# Assignment 1

Name: Ng Tze Kean

Student number: 721370290002

## Architecture

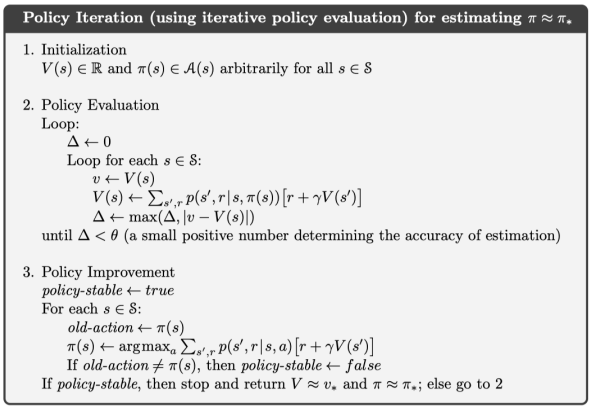
Classes are created to simulate the agents interacting with the gridworld. The gridworld acts as the environment which feeds the agent with the data. The agent will interact with the gridworld by calling for update of the current state with some action the agent wants to perform. The gridworld will return the reward and next state to the agent.

## Random algorithm

The random algorithm always performs the worst of the 3 since there is no use of information to select the optimal action. the random algorithm forms the trivial base line that the other 2 algorithms should beat to objectively say that there is an improvement.

## Iterative policy evaluation and iteration

The implementation of the policy evaluation and iteration is as shown in the textbook.



**Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$**

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   *policy-stable* $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      *old-action* $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Whereby the policy evaluation and iteration wrapped in a function and iteratively called till convergence of the algorithm.

## Value iteration

Value iteration on the other hand will compute the value of quality of the state that the agent is taking across all possible actions and its future quality given a state. The quality of the state is then updated with the maximum possible value that can be attained.

**Value Iteration, for estimating $\pi \approx \pi_*$**

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

The loop is repeated till convergence where by the optimal action is found. To compute the best action, we can either run another iteration of the training process to obtain the best action or we could create a function that finds the $\argmax_a V(s)$ and have the agent take that action.

## Results

Running the `main.py` program will run the *RandomAgent* agent, followed by *IterativePolicyEvaluation* then *ValueIteration*. We can see from the output how the agent moves through the gridworld and the number of steps taken.

Since there are 2 goal states, the agent will prioritize the goal state that is closest to the start location. As the random agent is not determinstic and optimal, it almost always performs poorly compared to the 2 algorithm.

```
================================================
Agent is created
This is agent IterativePolicyEvaluation
The starting point is 30
This IterativePolicyEvaluation will start now.
state was 30, action is e, state is now 31
state was 31, action is e, state is now 32
state was 32, action is e, state is now 33
state was 33, action is e, state is now 34
state was 34, action is e, state is now 35
Number of steps taken is 5
The action taken are ['e', 'e', 'e', 'e', 'e']
================================================
Agent is created
This is agent ValueIteration
The starting point is 30
This ValueIteration will start now.
state was 30, action is e, state is now 31
state was 31, action is e, state is now 32
state was 32, action is e, state is now 33
state was 33, action is e, state is now 34
state was 34, action is e, state is now 35
Number of steps taken is 5
The action taken are ['e', 'e', 'e', 'e', 'e']
================================================
```

### Use of program

To change the starting position of the program, one can modify line *184* of the code to change the starting position.

```
grid = Gridworld(start=30)
```

The other parameters of the agents can also be selected to change how the agent will learn from the environment. As for the other agents (not including the random agent), they would have to be first trained to learn the policy or value of the state before an optimal action can be learnt.