

eman ta zabal zazu



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

Técnicas de Inteligencia Artificial

Ingeniería Informática de Gestión y Sistemas de Información

Practica 4

Reinforcement Learning

Autor(es):

Xabier Gabiña

Diego Montoya

17 de diciembre de 2024

Índice general

1. Introducción	4
2. Ejercicios	5
2.1. Iteración de valores	5
2.2. Análisis de cruce de puentes	7
2.3. Q-Learning	9
2.4. Epsilon Greedy	11
2.5. Q-Learning y Pacman	13
2.6. Q-Learning aproximado	14
3. Resultados	16
3.1. Ejecución	16
3.1.1. Iteración de valores	16
3.1.2. Análisis de cruce de puentes	17
3.1.3. Q-Learning	18
3.1.4. Epsilon Greedy	19
3.1.5. Q-Learning y Pacman	20
3.1.6. Q-Learning aproximado	22
3.2. Autograder	25

Índice de figuras

3.1. Iteración de valores	16
3.2. Iteración de valores	16
3.3. Análisis de cruce de puentes	17
3.4. Q-Learning	18
3.5. Epsilon Greedy	19
3.6. Epsilon Greedy	19

Índice de Códigos

2.1. Iteración de valores	5
2.2. Análisis de cruce de puentes	7
2.3. Q-Learning	9
2.4. Epsilon Greedy	11
2.5. Q-Learning y Pacman	14
3.1. Ejecución de Q-Learning y Pacman 1	20
3.2. Ejecución de Q-Learning y Pacman 2	21
3.3. Q-Learning aproximado	22
3.4. Q-Learning aproximado	23
3.5. Q-Learning aproximado	24
3.6. Autograder	25

1. Introducción

En este proyecto de la asignatura de Técnicas de inteligencia artificial implementaremos iteración de valor y Q-learning; primero realizaremos un agente off-line de iteración de valores, pe lo que buscamos es implementar un agente inteligente basado en el modelo de Q-learning en el cual implementaremos el aprendizaje por refuerzo para nuestro Pacman. El objetivo es lograr que nuestro agente sea capaz de tomar decisiones mediante la obtención de recompensas basadas en las acciones que elija. Para lograr esto la practica nos llevará por 6 aspectos clave del aprendizaje por refuerzo.

- **Implementación básica de Q-learning:** Se comenzará con la creación de un agente que utiliza el algoritmo de Q-learning para aprender de sus interacciones con el entorno. Este agente actualizará sus valores de acción (Q-values) en función de las recompensas recibidas y las transiciones entre estados.
- **Selección de acciones con Epsilon-Greedy:** Para mejorar la exploración del entorno y evitar quedar atrapado en una política subóptima, se implementará una estrategia epsilon-greedy, donde el agente explorará acciones aleatorias en una fracción del tiempo.
- **Aplicación de Q-learning en Gridworld:** Inicialmente, el agente se entrenará en un entorno sencillo, conocido como Gridworld, para aprender a maximizar sus recompensas. A medida que el agente interactúa con el entorno, ajustará su política de manera iterativa.
- **Mejora de la política para Pacman:** Una vez que el agente ha aprendido en entornos más sencillos, se aplicará el algoritmo de Q-learning en el juego de Pacman, primero en un entorno pequeño (smallGrid), luego en entornos más complejos (mediumGrid). Durante esta fase, se evaluará la capacidad del agente para aprender y explotar su política en un entorno más desafiante.

Al finalizar esta practica tendremos un agente Pacman con la capacidad de aprender y así aplicar una estrategia óptima para maximizar su rendimiento en diferentes mapas.

2. Ejercicios

2.1. Iteración de valores

Descripción

El primer ejercicio se trata de implementar un agente que realice la iteración de valores. Un agente de Iteración de Valores toma un proceso de decisión de Markov en la inicialización y ejecuta la iteración de valores durante un número dado de iteraciones utilizando el factor de descuento proporcionado.

Implementación

```
1 class ValueIterationAgent(ValueEstimationAgent):
2     """
3         * Please read learningAgents.py before reading this.*
4
5         A ValueIterationAgent takes a Markov decision process
6         (see mdp.py) on initialization and runs value iteration
7         for a given number of iterations using the supplied
8         discount factor.
9     """
10    def __init__(self, mdp, discount = 0.9, iterations = 100):
11        """
12        Your value iteration agent should take an mdp on
13        construction, run the indicated number of iterations
14        and then act according to the resulting policy.
15
16        Some useful mdp methods you will use:
17            mdp.getStates()
18            mdp.getPossibleActions(state)
19            mdp.getTransitionStatesAndProbs(state, action)
20            mdp.getReward(state, action, nextState)
21            mdp.isTerminal(state)
22        """
23        self.mdp = mdp
24        self.discount = discount
25        self.iterations = iterations
26        self.values = util.Counter() # A Counter is a dict with default 0
27        self.runValueIteration()
28
29    def runValueIteration(self):
30        # Write value iteration code here
31        for _ in range(self.iterations):
32            new_values = self.values.copy()
33            for state in self.mdp.getStates():
34                if not self.mdp.isTerminal(state):
35                    new_values[state] = max(self.computeQValueFromValues(state, action) for action
36                    in self.mdp.getPossibleActions(state))
37            self.values = new_values
38
39    def getValue(self, state):
40        """
41        Return the value of the state (computed in __init__).
42        """
43        return self.values[state]
44
45
46    def computeQValueFromValues(self, state, action):
47        """
48        computeQValueFromValues(state, action) devuelve el valor Q del par (estado, acción)
49        dado por la función de valor dada por self.values. Nota: Recuerda que para calcular
50        el valor de un estado calcularemos los q_values (estado, acción), es decir los valores
51        de las acciones posibles para quedarnos con el mayor (max de entre los q_values).
52        """
53        q_value = 0
54        for next_state, prob in self.mdp.getTransitionStatesAndProbs(state, action):
```

```

55         q_value += prob * (self.mdp.getReward(state, action, next_state) + self.discount *
self.values[next_state])
56         return q_value
57
58
59     def computeActionFromValues(self, state):
60         """
61         computeActionFromValues(state) calcula la mejor acción de acuerdo con la función de
62         valor dada por self.values. Es decir, de acuerdo al estado siguiente al que nos lleva
63         cada acción y que mayor valor tiene. Este método llamará al siguiente.
64         """
65         if self.mdp.isTerminal(state):
66             return None
67
68         return max(self.mdp.getPossibleActions(state), key=lambda x: self.computeQValueFromValues(
state, x))
69
70
71     def getPolicy(self, state):
72         return self.computeActionFromValues(state)
73
74     def getAction(self, state):
75         "Returns the policy at the state (no exploration)."
```

```

76         return self.computeActionFromValues(state)
77
78     def getQValue(self, state, action):
79         return self.computeQValueFromValues(state, action)

```

Código 2.1: Iteración de valores

Comentarios

Implementamos el algoritmo de iteración de valores con los cuales fuimos capaces de calcular los óptimos de los estados y las políticas correspondientes de un MDP, gracias a esto nuestro agente es capaz de tomar decisiones basadas en la maximización de recompensas. Gracias a la validación mediante pruebas y simulaciones se asegura que el agente pueda planificar correctamente en diversos entornos, teniendo así una gran utilidad.

2.2. Análisis de cruce de puentes

Descripción

En este apartado analizaremos el comportamiento de un agente en el BridgeWorld, el cual es un GridWorld que presenta un escenario de decisiones críticas debido a un puente que separa nuestro agente; por un lado se encuentra un estado con recompensa alta y un estado con recompensa baja (caer al abismo). El escenario se ha configurado de tal manera que hay un factor de descuento gamma (γ) de 0.9 y un nivel de ruido establecido a 0.2. Lo que vamos a analizar es cómo el cambiar estos factores altera el comportamiento de nuestro agente y, así mismo, descifrar cómo es que cambiando ligeramente uno de estos dos valores logramos hacer una política que incentive a nuestro agente a cruzar el puente.

Factor de Descuento (γ) El factor de descuento determina que tanto el agente valorará las recompensas futuras en comparación con las recompensas inmediatas.

Si γ sube (se acerca a 1): El agente valora más las recompensas futuras, esto significa que será mas propenso a tomar riesgos inmediatos si es que estos lo llevan a obtener una mayor recompensa a largo plazo. De esta manera al incrementar este valor se puede incentivar al agente a cruzar el puente ya que se enfocara más en la recompensa alta del otro lado, sin embargo en el ejemplo presentado este factor ya se encuentra en 0.9, por lo que tendremos que optar por otra estrategia.

Si γ baja (se acerca a 0): El agente prioriza las recompensas inmediatas y descuida las recompensas futuras. Por lo que al disminuir este valor el agente será más propenso a quedarse en el estado de baja recompensa porque cruzar el puente representa un riesgo y las recompensas futuras tienen menor relevancia.

Nivel de Ruido El nivel de ruido introduce incertidumbre en las decisiones que toma el agente para moverse. Gracias a esto podemos acercarnos a la realidad en la cual no siempre podremos movernos en la dirección que queramos por diversas razones, este factor simula situaciones de este tipo en el sistema.

Si el ruido sube (más cercano a 1): Aumenta la probabilidad de terminar en estados inesperados, lo que introduce más incertidumbre en las decisiones. Por lo que un mayor nivel de ruido hace que el querer cruzar el puente sean menos atractivo ya que los riesgo de estados de recompensa negativa como lo es el caer en el abismo aumentan.

Si el ruido baja (más cercano a 0): Disminuye la probabilidad de terminar en estados no deseados, haciendo que las acciones sean más confiables. Reduciendo el ruido nuestro agente verá cruzar el puente más atractivo ya que puede confiar en que la acción que elija lo llevara al estado deseado sin tantos desvíos.

Implementación

```
1 def question2():
2     answerDiscount = 0.9
3     answerNoise = 0.01
4     return answerDiscount, answerNoise
5
6 if __name__ == '__main__':
7     print('Answers to analysis questions:')
8     import analysis
9     for q in [q for q in dir(analysis) if q.startswith('question')]:
10         response = getattr(analysis, q)()
11         print(' Question %s:\t%s' % (q, str(response)))
```

Código 2.2: Análisis de cruce de puentes

Comentarios

La solución más sencilla para este ejercicio es la de bajar el ruido por debajo de 0.01. Esto hace que las probabilidades de que el agente cometa un error y se caiga a un acantilado de puntos negativos sean muy bajas. No haría falta tocar el valor γ ya que al tener un factor alto, lo que conseguimos, es que las recompensas futuras tengan un peso mayor lo que conviene para no acabar llegando a la casilla +1 en vez de a la +10.

2.3. Q-Learning

Descripción

En este ejercicio se trata de implementar un agente que realice el aprendizaje por refuerzo mediante Q-Learning. Q-Learning es un algoritmo de aprendizaje por refuerzo que aprende una política óptima sin conocer el modelo del entorno. Se basa en ir probando acciones, en un principio de forma aleatoria, y actualizando los valores de la función Q en función de las recompensas recibidas. A medida que se va actualizando la función Q, el agente va aprendiendo la mejor acción a tomar en cada estado y va disminuyendo el número de acciones aleatorias.

Implementación

```
1 class QLearningAgent(ReinforcementAgent):
2     """
3     Q-Learning Agent
4
5     Functions you should fill in:
6     - computeValueFromQValues
7     - computeActionFromQValues
8     - getQValue
9     - getAction
10    - update
11
12    Instance variables you have access to
13    - self.epsilon (exploration prob)
14    - self.alpha (learning rate)
15    - self.discount (discount rate)
16
17    Functions you should use
18    - self.getLegalActions(state)
19      which returns legal actions for a state
20    """
21    def __init__(self, **args):
22        "You can initialize Q-values here..."
23        ReinforcementAgent.__init__(self, **args)
24        self.qValues = util.Counter()
25
26    def getQValue(self, state, action):
27        """
28        Returns Q(state,action)
29        Should return 0.0 if we have never seen a state
30        or the Q node value otherwise
31        """
32        return self.qValues[(state, action)] if (state, action) in self.qValues else 0.0
33
34
35    def computeValueFromQValues(self, state):
36        """
37        Returns max_action Q(state,action)
38        where the max is over legal actions. Note that if
39        there are no legal actions, which is the case at the
40        terminal state, you should return a value of 0.0.
41        """
42        legalActions = self.getLegalActions(state)
43        if not legalActions:
44            return 0.0
45        return max(self.getQValue(state, action) for action in legalActions)
46
47    def computeActionFromQValues(self, state):
48        """
49        Compute the best action to take in a state. Note that if there
50        are no legal actions, which is the case at the terminal state,
51        you should return None.
52        """
53        legalActions = self.getLegalActions(state)
54        if not legalActions:
55            return None
56
```

```

57     best_value = float('-inf')
58     best_actions = []
59
60     for action in legalActions:
61         q_value = self.getQValue(state, action)
62         if q_value > best_value:
63             best_value = q_value
64             best_actions = [action]
65         elif q_value == best_value:
66             best_actions.append(action)
67
68     return random.choice(best_actions)
69
70 def getAction(self, state):
71     """
72     Compute the action to take in the current state. With
73     probability self.epsilon, we should take a random action and
74     take the best policy action otherwise. Note that if there are
75     no legal actions, which is the case at the terminal state, you
76     should choose None as the action.
77
78     HINT: You might want to use util.flipCoin(prob)
79     HINT: To pick randomly from a list, use random.choice(list)
80     """
81     return self.computeActionFromQValues(state)
82
83 def update(self, state, action, nextState, reward):
84     """
85     The parent class calls this to observe a
86     state = action => nextState and reward transition.
87     You should do your Q-Value update here
88
89     NOTE: You should never call this function,
90     it will be called on your behalf
91     """
92     sample = reward + self.discount * self.computeValueFromQValues(nextState)
93     self.qValues[(state, action)] = (1 - self.alpha) * self.getQValue(state, action) + self.
alpha * sample
94
95 def getPolicy(self, state):
96     return self.computeActionFromQValues(state)
97
98 def getValue(self, state):
99     return self.computeValueFromQValues(state)

```

Código 2.3: Q-Learning

Comentarios

Gracias a la implementación de Q-Learning dentro de nuestro agente es que ahora podemos observar como es que el agente aprende a tomar decisiones óptimas mediante la prueba y error, sin necesidad de conocer previamente el entorno en el que se encuentra. Esto es útil para escenario en los cuales no conocemos las recompensas de antemano.

2.4. Epsilon Greedy

Descripción

En este ejercicio se nos pide modificar la función `getAction` de la clase `QLearningAgent` para que implemente la estrategia *epsilon-greedy*. Es decir, que se elijan acciones de forma aleatorio en base a un factor ϵ y se elija la mejor acción en base a la función `Q` en el resto de los casos.

Implementación

```
1 class QLearningAgent(ReinforcementAgent):
2     """
3     Q-Learning Agent
4
5     Functions you should fill in:
6     - computeValueFromQValues
7     - computeActionFromQValues
8     - getQValue
9     - getAction
10    - update
11
12    Instance variables you have access to
13    - self.epsilon (exploration prob)
14    - self.alpha (learning rate)
15    - self.discount (discount rate)
16
17    Functions you should use
18    - self.getLegalActions(state)
19      which returns legal actions for a state
20    """
21    def __init__(self, **args):
22        "You can initialize Q-values here..."
23        ReinforcementAgent.__init__(self, **args)
24        self.qValues = util.Counter()
25
26    def getQValue(self, state, action):
27        """
28        Returns Q(state,action)
29        Should return 0.0 if we have never seen a state
30        or the Q node value otherwise
31        """
32        return self.qValues[(state, action)] if (state, action) in self.qValues else 0.0
33
34
35    def computeValueFromQValues(self, state):
36        """
37        Returns max_action Q(state,action)
38        where the max is over legal actions. Note that if
39        there are no legal actions, which is the case at the
40        terminal state, you should return a value of 0.0.
41        """
42        legalActions = self.getLegalActions(state)
43        if not legalActions:
44            return 0.0
45        return max(self.getQValue(state, action) for action in legalActions)
46
47    def computeActionFromQValues(self, state):
48        """
49        Compute the best action to take in a state. Note that if there
50        are no legal actions, which is the case at the terminal state,
51        you should return None.
52        """
53        legalActions = self.getLegalActions(state)
54        if not legalActions:
55            return None
56
57        best_value = float('-inf')
58        best_actions = []
```

```

59
60     for action in legalActions:
61         q_value = self.getQValue(state, action)
62         if q_value > best_value:
63             best_value = q_value
64             best_actions = [action]
65         elif q_value == best_value:
66             best_actions.append(action)
67
68     return random.choice(best_actions)
69
70 def getAction(self, state):
71     """
72     Compute the action to take in the current state. With
73     probability self.epsilon, we should take a random action and
74     take the best policy action otherwise. Note that if there are
75     no legal actions, which is the case at the terminal state, you
76     should choose None as the action.
77
78     HINT: You might want to use util.flipCoin(prob)
79     HINT: To pick randomly from a list, use random.choice(list)
80     """
81     # Pick Action
82     legalActions = self.getLegalActions(state)
83     action = None
84
85     if util.flipCoin(self.epsilon):
86         action = random.choice(legalActions)
87     else:
88         action = self.computeActionFromQValues(state)
89
90     return action
91
92 def update(self, state, action, nextState, reward):
93     """
94     The parent class calls this to observe a
95     state = action => nextState and reward transition.
96     You should do your Q-Value update here
97
98     NOTE: You should never call this function,
99     it will be called on your behalf
100    """
101    sample = reward + self.discount * self.computeValueFromQValues(nextState)
102    self.qValues[(state, action)] = (1 - self.alpha) * self.getQValue(state, action) + self.alpha * sample
103
104 def getPolicy(self, state):
105     return self.computeActionFromQValues(state)
106
107 def getValue(self, state):
108     return self.computeValueFromQValues(state)

```

Código 2.4: Epsilon Greedy

Comentarios

Como hemos comentado ya en la descripción del ejercicio simplemente hemos modificado la implementación del temtodo `getAction` para que dependiendo del valor de epsilon se utilice una accion aleatoria entre las disponibles o se utiliza la mejor accion segun la funcion Q.

2.5. Q-Learning y Pacman

Comentarios

Este ejercicio consiste en ver como a medida que aumentamos el tamaño del mapa Pacman juega cada vez peor y va más lento. Esto es debido a la poca eficiencia del Q-Learning y el gran número de pruebas que se tendrían que hacer para que el agente aprenda. Para solucionar esto se propone el uso de Q-Learning aproximado en el siguiente ejercicio

2.6. Q-Learning aproximado

Descripción

En juegos como el Pacman existen demasiados estados posibles por lo que el Q-Learning convencional no es viable. Para ello se usa el Q-Learning aproximado el cual en vez de una tabla estado-acción reemplaza esta por una función de características o features.

$$Q(s, a) = w \cdot f(s, a)$$

Donde w es un vector de pesos y $f(s, a)$ es un vector de características. Entonces lo que hacemos es en lugar de aprender valores específicos para cada estado, aprendemos pesos para características importantes como la distancia al fantasma más cercano, distancia a la comida...

Implementación

```
1 class ApproximateQAgent(PacmanQAgent):
2     """
3     ApproximateQLearningAgent
4
5     You should only have to overwrite getQValue
6     and update. All other QLearningAgent functions
7     should work as is.
8     """
9     def __init__(self, extractor='IdentityExtractor', **args):
10         self.feateXtractor = util.lookup(extractor, globals())()
11         PacmanQAgent.__init__(self, **args)
12         self.weights = util.Counter()
13
14     def getWeights(self):
15         return self.weights
16
17     def getQValue(self, state, action):
18         """
19         Calcula Q(state,action) = w * featureVector
20         donde * es el operador de producto punto
21         """
22         features = self.feateXtractor.getFeatures(state, action)
23         qValue = 0
24         for feature in features:
25             qValue += features[feature] * self.weights[feature]
26         return qValue
27
28     def update(self, state, action, nextState, reward):
29         """
30         Actualiza los pesos basándose en la transición
31         """
32         # Obtener las características del estado actual y acción
33         features = self.feateXtractor.getFeatures(state, action)
34
35         # Calcular el valor máximo de Q para el siguiente estado
36         nextQValue = 0
37         if nextState: # si no es estado terminal
38             legalActions = self.getLegalActions(nextState)
39             if legalActions: # si hay acciones legales disponibles
40                 nextQValue = max([self.getQValue(nextState, nextAction)
41                                   for nextAction in legalActions])
42
43         # Calcular el error TD
44         difference = (reward + self.discount * nextQValue) - self.getQValue(state, action)
45
46         # Actualizar cada peso
47         for feature in features:
48             self.weights[feature] += self.alpha * difference * features[feature]
49
50     def final(self, state):
51         """
52         Llamado al final de cada juego
53         """
```

```

54     PacmanQAgent.final(self, state)
55
56     if self.episodesSoFar == self.numTraining:
57         print("Pesos finales:", self.weights)

```

Código 2.5: Q-Learning y Pacman

Comentarios

Lo que hemos hecho para implementar Q-Learning aproximado es:

- **getQValue:** Obtenemos el vector de características, calculamos el producto punto entre las características y los pesos.
- **update:** Calculamos el error TD y actualizamos los pesos. Aquí tenemos que tener en cuenta los estados terminales en los que no existen acciones disponibles ya que el valor Q debe basarse en recompensa inmediata y no en las futuras dado que no existen.

Lo más importante de esto es que ahora generalizamos entre estados que es lo que hace que sea mucho más eficiente.

3. Resultados

3.1. Ejecución

3.1.1. Iteración de valores

```
python gridworld.py -a value -i 100 -k 10
```



Figura 3.1: Iteración de valores

```
python gridworld.py -a value -i 5
```

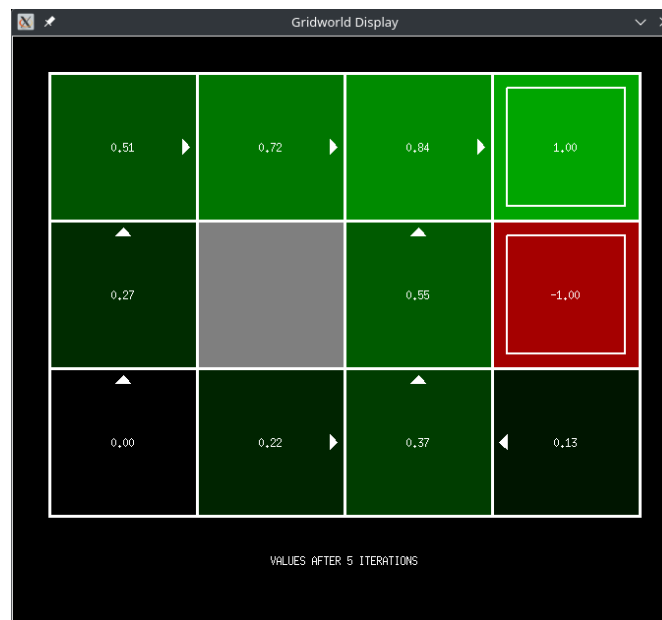


Figura 3.2: Iteración de valores

3.1.2. Análisis de cruce de puentes

python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0.2

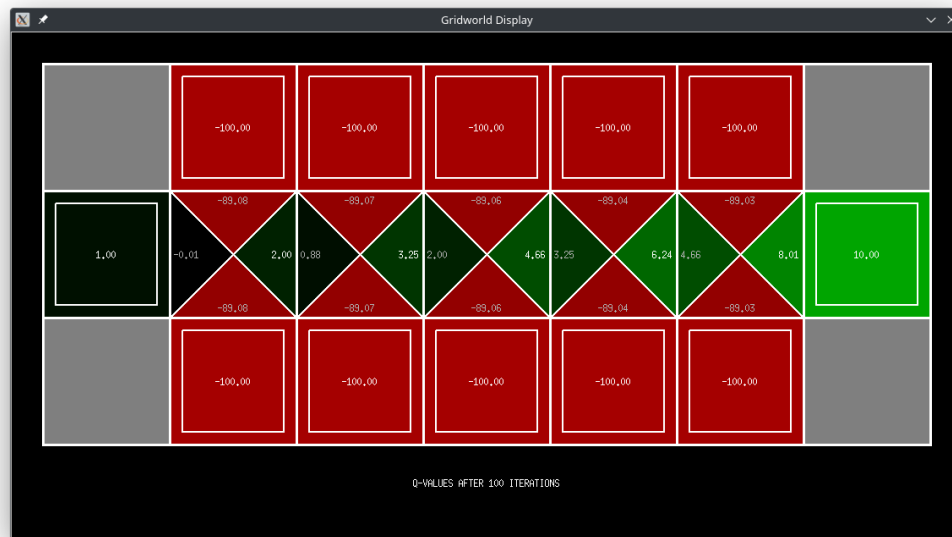


Figura 3.3: Análisis de cruce de puentes

3.1.3. Q-Learning

python gridworld.py -aq -k 5 -m

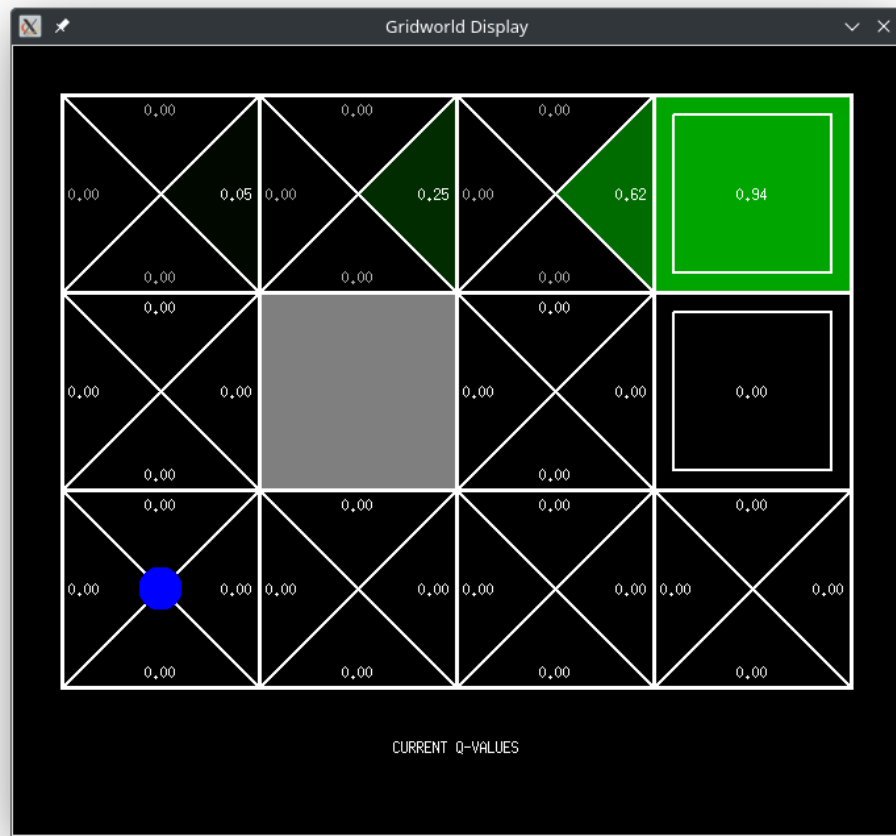


Figura 3.4: Q-Learning

3.1.4. Epsilon Greedy

`python gridworld.py -aq -k 100 - ruido 0.0 -e 0.1`



Figura 3.5: Epsilon Greedy

`python gridworld.py -aq -k 100 - ruido 0.0 -e 0.9`

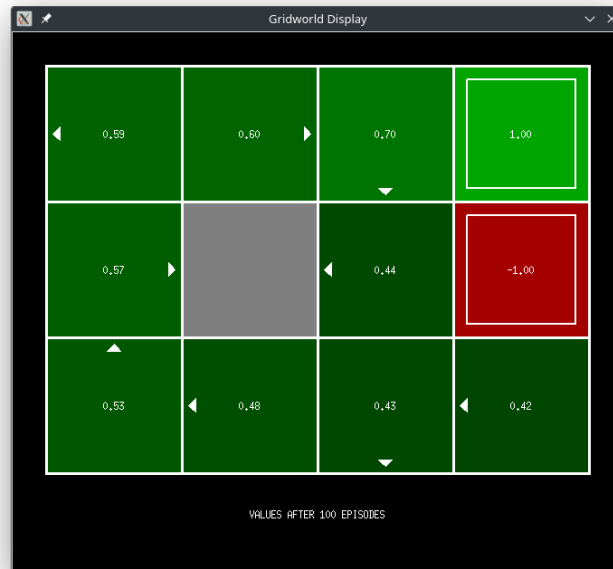


Figura 3.6: Epsilon Greedy

3.1.5. Q-Learning y Pacman

```
1 python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
2 Beginning 2000 episodes of Training
3 Reinforcement Learning Status:
4     Completed 100 out of 2000 training episodes
5     Average Rewards over all training: -510.01
6     Average Rewards for last 100 episodes: -510.01
7     Episode took 0.32 seconds
8 Reinforcement Learning Status:
9     Completed 200 out of 2000 training episodes
10    Average Rewards over all training: -510.93
11    Average Rewards for last 100 episodes: -511.84
12    Episode took 0.52 seconds
13 Reinforcement Learning Status:
14    Completed 300 out of 2000 training episodes
15    Average Rewards over all training: -491.66
16    Average Rewards for last 100 episodes: -453.12
17    Episode took 0.65 seconds
18 Reinforcement Learning Status:
19    Completed 400 out of 2000 training episodes
20    Average Rewards over all training: -444.42
21    Average Rewards for last 100 episodes: -302.71
22    Episode took 0.69 seconds
23 Reinforcement Learning Status:
24    Completed 500 out of 2000 training episodes
25    Average Rewards over all training: -405.61
26    Average Rewards for last 100 episodes: -250.38
27    Episode took 0.71 seconds
28 Reinforcement Learning Status:
29    Completed 600 out of 2000 training episodes
30    Average Rewards over all training: -381.12
31    Average Rewards for last 100 episodes: -258.65
32    Episode took 0.68 seconds
33 Reinforcement Learning Status:
34    Completed 700 out of 2000 training episodes
35    Average Rewards over all training: -373.79
36    Average Rewards for last 100 episodes: -329.83
37    Episode took 0.67 seconds
38 Reinforcement Learning Status:
39    Completed 800 out of 2000 training episodes
40    Average Rewards over all training: -354.59
41    Average Rewards for last 100 episodes: -220.17
42    Episode took 0.71 seconds
43 Reinforcement Learning Status:
44    Completed 900 out of 2000 training episodes
45    Average Rewards over all training: -332.85
46    Average Rewards for last 100 episodes: -158.96
47    Episode took 0.78 seconds
48 Reinforcement Learning Status:
49    Completed 1000 out of 2000 training episodes
50    Average Rewards over all training: -308.47
51    Average Rewards for last 100 episodes: -89.04
52    Episode took 0.76 seconds
53 Reinforcement Learning Status:
54    Completed 1100 out of 2000 training episodes
55    Average Rewards over all training: -288.34
56    Average Rewards for last 100 episodes: -86.98
57    Episode took 0.70 seconds
58 Reinforcement Learning Status:
59    Completed 1200 out of 2000 training episodes
60    Average Rewards over all training: -262.40
61    Average Rewards for last 100 episodes: 22.94
62    Episode took 0.77 seconds
63 Reinforcement Learning Status:
64    Completed 1300 out of 2000 training episodes
65    Average Rewards over all training: -228.72
66    Average Rewards for last 100 episodes: 175.45
67    Episode took 0.74 seconds
68 Reinforcement Learning Status:
69    Completed 1400 out of 2000 training episodes
```

```

70     Average Rewards over all training: -199.15
71     Average Rewards for last 100 episodes: 185.15
72     Episode took 0.75 seconds
73 Reinforcement Learning Status:
74     Completed 1500 out of 2000 training episodes
75     Average Rewards over all training: -174.12
76     Average Rewards for last 100 episodes: 176.33
77     Episode took 0.72 seconds
78 Reinforcement Learning Status:
79     Completed 1600 out of 2000 training episodes
80     Average Rewards over all training: -152.28
81     Average Rewards for last 100 episodes: 175.30
82     Episode took 0.75 seconds
83 Reinforcement Learning Status:
84     Completed 1700 out of 2000 training episodes
85     Average Rewards over all training: -133.01
86     Average Rewards for last 100 episodes: 175.43
87     Episode took 0.81 seconds
88 Reinforcement Learning Status:
89     Completed 1800 out of 2000 training episodes
90     Average Rewards over all training: -117.50
91     Average Rewards for last 100 episodes: 146.05
92     Episode took 0.71 seconds
93 Reinforcement Learning Status:
94     Completed 1900 out of 2000 training episodes
95     Average Rewards over all training: -98.88
96     Average Rewards for last 100 episodes: 236.37
97     Episode took 0.74 seconds
98 Reinforcement Learning Status:
99     Completed 2000 out of 2000 training episodes
100    Average Rewards over all training: -87.15
101    Average Rewards for last 100 episodes: 135.61
102    Episode took 0.73 seconds
103 Training Done (turning off epsilon and alpha)
104 -----
105 Pacman emerges victorious! Score: 499
106 Pacman emerges victorious! Score: 503
107 Pacman emerges victorious! Score: 499
108 Pacman emerges victorious! Score: 499
109 Pacman emerges victorious! Score: 499
110 Pacman emerges victorious! Score: 503
111 Pacman emerges victorious! Score: 499
112 Pacman emerges victorious! Score: 499
113 Pacman emerges victorious! Score: 499
114 Pacman emerges victorious! Score: 499
115 Average Score: 499.8
116 Scores:      499.0, 503.0, 499.0, 499.0, 499.0, 503.0, 499.0, 499.0, 499.0, 499.0
117 Win Rate:    10/10 (1.00)
118 Record:      Win, Win, Win, Win, Win, Win, Win, Win, Win, Win

```

Código 3.1: Ejecución de Q-Learning y Pacman 1

```

1 python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
2 Beginning 10 episodes of Training
3 Pacman died! Score: -507
4 Pacman died! Score: -507
5 Pacman died! Score: -508
6 Pacman died! Score: -505
7 Pacman died! Score: -518
8 Pacman died! Score: -507
9 Pacman died! Score: -508
10 Pacman died! Score: -510
11 Pacman died! Score: -515
12 Pacman died! Score: -506
13 Training Done (turning off epsilon and alpha)
14 -----
15 Average Score: -509.1
16 Scores:      -507.0, -507.0, -508.0, -505.0, -518.0, -507.0, -508.0, -510.0, -515.0, -506.0
17 Win Rate:    0/10 (0.00)
18 Record:      Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss

```

Código 3.2: Ejecución de Q-Learning y Pacman 2

3.1.6. Q-Learning aproximado

```
1 python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid
2 Beginning 2000 episodes of Training
3 Reinforcement Learning Status:
4     Completed 100 out of 2000 training episodes
5     Average Rewards over all training: -509.14
6     Average Rewards for last 100 episodes: -509.14
7     Episode took 0.43 seconds
8 Reinforcement Learning Status:
9     Completed 200 out of 2000 training episodes
10    Average Rewards over all training: -495.69
11    Average Rewards for last 100 episodes: -482.23
12    Episode took 0.69 seconds
13 Reinforcement Learning Status:
14    Completed 300 out of 2000 training episodes
15    Average Rewards over all training: -457.67
16    Average Rewards for last 100 episodes: -381.63
17    Episode took 0.81 seconds
18 Reinforcement Learning Status:
19    Completed 400 out of 2000 training episodes
20    Average Rewards over all training: -445.92
21    Average Rewards for last 100 episodes: -410.67
22    Episode took 0.81 seconds
23 Reinforcement Learning Status:
24    Completed 500 out of 2000 training episodes
25    Average Rewards over all training: -418.67
26    Average Rewards for last 100 episodes: -309.68
27    Episode took 0.85 seconds
28 Reinforcement Learning Status:
29    Completed 600 out of 2000 training episodes
30    Average Rewards over all training: -395.56
31    Average Rewards for last 100 episodes: -280.02
32    Episode took 0.87 seconds
33 Reinforcement Learning Status:
34    Completed 700 out of 2000 training episodes
35    Average Rewards over all training: -380.61
36    Average Rewards for last 100 episodes: -290.93
37    Episode took 0.88 seconds
38 Reinforcement Learning Status:
39    Completed 800 out of 2000 training episodes
40    Average Rewards over all training: -351.85
41    Average Rewards for last 100 episodes: -150.51
42    Episode took 0.97 seconds
43 Reinforcement Learning Status:
44    Completed 900 out of 2000 training episodes
45    Average Rewards over all training: -331.76
46    Average Rewards for last 100 episodes: -171.02
47    Episode took 1.02 seconds
48 Reinforcement Learning Status:
49    Completed 1000 out of 2000 training episodes
50    Average Rewards over all training: -302.39
51    Average Rewards for last 100 episodes: -38.04
52    Episode took 0.95 seconds
53 Reinforcement Learning Status:
54    Completed 1100 out of 2000 training episodes
55    Average Rewards over all training: -277.31
56    Average Rewards for last 100 episodes: -26.55
57    Episode took 0.94 seconds
58 Reinforcement Learning Status:
59    Completed 1200 out of 2000 training episodes
60    Average Rewards over all training: -254.64
61    Average Rewards for last 100 episodes: -5.31
62    Episode took 0.93 seconds
63 Reinforcement Learning Status:
64    Completed 1300 out of 2000 training episodes
65    Average Rewards over all training: -221.53
66    Average Rewards for last 100 episodes: 175.84
67    Episode took 0.94 seconds
68 Reinforcement Learning Status:
69    Completed 1400 out of 2000 training episodes
```

```

70     Average Rewards over all training: -193.09
71     Average Rewards for last 100 episodes: 176.57
72     Episode took 0.92 seconds
73 Reinforcement Learning Status:
74     Completed 1500 out of 2000 training episodes
75     Average Rewards over all training: -165.84
76     Average Rewards for last 100 episodes: 215.79
77     Episode took 0.94 seconds
78 Reinforcement Learning Status:
79     Completed 1600 out of 2000 training episodes
80     Average Rewards over all training: -143.87
81     Average Rewards for last 100 episodes: 185.66
82     Episode took 0.96 seconds
83 Reinforcement Learning Status:
84     Completed 1700 out of 2000 training episodes
85     Average Rewards over all training: -117.95
86     Average Rewards for last 100 episodes: 296.68
87     Episode took 0.98 seconds
88 Reinforcement Learning Status:
89     Completed 1800 out of 2000 training episodes
90     Average Rewards over all training: -98.24
91     Average Rewards for last 100 episodes: 236.87
92     Episode took 0.91 seconds
93 Reinforcement Learning Status:
94     Completed 1900 out of 2000 training episodes
95     Average Rewards over all training: -82.76
96     Average Rewards for last 100 episodes: 195.92
97     Episode took 0.96 seconds
98 Reinforcement Learning Status:
99     Completed 2000 out of 2000 training episodes
100     Average Rewards over all training: -64.30
101     Average Rewards for last 100 episodes: 286.37
102     Episode took 0.97 seconds
103 Training Done (turning off epsilon and alpha)
104 -----
105 Pacman emerges victorious! Score: 499
106 Pacman emerges victorious! Score: 499
107 Pacman emerges victorious! Score: 502
108 Pacman emerges victorious! Score: 499
109 Pacman emerges victorious! Score: 502
110 Pacman emerges victorious! Score: 495
111 Pacman emerges victorious! Score: 495
112 Pacman emerges victorious! Score: 503
113 Pacman emerges victorious! Score: 495
114 Pacman emerges victorious! Score: 503
115 Average Score: 499.2
116 Scores:      499.0, 499.0, 502.0, 499.0, 502.0, 495.0, 495.0, 503.0, 495.0, 503.0
117 Win Rate:    10/10 (1.00)
118 Record:      Win, Win, Win, Win, Win, Win, Win, Win, Win, Win

```

Código 3.3: Q-Learning aproximado

```

1 python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumGrid
2 Beginning 50 episodes of Training
3 Training Done (turning off epsilon and alpha)
4 -----
5 Pacman emerges victorious! Score: 525
6 Pacman emerges victorious! Score: 527
7 Pacman emerges victorious! Score: 521
8 Pacman emerges victorious! Score: 529
9 Pacman emerges victorious! Score: 529
10 Pacman emerges victorious! Score: 527
11 Pacman emerges victorious! Score: 529
12 Pacman emerges victorious! Score: 525
13 Pacman emerges victorious! Score: 529
14 Pacman emerges victorious! Score: 529
15 Average Score: 527.0
16 Scores:      525.0, 527.0, 521.0, 529.0, 529.0, 527.0, 529.0, 525.0, 529.0, 529.0
17 Win Rate:    10/10 (1.00)
18 Record:      Win, Win, Win, Win, Win, Win, Win, Win, Win, Win

```

Código 3.4: Q-Learning aproximado


```

1 python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumClassic
2 Beginning 50 episodes of Training
3 Training Done (turning off epsilon and alpha)
4 -----
5 Pacman emerges victorious! Score: 1330
6 Pacman emerges victorious! Score: 1337
7 Pacman emerges victorious! Score: 1335
8 Pacman emerges victorious! Score: 1329
9 Pacman emerges victorious! Score: 1332
10 Pacman died! Score: 68
11 Pacman emerges victorious! Score: 1324
12 Pacman emerges victorious! Score: 1322
13 Pacman emerges victorious! Score: 1336
14 Pacman emerges victorious! Score: 1329
15 Average Score: 1204.2
16 Scores:      1330.0, 1337.0, 1335.0, 1329.0, 1332.0, 68.0, 1324.0, 1322.0, 1336.0, 1329.0
17 Win Rate:    9/10 (0.90)
18 Record:      Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win

```

Código 3.5: Q-Learning aproximado

3.2. Autograder

```
1 python autograder.py
2 Starting on 12-17 at 17:23:30
3
4 Question q1
5 =====
6
7 *** PASS: test_cases/q1/1-tinygrid.test
8 *** PASS: test_cases/q1/2-tinygrid-noisy.test
9 *** PASS: test_cases/q1/3-bridge.test
10 *** PASS: test_cases/q1/4-discountgrid.test
11
12 ### Question q1: 4/4 ###
13
14
15 Question q2
16 =====
17
18 *** PASS: test_cases/q2/1-bridge-grid.test
19
20 ### Question q2: 1/1 ###
21
22
23 Question q3
24 =====
25
26 *** PASS: test_cases/q3/1-tinygrid.test
27 *** PASS: test_cases/q3/2-tinygrid-noisy.test
28 *** PASS: test_cases/q3/3-bridge.test
29 *** PASS: test_cases/q3/4-discountgrid.test
30
31 ### Question q3: 4/4 ###
32
33
34 Question q4
35 =====
36
37 *** PASS: test_cases/q4/1-tinygrid.test
38 *** PASS: test_cases/q4/2-tinygrid-noisy.test
39 *** PASS: test_cases/q4/3-bridge.test
40 *** PASS: test_cases/q4/4-discountgrid.test
41
42 ### Question q4: 2/2 ###
43
44
45 Question q5
46 =====
47
48 Beginning 2000 episodes of Training
49 Reinforcement Learning Status:
50   Completed 100 out of 2000 training episodes
51   Average Rewards over all training: -510.09
52   Average Rewards for last 100 episodes: -510.09
53   Episode took 0.32 seconds
54 Reinforcement Learning Status:
55   Completed 200 out of 2000 training episodes
56   Average Rewards over all training: -511.44
57   Average Rewards for last 100 episodes: -512.78
58   Episode took 0.52 seconds
59 Reinforcement Learning Status:
60   Completed 300 out of 2000 training episodes
61   Average Rewards over all training: -474.83
62   Average Rewards for last 100 episodes: -401.61
63   Episode took 0.58 seconds
64 Reinforcement Learning Status:
65   Completed 400 out of 2000 training episodes
66   Average Rewards over all training: -443.94
67   Average Rewards for last 100 episodes: -351.28
68   Episode took 0.66 seconds
```

```

69 Reinforcement Learning Status:
70   Completed 500 out of 2000 training episodes
71   Average Rewards over all training: -425.37
72   Average Rewards for last 100 episodes: -351.08
73   Episode took 0.69 seconds
74 Reinforcement Learning Status:
75   Completed 600 out of 2000 training episodes
76   Average Rewards over all training: -392.55
77   Average Rewards for last 100 episodes: -228.44
78   Episode took 0.66 seconds
79 Reinforcement Learning Status:
80   Completed 700 out of 2000 training episodes
81   Average Rewards over all training: -361.91
82   Average Rewards for last 100 episodes: -178.11
83   Episode took 0.68 seconds
84 Reinforcement Learning Status:
85   Completed 800 out of 2000 training episodes
86   Average Rewards over all training: -345.18
87   Average Rewards for last 100 episodes: -228.07
88   Episode took 0.66 seconds
89 Reinforcement Learning Status:
90   Completed 900 out of 2000 training episodes
91   Average Rewards over all training: -331.22
92   Average Rewards for last 100 episodes: -219.49
93   Episode took 0.76 seconds
94 Reinforcement Learning Status:
95   Completed 1000 out of 2000 training episodes
96   Average Rewards over all training: -312.16
97   Average Rewards for last 100 episodes: -140.62
98   Episode took 0.79 seconds
99 Reinforcement Learning Status:
100  Completed 1100 out of 2000 training episodes
101  Average Rewards over all training: -288.21
102  Average Rewards for last 100 episodes: -48.77
103  Episode took 0.79 seconds
104 Reinforcement Learning Status:
105  Completed 1200 out of 2000 training episodes
106  Average Rewards over all training: -270.74
107  Average Rewards for last 100 episodes: -78.52
108  Episode took 0.76 seconds
109 Reinforcement Learning Status:
110  Completed 1300 out of 2000 training episodes
111  Average Rewards over all training: -243.37
112  Average Rewards for last 100 episodes: 85.10
113  Episode took 0.71 seconds
114 Reinforcement Learning Status:
115  Completed 1400 out of 2000 training episodes
116  Average Rewards over all training: -220.66
117  Average Rewards for last 100 episodes: 74.50
118  Episode took 0.72 seconds
119 Reinforcement Learning Status:
120  Completed 1500 out of 2000 training episodes
121  Average Rewards over all training: -190.82
122  Average Rewards for last 100 episodes: 226.94
123  Episode took 0.69 seconds
124 Reinforcement Learning Status:
125  Completed 1600 out of 2000 training episodes
126  Average Rewards over all training: -161.59
127  Average Rewards for last 100 episodes: 276.94
128  Episode took 0.70 seconds
129 Reinforcement Learning Status:
130  Completed 1700 out of 2000 training episodes
131  Average Rewards over all training: -141.72
132  Average Rewards for last 100 episodes: 176.13
133  Episode took 0.70 seconds
134 Reinforcement Learning Status:
135  Completed 1800 out of 2000 training episodes
136  Average Rewards over all training: -121.80
137  Average Rewards for last 100 episodes: 216.92
138  Episode took 0.69 seconds
139 Reinforcement Learning Status:

```

```

140 Completed 1900 out of 2000 training episodes
141 Average Rewards over all training: -105.07
142 Average Rewards for last 100 episodes: 195.96
143 Episode took 0.76 seconds
144 Reinforcement Learning Status:
145 Completed 2000 out of 2000 training episodes
146 Average Rewards over all training: -89.46
147 Average Rewards for last 100 episodes: 207.10
148 Episode took 0.67 seconds
149 Training Done (turning off epsilon and alpha)
150 -----
151 Pacman emerges victorious! Score: 503
152 Pacman emerges victorious! Score: 503
153 Pacman emerges victorious! Score: 499
154 Pacman emerges victorious! Score: 499
155 Pacman emerges victorious! Score: 503
156 Pacman emerges victorious! Score: 503
157 Pacman emerges victorious! Score: 503
158 Pacman emerges victorious! Score: 503
159 Pacman emerges victorious! Score: 495
160 Pacman emerges victorious! Score: 503
161 Pacman emerges victorious! Score: 499
162 Pacman emerges victorious! Score: 503
163 Pacman emerges victorious! Score: 503
164 Pacman emerges victorious! Score: 503
165 Pacman emerges victorious! Score: 503
166 Pacman emerges victorious! Score: 503
167 Pacman emerges victorious! Score: 495
168 Pacman emerges victorious! Score: 503
169 Pacman emerges victorious! Score: 503
170 Pacman emerges victorious! Score: 503
171 Pacman emerges victorious! Score: 503
172 Pacman emerges victorious! Score: 503
173 Pacman emerges victorious! Score: 503
174 Pacman emerges victorious! Score: 503
175 Pacman emerges victorious! Score: 499
176 Pacman emerges victorious! Score: 495
177 Pacman emerges victorious! Score: 503
178 Pacman emerges victorious! Score: 503
179 Pacman emerges victorious! Score: 495
180 Pacman emerges victorious! Score: 495
181 Pacman emerges victorious! Score: 495
182 Pacman emerges victorious! Score: 499
183 Pacman emerges victorious! Score: 499
184 Pacman emerges victorious! Score: 503
185 Pacman emerges victorious! Score: 503
186 Pacman emerges victorious! Score: 503
187 Pacman emerges victorious! Score: 503
188 Pacman emerges victorious! Score: 495
189 Pacman emerges victorious! Score: 499
190 Pacman emerges victorious! Score: 503
191 Pacman emerges victorious! Score: 495
192 Pacman emerges victorious! Score: 499
193 Pacman emerges victorious! Score: 503
194 Pacman emerges victorious! Score: 503
195 Pacman emerges victorious! Score: 503
196 Pacman emerges victorious! Score: 503
197 Pacman emerges victorious! Score: 499
198 Pacman emerges victorious! Score: 503
199 Pacman emerges victorious! Score: 495
200 Pacman emerges victorious! Score: 503
201 Pacman emerges victorious! Score: 503
202 Pacman emerges victorious! Score: 503
203 Pacman emerges victorious! Score: 503
204 Pacman emerges victorious! Score: 499
205 Pacman emerges victorious! Score: 503
206 Pacman emerges victorious! Score: 499
207 Pacman emerges victorious! Score: 503
208 Pacman emerges victorious! Score: 503
209 Pacman emerges victorious! Score: 499
210 Pacman emerges victorious! Score: 503

```



```

269
270 Question q6
271 =====
272
273 *** PASS: test_cases/q6/1-tinygrid.test
274 *** PASS: test_cases/q6/2-tinygrid-noisy.test
275 *** PASS: test_cases/q6/3-bridge.test
276 *** PASS: test_cases/q6/4-discountgrid.test
277 *** PASS: test_cases/q6/5-coord-extractor.test
278
279 ### Question q6: 3/3 ###
280
281
282 Finished at 17:23:45
283
284 Provisional grades
285 =====
286 Question q1: 4/4
287 Question q2: 1/1
288 Question q3: 4/4
289 Question q4: 2/2
290 Question q5: 1/1
291 Question q6: 3/3
292 -----
293 Total: 15/15
294
295 Your grades are NOT yet registered. To register your grades, make sure
296 to follow your instructor's guidelines to receive credit on your project.

```

Código 3.6: Autograder