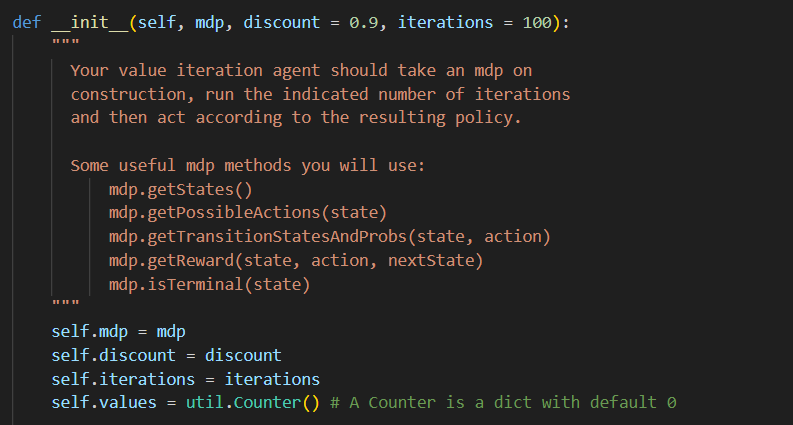
### ***Question 1 (6 points)*** Write a value iteration agent in ValueIterationAgent, which has been partially specified for you in [valueIterationAgents.py](https://classes.engineering.wustl.edu/cse511/p3/docs/valueIterationAgents.php). Your value iteration agent is an offline planner, not a reinforcement learning agent, and so the relevant training option is the number of iterations of value iteration it should run (option -i) in its initial planning phase. ValueIterationAgent takes an MDP on construction and runs value iteration for the specified number of iterations before the constructor returns.

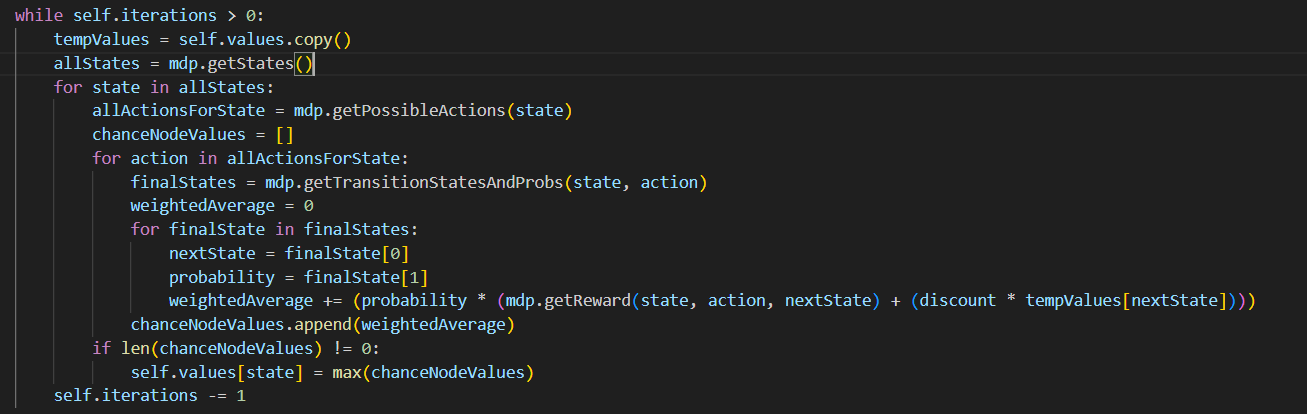


**mdp**: môi trường mà agent sẽ tương tác (Markov Decision Process).

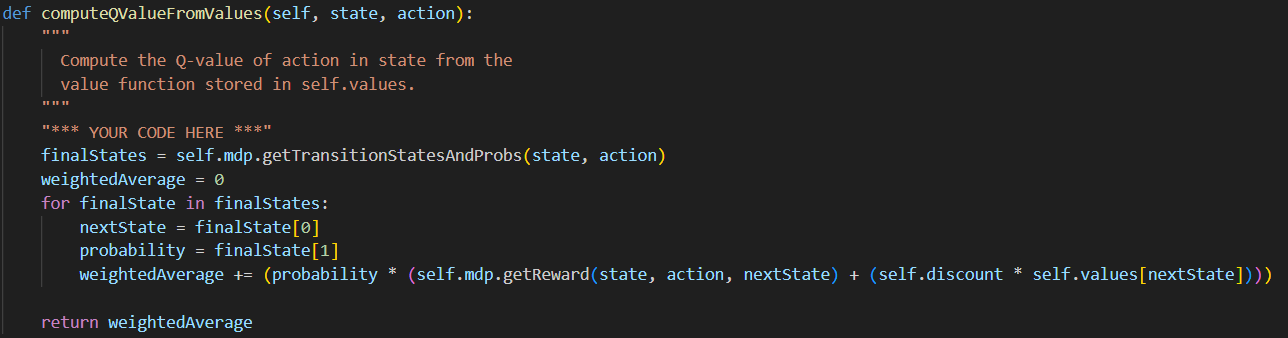
**discount**: Hệ số chiết khấu giúp giảm tầm quan trọng của phần thưởng tương lai

**iterations**: Số lần lặp mà Value Iteration sẽ thực hiện.

**values:** Khởi tạo các giá trị của trạng thái bằng 0



Vòng lặp này thực hiện thuật toán **Value Iteration** để tính toán và cập nhật giá trị của các trạng thái trong một MDP (Markov Decision Process). Mục tiêu là tìm giá trị tối ưu cho từng trạng thái thông qua nhiều lần lặp.



Hàm computeQValueFromValues(self, state, action) được thiết kế để tính toán **giá trị Q (Q-value)** cho một hành động cụ thể từ một trạng thái đã cho trong Markov Decision Process (MDP). Giá trị Q biểu thị mức độ kỳ vọng về phần thưởng khi thực hiện hành động đó từ trạng thái hiện tại, có tính đến xác suất chuyển tiếp tới các trạng thái tiếp theo và giá trị của chúng.

