In [1]:
```
!pip install mdp
```

```
Requirement already satisfied: mdp in c:\users\sanah quazi\anaconda3\lib\site-pack
ages (3.6)
Requirement already satisfied: future in c:\users\sanah quazi\anaconda3\lib\site-p
ackages (from mdp) (0.18.2)
Requirement already satisfied: numpy in c:\users\sanah quazi\anaconda3\lib\site-pa
ckages (from mdp) (1.22.4)
```

```
WARNING: Ignoring invalid distribution -oogle-auth (c:\users\sanah quazi\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -oogle-auth (c:\users\sanah quazi\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -oogle-auth (c:\users\sanah quazi\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -oogle-auth (c:\users\sanah quazi\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -oogle-auth (c:\users\sanah quazi\anaconda3
\lib\site-packages)
WARNING: Ignoring invalid distribution -oogle-auth (c:\users\sanah quazi\anaconda3
\lib\site-packages)
```

In [2]:
```
transition_probs = {
    's0': {
        'a0': {'s0': 0.5, 's2': 0.5},
        'a1': {'s2': 1}
    },
    's1': {
        'a0': {'s0': 0.7, 's1': 0.1, 's2': 0.2},
        'a1': {'s1': 0.95, 's2': 0.05}
    },
    's2': {
        'a0': {'s0': 0.4, 's2': 0.6},
        'a1': {'s0': 0.3, 's1': 0.3, 's2': 0.4}
    }
}
rewards = {
    's1': {'a0': {'s0': +5}},
    's2': {'a1': {'s0': -1}}
}

from mdp import MDP
mdp = MDP(transition_probs, rewards, initial_state='s0')
```

In [3]:
```
import gym

# Create the Mountain Car environment
env = gym.make('MountainCar-v0')

# Reset the environment and get the initial state
initial_state = env.reset()
print('initial state =', initial_state)

# Take a step in the environment
action = 1  # Example action (0 or 1)
next_state, reward, done, info = env.step(action)
print('next_state = %s, reward = %s, done = %s' % (next_state, reward, done))
```

```
initial state = [-0.5529246  0.        ]
next_state = [-5.5270493e-01  2.1965985e-04], reward = -1.0, done = False
```

In [4]:
```
# Get all states
all_states = env.observation_space
```

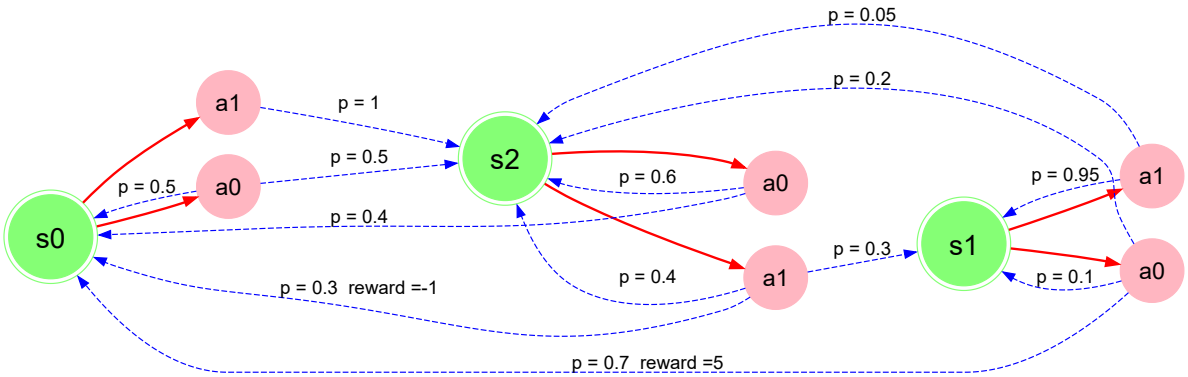
```
print("mdp.get_all_states =", all_states)

# Get possible actions for a state
state = 's1'  # Example state
possible_actions = env.action_space.n
print("mdp.get_possible_actions('s1') =", possible_actions)

# Get next states and their transition probabilities for a state-action pair
state = 's1'  # Example state
action = 0  # Example action (0 or 1)
next_states = env.reset()  # Reset the environment
env.state = state  # Set the environment state to the desired state
_, _, _, info = env.step(action)  # Take a step in the environment
next_states = env.state  # Get the resulting state
transition_probs = {s: 1.0 for s in next_states}  # Assuming deterministic transit
print("mdp.get_next_states('s1', 'a0') =", next_states)
print("mdp.get_transition_prob('s1', 'a0', 's0') =", transition_probs)

# Get reward for a state-action-next state transition
state = 's1'  # Example state
action = 0  # Example action (0 or 1)
next_state = 's0'  # Example next state
reward = env.reward_range[1] if next_state == 's0' and action == 0 and state == 's
print("mdp.get_reward('s1', 'a0', 's0') =", reward)
```

```
mdp.get_all_states = Box([-1.2  -0.07], [0.6  0.07], (2,), float32)
mdp.get_possible_actions('s1') = 3
mdp.get_next_states('s1', 'a0') = s1
mdp.get_transition_prob('s1', 'a0', 's0') = {'s': 1.0, '1': 1.0}
mdp.get_reward('s1', 'a0', 's0') = inf
```

