We start by instaling the MDP library that we will use throughout the workshop.

```
!pip install mdp
```

```
Requirement already satisfied: mdp in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
```

```
import os, inspect
stdlib = os.path.dirname(inspect.getfile(os))
print(stdlib)
```

```
/usr/lib/python3.7
```

And we now mount our Drive to make use of the additional files. Here I have put these on the Colab Notebooks directoy on my Drive. If you use a different directory, the path should be changed accordingly.

```
 from google.colab import drive
 import sys
 drive.mount('/content/drive')
 sys.path.insert(0,'/content/drive/MyDrive/Colab Notebooks')
 import _mdp as mdp
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

We will now setup the practical scrpt and the virtual display for the visualisation tasks throughout. The cell may have to be run a couple of times so as to allow xvfb to start.

```
import sys, os
if 'google.colab' in sys.modules and not os.path.exists('.setup_complete'):
  # Insert the directoryimport sys
  # Run the scripts
  !setup_colab_practical3.sh -O- | bash
  !touch .setup_complete

# This code creates a virtual display to draw game images on.
# It will have no effect if your machine has a monitor.
if type(os.environ.get("DISPLAY")) is not str or len(os.environ.get("DISPLAY")
    !bash ../xvfb start
    os.environ['DISPLAY'] = ':1'
```

```
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')
```

We can now use MDP just as any other gym environment:

```
print('initial state =', mdp.reset())
next_state, reward, done, info = mdp.step('a1')
print('next_state = %s, reward = %s, done = %s' % (next_state, reward, done))
```

```
initial state = s0
next_state = s2, reward = 0.0, done = False
```

but it also has other methods that you'll need for Value Iteration

```
print("mdp.get_all_states =", mdp.get_all_states())
print("mdp.get_possible_actions('s1') = ", mdp.get_possible_actions('s1'))
print("mdp.get_next_states('s1', 'a0') = ", mdp.get_next_states('s1', 'a0'))
print("mdp.get_reward('s1', 'a0', 's0') = ", mdp.get_reward('s1', 'a0', 's0'))
print("mdp.get_transition_prob('s1', 'a0', 's0') = ", mdp.get_transition_prob(
```

```
mdp.get_all_states = ('s0', 's1', 's2')
mdp.get_possible_actions('s1') =  ('a0', 'a1')
mdp.get_next_states('s1', 'a0') =  {'s0': 0.7, 's1': 0.1, 's2': 0.2}
mdp.get_reward('s1', 'a0', 's0') =  5
mdp.get_transition_prob('s1', 'a0', 's0') =  0.7
```

Visualising MDPs

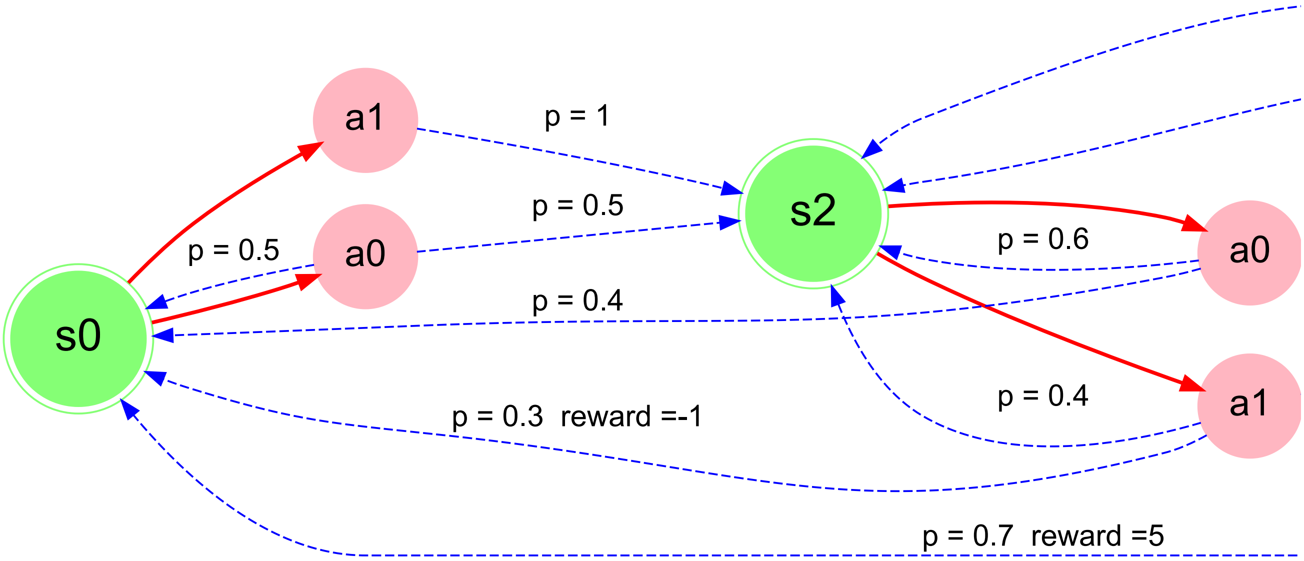
You can visualize any MDP with the drawing fuction donated by [neer201](). To do so, we have to install graphviz.

