# Lab 3: Maze

You Tao

2021-11-30

# Maze World Under MDP

- States: (col, row)； cols*rows states in total
  - Start state： State where agent starts the game
  - State pair： Jump immediately to another state in a pair
  - Goal state： End the game when agent reaches this state
- Actions

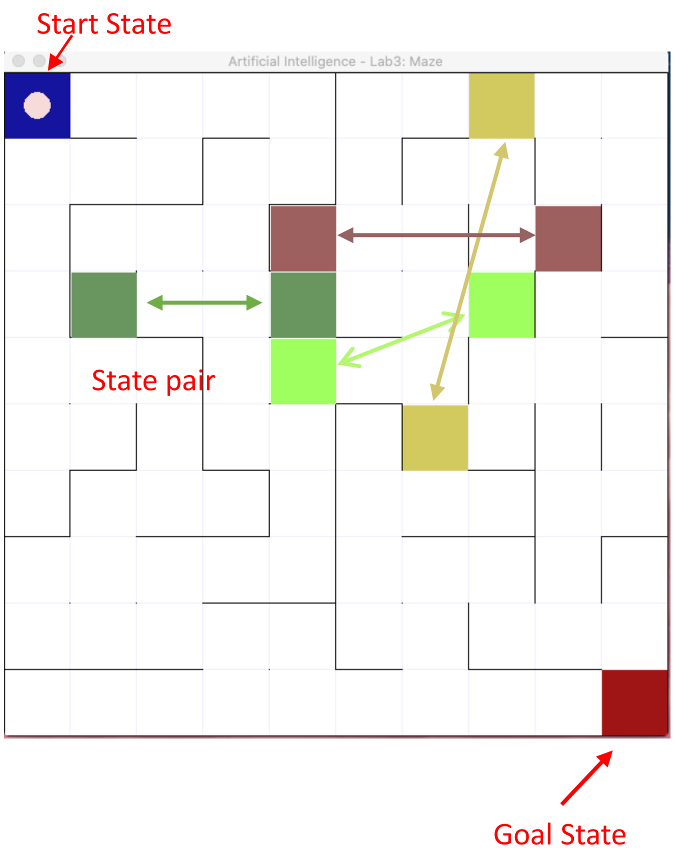
  Up / Down / Left / Right；

  Action result is determined
- Rewards

  $n = \#cells\ in\ this\ maze$

$$R(s,a) = \begin{cases} -\dfrac{5}{n}, & default\ reward \\ -\dfrac{10}{n}, & out\ of\ board\ /\ wall\ hitting \\ 10, & \text{at } the\ goal \end{cases}$$

- Discount： 0.99



# Maze World

- Problem：
  - Solve the Maze Problem based on MDP
  - Address：[10.192.9.82](10.192.9.82)
- Methods for Solving MDP
  - Value Iteration
  - Policy Iteration

# Value Iteration

- Bellman Equation:

$$V^*(s) = \max_{a \in A(s)} R(s,a) + \gamma \sum_{s'} P(s'|s,a) * V^*(s')$$

  - Synchronous Update

    Use the state value of the last iteration as $V^*(s')$

  - Asynchronous Update

    Use the latest state value as $V^*(s')$

# Policy Iteration

- Policy Evaluation

$$V^{\pi}(s) = R(s,\pi(s)) + \gamma \sum_{s'} P(s'|s,\pi(s)) * V^{\pi}(s')$$

- Policy Improvement

$$\pi(s) = \arg\max_{a \in A(s)} Q(s,a)$$

# Code Template

- game.py： For play and visualization
  - Study mode : run your AI agent
  - Human mode : play by yourself
- maze_template.py：
  - Maze Environment
  - Your MazeRLAgent

# Requirement for Lab

- Complete 3 functions in maze_template.py

  jump directly to line 256 and start reading

  - Line 396: policy_evaluation
  - Line 412: policy_iteration
  - Line 428: value_iteration

- Output： Print the iteration numbers and optimal values of all states using value iteration and policy iteration

# Reminder

- Use synchronous update when you update state values.
- Use the given method in class ‘MazeEnv’ to get legal actions of states and etc.

# Pseudocode

- Value Iteration

**function** VALUE-ITERATION($mdp$, $\epsilon$) **returns** a utility function

**inputs**: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s'|s,a)$,
rewards $R(s)$, discount $\gamma$ .
$\epsilon$, the accumulative error allowed in the utility of all states

**local variables:** $U, U'$**,** dict of utilities for states in $S$**,** initially zero
$\delta$, the accumulative change in the utility of any stage in an iteration

**repeat**
$U \leftarrow U'$; $\delta \leftarrow 0$
**for each state** $s$ **in** $S$ **do**
$U'[s] \leftarrow \max_{a \in A(s)} R(s,a) + \gamma \sum_{s'} P(s'|s,a)\, U[s']$
$\delta \leftarrow \delta + |U'[s] - U[s]|$
**until** $\delta < \epsilon$
**return** $U$

# Pseudocode

- Policy Iteration

**function** POLICY-ITERATION($mdp$) **returns** a policy

**inputs**: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s'|s,a)$
**local variables:** $U$, a dict of utilities for states in $S$**,** initially zero
$\pi$, a dict of policy whose key is state, initially random

**repeat**
$U \leftarrow \text{POLICY\_EVALUATION}(\pi, U, mdp)$
$unchanged? \leftarrow$ true
**for each state** $s$ **in** $S$ **do**
**if** $\max_{a \in A(s)} Q(s,a) > Q(s, \pi[s])$ **then do**
$\pi[s] \leftarrow \arg\max_{a \in A(s)} Q(s,a)$
$unchanged? \leftarrow$ false
**until** $unchanged?$
**return** $\pi$

Think by yourself :
How to compute the
Q-value in this problem?