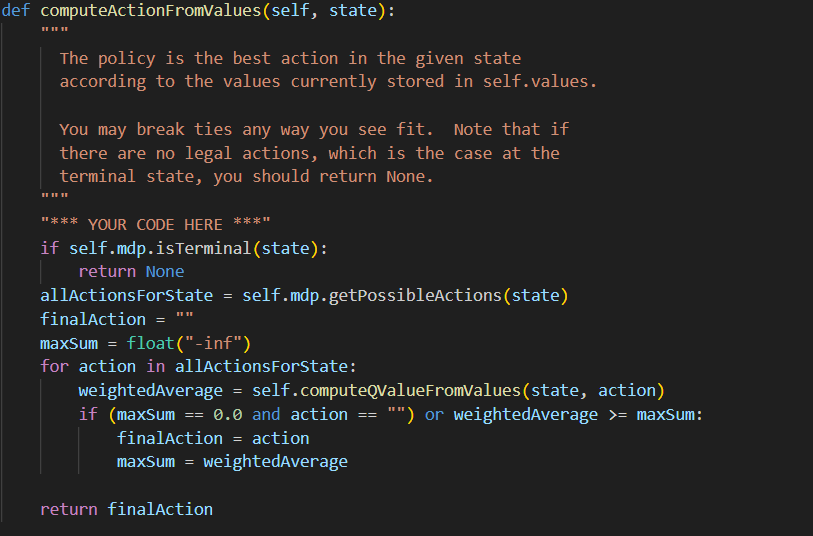
* Sử dụng phương thức mdp.getTransitionStatesAndProbs(state, action) để lấy danh sách các cặp **(nextState, probability)**:
* **nextState**: Trạng thái tiếp theo mà agent có thể chuyển đến khi thực hiện hành động.
* **probability**: Xác suất chuyển tiếp từ trạng thái hiện tại tới trạng thái tiếp theo khi thực hiện hành động đó.
* Biến weightedAverage được khởi tạo bằng 0 để lưu trữ giá trị Q (giá trị kỳ vọng) cho hành động.
* Hàm lặp qua từng cặp **(nextState, probability)**:
* **nextState**: Trạng thái tiếp theo.
* **probability**: Xác suất dẫn tới trạng thái đó.
* **Tính giá trị kỳ vọng** sử dụng công thức:

weightedAverage += probability \* (reward + discount \* valueOfNextState)

* **reward**: Được tính bởi mdp.getReward(state, action, nextState) và biểu thị phần thưởng tức thời khi chuyển từ trạng thái hiện tại tới trạng thái tiếp theo thông qua hành động.
* **discount**: Hệ số giảm giá, phản ánh tầm quan trọng của phần thưởng trong tương lai.
* **valueOfNextState**: Giá trị của trạng thái tiếp theo đã được tính trước trong quá trình Value Iteration.
* Giá trị Q của hành động được tính bằng cách cộng dồn các giá trị kỳ vọng từ các trạng thái tiếp theo:

Q(state, action) = Σ (probability \* (reward + discount \* valueOfNextState))

* Mỗi phần của tổng đại diện cho giá trị kỳ vọng của việc thực hiện hành động đó, xét tới xác suất, phần thưởng và giá trị giảm giá của trạng thái tiếp theo.
* Sau khi tính toán giá trị kỳ vọng cho tất cả các trạng thái tiếp theo, hàm sẽ trả về giá trị **weightedAverage**. Đây chính là giá trị Q của hành động tại trạng thái đó.



Hàm này có nhiệm vụ tìm và trả về **hành động tốt nhất** từ một trạng thái cụ thể dựa trên các giá trị (value) đã được tính toán trước đó. Giá trị này được tính toán bằng thuật toán Value Iteration.

* Sử dụng mdp.isTerminal(state) để kiểm tra xem trạng thái hiện tại có phải là trạng thái kết thúc không.
* Nếu là trạng thái kết thúc, không có hành động nào cần thực hiện và hàm sẽ trả về None.
* Dùng mdp.getPossibleActions(state) để lấy tất cả các hành động khả thi từ trạng thái này.
* Nếu không có hành động nào khả dụng, hàm sẽ không tính tiếp.
* finalAction: Khởi tạo là chuỗi rỗng, dùng để lưu lại hành động tốt nhất.
* maxSum: Khởi tạo là giá trị âm vô cùng (-inf), dùng để theo dõi giá trị lớn nhất của các hành động.
* Lặp qua từng hành động trong danh sách hành động khả thi.
* Sử dụng hàm computeQValueFromValues(state, action) để tính giá trị Q (Q-value) cho từng hành động. Giá trị Q thể hiện giá trị kỳ vọng của hành động dựa trên các trạng thái tiếp theo và xác suất chuyển tới chúng.
* Sau khi tính giá trị Q của hành động, so sánh nó với giá trị lớn nhất hiện tại (maxSum).

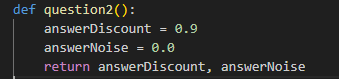
### ***Question 2 (1 point)*** BridgeGrid is a grid world map with the a low-reward terminal state and a high-reward terminal state separated by a narrow "bridge", on either side of which is a chasm of high negative reward. The agent starts near the low-reward state. With the default discount of 0.9 and the default noise of 0.2, the optimal policy does not cross the bridge. Change only ONE of the discount and noise parameter

### s so that the optimal policy causes the agent to attempt to cross the bridge. Put your answer in question2() of [analysis.py](https://classes.engineering.wustl.edu/cse511/p3/docs/analysis.php). (Noise refers to how often an agent ends up in an unintended successor state when they perform an action.)

