Práctica 1

Laura Rodríguez Navas
rodrigueznavas@posgrado.uimp.es

2 de abril de 2021

Ejercicio 1

1. Descargar el código fuente para esta práctica, *softpractica1.zip*, de la página web de la asignatura.
2. Descomprimir el fichero anterior.
3. Abrir un terminal o consola de comandos y entrar dentro de la carpeta *softpractica1*.
4. Para empezar vamos a ejecutar GridWorld en el modo de control manual, usando el comando *python gridworld.py -m -n 0*, que utiliza las teclas de flecha.
5. El objetivo es lograr llegar lo antes posible a la celda etiquetada con un 1, evitando caer en la celda con un -1.

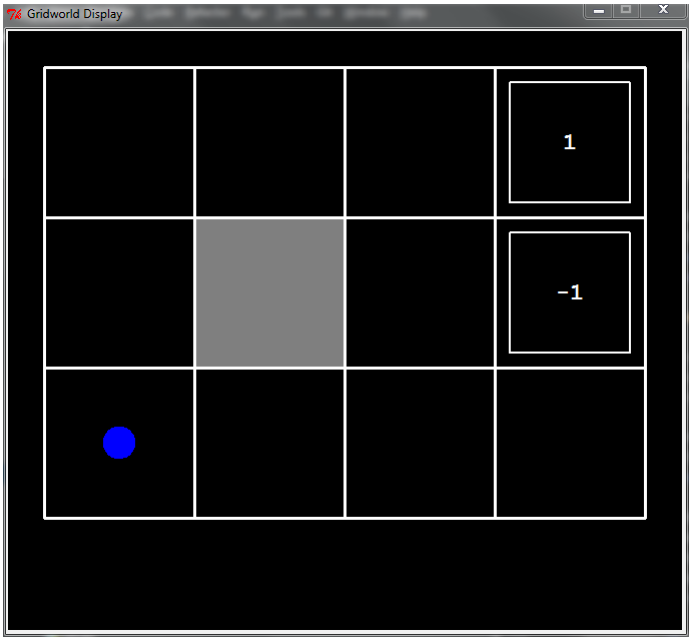

Figura 1: Interfaz del dominio GridWorld en el modo de control manual.

Preguntas

1. ¿Cuántas celdas/estados aparecen en el tablero? ¿Cuántas acciones puede ejecutar el agente? Si quisieras resolver el juego mediante aprendizaje por refuerzo, ¿cómo lo harías?

   En el tablero aparecen 11 celdas/estado y el agente puede ejecutar 4 acciones: arriba (*north*), abajo (*south*), izquierda (*west*) y derecha (*east*).

   Para resolver el juego usaremos el algoritmo Q-learning con el objetivo de que el agente llegue lo antes posible a la celda etiquetada con un 1 (estado *done*), evitando caer en la celda con un -1 (estado *exit*). A medida que el agente se mueve por el laberinto, pierde salud gradualmente, por lo que tiene que moverse con un propósito.

   La clave del algoritmo Q-learning será la construcción de su tabla Q, que es una matriz donde tendremos las recompensas que obtendrá el agente para cada acción y en cada estado, es decir, los valores $Q(s, a)$. El algoritmo hará que el agente vaya tomando decisiones y a cada decisión actualizará uno de los valores $Q(s, a)$ de la tabla Q. A la hora de tomar las decisiones el algoritmo usa la estrategia ϵ-greedy. Esta estrategia consiste en que todas las acciones sean tomadas buscando el valor máximo de $Q(s, a)$ pero existiendo una probabilidad pequeña ϵ, de tomar una decisión aleatoria para que el agente explore todo el espacio de soluciones. Para la actualización de la tabla Q e ir completándola, cada decisión tomada por el agente se evaluará por la siguiente expresión:

$$Q^*(s_t, a_t) \leftarrow (1 - \alpha)\, Q(s_t, a_t) + \alpha_t \left[r_{s_t,a_t} + \gamma\, max_a\, Q(s_{t+1}, a)\right] \tag{1}$$

   En esta expresión $Q(s_t, a_t)$ es el valor de la tabla Q del estado s_t y la acción a_t. $Q^*(s_t, a_t)$ será el nuevo valor de la tabla Q para dicho estado y acción. El valor γ es el *learning rate*, que puede tomar valores entre 0 y 1. Finalmente r_{s_t,a_t}, determina la recompensa inmediata asociada a la acción tomada y es el *discount factor* que también puede tomar valores entre 0 y 1.

2. Abrir el fichero *qlearningAgents.py* y buscar la clase *QLearningAgent*. Describir los métodos que aparecen en ella.

   Los métodos que aparecen en la clase *QLearningAgent* son:

   - **__init__**: Inicializa la *Q-Table* a partir del fichero *qtable.txt*, es decir, la *Q-Table* se inicializa a cero.
   - **readQtable**: Lee la *Q-Table* del fichero *qtable.txt*.
   - **writeQtable**: Escribe la *Q-Table* en el fichero *qtable.txt*.
   - **__del__**: Llama al método *writeQtable* que escribe el resultado final de la *Q-Table* en el fichero *qtable.txt*.
   - **computePosition**: Calcula la fila de la *Q-Table* para un estado dado.
   - **getQValue**: Devuelve el valor $Q(s, a)$ para un estado y una acción dados. De lo contrario, devuelve 0.0, si nunca hemos visto el estado o el valor del nodo Q.
   - **computeValueFromQValues**: Devuelve el valor máximo de $Q(s, a)$ para un estado dado. Este valor se encuentra por encima de las acciones válidas. Si no hay acciones válidas, como en el caso del estado *exit*, devuelve 0.0.
   - **computeActionFromQValues**: Calcula la mejor acción a realizar para un estado dado. Si no hay acciones válidas, como en el caso del estado *exit*, devuelve *None*.

- **getAction**: Calcula la acción a realizar para un estado dado. En caso contrario, con probabilidad *self.epsilon*, elige una acción aleatoria y la mejor acción política. Si no hay acciones válidas, como en el caso del estado *exit*, elige *None* como acción.
- **update**: Actualiza la *Q-Table*. El método para un acción dada, observa una recompensa, introduce un estado nuevo (que depende del estado anterior y de la acción dada), y actualiza el valor $Q(s, a)$.

  Si el nuevo estado introducido es el estado *exit*, se sigue la regla:

  $$Q(state, action) < -(1 - self.alpha) * Q(state, action) + self.alpha * (reward + 0)$$

  De lo contrario, si el nuevo estado introducido no es el estado *exit*, se sigue la regla:

  $$Q(state, action) < -(1 - self.alpha) * Q(state, action) + self.alpha * (reward + self.discount * max_{a'} Q(nextState, a'))$$

- **getPolicy**: Devuelve la mejor acción de la *Q-Table* para un estado dado.
- **getValue**: Devuelve el valor $Q(s, a)$ más alto para un estado dado.

3. Ejecuta ahora el agente anterior con: *python gridworld.py -a q -k 100 -n 0.*

   A diferencia de la primera ejecución, en esta ejecución le indicamos el tipo de agente, que en este caso es q, y el número de movimientos MDP a realizar, que en este caso son 100.

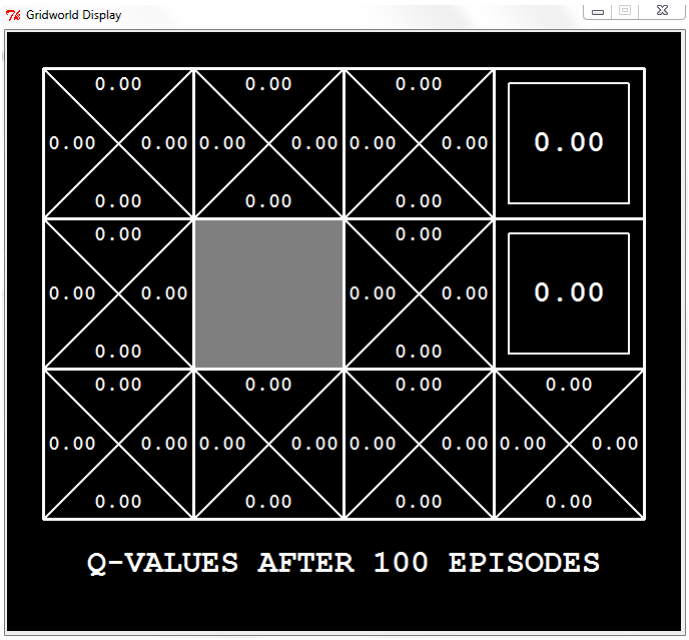

Figura 2: Interfaz del dominio GridWorld usando *python gridworld.py -a q -k 100 -n 0.*

4. ¿Qué información se muestra en el laberinto? ¿Qué aparece por terminal cuando se realizan los movimientos en el laberinto?

   Si observamos la Figura 2, vemos que la información que se muestra en el laberinto son los valores de $Q(s, a)$ después de realizarse 100 movimientos.

En la terminal cuando se realizan los movimientos en el laberinto, vemos que aparecen los siguientes valores por cada movimiento:

- La posición (x, y) donde empieza el estado. Por ejemplo: (2, 1).
- La acción tomada. Por ejemplo: derecha (*east*).
- La posición (x, y) donde acaba el estado. Por ejemplo: (3, 1).
- La recompensa obtenida. Por ejemplo: 0.0, en este caso no ha habido recompensa.

5. ¿Qué clase de movimiento realiza el agente anterior?

   El agente desde una posición y salud, siempre que decida moverse en una dirección por el laberinto, lo hace en esa dirección con probabilidad igual a 1, es decir, se mueve una posición.

6. ¿Se pueden sacar varias políticas óptimas? Describe todas las políticas óptimas para este problema.

   TODO

7. Escribir el método *update* de la clase *QLearningAgent* utilizando las funciones de actualización del algoritmo *Q-Learning*. Para ello, inserta el código necesario allí donde aparezca la etiqueta INSERTA TU CÓDIGO AQUÍ siguiendo las instrucciones que se proporcionan, con el fin de conseguir el comportamiento deseado.

   El código que se ha insertado en el método *update* de la clase *QLearningAgent* es:

```
position = self.computePosition(state)
action_column = self.actions[action]

if nextState != 'TERMINAL_STATE':
  # Q(state,action) <- (1-self.alpha) * Q(state,action) + self.alpha * (reward
    + self.discount * max a' Q(nextState, a'))
  sample = (1 - self.alpha) * self.getQValue(state, action) + self.alpha * (
    reward + self.discount * self.computeValueFromQValues(nextState))
  self.q_table[position][action_column] = sample

elif nextState == 'TERMINAL_STATE':
  # Q(state,action) <- (1-self.alpha) * Q(state,action) + self.alpha * (reward
    + 0)
  sample = (1 - self.alpha) * self.getQValue(state, action) + self.alpha * (
    reward + 0)
  self.q_table[position][action_column] = sample
```

   En la Figura 3 podemos observar que con el código insertado anteriormente, los valores de $Q(s, a)$ se han actualizado después de volver a ejecutar *python gridworld.py -a q -k 100 -n 0.*

8. Establece en el constructor de la clase *QLearningAgent* el valor de la variable *epsilon* a 0,05. Ejecuta nuevamente con: *python gridworld.py -a q -k 100 -n 0.* ¿Qué sucede?

   Si consideramos que un episodio termina si se gana (obtenemos un valor de recompensa positivo) o se pierde (obtenemos un a valor de recompensa negativo), decimos que se gana si el agente alcanza el objetivo, la celda etiquetada con un 1, en el estado TERMINAL_STATE. Por el contrario decimos que se pierde si el agente no alcanza la celda etiquetada con un 1, alcanza la celda etiquetada con un -1 en el estado TERMINAL_STATE. En ese caso, cuando *epsilon* es igual a 0,05 (ver Figura 4), se logra llegar más veces a la celda etiquetada con un 1, es

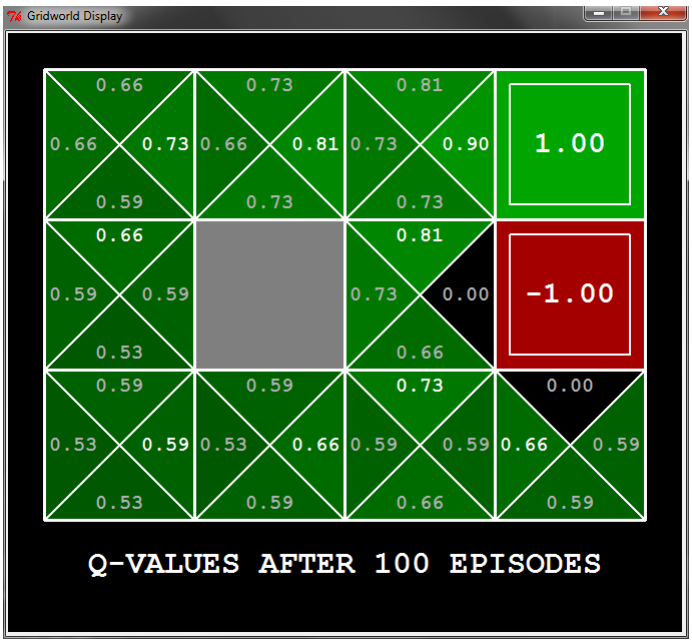

Figura 3: Interfaz del dominio GridWorld cuando *epsilon* es igual a 1.

decir, existen más episodios donde se gana que donde se pierde. Por el contrario cuando *epsilon* es igual a 1, se logra llegar más veces a la celda etiquetada con un -1, es decir, existen más episodios donde se pierde que donde se gana.

Cuando *epsilon* es igual a 0,05:

- EPISODE 100 COMPLETE: RETURN WAS 0.59049
- AVERAGE RETURNS FROM START STATE: 0.530784257974

Cuando *epsilon* es igual a 1:

- EPISODE 100 COMPLETE: RETURN WAS 0.0119725151826
- AVERAGE RETURNS FROM START STATE: -0.079260284124

Comprobamos que cuando *epsilon* es igual a 0,05 ganamos: 0.530784257974 >= 0; y cuando *epsilon* es igual a 1 perdemos: -0.079260284124 <= 0.

9. Después de la ejecución anterior, abrir el fichero *qtable.txt*. ¿Qué contiene?

   El fichero *qtable.txt* contiene los valores de $Q(s, a)$ después de todas las actualizaciones en todos los episodios. Podemos ver el contenido del fichero fichero *qtable.txt* a continuación:

Ejercicio 2

En el ejercicio anterior, siempre que el agente decidía moverse hacia una dirección se movía en esa dirección con probabilidad 1. Es decir, se trataba de un MDP determinista. Ahora vamos a crear un MDP estocástico.

Figura 4: Interfaz del dominio GridWorld cuando *epsilon* es igual a 0,05.

1. Ejecuta y juega un par de partidas con el agente manual: *python gridworld.py -m -n 0.3*. ¿Qué sucede? ¿Crees que el agente *QLearningAgent* será capaz de aprender en este nuevo escenario?

2. Reiniciar los valores de la tabla Q del fichero *qtable.txt*. Para ello ejecutar desde el terminal: *cp qtable.ini.txt qtable.txt*.

3. Ejecutar el agente *QLearningAgent*: *python gridworld.py -a q -k 100 -n 0.3*.

4. Tras unas cuantos episodios, ¿se genera la política óptima? Y si se genera, ¿se tarda más o menos que en el caso determinista?

0.59049	0.119569809907	0.0	0.0	0.0
0.0	0.0	0.0	0.398575844342	0.0
0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0
0.6561	0.153677329102	0.0	0.4428675	0.0
0.0	0.0	0.0	0.0	0.0
0.568245849609	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	-0.75
0.32805	0.729	0.295245	0.327896389246	0.0
0.0	0.81	0.364499433769	0.0	0.0
0.0	0.9	0.148078125	0.0	0.0
0.0	0.0	0.0	0.0	1.0

Tabla 1: Contenido del fichero *qtable.txt*.

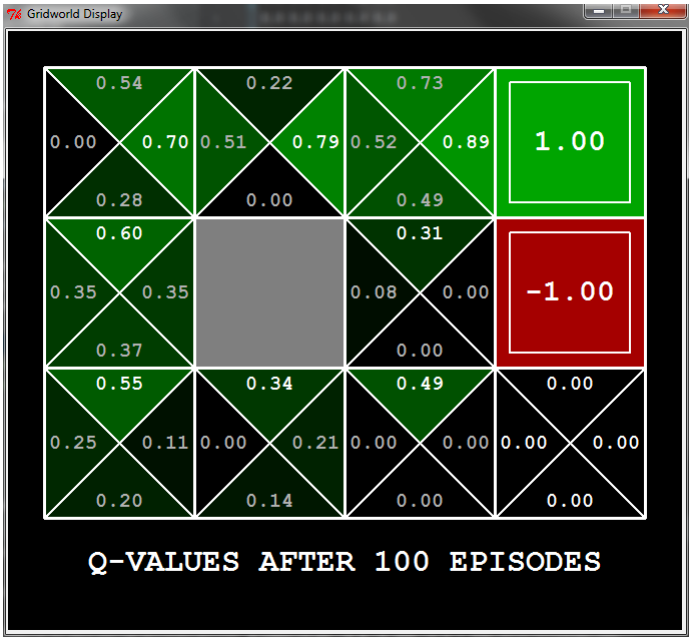

Figura 5: Interfaz del dominio GridWorld.