In [5]:
```
from mdp import has_graphviz
from IPython.display import display
print("Graphviz available:", has_graphviz)
```

```
Graphviz available: True
```

In [6]:
```
import os
os.environ["PATH"] += os.pathsep + r'C:\Program Files\Graphviz\bin'

if has_graphviz:
    from mdp import plot_graph, plot_graph_with_state_values, plot_graph_optimal_s
    display(plot_graph(mdp))
```



In [7]:
```
def get_action_value(mdp, state_values, state, action, gamma):
    # Initialize Q
    Q = 0
    for s in mdp.get_all_states():
        # Compute Q using the equation above
        Q += mdp.get_transition_prob(state, action, s) * (mdp.get_reward(state, ac
                                                          gamma * state_values[s])
    return Q
```

In [8]:
```
import numpy as np

# Test the get_action_value function
test_Vs = {s: i for i, s in enumerate(sorted(mdp.get_all_states()))}
print(test_Vs)
print(get_action_value(mdp, test_Vs, 's2', 'a1', 0.9))
assert np.isclose(get_action_value(mdp, test_Vs, 's2', 'a1', 0.9), 0.69)
print(get_action_value(mdp, test_Vs, 's1', 'a0', 0.9))
assert np.isclose(get_action_value(mdp, test_Vs, 's1', 'a0', 0.9), 3.95)
```

```
{'s0': 0, 's1': 1, 's2': 2}
0.6900000000000002
3.9499999999999997
```

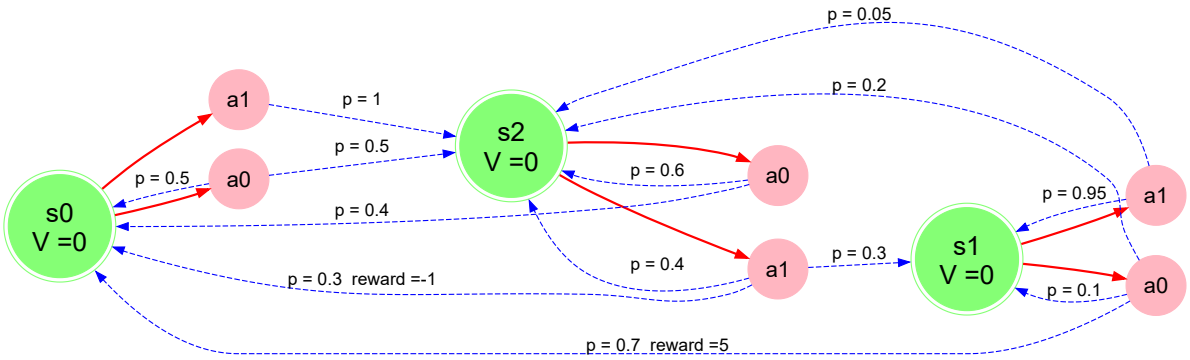
In [9]:
```
import numpy as np

def get_new_state_value(mdp, state_values, state, gamma):
    # Computes next V(s) as in the formula above. Do not change state_values in the
    if mdp.is_terminal(state):
        return 0

    # Initialize the dict
    A = [a for a in mdp.get_possible_actions(state)]
    v = np.zeros(len(mdp.get_possible_actions(state)))
    i = 0

    # Compute all possible options
    for a in mdp.get_possible_actions(state):
        v[i] = get_action_value(mdp, state_values, state, a, gamma)
        A[i] = a
        i = i + 1

    # Recover V(s) and π*(s) as per the formula above
    V = {A[np.argmax(v)]: v[np.argmax(v)]}

    return V
```