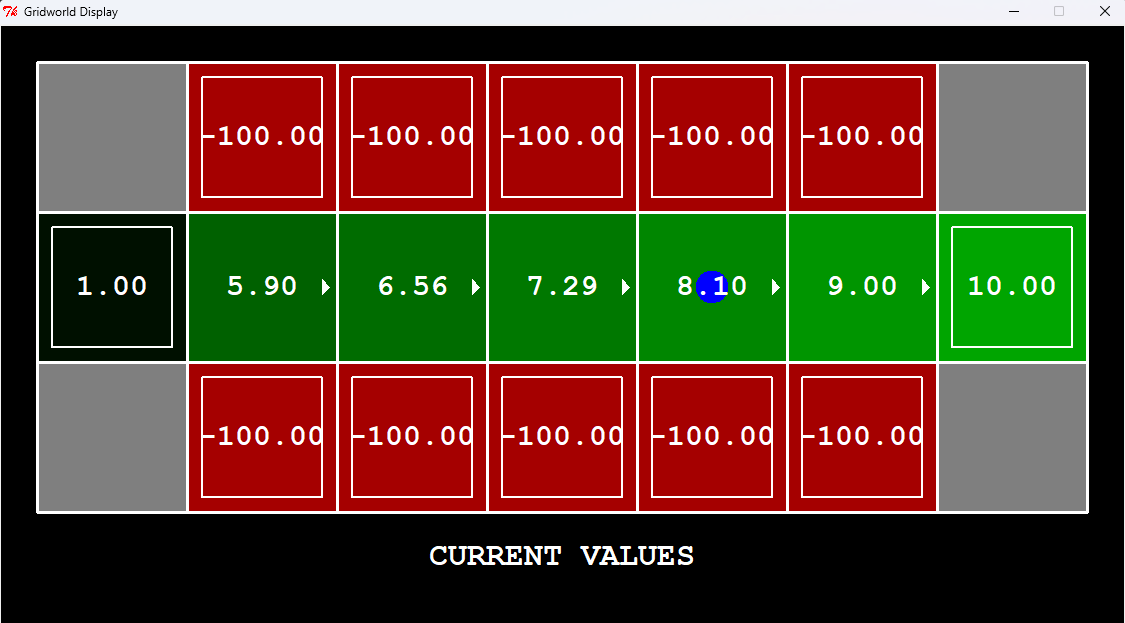
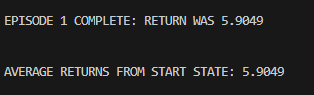
Đề bài yêu cầu điều chỉnh một trong hai tham số: discount (hệ số chiết khấu) hoặc noise (độ nhiễu) để khiến agent (tác nhân) trong môi trường BridgeGrid chọn chiến lược đi qua cây cầu, thay vì tránh nó.

### **Phân tích:**

1. **Giảm độ nhiễu (noise)**:
   * Nếu giảm độ nhiễu, các hành động của tác nhân sẽ chính xác hơn, nghĩa là xác suất tác nhân đến đúng nơi mong muốn sẽ cao hơn. Khi đó, tác nhân có thể tự tin hơn khi chọn con đường qua cây cầu hẹp.
2. **Tăng hệ số chiết khấu (discount)**:
   * Nếu tăng hệ số chiết khấu, tác nhân sẽ quan tâm hơn đến các phần thưởng lớn ở tương lai (phần thưởng cao ở đầu cầu bên kia) thay vì chỉ tập trung vào phần thưởng thấp gần đó. Điều này có thể khiến tác nhân cố gắng đi qua cầu để đạt phần thưởng cao.



Kết quả nhận được:



### ***Question 3 (5 points)*** Consider the DiscountGrid layout, shown below. This grid has two terminal states with positive payoff (shown in green), a close exit with payoff +1 and a distant exit with payoff +10. The bottom row of the grid consists of terminal states with negative payoff (shown in red); each state in this "cliff" region has payoff -10. The starting state is the yellow square. We distinguish between two types of paths: (1) paths that "risk the cliff" and travel near the bottom row of the grid; these paths are shorter but risk earning a large negative payoff, and are represented by the red arrow in the figure below. (2) paths that "avoid the cliff" and travel along the top edge of the grid. These paths are longer but are less likely to incur huge negative payoffs. These paths are represented by the green arrow in the figure below.

**Câu hỏi :**

Trong lưới DiscountGrid, có hai trạng thái kết thúc với phần thưởng dương (lối ra gần +1, lối ra xa +10) và một hàng dưới cùng với phần thưởng âm (-10). Bạn sẽ chọn các tham số chiết khấu (discount), nhiễu (noise), và phần thưởng sống (living reward) để tạo ra các chính sách tối ưu khác nhau. Nếu không thể đạt được hành vi nào đó, hãy trả về 'NOT POSSIBLE'.

**Các chính sách cần đạt:**

* + Ưu tiên lối ra gần (+1), mạo hiểm với vách đá (-10)
  + Ưu tiên lối ra gần (+1), nhưng tránh vách đá (-10)
  + Ưu tiên lối ra xa (+10), mạo hiểm với vách đá (-10)
  + Ưu tiên lối ra xa (+10), nhưng tránh vách đá (-10)
  + Tránh cả lối ra và vách đá

**Khái niệm các tham số:**

### **1. Discount (Chiết khấu):**

* **Giá trị chiết khấu** xác định mức độ mà tác nhân (agent) ưu tiên các phần thưởng trong tương lai.
  + **Giá trị cao (gần 1)**: Tác nhân ưu tiên phần thưởng dài hạn, thường chọn các hành động dẫn đến kết quả tốt hơn trong tương lai.
  + **Giá trị thấp (gần 0)**: Tác nhân ưu tiên phần thưởng ngay lập tức, không quan tâm nhiều đến các phần thưởng trong tương lai.

### **2. Noise (Nhiễu):**

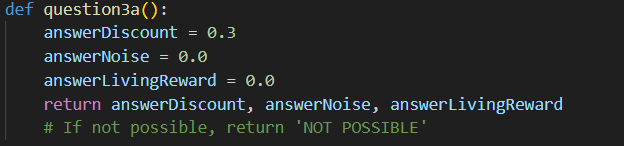
* **Nhiễu** là mức độ không chắc chắn trong môi trường. Nếu nhiễu cao, tác nhân có thể không luôn thực hiện chính xác hành động mong muốn.
  + **Nhiễu thấp (gần 0):** Tác nhân có thể kiểm soát tốt hơn hành động của mình.
  + **Nhiễu cao (gần 1):** Tác nhân gặp nhiều bất định hơn và có khả năng mắc lỗi khi hành động.