```
from _mdp import has_graphviz
from IPython.display import display
print("Graphviz available:", has_graphviz)
```

```
Graphviz available: True
```

```
if has_graphviz:
```

```
from _mdp import plot_graph, plot_graph_with_state_values, plot_graph_opti
display(plot_graph(mdp))
```



Policy iteration algorithm

Here's the pseudo-code for PI:

---

1. Pick an random policy π
2. Iterate until π is unchanged (converges).

Policy evalution:

$$V_{i+1}(s) = \sum_{s'} P(s'|s, \pi(s)) \cdot [r(s, \pi(s), s') + \gamma V_i(s')]$$

until value converges

Policy improvement:

$$\pi(s) = \max_a \sum_{s'} P(s'|s, a) \cdot [r(s, a, s') + \gamma V_i(s')]$$

First, let's write a function to compute the state-action value function $Q(\pi)$, defined as follows

$$Q_i(s, a) = \sum_{s'} P(s'|s, a) \cdot [r(s, a, s') + \gamma V_i(s')]$$

Using $Q(s, a)$ we calculate the value at state s with policy $\pi(s)$.

$$V_{i+1}(s) = \sum_{s'} P(s'|s, \pi(s)) \cdot [r(s, \pi(s), s') + \gamma V_i(s')]$$

and loop until the value if coverage, the different from current loop with previous is not much.

Then, We try update the policy base on the new value of V

$$\pi(s) = \max_a \sum_{s'} P(s'|s, a) \cdot [r(s, a, s') + \gamma V_i(s')]$$

loop until π is converges.

```
# Compute the Q-value using the formula above
def get_action_value(mdp, state_values, state, action, gamma):
    # Initialise Q
    Q = 0
    for s in mdp.get_all_states():
      # Compute Q using the equation above
      Q = Q + mdp.get_transition_prob(state, action, s)*(mdp.get_reward(state,
                                                                    gamma*state_val
    return Q
```

```
# initial simple policy
from collections import defaultdict
def init_policy():
    policy = defaultdict(lambda: {})
    for state in mdp.get_all_states():
        actions = mdp.get_possible_actions(state)
        for action in actions:
            policy[state][action] = 1. / len(actions)
    return policy
```

```
def policy_eval(mdp, policy, gamma, theta=0.00001):
    num_iter = 100  # maximum iterations, excluding initialization
    # initialize V(s)
    V = {s: 0 for s in mdp.get_all_states()}

    for i in range(num_iter):
        delta = 0
        # Compute new state values using the functions defined above.
        new_V = {}
        for s in mdp.get_all_states():
          nsv = get_action_table(mdp, V, s, gamma)
          # new_V[s] = max(nsv.items(), key=operator.itemgetter(1))[1]
          action_policy = best_action(policy[s])
          new_V[s] = nsv[action_policy]

          delta = max(delta, abs(new_V[s] - V[s]))
        assert isinstance(new_V, dict)
        V = new_V
        if delta < theta:
            break

    return V
```

```
def get_action_table(mdp, state_values, state, gamma):
    A = {a: 0 for a in mdp.get_possible_actions(state)}
    i = 0
    # Compute all possible options
    for a in mdp.get_possible_actions(state):
        v = get_action_value(mdp, state_values, state, a, gamma)
        A[a] = v
    return A
```

```python
def best_action(policy_state):
  return max(policy_state.items(), key=operator.itemgetter(1))[0] # argmax
```

```python
import operator
def policy_improvement(mdp, policy_eval_fn=policy_eval, gamma=0.9):
    num_iter = 100  # maximum iterations, excluding initialization

    policy = init_policy()
    for i in range(num_iter):
      V = policy_eval_fn(mdp, policy, gamma)
      print(V)
      # Will be set to false if we make any changes to the policy
      policy_stable = True
      # For each state...
      for state in mdp.get_all_states():
          # The best action we would take under the current policy
          chosen_a = best_action(policy[state])