In [10]:
```
test_Vs_copy = dict(test_Vs)
V = get_new_state_value(mdp, test_Vs, 's0', 0.9)
print(test_Vs)
a = list(V)[0]
v = V[a]
print(a, v)
assert np.isclose(v, 1.8)
assert test_Vs == test_Vs_copy, "Please do not change state_values in get_new_state
```

```
{'s0': 0, 's1': 1, 's2': 2}
a1 1.8
```

In [12]:
```
# parameters
gamma = 0.9            # discount for MDP
num_iter = 1000         # maximum iterations, excluding initialization
# stop VI if new values are this close to old values (or closer)
min_difference = 0.0001
```

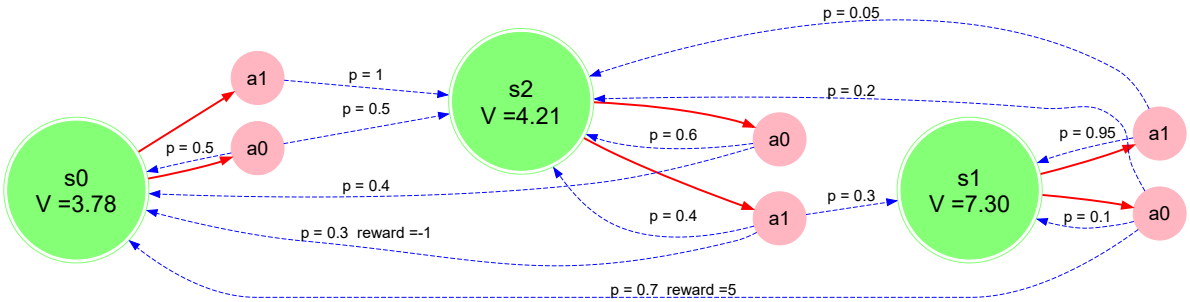
```
# initialize V(s)
state_values = {s: 0 for s in mdp.get_all_states()}

if has_graphviz:
    display(plot_graph_with_state_values(mdp, state_values))

for i in range(num_iter):
    # Compute new state values using the functions defined above.
    # It must be a dict {state : float V_new(state)}
    new_state_values = {}
    for s in mdp.get_all_states():
        nsv = get_new_state_value(mdp, state_values, s, gamma)
        a = list(nsv)[0]
        v = nsv[a]
        new_state_values[s] = v

    assert isinstance(new_state_values, dict)

    # Compute difference
    diff = max(abs(new_state_values[s] - state_values[s]) for s in mdp.get_all_sta
    print("iter %4i   |   diff: %6.5f   |   " % (i, diff), end="")
    print('   '.join("V(%s) = %.3f" % (s, v) for s, v in state_values.items()))
    state_values = new_state_values

    if diff < min_difference:
        print("Terminated")
        break
```



```
iter    0   |   diff: 3.50000   |   V(s0) = 0.000   V(s1) = 0.000   V(s2) = 0.000
iter    1   |   diff: 0.64500   |   V(s0) = 0.000   V(s1) = 3.500   V(s2) = 0.000
iter    2   |   diff: 0.58050   |   V(s0) = 0.000   V(s1) = 3.815   V(s2) = 0.645
iter    3   |   diff: 0.43582   |   V(s0) = 0.581   V(s1) = 3.959   V(s2) = 0.962
iter    4   |   diff: 0.30634   |   V(s0) = 0.866   V(s1) = 4.395   V(s2) = 1.272
iter    5   |   diff: 0.27571   |   V(s0) = 1.145   V(s1) = 4.670   V(s2) = 1.579
iter    6   |   diff: 0.24347   |   V(s0) = 1.421   V(s1) = 4.926   V(s2) = 1.838
iter    7   |   diff: 0.21419   |   V(s0) = 1.655   V(s1) = 5.169   V(s2) = 2.075
iter    8   |   diff: 0.19277   |   V(s0) = 1.868   V(s1) = 5.381   V(s2) = 2.290
iter    9   |   diff: 0.17327   |   V(s0) = 2.061   V(s1) = 5.573   V(s2) = 2.481
iter   10   |   diff: 0.15569   |   V(s0) = 2.233   V(s1) = 5.746   V(s2) = 2.654
iter   11   |   diff: 0.14012   |   V(s0) = 2.389   V(s1) = 5.902   V(s2) = 2.810
iter   12   |   diff: 0.12610   |   V(s0) = 2.529   V(s1) = 6.042   V(s2) = 2.950
iter   13   |   diff: 0.11348   |   V(s0) = 2.655   V(s1) = 6.168   V(s2) = 3.076
iter   14   |   diff: 0.10213   |   V(s0) = 2.769   V(s1) = 6.282   V(s2) = 3.190
