Kerim Sever
Professor Jing Li
Ds699-852
07 March 2025

Part 1:

(a) Implement policy_evaluation and policy_improvement, and then use these two functions to finish policy_iteration. Please be careful about the definition of iteration. The iteration does not include the steps of policy evaluation.

Policy Evaluation:

```
    # evaluation_steps: the number of steps needed for policy evaluation in each iteration
    evaluation_steps = 0

    ############################
    # Your Code #
    # Please use np.linalg.norm(x, np.inf) to calculate the infinity norm. #
    # Please use while loop to finish this part. #
    # Remember to update the evaluation_steps. #

    while True:
        max_diff = 0  # Track maximum change in value function
        new_value_function = np.copy(value_function)  # Avoid in-place modification

        for state in range(nS):
            prev_value = value_function[state]  # Store old value

            # Compute new value using Bellman equation
            new_value = 0
            action = policy[state]  # Get action from policy

            for prob, next_state, reward, _ in P[state][action]:
                new_value += prob * (reward + gamma * value_function[next_state])

            new_value_function[state] = new_value
            max_diff = max(max_diff, abs(prev_value - new_value))  # Track max change

        value_function = new_value_function
        evaluation_steps += 1

        # Stop when max change is below threshold
        if max_diff < epsilon:
            break
    ############################
```

Policy_improvement

```python
def policy_improvement(P, nS, nA, value_function, gamma=0.9):
    """
    Use the value function to improve the policy.
    :param P: transition probability
    :param nS: number of states
    :param nA: number of actions
    :param value_function: value function from policy iteration
    :param gamma: gamma parameter used in policy improvement
    :return: new_policy: An array of integers. Each integer is the optimal action
                the environment dynamics and the given value function.
    """

    new_policy = np.zeros(nS, dtype="int")

    ############################
    # Your Code #
    # Please use np.argmax to select the best actions after getting the q value of

    # hold q values for all actions of all states
    q_values = np.zeros((nS,nA))

    for state in range(nS):
        for action in range(nA):
            q_values[state, action] = sum(
                prob * (reward + gamma * value_function[next_state])
                for prob, next_state, reward, _ in P[state][action]
            )

    new_policy = np.argmax(q_values, axis=1)
    ############################
    return new_policy
```

Policy_iteration

```
    # previous policy: the policy of last iteration.
    policy_prev = init_policy

    ############################
    # Your Code #
    # Please call the policy_evaluation() and policy_improvement() to update the policy. #
    # Remember to update the iteration and policy_prev. #
    # Please use while loop to finish this part. #
    # The time complexity of the code within the while loop represents the running time requ


    while True:
        # Policy evaluation function call
        value_function, steps = policy_evaluation(P, nS, policy_prev, gamma, epsilon)
        # Policy improvement function call
        new_policy = policy_improvement(P, nS, nA, value_function, gamma)
        # computes the difference in both policies and if no change means that policy is goo
        if np.linalg.norm(new_policy - policy_prev, ord=1) == 0:
            break
        # assgin new policy as previous policy
        policy_prev = new_policy
        # increase iteration
        iteration += 1
    iteration = iteration + steps

    ############################

    print(f"There are {iteration} iterations in policy iteration.")
    return value_function, policy_prev, iteration
```

(b) Implement value_iteration.

value_iteration

```python
def value_iteration(P, nS, nA, init_value=0.0, gamma=0.9, epsilon=1e-3):
    while True:
        max_value_change = 0
        new_value_function = np.copy(value_function)  # Create a copy to store updated values

        # Value update step
        for state in range(nS):
            if not P[state]:  # Handle terminal states
                continue

            q_values = np.zeros(nA)  # Store Q-values for all actions

            for action in range(nA):
                for prob, next_state, reward, terminal in P[state][action]:
                    q_values[action] += prob * (reward + gamma * value_function[next_state])

            # Bellman update: max action value
            new_value_function[state] = np.max(q_values)
            max_value_change = max(max_value_change, abs(new_value_function[state] - value_function[state]))

        value_function = new_value_function  # Update value function
        iteration += 1

        if max_value_change < epsilon:  # Check for convergence
            break

    # Policy extraction (Greedy policy)
    for state in range(nS):
        if not P[state]:  # Handle terminal states
            continue

        q_values = np.zeros(nA)
        for action in range(nA):
            for prob, next_state, reward, terminal in P[state][action]:
                q_values[action] += prob * (reward + gamma * value_function[next_state])

        policy[state] = np.argmax(q_values)  # Select best action
```

Part 2:

(a) Set `seeds` $= 1$ in `get_args.py` and take the screenshots of the all the output results after running `policy_iteration` and `value_iteration`.
(The screenshots should include the method you are using, the policy generated, the total running time and the episode reward.)

```
11
12
13
14
There are 29 iterations in policy iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: UserWarni
 variables from other wrappers is deprecated and will be removed in v1.0, to get this variabl
ed.nrow` for environment variables or `env.get_wrapper_attr('nrow')` that will search the re
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: UserWarni
 variables from other wrappers is deprecated and will be removed in v1.0, to get this variabl
ed.ncol` for environment variables or `env.get_wrapper_attr('ncol')` that will search the re
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.8146569728851318  average number of iteration is: 29.0
Episode reward: 1.0
PS E:\Ds699\Hw1>
```

```
  logger.warn(
---- Value Iteration----

c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: U
s from other wrappers is deprecated and will be removed in v1.0, to get this variable
onment variables or `env.get_wrapper_attr('P')` that will search the reminding wrappe
  logger.warn(
There are 15 iterations in value iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: U
bles from other wrappers is deprecated and will be removed in v1.0, to get this varia
 environment variables or `env.get_wrapper_attr('nrow')` that will search the remindi
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: U
bles from other wrappers is deprecated and will be removed in v1.0, to get this varia
 environment variables or `env.get_wrapper_attr('ncol')` that will search the remindi
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7232534885406494  average number of iteration is: 15.0
Episode reward: 1.0
PS E:\Ds699\Hw1>
```

(b) Please modify the parameter `seeds` in `get_args` and run each of `policy_iteration` and `value_iteration` **50 times** and take the screenshots of the total running time starting with the line: "`Total running time is`".

```
 7
 8
 9
 10
 11
 12
 13
 14
 There are 29 iterations in policy iteration.
 policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
  ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
  ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
  ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
  ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
  ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
  ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
  ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
 Total running time is: 13.025946617126465  average number of iteration is: 28.42
 Episode reward: 1.0
 PS E:\Ds699\Hw1>
```

```
  ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
 ---- Value Iteration----

 There are 15 iterations in value iteration.
 policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
  ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
  ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
  ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
  ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
  ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
  ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
  ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
 ---- Value Iteration----

 There are 15 iterations in value iteration.
 policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
  ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
  ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
  ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
  ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
  ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
  ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
  ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
 Total running time is: 12.966875076293945  average number of iteration is: 15.0
 Episode reward: 1.0
 PS E:\Ds699\Hw1>
```

(c) **Question:** What is the time complexity of **policy** iteration in one iteration? Please provide your answer in terms of $|S|$ (the length of the state vector) and $|A|$ (the length of the action vector). (**Hint:** "in one iteration" means we only consider the running time for just one run of policy evaluation and policy improvement function. You may also refer to the annotation in the codebase to calculate the running time.)

The time complexity in policy iteration for 1 iteration has 2 steps which are policy evaluation and policy improvement. Policy evaluation solves the Bellman equation for all states using O(k |S|^2 |A|). K is the number of iterations needed for the convergence. Policy Improvement does updates by finding the best action for each state. O(|S| |A|) since its only 1 iteration.

(d) **Question:** What is the time complexity of **value** iteration in one iteration? Please provide your answer in terms of $|S|$ (the length of the state vector) and $|A|$ (the length of the action vector). (**Hint:** "in one iteration" means we only consider the running time for just one time's value updating. You may also refer to the annotation in the codebase to calculate the running time.)

For value iteration it updates the value function for each state. Every state looks at all possible actions to find the best value. O(|S| |A|) for each iteration to find the maximum value. Since there are |S| states with a required |A| action value the cost for each iteration is the same. This only uses a single value update across all states and does not need to do a convergence like policy iteration.

(e) **Question:** Based on your answer for part(c) and part(d), theoretically which algorithm is faster in one iteration? Aside from the running time in a single iteration, what other factors could influence the **total** running time?

Value iteration was faster in both iteration examples because it has a simpler time complexity of O(|S| |A|) vs policy iterations policy evaluation step O(k |S|^2 |A|). Policy iteration has less convergence but the extra step increases the processing time. Value iteration updates until convergence and policy iteration stops when the policy converges. Other factors could be stopping criteria, state space size, and discount factor.

Part 3:

(a) Reset the parameter `seeds` as 1 in `get_args.py`. Use the value iteration method by setting the `method` as `value_iteration`. Set value the of ε as 0.5. Rerun the program and take the screenshot of all the output results.
(The screenshots should include the method you are using, the policy generated, the total running time and the episode reward.)

```
  logger.warn(
---- Value Iteration----

c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: Use
s from other wrappers is deprecated and will be removed in v1.0, to get this variable y
onment variables or `env.get_wrapper_attr('P')` that will search the reminding wrappers
  logger.warn(
There are 8 iterations in value iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: Use
bles from other wrappers is deprecated and will be removed in v1.0, to get this variabl
 environment variables or `env.get_wrapper_attr('nrow')` that will search the reminding
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: Use
bles from other wrappers is deprecated and will be removed in v1.0, to get this variabl
 environment variables or `env.get_wrapper_attr('ncol')` that will search the reminding
  logger.warn(
policy: [['L' 'L' 'L' 'L' 'L' 'D' 'D' 'D']
 ['L' 'L' 'L' 'L' 'D' 'D' 'D' 'D']
 ['L' 'L' 'L' 'L' 'D' 'R' 'D' 'D']
 ['L' 'L' 'R' 'R' 'D' 'L' 'D' 'D']
 ['L' 'L' 'L' 'L' 'D' 'D' 'R' 'D']
 ['L' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['L' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['L' 'L' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7102460861206055  average number of iteration is: 8.0
The agent didn't reach a terminal state in 100 steps.
PS E:\Ds699\Hw1>
```

(b) **Question**: How many iterations does the value iteration method need to generate the policy? Is the policy the optimal policy?

8

(c) **Question:** How many iterations does value iteration method need to generate the policy from part II.(a)? Is it less or more than the iterations you get when setting $\varepsilon = 0.5$?

