Project 1：强化学习

2024.4.24

基本要求

- 1-2 人组队完成
- 通过 elearning 提交
- 提交内容：两个 Python 代码文件 + 一份 PDF 报告
  - 计算题和问答题（除 Q2.2 外）在报告中回答
  - 编程题填写到代码框架文件中
- 截止时间：5.12 晚上 24:00

代码结构

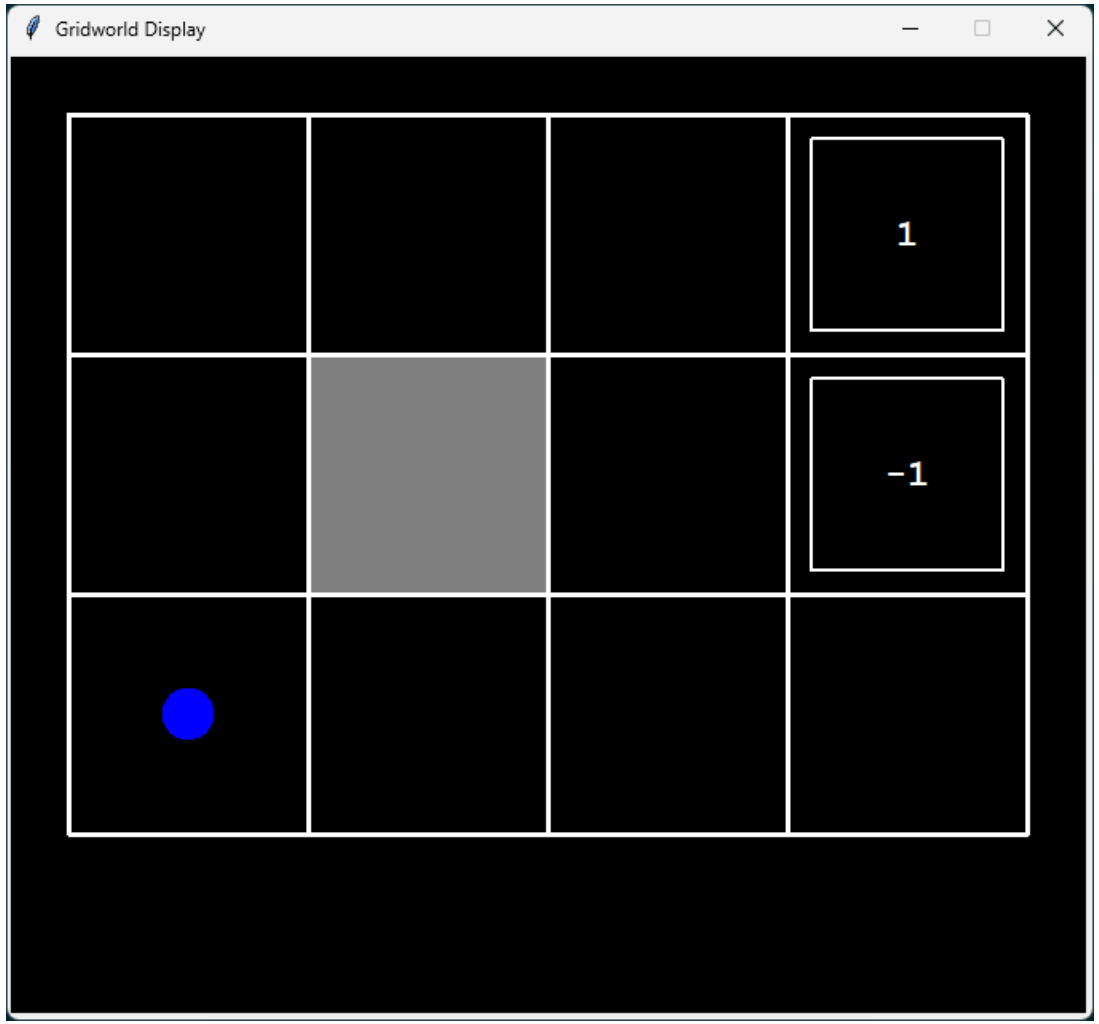
- 需要填写并提交的代码
  - qlearningAgents.py、analysis.py
- 推荐填写的代码
  - valueIterationAgents.py
- 推荐浏览的代码
  - mdp.py、learningAgents.py、util.py、gridworld.py、featureExtractors.py
- 代码自动评测命令
  - python autograder.py

