The graph below shows a grid world that for an agent to explore. There are three terminal states and their values as indicate in the corresponding cells. All the other cells are non-terminal states except the blue cell, which represents a wall that the agent can not enter. Your job is to implement MDP method using the "Policy Iteration" approach to generate an optimal policy for each cell.

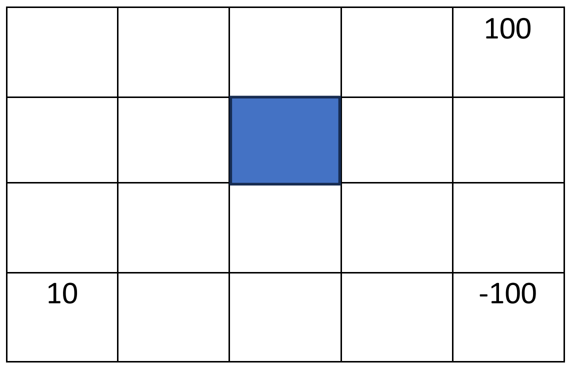
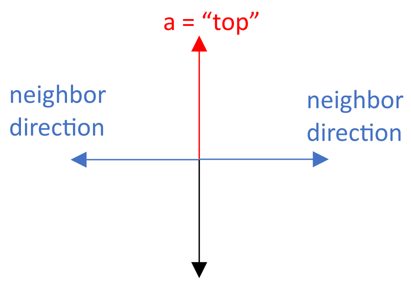


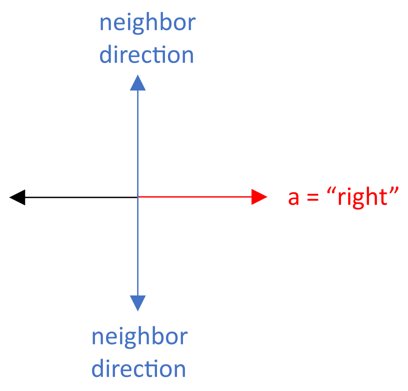
Fig: Grid world for MDP

Below are the basic setting for the Grid world environment:

(1) Action Space: {"left", "right", "up", "down"}.
At each non-terminal state, the agent can take one of the four possible actions.

(2) Transition Function Distribution: {0.6, 0.15, 0.15, 0.1}
0.6 or 60% lands on the state that is at the action direction.
0.15 or 15% lands on the two adjacent directions of the action direction.
0.1 or 10% stays at the current state
For example, a = "top", 60% lands on the top neighbor; 15% lands on the left neighbor and 15% lands on the right neighbor; 10% stay at the current state. The same rule apply to others.





If the corresponding neighbor is not available, the agent will stay at the same position. For example, if the agent locates at the top row of the gridworld, the agent takes action a = "top", then the agent has 60% + 10% = 70% stays at the current position. Because the top neighbor is not available, the 60% contribution goes to the current state.

(3) Reward = {10, 2, 2, -1}
  10 – if the agent lands on the action direction’s state.
  2 – if the agent lands on one of the two neighbor states of the action direction.
  -1 – if the agent stays at the current state

(4) Discounting factor: 0.9

Requirement:
Write a similar code as the demo in the class. Feel free to use “numpy” module or regular “list” to hold state value and Q-values. But no other additional libraries or modules are allowed to implement this project.

(1) Properly use “numpy” or “list” to hold state values and Q-values. (10%)
(2) Correctly use Policy Iteration to iteratively update policy. (20%)
(3) Correctly compute Q-value and state-value at the “Value Iteration” stage. (20%).
(4) Correctly assign an updated policy at the “Policy Improvement” stage. (20%)
(5) The algorithm converges successfully. (10%)
(6) Provide a document to show your iteratively output result (snapshot is fine) and the final output policy, or you can discuss any part not implement well. (20%)

Attention: though in the class, I mentioned you can use ChatGPT to solve problems or debug, you are not allowed to simply use ChatGPT to generate a code, which can be easily detected. Your code should follow the style of our demo code in the class.