Part 2a had 15 iterations versus epsilon value being 0.5 giving 8 iterations.

(d) **Question**: Based on the comparison between part (b) and part (c), what will happen if ε is decreased? Conversely, what will happen if ε is increased? Can you explain the reason?
(**Hint:** Based on the definition from lecture that the condition of convergence is the maximum difference $\|\mathbf{V_{k+1}} - \mathbf{V_k}\|_\infty \leq \varepsilon$, how does ε affect the number of iteration and the optimal policy?)

Epsilon = 0.001

```
---- Value Iteration----

c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: U
s from other wrappers is deprecated and will be removed in v1.0, to get this variable
onment variables or `env.get_wrapper_attr('P')` that will search the reminding wrappe
  logger.warn(
There are 15 iterations in value iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: U
bles from other wrappers is deprecated and will be removed in v1.0, to get this varia
 environment variables or `env.get_wrapper_attr('nrow')` that will search the remindi
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: U
bles from other wrappers is deprecated and will be removed in v1.0, to get this varia
 environment variables or `env.get_wrapper_attr('ncol')` that will search the remindi
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7265369892120361  average number of iteration is: 15.0
Episode reward: 1.0
PS E:\Ds699\Hw1>
```

Epsilon = 5

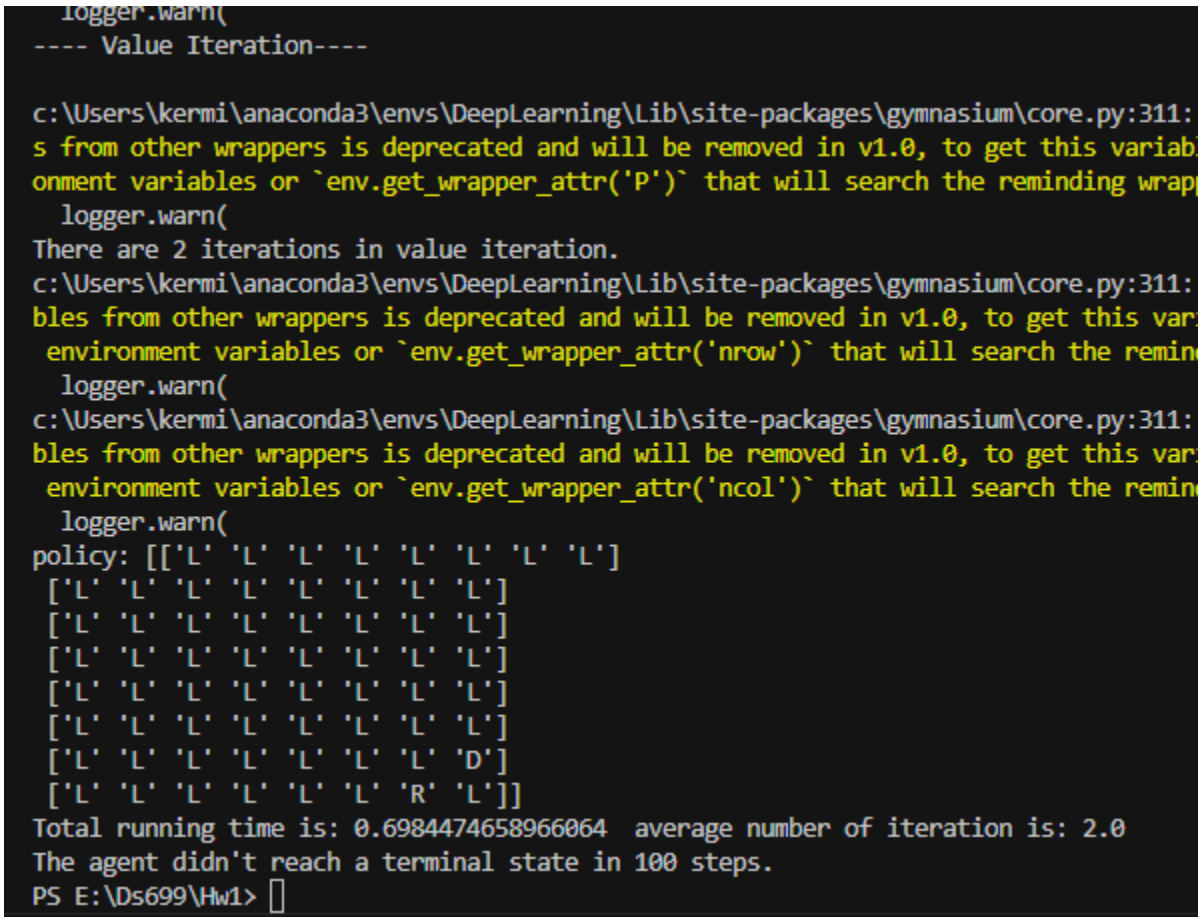
```
  logger.warn(
---- Value Iteration----

c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311:
s from other wrappers is deprecated and will be removed in v1.0, to get this variabl
onment variables or `env.get_wrapper_attr('P')` that will search the reminding wrapp
  logger.warn(
There are 1 iterations in value iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311:
bles from other wrappers is deprecated and will be removed in v1.0, to get this vari
 environment variables or `env.get_wrapper_attr('nrow')` that will search the remind
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311:
bles from other wrappers is deprecated and will be removed in v1.0, to get this vari
 environment variables or `env.get_wrapper_attr('ncol')` that will search the remind
  logger.warn(
policy: [['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'D']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'D']
 ['L' 'L' 'L' 'L' 'L' 'R' 'R' 'L']]
Total running time is: 0.6942884922027588  average number of iteration is: 1.0
The agent didn't reach a terminal state in 100 steps.
PS E:\Ds699\Hw1>
```