iter   15   |   diff: 0.09192   |   V(s0) = 2.871   V(s1) = 6.384   V(s2) = 3.292
iter   16   |   diff: 0.08272   |   V(s0) = 2.963   V(s1) = 6.476   V(s2) = 3.384
iter   17   |   diff: 0.07445   |   V(s0) = 3.045   V(s1) = 6.558   V(s2) = 3.467
iter   18   |   diff: 0.06701   |   V(s0) = 3.120   V(s1) = 6.633   V(s2) = 3.541
iter   19   |   diff: 0.06031   |   V(s0) = 3.187   V(s1) = 6.700   V(s2) = 3.608
iter   20   |   diff: 0.05428   |   V(s0) = 3.247   V(s1) = 6.760   V(s2) = 3.668
iter   21   |   diff: 0.04885   |   V(s0) = 3.301   V(s1) = 6.814   V(s2) = 3.723
iter   22   |   diff: 0.04396   |   V(s0) = 3.350   V(s1) = 6.863   V(s2) = 3.771
iter   23   |   diff: 0.03957   |   V(s0) = 3.394   V(s1) = 6.907   V(s2) = 3.815
iter   24   |   diff: 0.03561   |   V(s0) = 3.434   V(s1) = 6.947   V(s2) = 3.855
iter   25   |   diff: 0.03205   |   V(s0) = 3.469   V(s1) = 6.982   V(s2) = 3.891
iter   26   |   diff: 0.02884   |   V(s0) = 3.502   V(s1) = 7.014   V(s2) = 3.923
iter   27   |   diff: 0.02596   |   V(s0) = 3.530   V(s1) = 7.043   V(s2) = 3.951
iter   28   |   diff: 0.02336   |   V(s0) = 3.556   V(s1) = 7.069   V(s2) = 3.977
iter   29   |   diff: 0.02103   |   V(s0) = 3.580   V(s1) = 7.093   V(s2) = 4.001
iter   30   |   diff: 0.01892   |   V(s0) = 3.601   V(s1) = 7.114   V(s2) = 4.022
iter   31   |   diff: 0.01703   |   V(s0) = 3.620   V(s1) = 7.133   V(s2) = 4.041
iter   32   |   diff: 0.01533   |   V(s0) = 3.637   V(s1) = 7.150   V(s2) = 4.058
iter   33   |   diff: 0.01380   |   V(s0) = 3.652   V(s1) = 7.165   V(s2) = 4.073
iter   34   |   diff: 0.01242   |   V(s0) = 3.666   V(s1) = 7.179   V(s2) = 4.087
iter   35   |   diff: 0.01117   |   V(s0) = 3.678   V(s1) = 7.191   V(s2) = 4.099
iter   36   |   diff: 0.01006   |   V(s0) = 3.689   V(s1) = 7.202   V(s2) = 4.110
iter   37   |   diff: 0.00905   |   V(s0) = 3.699   V(s1) = 7.212   V(s2) = 4.121
iter   38   |   diff: 0.00815   |   V(s0) = 3.708   V(s1) = 7.221   V(s2) = 4.130
iter   39   |   diff: 0.00733   |   V(s0) = 3.717   V(s1) = 7.230   V(s2) = 4.138
iter   40   |   diff: 0.00660   |   V(s0) = 3.724   V(s1) = 7.237   V(s2) = 4.145
iter   41   |   diff: 0.00594   |   V(s0) = 3.731   V(s1) = 7.244   V(s2) = 4.152
iter   42   |   diff: 0.00534   |   V(s0) = 3.736   V(s1) = 7.249   V(s2) = 4.158
iter   43   |   diff: 0.00481   |   V(s0) = 3.742   V(s1) = 7.255   V(s2) = 4.163
iter   44   |   diff: 0.00433   |   V(s0) = 3.747   V(s1) = 7.260   V(s2) = 4.168
iter   45   |   diff: 0.00390   |   V(s0) = 3.751   V(s1) = 7.264   V(s2) = 4.172
iter   46   |   diff: 0.00351   |   V(s0) = 3.755   V(s1) = 7.268   V(s2) = 4.176
iter   47   |   diff: 0.00316   |   V(s0) = 3.758   V(s1) = 7.271   V(s2) = 4.179