注意事项

- 请勿修改其他文件或自行添加新的文件
  - 提交时代码部分只需要包括两个需要提交的文件
- 请勿修改给定的函数名或类名
- 推荐用 LaTeX 生成 PDF 报告
  - 允许扫描手写，但需要保证答案过程清晰

格子世界简介

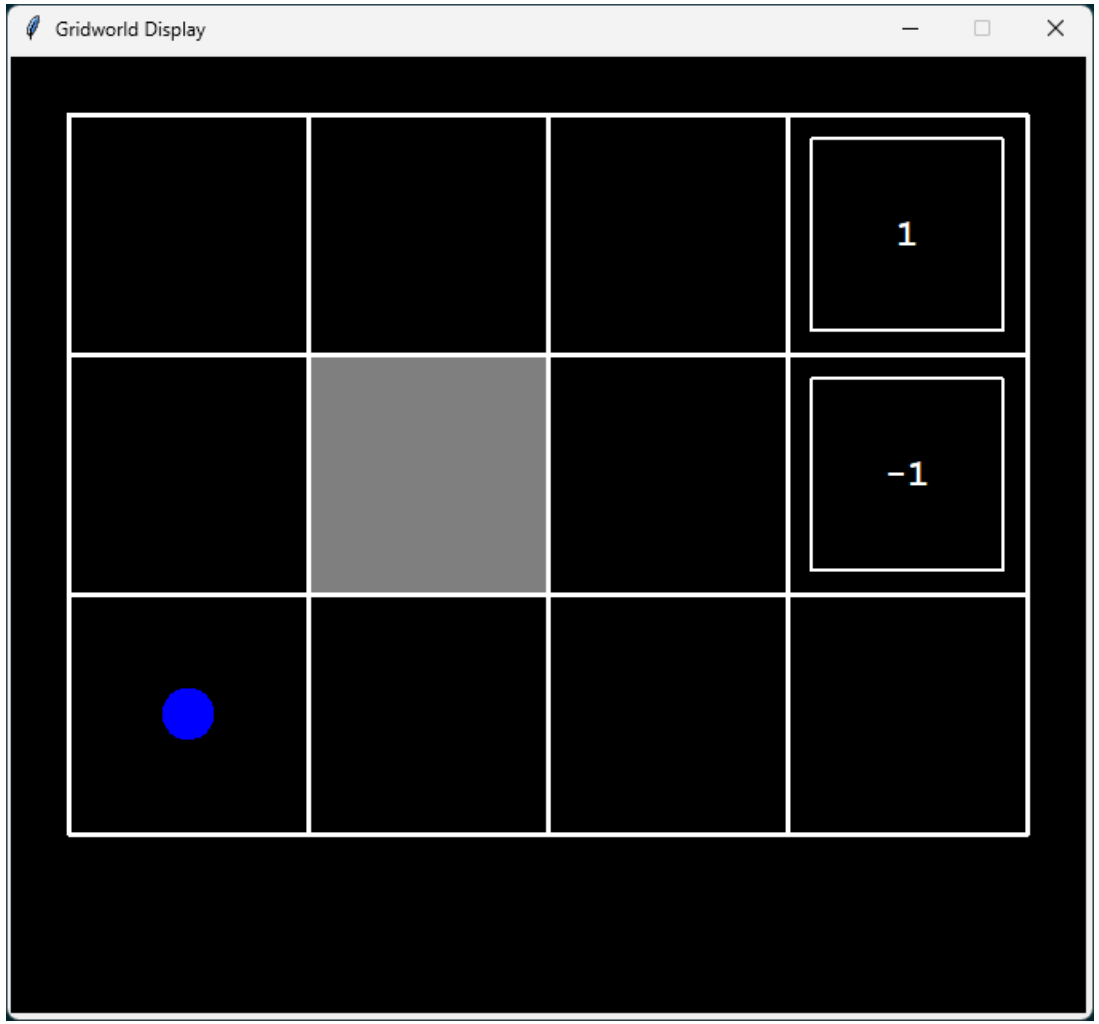
- **键盘操作 agent**
  - python gridworld.py –m
  - 默认设置下 agent 有概率随机移动
  - 进入终止方格（双框）后，需要再操作一次才能进入真正的终止状态
  - 控制台会显示状态转移信息
- **通过 -h 选项查看可用的选项**
  - 例（更改地图）：python gridworld.py -g MazeGrid