When epsilon is 5 the iteration count is 1 and when it is 0.001 epsilon count is 15. It seems like the higher the epsilon count the lower the iteration count is and the lower the epsilon count the

higher iteration. The main difference however is that the agent did not complete when epsilon was 5 but did complete at 0.001. For 0.001 since epsilon is decreased the convergence is stricter and the code will require a smaller difference between the current epsilon and previous epsilon which causes the count in iterations to increase. When it is increased to 5 however the convergence is a lot less strict and the code stops a lot sooner. This is not good however because the agent did not reach the terminal which explains the precision being lower. Higher epsilon will lead to a faster convergence with less accuracy and a potential incomplete with the agent but a smaller epsilon will lead to more iterations but a harder time with convergence.

Part 4:

(a) Set $\gamma = 0$, and take the screenshot of the all the output results from your code.
(The screenshots should include the method you are using, the policy generated, the total running time and the episode reward.)

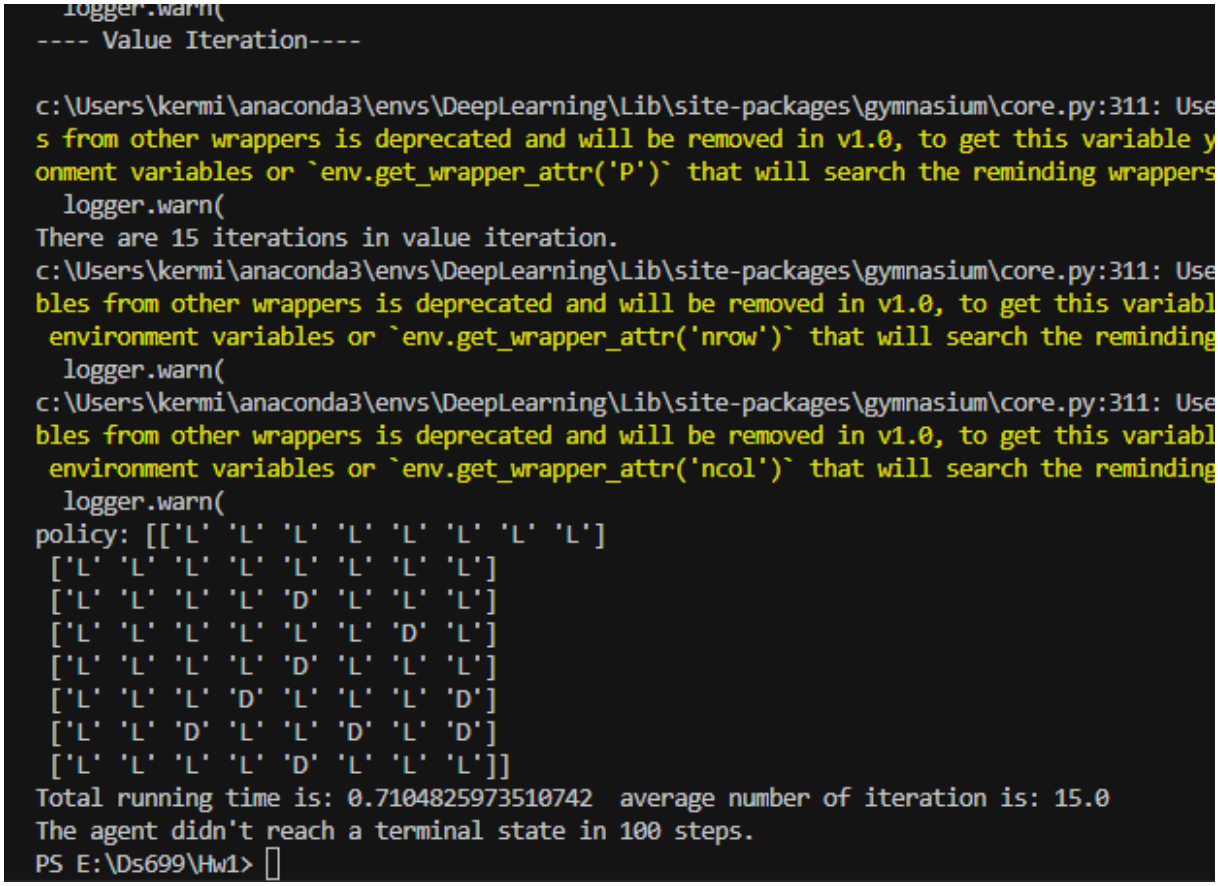


(b) **Question:** A computer science student Micheal conducted an experiment setting the value of γ to be 0, 1 and 2, and respectively obtained the results shown in the screenshots of Figure 1. Based on his results, can you analysis:
When $\gamma = 0$, does the value iteration generate the optimal policy? If not, can you explain the reason?
When $\gamma = 1$, does the value iteration generate the optimal policy? If not, can you explain the reason?
When $\gamma = 2$, can you explain why we got the value function and the policy as shown?