### **3. Living Reward (Phần thưởng sống):**

* **Phần thưởng sống** là phần thưởng mà tác nhân nhận được ở mỗi bước di chuyển (không phải là phần thưởng từ trạng thái kết thúc). Điều này có thể khuyến khích hoặc ngăn tác nhân khám phá nhiều trạng thái khác nhau.
  + **Phần thưởng sống cao (lớn hơn 0):** Tác nhân sẽ có xu hướng di chuyển nhiều hơn, tránh các trạng thái kết thúc (lối ra).
  + **Phần thưởng sống thấp hoặc âm:** Tác nhân sẽ tìm cách kết thúc sớm hơn để tối đa hóa phần thưởng.

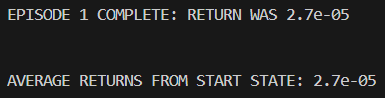
### **Các chính sách :**

**1/ Ưu tiên lối ra gần (+1), mạo hiểm với vách đá (-10):**

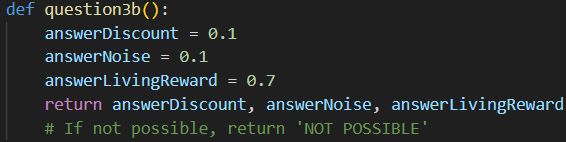


* **Chiết khấu**: Giá trị chiết khấu vừa phải (0.3) để tác nhân không quá ưu tiên phần thưởng dài hạn.
* **Nhiễu**: Nhiễu rất thấp (0.0) để đảm bảo tác nhân có thể điều khiển chính xác và mạo hiểm với vách đá.
* **Phần thưởng sống**: Không cần phần thưởng sống (0.0) vì tác nhân muốn kết thúc sớm với phần thưởng gần.

**Kết quả :**

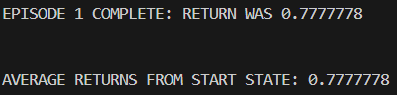
****

**2/Ưu tiên lối ra gần (+1), tránh vách đá (-10):**

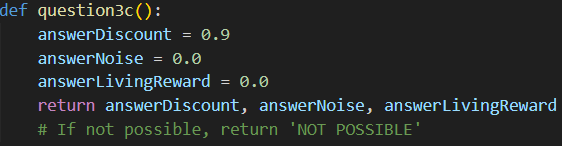
****

* **Chiết khấu**: Giá trị chiết khấu thấp (0.1), tập trung vào phần thưởng ngắn hạn.
* **Nhiễu**: Nhiễu nhỏ (0.1), cho phép tác nhân tránh rủi ro từ vách đá.
* **Phần thưởng sống**: Phần thưởng sống dương (0.7) khuyến khích tác nhân tránh lối đi nguy hiểm.

**Kết quả:**

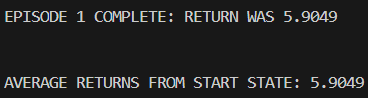


**3/Ưu tiên lối ra xa (+10), mạo hiểm với vách đá (-10):**

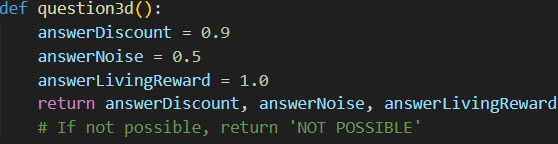
****

* **Chiết khấu:** Giá trị chiết khấu cao (0.9), ưu tiên phần thưởng dài hạn (lối ra xa +10).
* **Nhiễu:** Không có nhiễu (0.0) để tác nhân có thể điều khiển chính xác.
* **Phần thưởng sống:** Không có phần thưởng sống (0.0) để khuyến khích tác nhân đi nhanh đến lối ra xa.

**Kết quả:**

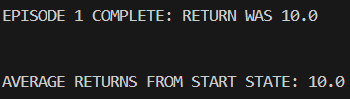


**4/Ưu tiên lối ra xa (+10), tránh vách đá (-10)**:

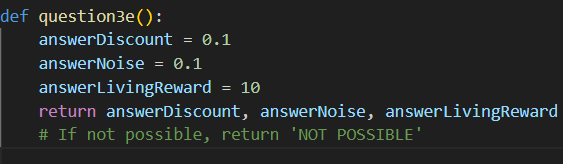


* **Chiết khấu**: Giá trị chiết khấu cao (0.9) để tác nhân ưu tiên phần thưởng dài hạn.
* **Nhiễu**: Nhiễu cao hơn (0.5), khiến tác nhân cẩn trọng và tránh đi qua vách đá.
* **Phần thưởng sống**: Phần thưởng sống cao (1.0), giúp tác nhân ưu tiên sống lâu hơn và tránh rủi ro.

**Kết quả:**



**5/Tránh cả lối ra và vách đá:**

****

* **Chiết khấu thấp (0.1)**: Không ưu tiên phần thưởng dài hạn.
* **Nhiễu thấp (0.1)**: Có một ít rủi ro trong hành động nhưng tác nhân phần lớn kiểm soát được.
* **Phần thưởng sống cao (10)**: Tác nhân có động lực để kéo dài cuộc sống thay vì kết thúc nhanh bằng cách đi đến lối ra, khiến nó tránh cả lối ra và vách đá.

**Kết quả :** Tác nhân ở trong một chuỗi hành động k hồi kết

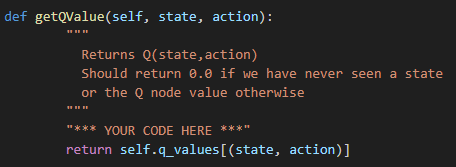
### ***Question 4 (5 points)*** You will now write a Q-learning agent, which does very little on construction, but instead learns by trial and error from interactions with the environment through its update(state, action, nextState, reward) method. A stub of a Q-learner is specified in QLearningAgent in [qlearningAgents.py](https://classes.engineering.wustl.edu/cse511/p3/docs/qlearningAgents.php), and you can select it with the option '-a q'. For this question, you must implement the update, getValue, getQValue, and getPolicy methods.