Q1：值迭代

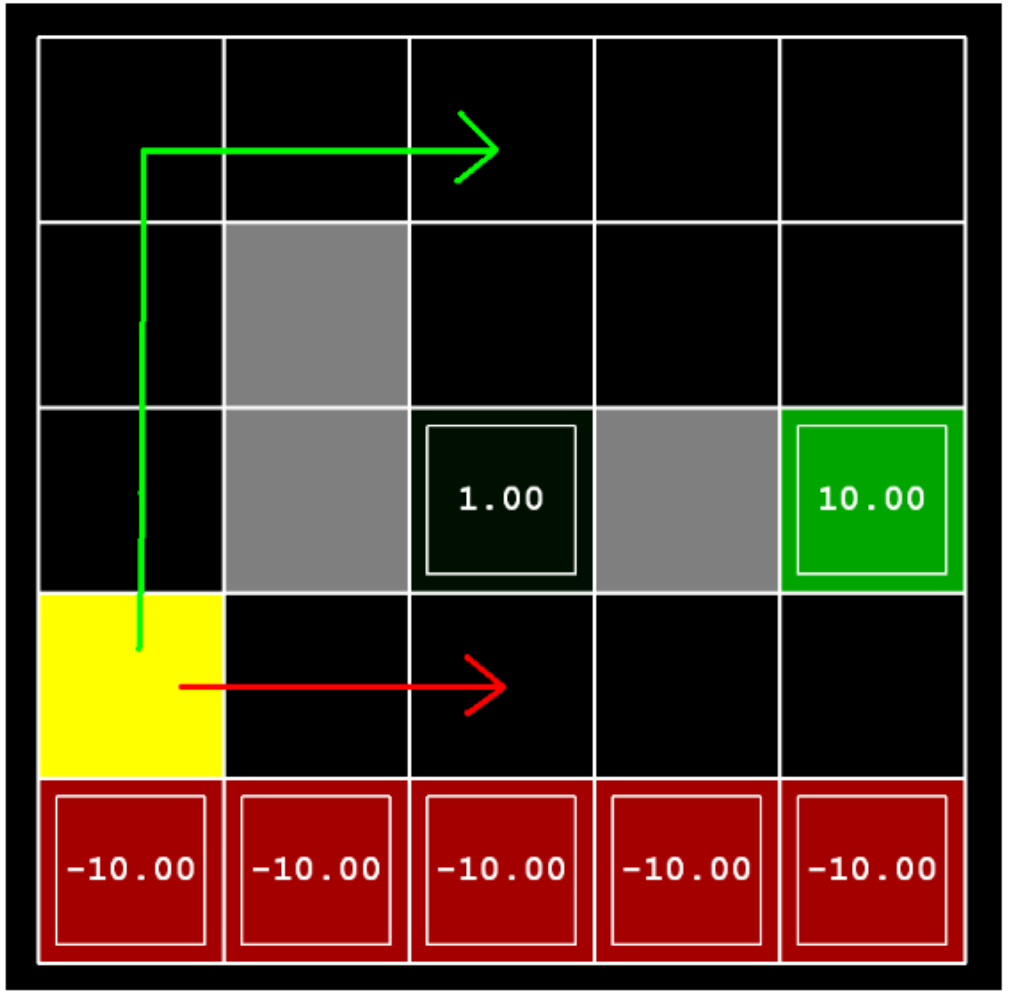
- 计算两轮值迭代后各个状态的值
  - 所有状态初始值为 0
  - 噪声为 0.2
    - 除终止方格外，任何方格的动作都有20%的概率向某个垂直方向移动
  - agent 撞墙会停在原地
  - 除终止动作外奖励值为 0
- 请采用同步更新方法计算