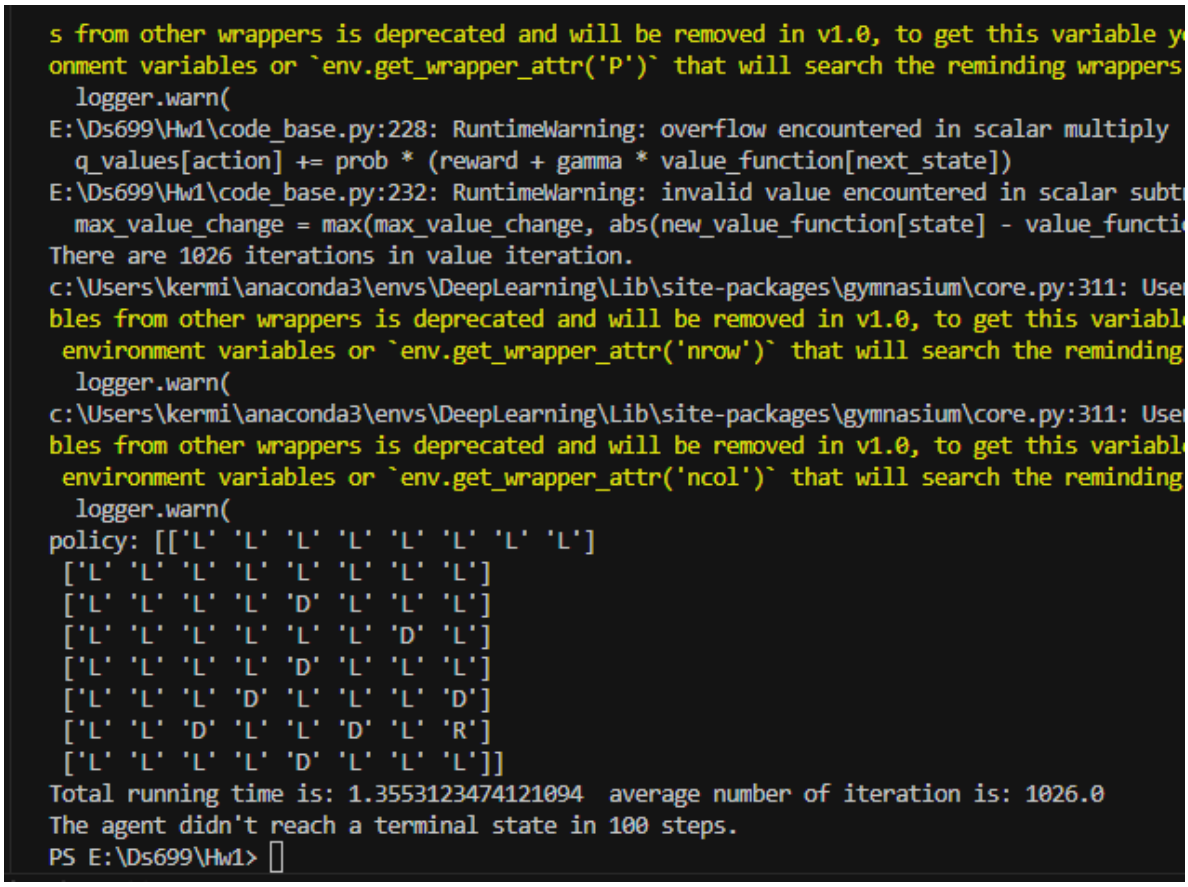
```
  logger.warn(
---- Value Iteration----

c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311:
s from other wrappers is deprecated and will be removed in v1.0, to get this variab
onment variables or `env.get_wrapper_attr('P')` that will search the reminding wrap
  logger.warn(
There are 2 iterations in value iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311:
bles from other wrappers is deprecated and will be removed in v1.0, to get this var
 environment variables or `env.get_wrapper_attr('nrow')` that will search the remin
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311:
bles from other wrappers is deprecated and will be removed in v1.0, to get this var
 environment variables or `env.get_wrapper_attr('ncol')` that will search the remin
  logger.warn(
policy: [['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'L']
 ['L' 'L' 'L' 'L' 'L' 'L' 'L' 'D']
 ['L' 'L' 'L' 'L' 'L' 'L' 'R' 'L']]
Total running time is: 0.6984474658966064  average number of iteration is: 2.0
The agent didn't reach a terminal state in 100 steps.
PS E:\Ds699\Hw1>
```

0

When the gamma is 0 the iteration is 2 with a time of 0.7 and the agent did not complete. The agent is ignoring future rewards and only focuses on the current reward. This is treating each state as an independent decision. The value function is not considering anything long term which is why the agent did not complete. This is mainly because the agent is not considering cumulative rewards over time.



When the gamma is 1 the iteration is 15 with a time of 0.71 and the agent did not complete. This had a better result then gamma being equal to 0 because it is hitting 15 iterations even though the agent did not complete. Gamma as 1 is allowing the agent to consider future rewards throughout the iterations but still did not complete because the value iteration process is running into issues with the convergence or computational efficiency to stop the agent from completing. This could be from the stopping criteria in the code since it did 15 iterations but worse in time or hanging on convergence.



