|s,a') V_k'(s') \right] \|$$

Sebastian Hurubaru (CF) said we could add the subscript k

$$\leq \max_a \| \left[R(s,a) + \gamma_k \sum_{s'} P(s'|s,a) V_k(s') \right]$$

$$- \left[R(s,a) + \gamma_k \sum_{s'} P(s'|s,a) V_k'(s') \right] \|$$

$$= \gamma_k \max_a \| \sum_{s'} P(s'|s,a) \left(V_k(s') - V_k'(s') \right) \|$$

By the definition of infinity norm $\|V_k - V_k'\| = \max_s |V_k(s) - V_k'(s)|$

$$\leq \gamma_k \max_a \| \sum_{s'} P(s'|s,a) \| V_k - V_k' \| \quad \|$$

$$= \gamma_k \| V_k - V_k' \| \max_a \| \underbrace{\sum_{s'} P(s'|s,a)}_{1} \|$$

$$= \gamma_k \| V_k - V_k' \|$$

From 3-(a) we know that $\|B_k V_k - B_k V_k'\| \le \gamma_k \|V_k - V_k'\|$.

Since $\gamma_k = 1 - \frac{1}{k+1} > 0$, $\forall k = 1, 2, \cdots K$,

we have $\|V_k - V_k'\| \ge \frac{1}{\gamma_k} \|B_k V_k - B_k V_k'\|$, $\forall k = 1, 2, \cdots K$ --- (*)

$$\|V_K - V_K'\| \overset{(*)}{\ge} \frac{1}{\gamma_K} \|B_K V_K - B_K V_K'\| = \frac{1}{\gamma_K} \boxed{\|V_{K-1} - V_{K-1}'\|}$$

$$\overset{(*)}{\ge} \frac{1}{\gamma_K} \boxed{\frac{1}{\gamma_{K-1}} \|B_{K-1} V_{K-1} - B_{K-1} V_{K-1}'\|}$$

... continuously apply (*)

$$\overset{(*)}{\ge} \frac{1}{\prod_{k=K}^{k=1} \gamma_k} \|B_1 V_1 - B_1 V_1'\| = \frac{1}{\prod_{k=1}^{k=K} \gamma_k} \|B_1 V_1 - B_1 V_1'\|$$

$$\left(\prod_{k=1}^{k=K} \gamma_k \right) \|V_K - V_K'\| \ge \|B_1 V_1 - B_1 V_1'\|$$

$V_{k-1} \overset{\text{def}}{=} B_k V_k$

$$= \|B_1 B_2 V_2 - B_1 B_2 V_2'\|$$

$$= \|B_1 B_2 B_3 V_3 - B_1 B_2 B_3 V_3'\|$$

$$\cdots$$

$$= \|B_1 B_2 B_3 \cdots B_K V_K - B_1 B_2 B_3 \cdots B_K V_K'\|$$

$$\gamma_i = 1 - \frac{1}{i+1} = \frac{i+1-1}{i+1} = \frac{i}{i+1}$$

$$\gamma_1 \gamma_2 \cdots \gamma_K = \frac{1}{2}\frac{2}{3}\cdots\frac{K}{K+1} = \frac{1}{K+1}$$

$$\Rightarrow \frac{1}{K+1}\|V_K - V_K'\|$$