iter   48   |   diff: 0.00284   |   V(s0) = 3.762   V(s1) = 7.275   V(s2) = 4.183
iter   49   |   diff: 0.00256   |   V(s0) = 3.764   V(s1) = 7.277   V(s2) = 4.185
iter   50   |   diff: 0.00230   |   V(s0) = 3.767   V(s1) = 7.280   V(s2) = 4.188
iter   51   |   diff: 0.00207   |   V(s0) = 3.769   V(s1) = 7.282   V(s2) = 4.190
iter   52   |   diff: 0.00186   |   V(s0) = 3.771   V(s1) = 7.284   V(s2) = 4.192
iter   53   |   diff: 0.00168   |   V(s0) = 3.773   V(s1) = 7.286   V(s2) = 4.194
iter   54   |   diff: 0.00151   |   V(s0) = 3.775   V(s1) = 7.288   V(s2) = 4.196
iter   55   |   diff: 0.00136   |   V(s0) = 3.776   V(s1) = 7.289   V(s2) = 4.197
iter   56   |   diff: 0.00122   |   V(s0) = 3.778   V(s1) = 7.291   V(s2) = 4.199
iter   57   |   diff: 0.00110   |   V(s0) = 3.779   V(s1) = 7.292   V(s2) = 4.200
iter   58   |   diff: 0.00099   |   V(s0) = 3.780   V(s1) = 7.293   V(s2) = 4.201
iter   59   |   diff: 0.00089   |   V(s0) = 3.781   V(s1) = 7.294   V(s2) = 4.202
iter   60   |   diff: 0.00080   |   V(s0) = 3.782   V(s1) = 7.295   V(s2) = 4.203
iter   61   |   diff: 0.00072   |   V(s0) = 3.783   V(s1) = 7.296   V(s2) = 4.204
iter   62   |   diff: 0.00065   |   V(s0) = 3.783   V(s1) = 7.296   V(s2) = 4.205
iter   63   |   diff: 0.00058   |   V(s0) = 3.784   V(s1) = 7.297   V(s2) = 4.205
```

```
iter   64   |   diff: 0.00053   |   V(s0) = 3.785   V(s1) = 7.298   V(s2) = 4.206
iter   65   |   diff: 0.00047   |   V(s0) = 3.785   V(s1) = 7.298   V(s2) = 4.206
iter   66   |   diff: 0.00043   |   V(s0) = 3.786   V(s1) = 7.299   V(s2) = 4.207
iter   67   |   diff: 0.00038   |   V(s0) = 3.786   V(s1) = 7.299   V(s2) = 4.207
iter   68   |   diff: 0.00035   |   V(s0) = 3.786   V(s1) = 7.299   V(s2) = 4.208
iter   69   |   diff: 0.00031   |   V(s0) = 3.787   V(s1) = 7.300   V(s2) = 4.208
iter   70   |   diff: 0.00028   |   V(s0) = 3.787   V(s1) = 7.300   V(s2) = 4.208
iter   71   |   diff: 0.00025   |   V(s0) = 3.787   V(s1) = 7.300   V(s2) = 4.209
iter   72   |   diff: 0.00023   |   V(s0) = 3.788   V(s1) = 7.301   V(s2) = 4.209
iter   73   |   diff: 0.00020   |   V(s0) = 3.788   V(s1) = 7.301   V(s2) = 4.209
iter   74   |   diff: 0.00018   |   V(s0) = 3.788   V(s1) = 7.301   V(s2) = 4.209
iter   75   |   diff: 0.00017   |   V(s0) = 3.788   V(s1) = 7.301   V(s2) = 4.209
iter   76   |   diff: 0.00015   |   V(s0) = 3.788   V(s1) = 7.301   V(s2) = 4.210
iter   77   |   diff: 0.00013   |   V(s0) = 3.789   V(s1) = 7.302   V(s2) = 4.210
iter   78   |   diff: 0.00012   |   V(s0) = 3.789   V(s1) = 7.302   V(s2) = 4.210
iter   79   |   diff: 0.00011   |   V(s0) = 3.789   V(s1) = 7.302   V(s2) = 4.210
iter   80   |   diff: 0.00010   |   V(s0) = 3.789   V(s1) = 7.302   V(s2) = 4.210
Terminated
```