2

When the gamma is 2 the iteration is 1026 with a time of 1.4 and the agent did not complete. When gamma is 2 the agent is considering future rewards too heavily which is causing the 1026 iterations. This is causing the variable iteration process to be incorrectly assessing future rewards and are treated incorrectly. This also did not allow the agent to complete because the learning process is no longer stable.

Overall each of these gamma values lead to different actions within the value iteration process. Gamma as 0 fails to see future rewards, gamma as 1 is better by seeing current action and future rewards but is still not completing likely due to convergence, and gamma as 2 is unstable because it is heavily dependent on future rewards

(c) Set $\gamma = 0.5$, and take the screenshot of the output.
Question: Does the value iteration generate the optimal policy? How is γ related with ε?

```
---- Value Iteration----

c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:3
s from other wrappers is deprecated and will be removed in v1.0, to get this var
onment variables or `env.get_wrapper_attr('P')` that will search the reminding w
  logger.warn(
There are 11 iterations in value iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:3
bles from other wrappers is deprecated and will be removed in v1.0, to get this
 environment variables or `env.get_wrapper_attr('nrow')` that will search the re
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:3
bles from other wrappers is deprecated and will be removed in v1.0, to get this
 environment variables or `env.get_wrapper_attr('ncol')` that will search the re
  logger.warn(
policy: [['L' 'L' 'D' 'D' 'D' 'D' 'D' 'D']
 ['L' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['L' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7106831073760986  average number of iteration is: 11.0
The agent didn't reach a terminal state in 100 steps.
PS E:\Ds699\Hw1>
```

0.5

When the gamma is 0.5 the iteration is 11 with a time of 0.71 and the agent did not complete. Gamma and Epsilon determine how much the agent looks at future rewards and its current rewards. The convergence is what allows the agent to stabilize and by gamma being set to 0.5 the agent does not look at future rewards or immediate rewards as it treats them both the same. I think this makes the agent confused on which option to take as it does not know if it wants the immediate reward or the long term reward.

(d) **Question:** If the policy found in the previous question is not optimal, can you help the value iteration find the optimal policy by modifying the value of ε? Please show your modified value of ε and provide a screenshot.

By changing the epsilon to 1e-2 and keeping the gamma 0.5 it still failed with 8 iterations but when I changed the epsilon to 1e-4 and gamma to 0.5 it passed with 15 iterations. This makes sense because it is considering the long term value more with the epsilon value as 1e-4. The larger epsilon value of 1e-2 is converging too quickly which is causing less iterations and does not allow the agent to complete.

1e-2

```
---- Value Iteration----

c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:31
his variable you can do `env.unwrapped.P` for environment variables or `env.get_w
  logger.warn(
There are 8 iterations in value iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:31
t this variable you can do `env.unwrapped.nrow` for environment variables or `env
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:31
t this variable you can do `env.unwrapped.ncol` for environment variables or `env
  logger.warn(
policy: [['L' 'L' 'L' 'L' 'L' 'D' 'D' 'D']
 ['L' 'L' 'L' 'L' 'D' 'D' 'D' 'D']
 ['L' 'L' 'L' 'L' 'D' 'R' 'D' 'D']
 ['L' 'L' 'R' 'R' 'D' 'L' 'D' 'D']
 ['L' 'L' 'L' 'L' 'D' 'D' 'R' 'D']
 ['L' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['L' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['L' 'L' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7304694652557373  average number of iteration is: 8.0
The agent didn't reach a terminal state in 100 steps.
PS E:\Ds699\Hw1>
```

1e-4

```
  logger.warn(
---- Value Iteration----

c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311
his variable you can do `env.unwrapped.P` for environment variables or `env.get_wra
  logger.warn(
There are 15 iterations in value iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311
t this variable you can do `env.unwrapped.nrow` for environment variables or `env.g
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311
t this variable you can do `env.unwrapped.ncol` for environment variables or `env.g
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7159507274627686  average number of iteration is: 15.0
Episode reward: 1.0
PS E:\Ds699\Hw1>
```

Part 5:

(a) Please change the initial action to "go left" for all states and rerun the policy iteration. Take the screenshot of all the output results from your code. Please use the `init_action` in `get_args.py` to control the initialization of the policy.
(The screenshots should include the method you are using, the policy generated, the total running time and the episode reward.)

Default = 0 (left)

```
Evaluation steps: 9
Evaluation steps: 10
Evaluation steps: 11
Evaluation steps: 12
Evaluation steps: 13
Evaluation steps: 14
There are 29 iterations in policy iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311
t this variable you can do `env.unwrapped.nrow` for environment variables or `env
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311
t this variable you can do `env.unwrapped.ncol` for environment variables or `env
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7985923290252686  average number of iteration is: 29.0
Episode reward: 1.0
PS E:\Ds699\Hw1>
```

(b) **Question:** Is the policy the optimal policy? How many iterations does the policy iteration need?

This policy is optimal because the agent was successfully able to complete and did it in 29 iterations with a time of 0.8.

(c) Please try "go right", "go up", and "go down" for all states and rerun the policy iteration method. Take a screenshot for each case.