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s)]$$



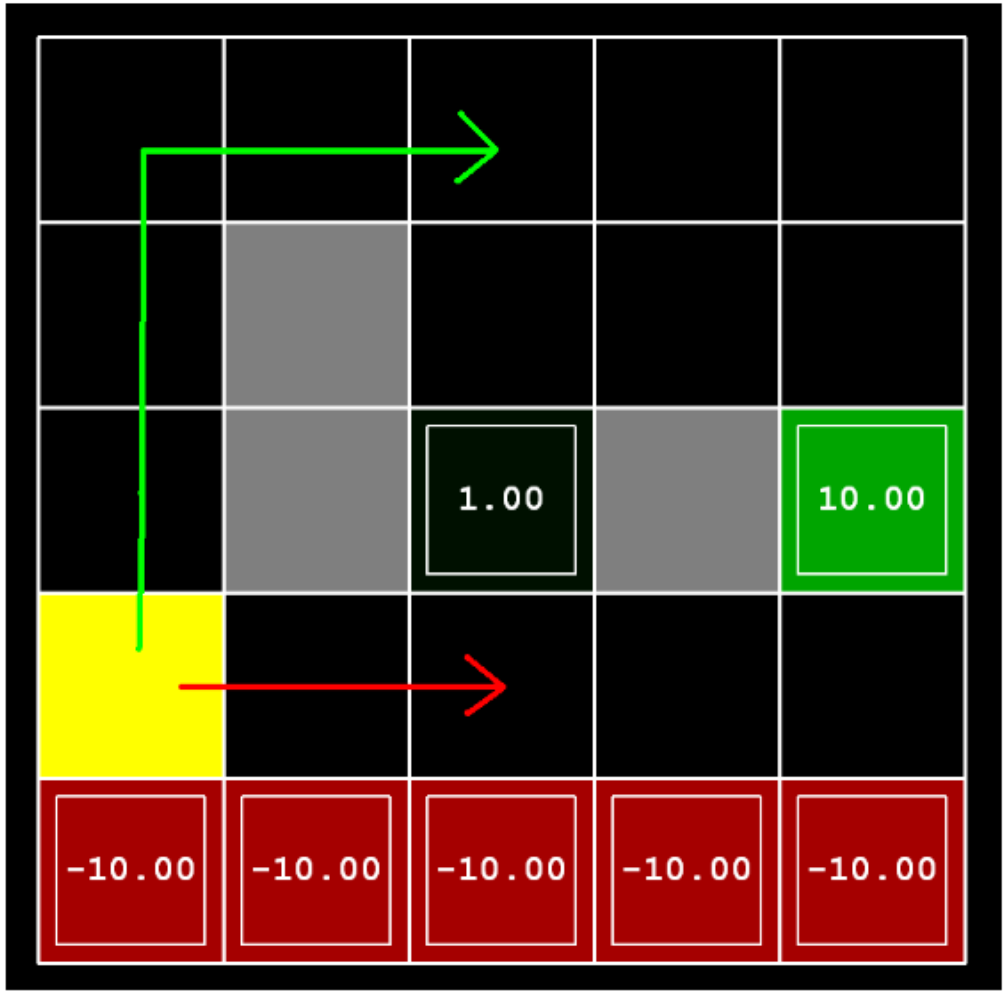
Q2：策略构建

- 新地图：DiscountGrid
  - 两个正奖励出口：近+1，远+10
  - 底部一排负奖励（-10）出口
- 两种获得正奖励的路线
  - 红色：冒险路线
  - 绿色：稳妥路线
- 考察三种参数对最优策略的影响
  - 噪声 n、折扣系数 γ、生存奖励 r