          action_values = get_action_table(mdp, V, state, gamma)
          best_a = best_action(action_values)
          # Greedily update the policy
          if chosen_a != best_a:
              policy_stable = False
          for action in policy[state]:
            if action == best_a:
              policy[state][action] = 1
            else:
              policy[state][action] = 0
      # If the policy is stable we've found an optimal policy. Return it
      if policy_stable:
          return policy, V
    return policy, V
```

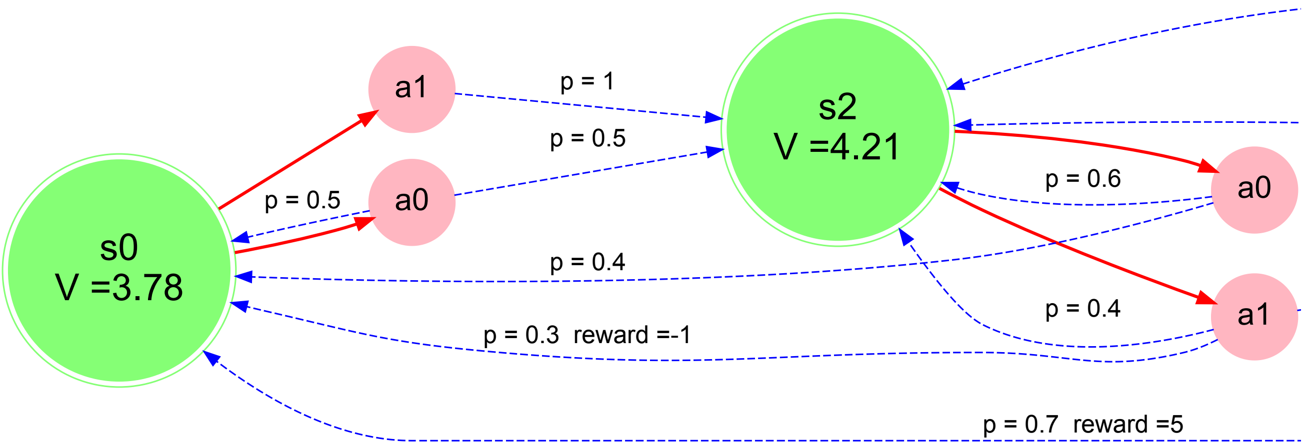
```python
import pprint
policy, v = policy_improvement(mdp)
print("Policy Probability Distribution:")
pprint.pprint(policy)
print("")
print(v)
```

```
{'s0': 0.0, 's1': 3.8461536621935, 's2': 0.0}
{'s0': 2.8398740312453152, 's1': 6.498777111635999, 's2': 3.4709784640026404}
{'s0': 3.7898252943341277, 's1': 7.302796844653717, 's2': 4.210930696013536}
Policy Probability Distribution:
defaultdict(<function init_policy.<locals>.<lambda> at 0x7f651d5efb90>,
            {'s0': {'a0': 0, 'a1': 1},
             's1': {'a0': 1, 'a1': 0},
             's2': {'a0': 0, 'a1': 1}})

{'s0': 3.7898252943341277, 's1': 7.302796844653717, 's2': 4.210930696013536}
```

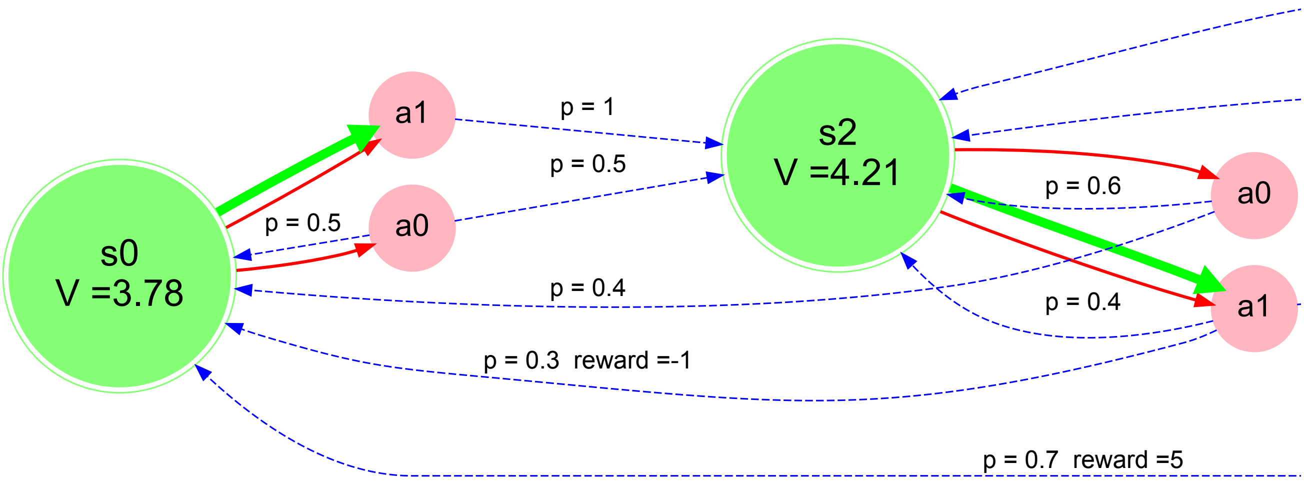
Finally, we can now plot the final graphical model

```
if has_graphviz:
    display(plot_graph_with_state_values(mdp, v))
```



Also, we can now look at te final state values.

```
if has_graphviz:
    display(plot_graph_optimal_strategy_and_state_values(mdp, v, get_action_va
```




```
def get_optimal_action(mdp, state):
    # Finds optimal action using formula above.
    if mdp.is_terminal(state):
        return None

    return best_action(policy[state])
```

```
gamma=0.9
assert get_optimal_action(mdp, 's0') == 'a1'
assert get_optimal_action(mdp, 's1') == 'a0'
assert get_optimal_action(mdp, 's2') == 'a1'
```

```
import numpy as np
```

```
# Measure agent's average reward

s = mdp.reset()
rewards = []
gamma=0.9

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

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

assert(0.40 < np.mean(rewards) < 0.55)
```

```
average reward:  0.453
```