Default = 2 (down)

```
13
14
There are 27 iterations in policy iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:31
bles from other wrappers is deprecated and will be removed in v1.0, to get this v
 environment variables or `env.get_wrapper_attr('nrow')` that will search the rem
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:31
bles from other wrappers is deprecated and will be removed in v1.0, to get this v
 environment variables or `env.get_wrapper_attr('ncol')` that will search the rem
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7465660572052002  average number of iteration is: 27.0
Episode reward: 1.0
PS E:\Ds699\Hw1>
```

Default = 3 (up)

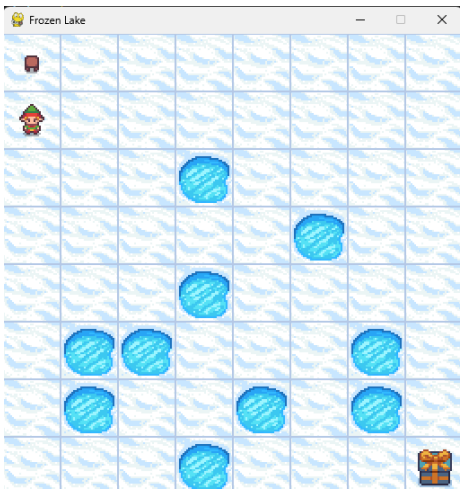
```
14
There are 29 iterations in policy iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:3
bles from other wrappers is deprecated and will be removed in v1.0, to get this
 environment variables or `env.get_wrapper_attr('nrow')` that will search the re
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:3
bles from other wrappers is deprecated and will be removed in v1.0, to get this
 environment variables or `env.get_wrapper_attr('ncol')` that will search the re
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.748060941696167  average number of iteration is: 29.0
Episode reward: 1.0
PS E:\Ds699\Hw1>
```

Default = 1 (right)

```
13
14
There are 27 iterations in policy iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:3
bles from other wrappers is deprecated and will be removed in v1.0, to get this
 environment variables or `env.get_wrapper_attr('nrow')` that will search the re
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:3
bles from other wrappers is deprecated and will be removed in v1.0, to get this
 environment variables or `env.get_wrapper_attr('ncol')` that will search the re
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7618312835693359  average number of iteration is: 27.0
Episode reward: 1.0
PS E:\Ds699\Hw1>
```

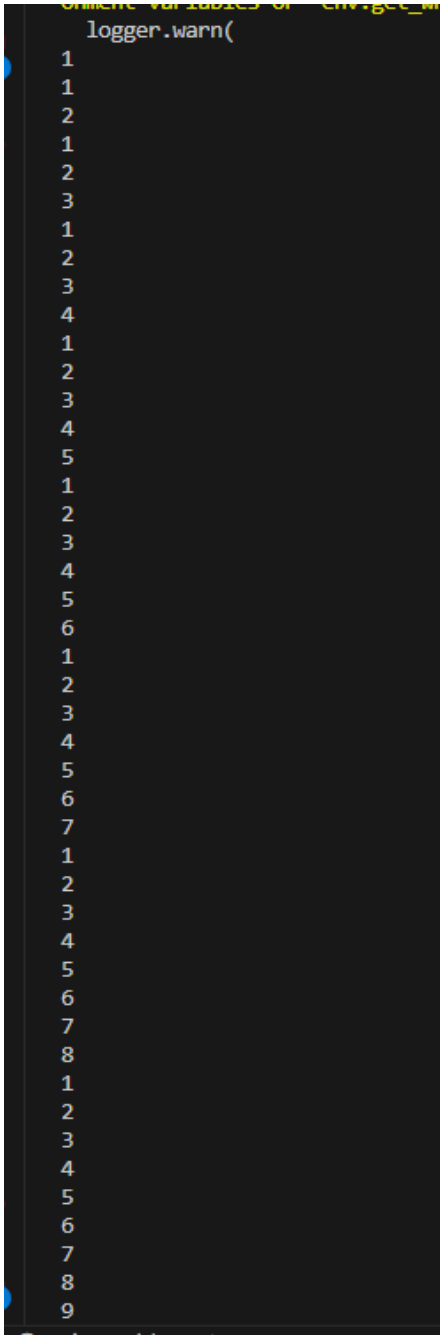
(d) **Question:** Which action(s) initialization can reduce the number of iterations? Can you explain the reason?