### **Tổng quan:**

Trong câu hỏi này, yêu cầu chúng ta phải triển khai một Q-learning agent trong file qlearningAgents.py. Q-learning là một thuật toán học tăng cường không yêu cầu biết trước MDP (Markov Decision Process), mà thay vào đó học thông qua việc tương tác với môi trường. Agent sẽ sử dụng phương pháp cập nhật giá trị Q(s,a)dựa trên phần thưởng nhận được sau mỗi lần thực hiện hành động.

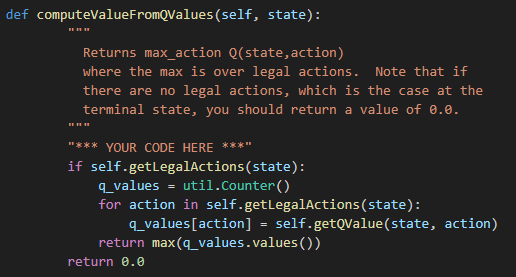
### **Các phương thức cần triển khai:**

1. **getQValue(state, action)**:
   * Trả về giá trị Q của cặp (trạng thái, hành động) dựa trên bảng Q-values. Nếu trạng thái và hành động này chưa được khám phá trước đó, giá trị mặc định là 0.



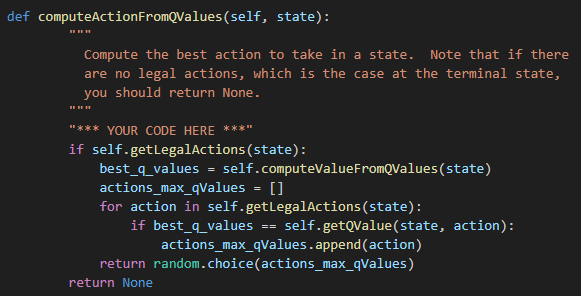
1. **getValue(state)**:
   * Trả về giá trị tối ưu của một trạng thái, tức là giá trị Q lớn nhất giữa các hành động khả thi từ trạng thái đó. Nếu trạng thái không có hành động hợp lệ (như trạng thái kết thúc), trả về 0.



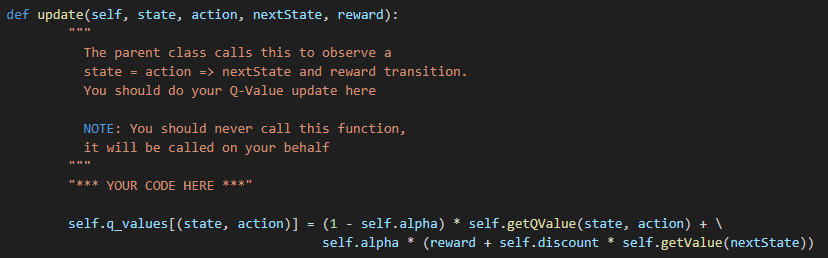
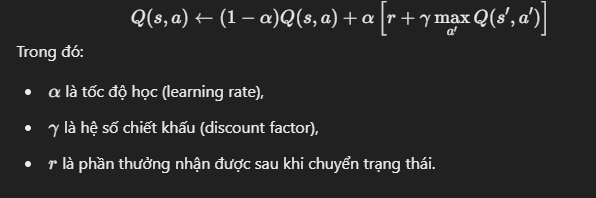


1. **getPolicy(state)**:
   * Trả về hành động tốt nhất từ các hành động khả thi, dựa trên giá trị Q lớn nhất. Nếu có nhiều hành động có cùng giá trị Q, chọn ngẫu nhiên giữa các hành động này để đảm bảo sự ngẫu nhiên trong chính sách.





1. **update(state, action, nextState, reward)**:
   * Cập nhật giá trị Q cho cặp (trạng thái, hành động) sau khi agent thực hiện hành động, chuyển sang trạng thái tiếp theo, và nhận được phần thưởng. Phương pháp cập nhật dựa trên công thức:

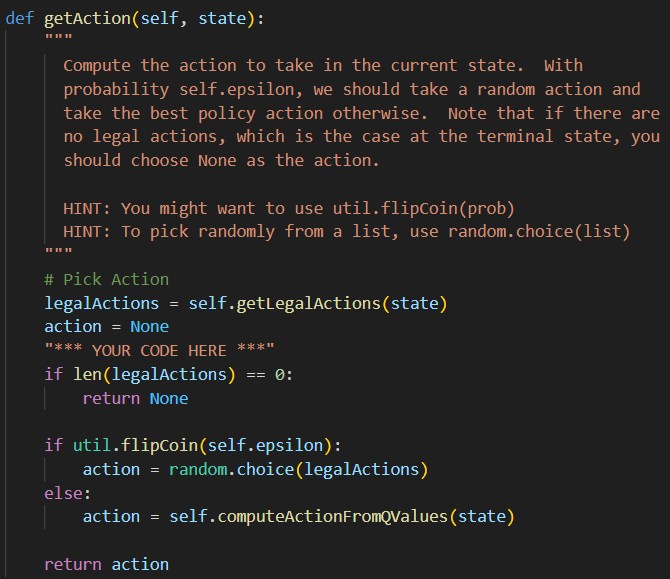


### 

### ***Question 5 (2 points)*** Complete your Q-learning agent by implementing epsilon-greedy action selection in getAction, meaning it chooses random actions an epsilon fraction of the time, and follows its current best Q-values otherwise

Đây là phần code của Q-learning agent bằng việc áp dụng Epsilon-Greedy action in getAction thì nó chọn các hành động ngẫu nhiên trong một phần epsilon thời gian và tuân theo các giá trị Q tốt nhất hiện tại của nó.