Q2：策略构建

- Q2.1：假定没有噪声，计算生成下列最优策略的参数范围
  - 倾向于选择+1出口，冒险路线；
  - 倾向于选择+10出口，冒险路线；
  - 倾向于选择+1出口，稳妥路线；
  - 倾向于选择+10出口，稳妥路线；
  - 避免走到出口，整个过程不会结束；
  - 放弃游戏，直接进入-10出口。
- 给出 γ 和 r 的关系式或证明无解



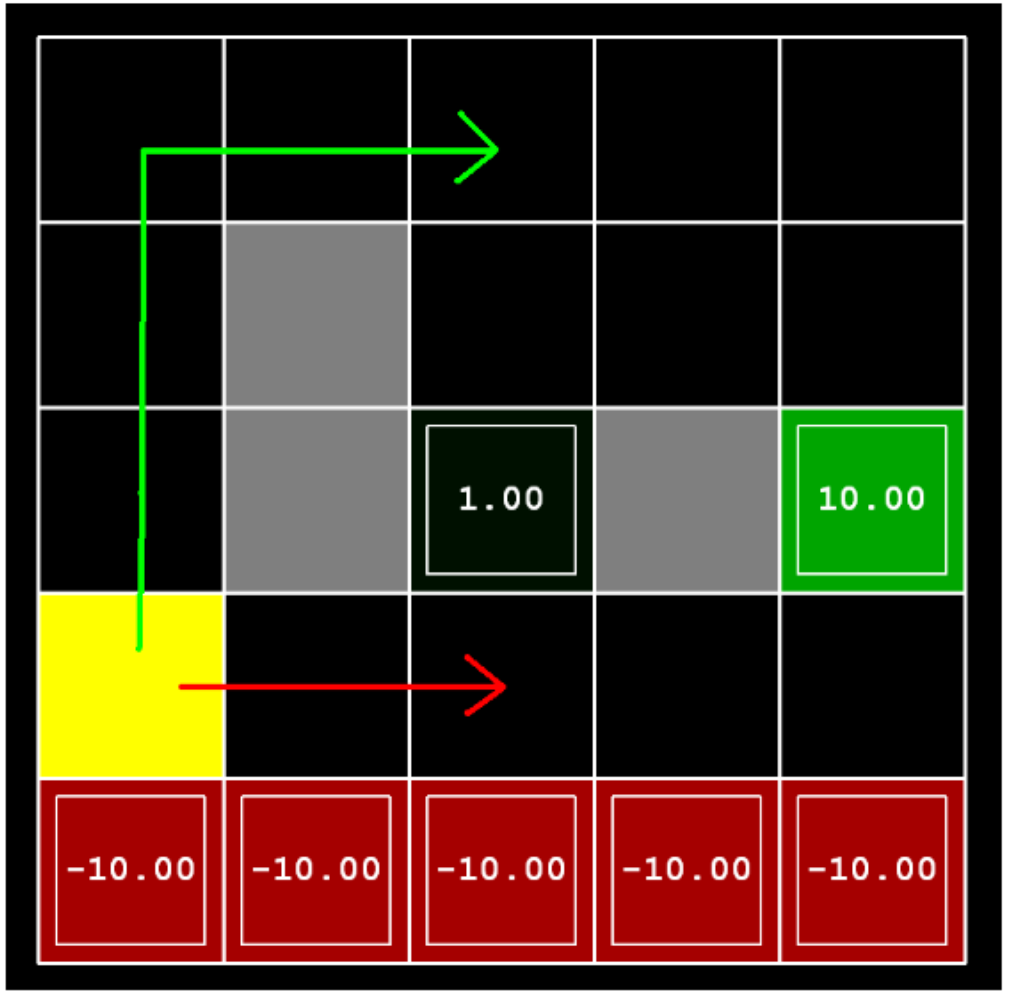
Q2：策略构建

- Q2.2：给出生成 Q2.1 中前五种最优策略的参数组合 (n, γ, r)
  - 可以利用 Q2.1 的结果
  - 如果 Q2.1 无解，就需要改变噪声 n
- 结果填写在 analysis.py 中
  - 若无解，返回 "NOT POSSIBLE"

```python
def question2a():
    """
      Prefer the close exit (+1), risking the cliff (-10).
    """
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question2b():
    """
      Prefer the close exit (+1), but avoiding the cliff (-10).
    """
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question2c():
    """
      Prefer the distant exit (+10), risking the cliff (-10).
    """
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question2d():
    """
      Prefer the distant exit (+10), avoiding the cliff (-10).
    """
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'

def question2e():
    """
      Avoid both exits and the cliff (so an episode should never terminate).
    """
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

Q2：策略构建

- Q2.3（附加题）：不同的生存奖励 r 会产生哪些最优策略?
  - $n = 0.3$、$\gamma \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$
  - 仅无关格子行动不同的两个策略等价
  - 不要求解析解，给出数值范围即可
- 提示：固定策略的效用值随 r 线性变化
  - 为什么?



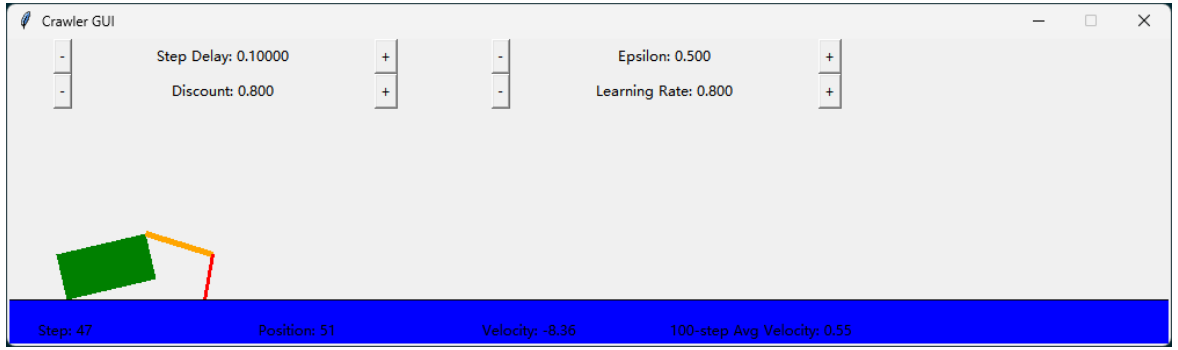
Q3：Q-Learning

- 编写 Q-Learning 的 agent（QLearningAgent）
  - 需要实现 update、computeValueFromQValues、getQValue 和 computeActionFromQValues
  - 注意**相等**和**未见动作**的 Q 值
- 手动运行 gridworld 观察
  - python gridworld.py -a q -k 5 -m

```
def getQValue(self, state, action):
    """
      Returns Q(state,action)
      Should return 0.0 if we have never seen a state
      or the Q node value otherwise
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

def computeValueFromQValues(self, state):
    """
      Returns max_action Q(state,action)
      where the max is over legal actions.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return a value of 0.0.
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

def computeActionFromQValues(self, state):
    """
      Compute the best action to take in a state.  Note that if there
      are no legal actions, which is the case at the terminal state,
      you should return None.
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

def getAction(self, state):
    """
      Compute the action to take in the current state.  With
      probability self.epsilon, we should take a random action and
      take the best policy action otherwise.  Note that if there are
      no legal actions, which is the case at the terminal state, you
      should choose None as the action.
      HINT: You might want to use util.flipCoin(prob)
      HINT: To pick randomly from a list, use random.choice(list)
    """
    # Pick Action
    legalActions = self.getLegalActions(state)
    action = None
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