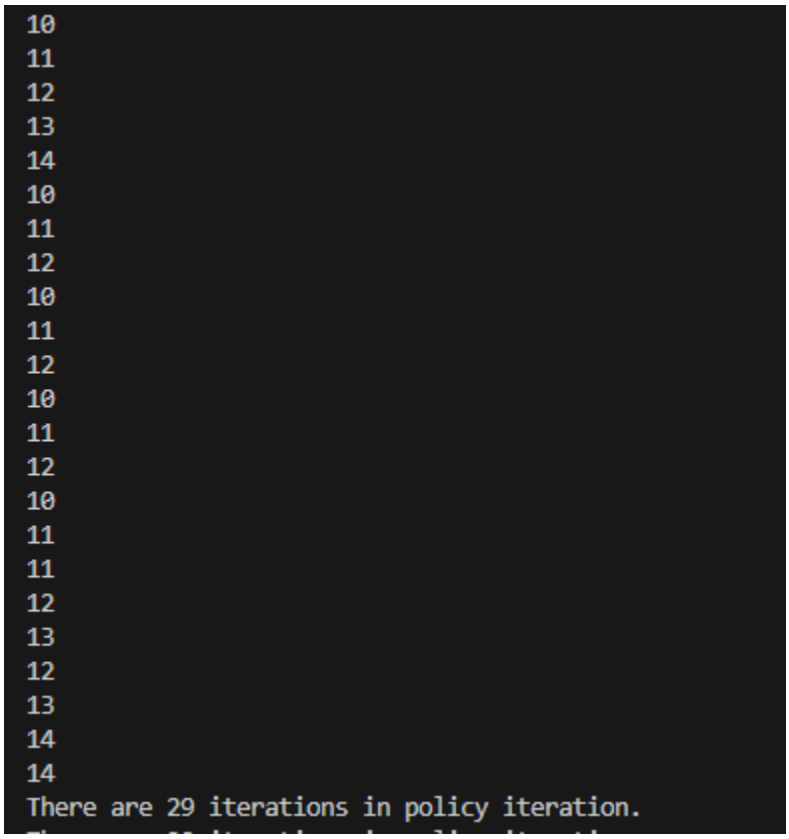
Through the different directions it seems like default as 1 and default as 2 were the best initialization methods because it was able to complete the agent in 27 iterations with times around 0.75. I believe this is because the direction of the initialization is towards the reward whereas left and up are not helpful for the starting point. The agent would but looking at the direction of the border and cannot move forward while down and right are movements the agent can make.



(e) In `policy_evaluation` function of the policy iteration method, we initialize the value function with all 0 by: `value_function = np.zeros(nS)`

Question: How many evaluation steps do policy evaluation need to converge? Please print the `evaluation_steps` for each iteration and take a screenshot.





```
Evaluation steps: 14
There are 29 iterations in policy iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:31
t this variable you can do `env.unwrapped.nrow` for environment variables or `env
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:31
t this variable you can do `env.unwrapped.ncol` for environment variables or `env
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7849111557006836  average number of iteration is: 29.0
Episode reward: 1.0
PS E:\Ds699\Hw1>
```

(f) Can you optimize the initialization to reduce the number of evaluation steps? You can add a new parameter to the policy_iteration function and use this parameter to control the initialization. Please print out **evaluation_steps** for each iteration and take the screenshot.

Question: Please describe the main idea of your initialization method.

Initialization_factor = 1

```
Evaluation steps: 6
Evaluation steps: 7
Evaluation steps: 8
Evaluation steps: 9
Evaluation steps: 10
Evaluation steps: 11
Evaluation steps: 12
Evaluation steps: 13
There are 28 iterations in policy iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: Use
t this variable you can do `env.unwrapped.nrow` for environment variables or `env.get_w
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311: Use
t this variable you can do `env.unwrapped.ncol` for environment variables or `env.get_w
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.8913533687591553  average number of iteration is: 28.0
Episode reward: 1.0
```

Initialization_factor = 3

```
Evaluation steps: 7
Evaluation steps: 8
Evaluation steps: 9
Evaluation steps: 10
Evaluation steps: 11
Evaluation steps: 12
Evaluation steps: 13
Evaluation steps: 14
There are 29 iterations in policy iteration.
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311:
t this variable you can do `env.unwrapped.nrow` for environment variables or `env.g
  logger.warn(
c:\Users\kermi\anaconda3\envs\DeepLearning\Lib\site-packages\gymnasium\core.py:311:
t this variable you can do `env.unwrapped.ncol` for environment variables or `env.g
  logger.warn(
policy: [['D' 'D' 'D' 'D' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'R' 'D' 'D' 'D' 'D']
 ['D' 'D' 'D' 'L' 'D' 'R' 'D' 'D']
 ['R' 'R' 'R' 'R' 'D' 'L' 'D' 'D']
 ['R' 'R' 'U' 'L' 'D' 'D' 'R' 'D']
 ['D' 'L' 'L' 'R' 'R' 'D' 'L' 'D']
 ['D' 'L' 'R' 'U' 'L' 'D' 'L' 'D']
 ['R' 'R' 'U' 'L' 'R' 'R' 'R' 'L']]
Total running time is: 0.7819430828094482  average number of iteration is: 29.0
Episode reward: 1.0
```

```python
value_function = np.zeros(nS)
for state in range(nS):
    action = policy[state]
    # Initialize with some weighted reward value to better approximate the true value
    value_function[state] = initialization_factor * np.sum([prob * reward for prob, next_state, reward, _ in P[state][action]])

# Track the number of evaluation steps
evaluation_steps = 0
```

I tried using an initialization factor with a default of 0.5 within the policy evaluation process. The idea is to improve the starting point by using a weighted sum instead of initialization with zeros. By calculating the expected reward for each state based on the transition probabilities. By setting the initialization factor to 0.5 scales the weighted sum to change the initial starting point. In theory this would help the convergence of the policy evaluation and reduce the number of iterations. This did not happen but it was the idea I was shooting for. I was hoping the initialization factor would allow convergence to happen sooner and allow the function to stabilize. By increasing it to 1 however it did reduce the amount of iterations from 29 to 28 which means there is progress. When I tried 3 it went back to 29. This shows that it does work but only by a minimal amount.