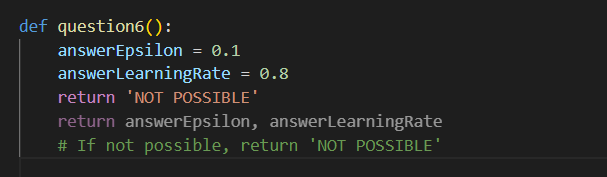
Đây là phần code của bài:



Trong đó ta có thể hiểu như sau:

* getAction(self, state) là phương thức quyết định hành động nào mà tác nhân (Pacman) nên thực hiện khi ở trạng thái cụ thể.
* self.getLegalActions(state) truy xuất danh sách các hành động hợp lệ (có sẵn) mà Pacman có thể thực hiện từ trạng thái hiện tại. Nếu không có hành động hợp lệ nào (ví dụ: ở trạng thái đầu cuối), nó sẽ trả về danh sách trống.
* action = None: Điều này khởi tạo biến hành động, biến này sẽ lưu trữ hành động mà Pacman cuối cùng sẽ thực hiện.
* Util.flipCoin(self.epsilon) mô phỏng việc tung đồng xu với xác suất epsilon. Nếu kết quả tung đồng xu là **True**, Pacman sẽ **khám phá** bằng cách chọn một hành động ngẫu nhiên.
* Self.\_epsilon là một tham số điều khiển **tỷ lệ khám phá**: giá trị epsilon càng cao, Pacman càng có khả năng chọn một hành động ngẫu nhiên.
* Random.choice(legalActions) chọn một hành động ngẫu nhiên từ danh sách các hành động hợp lệ.

### ***Question 6 (1 points)*** First, train a completely random Q-learner with the default learning rate on the noiseless BridgeGrid for 50 episodes and observe whether it finds the optimal policy.



answerEpsilon = 0.1(Giá trị epsilon):

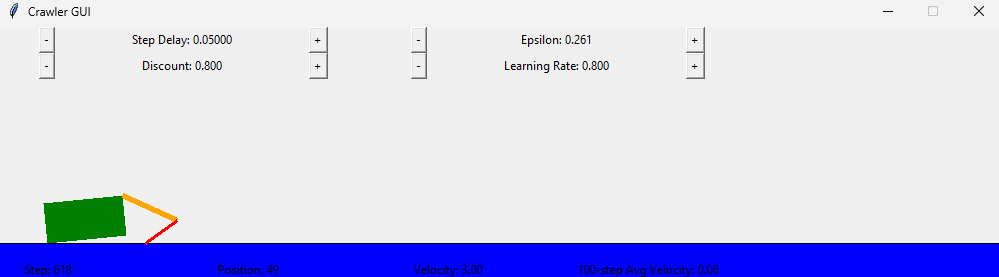
* Đây là giá trị của **epsilon**, tham số trong chiến lược **epsilon-greedy** được sử dụng trong Q-learning. Với epsilon-greedy, tác nhân sẽ chọn hành động ngẫu nhiên với xác suất **epsilon** và chọn hành động tốt nhất dựa trên giá trị Q đã biết với xác suất **1 - epsilon**.
* Ở đây, giá trị **0.1** có nghĩa là tác nhân sẽ chọn một hành động ngẫu nhiên với xác suất **10%** và khai thác hành động tốt nhất với xác suất **90%**.

answerLearningRate = 0.8(Tốc độ học):

* Giá trị này xác định mức độ mà giá trị Q mới sẽ ảnh hưởng đến giá trị Q hiện tại. Nếu **alpha** gần bằng 1, tác nhân sẽ ưu tiên giá trị phần thưởng hiện tại nhiều hơn. Nếu **alpha** nhỏ hơn, tác nhân sẽ ít thay đổi giá trị Q hiện tại.
* Với giá trị **0.8**, tác nhân sẽ cập nhật giá trị Q rất nhiều dựa trên phần thưởng nhận được từ hành động gần nhất.

Huấn luyện tác nhân Q-Learning với **chính sách ngẫu nhiên** trên **Bridge Grid không có nhiễu** trong 50 tập đã cho thấy rằng tác nhân có thể bắt đầu học cách phân biệt giữa các trạng thái tốt và xấu. Tuy nhiên, chính sách hiện tại **chưa tối ưu** do tính chất ngẫu nhiên của việc chọn hành động.=> tăng số tập lên hoặc dùng epsilon-greedy

### ***Question 7 (1 point)*** With no additional code, you should now be able to run a Q-learning crawler robot:



Dễ dàng nhận thấy có 4 tham số để thay đổi bao gồm  
- Step delay:

**Định nghĩa**: Đây là thời gian chờ giữa các bước của robot trong mô phỏng. Nó không liên quan trực tiếp đến quá trình học tập của robot, mà chủ yếu được sử dụng để làm chậm hoặc tăng tốc độ hiển thị của mô phỏng để người dùng có thể quan sát quá trình.

**Ảnh hưởng**:

* **Tăng Step Delay**: Khi độ trễ được tăng lên, mỗi bước của robot sẽ được thực hiện chậm hơn, giúp bạn có nhiều thời gian hơn để quan sát các hành động của robot.
* **Giảm Step Delay**: Nếu độ trễ giảm, các bước của robot sẽ được thực hiện nhanh hơn, tăng tốc độ mô phỏng nhưng có thể khó quan sát chi tiết.

- Epsilon:

**Định nghĩa**: epsilon là xác suất mà tác nhân sẽ thực hiện một hành động ngẫu nhiên thay vì hành động tốt nhất dựa trên chính sách hiện tại (chính sách được tạo ra từ các giá trị Q). Đây là một cơ chế để duy trì việc khám phá các trạng thái và hành động mới, tránh việc tác nhân chỉ tập trung vào các hành động mà nó đã biết là tốt nhất (khai thác - exploitation).

**Ảnh hưởng**:

* **Tăng Epsilon**: Nếu bạn tăng giá trị epsilon, tác nhân sẽ thực hiện nhiều hành động ngẫu nhiên hơn, giúp khám phá môi trường tốt hơn nhưng có thể dẫn đến việc thực hiện các hành động không tối ưu trong thời gian ngắn.
* **Giảm Epsilon**: Nếu bạn giảm epsilon, tác nhân sẽ ít khám phá hơn và chủ yếu dựa vào các hành động đã học, có thể nhanh chóng dẫn đến việc tập trung vào các hành động tốt nhất nhưng không khám phá đủ môi trường, có nguy cơ bỏ lỡ những chiến lược tốt hơn.

**Tối ưu hóa**: Ban đầu, epsilon thường được đặt cao hơn để thúc đẩy việc khám phá môi trường và dần giảm theo thời gian (epsilon-greedy decay), cho phép tác nhân học chính sách tối ưu sau khi đã khám phá đủ các trạng thái và hành động.

### ***Question 8 (1 points)*** Time to play some Pacman! Pacman will play games in two phases. In the first phase, *training*, Pacman will begin to learn about the values of positions and actions. Because it takes a very long time to learn accurate Q-values even for tiny grids, Pacman's training games run in quiet mode by default, with no GUI (or console) display. Once Pacman's training is complete, he will enter *testing* mode. When testing, Pacman's self.epsilon and self.alpha will be set to 0.0, effectively stopping Q-learning and disabling exploration, in order to allow Pacman to exploit his learned policy. Test games are shown in the GUI by default. Without any code changes you should be able to run Q-learning Pacman for very tiny grids as follows:

### **Tổng quan:**

Câu hỏi này yêu cầu chạy Pacman sử dụng **PacmanQAgent**, một agent dựa trên Q-learning. Pacman sẽ được huấn luyện qua hai giai đoạn: **huấn luyện (training)** và **kiểm tra (testing)**. Trong giai đoạn huấn luyện, Pacman sẽ học về giá trị của các vị trí và hành động, trong khi giai đoạn kiểm tra, Pacman sẽ khai thác chính sách đã học để chơi game.

### **Các yêu cầu chính:**

1. **Huấn luyện Pacman trong môi trường smallGrid**:
   * Bạn sẽ huấn luyện Pacman trên một lưới nhỏ với 2000 game để học các giá trị Q và chính sách tối ưu. Sau khi kết thúc quá trình huấn luyện, Pacman sẽ kiểm tra chính sách đã học trong 10 game.

python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid

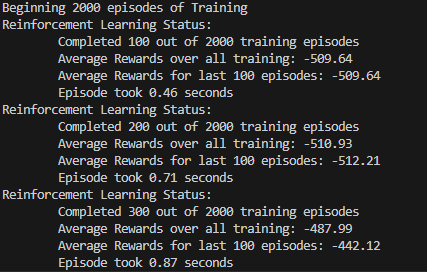
Trong đó:

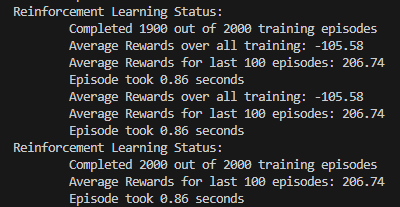
* + **-p PacmanQAgent**: Chạy Pacman với agent đã triển khai Q-learning.
  + **-x 2000**: Chạy 2000 game huấn luyện mà không hiển thị GUI.
  + **-n 2010**: Tổng cộng 2010 game, trong đó 10 game cuối sẽ được hiển thị để kiểm tra (testing).
  + **-l smallGrid**: Sử dụng lưới nhỏ cho Pacman.

1. **Điều chỉnh epsilon và alpha**:
   * Trong giai đoạn huấn luyện, **epsilon** sẽ được đặt ở giá trị mặc định (0.05), giúp Pacman vừa khám phá các hành động mới nhưng vẫn tập trung khai thác hành động đã học.
   * **Alpha** (learning rate) cũng sẽ được đặt ở giá trị mặc định (0.2), giúp Pacman cập nhật giá trị Q-value với tốc độ vừa phải.
2. **Quá trình kiểm tra**:
   * Khi Pacman hoàn thành 2000 game huấn luyện, giá trị **epsilon** và **alpha** sẽ được đặt về 0.0 để dừng việc học và cho phép Pacman khai thác chính sách tốt nhất mà không khám phá nữa.
   * Pacman cần phải thắng ít nhất **80%** số game để được tính điểm đầy đủ cho câu hỏi này.

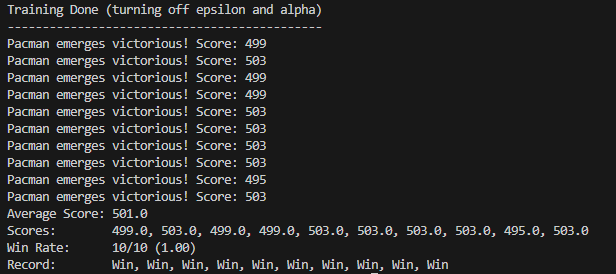
### **Mô tả quá trình huấn luyện và kiểm tra:**

* **Huấn luyện (training phase)**:
  + Trong giai đoạn huấn luyện, Pacman sẽ học từ các hành động của mình và cập nhật bảng Q-values. Các giá trị Q sẽ dần phản ánh giá trị của các vị trí và hành động trong game.
  + Trong quá trình này, **epsilon** sẽ khác 0, nghĩa là Pacman thỉnh thoảng sẽ chọn các hành động ngẫu nhiên để khám phá môi trường và tìm kiếm những hành động có thể mang lại phần thưởng cao hơn.





* **Kiểm tra (testing phase)**:
  + Sau 2000 game huấn luyện, **epsilon** sẽ được đặt về 0, nghĩa là Pacman sẽ chỉ dựa vào các giá trị Q đã học để đưa ra hành động tối ưu. Pacman không còn khám phá các hành động mới mà chỉ khai thác chính sách hiện có để cố gắng thắng game.
  + Kết quả của Pacman trong giai đoạn này sẽ cho thấy khả năng học của agent.

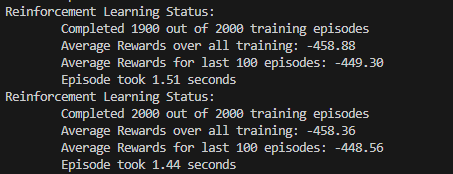


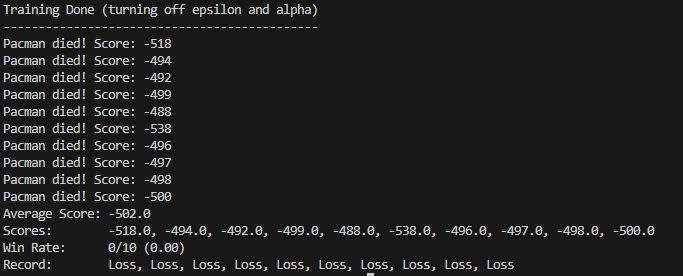
### **Khó khăn với lưới lớn hơn:**

Một vấn đề nổi lên khi ta huấn luyện Pacman trên lưới lớn hơn như **mediumGrid**. Mặc dù có vẻ đơn giản, nhưng các cấu hình bảng phức tạp hơn khiến Pacman gặp khó khăn trong việc học chính sách tối ưu. Nguyên nhân chính bao gồm:

* Mỗi cấu hình bảng riêng lẻ là một trạng thái khác nhau, dẫn đến Pacman cần nhiều thời gian để học các giá trị Q cho tất cả các trạng thái.
* Pacman không thể tổng quát hóa việc né tránh ghost trên các trạng thái khác nhau, khiến việc học chính sách trở nên kém hiệu quả trên lưới lớn.

python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l mediumGrid



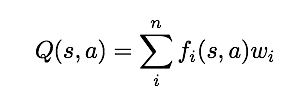


### ***Question 9 (3 points)*** Implement an approximate Q-learning agent that learns weights for features of states, where many states might share the same features. Write your implementation in ApproximateQAgent class in [qlearningAgents.py](https://classes.engineering.wustl.edu/cse511/p3/docs/qlearningAgents.php), which is a subclass of PacmanQAgent.

Trong câu hỏi này, đề yêu triển khai một Approximate Q-learning Agent để học các trọng số cho các đặc trưng (features) của trạng thái và hành động. Các đặc trưng này giúp Pacman có thể tổng quát hóa giữa nhiều trạng thái khác nhau nhưng có chung đặc trưng, giúp agent có thể học nhanh hơn và hiệu quả hơn.

**Công thức Q-learning gần đúng (Approximate Q-learning):**

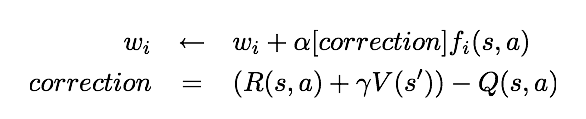
Approximate Q-learning sử dụng công thức sau để tính toán giá trị Q cho mỗi cặp (state, action):



* fi(s,a) là giá trị của đặc trưng thứ iii cho cặp (state, action).
* 𝑤𝑖​ là trọng số tương ứng với đặc trưng đó.
* Tổng tất cả các đặc trưng và trọng số tạo thành giá trị Q(s,a), Q(s, a), Q(s,a).

**Cập nhật trọng số:**

Các trọng số 𝑤𝑖 sẽ được cập nhật theo công thức:



Trong đó:

* 𝛼 là learning rate.
* 𝛾 là discount factor.
* 𝑅(𝑠,𝑎)là phần thưởng khi thực hiện hành động 𝑎 tại trạng thái 𝑠.
* 𝑉(𝑠′) là giá trị Q tối ưu cho trạng thái tiếp theo 𝑠′.
* 𝑄(𝑠,𝑎) là giá trị Q hiện tại của cặp (state, action).

### **Cách triển khai trong lớp ApproximateQAgent:**

1. **Phương thức getQValue(state, action)**:
   * Tính toán giá trị Q cho cặp (state, action) bằng cách nhân các trọng số với các đặc trưng (features) tương ứng và tổng các giá trị đó lại.
2. **Phương thức update(state, action, nextState, reward)**:
   * Tính toán giá trị **correction** cho quá trình cập nhật, tức là sự chênh lệch giữa phần thưởng dự kiến và giá trị hiện tại Q(s,a).
   * Cập nhật các trọng số 𝑤𝑖​ tương ứng với từng đặc trưng bằng công thức trên.

**Triển khai code:**



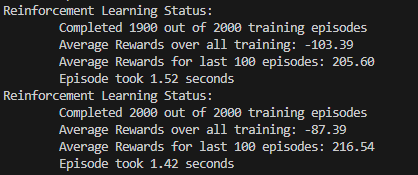
### **Mô tả từng phần của code:**

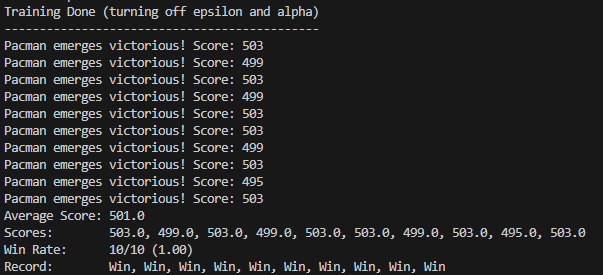
1. **getQValue(state, action)**:
   * Phương thức này sử dụng các đặc trưng của cặp (state, action) và nhân các trọng số tương ứng để tính giá trị Q. Kết quả là tổng của tất cả các đặc trưng nhân với trọng số của chúng.
2. **update(state, action, nextState, reward)**:
   * Tính toán sự khác biệt giữa phần thưởng dự kiến (dựa trên trạng thái tiếp theo và phần thưởng tức thời) và giá trị Q hiện tại của cặp (state, action). Sự khác biệt này gọi là **correction**.
   * Cập nhật từng trọng số 𝑤𝑖 dựa trên đặc trưng fi(s,a) và giá trị **correction** vừa tính được.

### **Kiểm tra Approximate Q-learning Agent:**

1. **Kiểm tra với IdentityExtractor**:
   * Chạy thử lệnh sau để kiểm tra agent với extractor mặc định (IdentityExtractor):

python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid





Agent này sẽ hoạt động giống hệt với PacmanQAgent, nhưng với khả năng tổng quát hóa khi các đặc trưng được trích xuất.

1. **Kiểm tra với SimpleExtractor**:
   * Khi mọi thứ hoạt động đúng, bạn có thể thử nghiệm với **SimpleExtractor**:

python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumGrid

**SimpleExtractor** sẽ giúp agent tổng quát hóa tốt hơn giữa các trạng thái khác nhau và học nhanh hơn.