In [13]:
```
if has_graphviz:
    display(plot_graph_with_state_values(mdp, state_values))
```



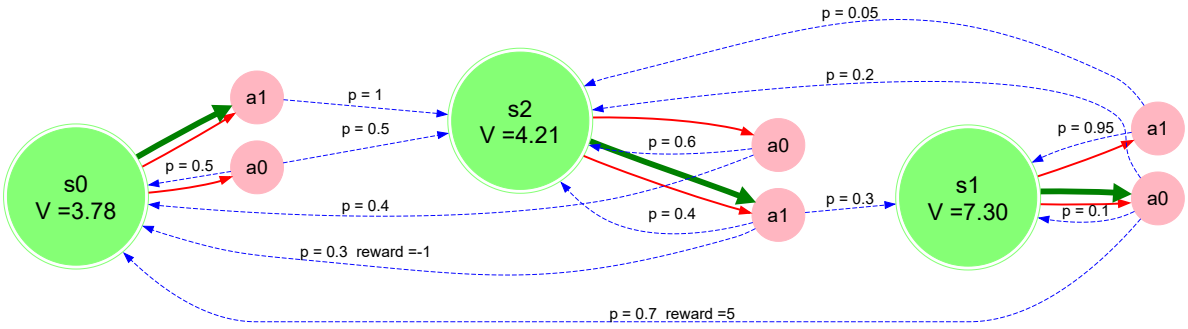
In [14]:
```
print("Final state values:", state_values)

assert abs(state_values['s0'] - 3.781) < 0.01
assert abs(state_values['s1'] - 7.294) < 0.01
assert abs(state_values['s2'] - 4.202) < 0.01
```

```
Final state values: {'s0': 3.7890708157821975, 's1': 7.3020423661017855, 's2': 4.2
101076217461606}
```

In [15]:
```
def get_optimal_action(mdp, state_values, state, gamma):
    # Finds optimal action using formula above.
    if mdp.is_terminal(state):
        return None

    nsv = get_new_state_value(mdp, state_values, state, gamma)
    a = max(nsv, key=nsv.get)

    return a
```

In [16]:
```
gamma=0.9
assert get_optimal_action(mdp, state_values, 's0', gamma) == 'a1'
assert get_optimal_action(mdp, state_values, 's2', gamma) == 'a1'
```

In [17]:
```
if has_graphviz:
    display(plot_graph_optimal_strategy_and_state_values(mdp, state_values, get_ac
```



In [18]:
```
s = mdp.reset()
rewards = []
gamma = 0.9

for _ in range(1000):
    s, r, done, _ = mdp.step(get_optimal_action(mdp, state_values, s, gamma))
    rewards.append(r)

average_reward = np.mean(rewards)
print("average reward:", average_reward)

assert 0.40 < average_reward < 0.55
```

```
average reward: 0.459
```

In [ ]: