Práctica 1

Laura Rodríguez Navas
rodrigueznavas@posgrado.uimp.es

6 de marzo 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, que utiliza las teclas de flecha (ver Figura 1).
   python gridworld.py -m -n 0
5. El objetivo es lograr llegar lo antes posible a la celda etiquetada con un 1, evitando caer en la celda con un -1.

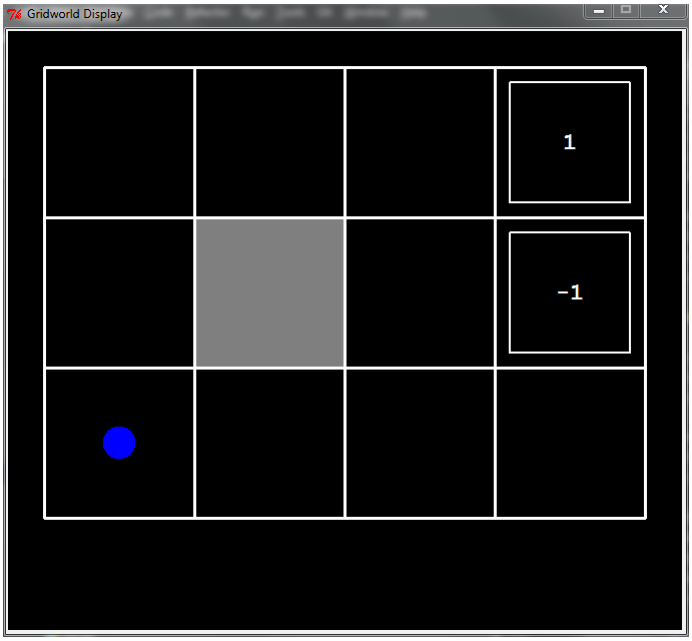

Figura 1: Interfaz del dominio GridWorld.

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.

   El agente puede ejecutar 6 acciones: *north*, *west*, *south*, *east*, *exit* y *done*.

   Para resolver el juego mediante el algoritmo Q-Learning: **TODO**

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:

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

   - 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.

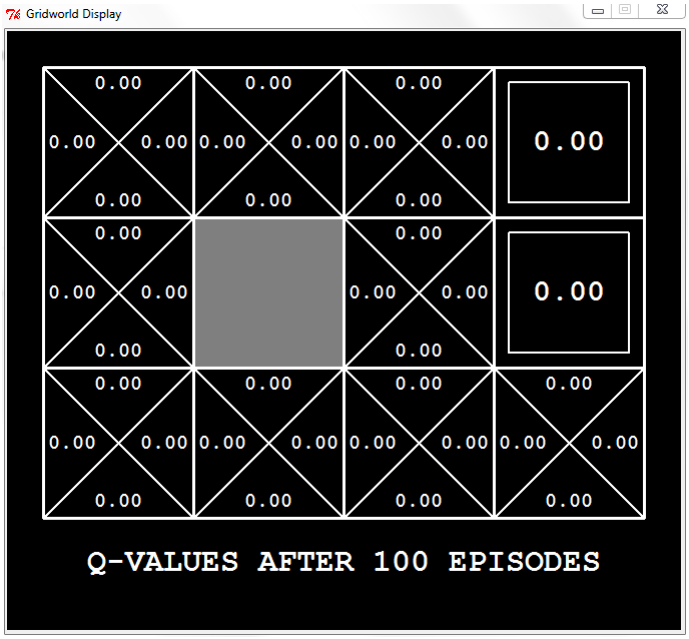

Figura 2: Interfaz del dominio GridWorld.

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

   A diferencia de la primera ejecución, en esta le indicamos el tipo de agente (q) y el número de episodios del MDP a ejecutar (100).

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 son los valores de $Q(s, a)$.

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

   - La posición (x, y) donde empieza el estado. Por ejemplo: (2, 1).
   - La acción tomada. Por ejemplo: *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?

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

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.

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?

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

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:

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?