    return action

def update(self, state, action, nextState, reward: float):
    """
      The parent class calls this to observe a
      state = action => nextState and reward transition.
      You should do your Q-Value update here
      NOTE: You should never call this function,
      it will be called on your behalf
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()
```

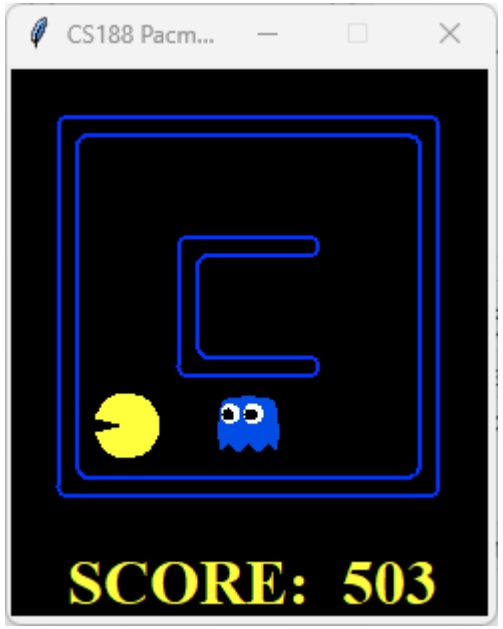
Q4：ε-greedy

- 实现 Q3 中 agent 的 ε-greedy
  - 需要实现 getAction
- 自动运行 gridworld 观察
  - python gridworld.py -a q -k 100
  - 通过 –e 选项调整探索概率
- 彩蛋：单臂机器人
  - python crawler.py
  - 实现正确后可以观看机器人学习爬行



Q5：Q-Learning Pacman

- 在 Pacman 中检验 Q-Learning
  - python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
  - 无需填写代码
- 观察训练过程
  - python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
- 注意未见动作的 Q 值



Q6：值函数近似 Q-Learning

- 编写值函数近似 Q-Learning 的 agent（ApproximateQAgent）
  - 给定特征向量 $f(s,a)$，维护权重向量 w，使得 $Q(s,a)=\sum_{i=1}^{n} f_i(s,a)w_i$
- 权重更新公式
  - $w_i \leftarrow w_i + \alpha\Delta \cdot f_i(s,a)$
  - $\Delta = \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$
- 正确实现后可以学习大地图
  - -l mediumGrid / –l mediumClassic

```python
class ApproximateQAgent(PacmanQAgent):
    """
       ApproximateQLearningAgent
       You should only have to overwrite getQValue
       and update.  All other QLearningAgent functions
       should work as is.
    """
    def __init__(self, extractor='IdentityExtractor', **args):
        self.featExtractor = util.lookup(extractor, globals())()
        PacmanQAgent.__init__(self, **args)
        self.weights = util.Counter()

    def getWeights(self):
        return self.weights

    def getQValue(self, state, action):
        """
          Should return Q(state,action) = w * featureVector
          where * is the dotProduct operator
        """
        "*** YOUR CODE HERE ***"
        util.raiseNotDefined()

    def update(self, state, action, nextState, reward: float):
        """
           Should update your weights based on transition
        """
        "*** YOUR CODE HERE ***"
        util.raiseNotDefined()

    def final(self, state):
        """Called at the end of each game."""
        # call the super-class final method
        PacmanQAgent.final(self, state)

        # did we finish training?
        if self.episodesSoFar == self.numTraining:
            # you might want to print your weights here for debugging
            "*** YOUR CODE HERE ***"
            pass
```

基本要求 x2

- 1-2 人组队完成
- 通过 elearning 提交
- 提交内容：两个 Python 代码文件 + 一份 PDF 报告
  - 计算题和问答题（除 Q2.2 外）在报告中回答
  - 编程题填写到代码框架文件中
- 截止时间：5.12 晚上 24:00


SCHOOL OF DATA SCIENCE FUDAN UNIVERSITY
复旦大学大数据学院


FUDAN UNIVERSITY
1905

Thanks!