PRÁCTICA INTELIGENCIA ARTIFICIAL
Q-LEARNING

Grado en Diseño y Desarrollo de Videojuegos

Universidad Rey Juan Carlos

Nerea Díaz Jérica

Alejandro Cavero Sebastián

1. INTRODUCCIÓN **3**
1.1. Aprendizaje por refuerzo: Q-learning 3
1.2. Elementos involucrados 3
1.3. Fases 3
1.4. Requerimientos previos 4
2. APRENDIZAJE CON EXPLORACIÓN 100% **4**
2.1. Método Get_Q 5
2.2. Método Get_Reward 5
2.3. Método Get_maxQ 5
2.4. Método Update_Rule. 6
2.5. Método Update_tableQ 6
2.6. Evalute_Stop 6
2.7. CopyResult 6
2.8. Fichero 6
3. EXPLORACIÓN (TABLA YA APRENDIDA) **7**
3.1. Get_Best_Action 7
4. MOVER AL PERSONAJE **7**
5. RESULTADOS OBTENIDOS **7**

1. INTRODUCCIÓN

1.1. Aprendizaje por refuerzo: Q-learning

En esta práctica se va a implementar el algoritmo Q-learning para que la máquina aprenda a encontrar el mejor camino para llegar a la meta, fomentando con recompensas para lograr el objetivo. Para ello se va a crear un array que guarde una de las direcciones a las que puede acceder cada celda: norte sur, este y oeste, y en cada iteración se generarán un estado y acción aleatorios para los cuales se calculará su valor de Q, guardando el resultado en la tabla de forma que el agente vaya tomando decisiones y aprendiendo a recorrer el tablero de forma óptima.

1.2. Elementos involucrados

Para poder llevar a cabo este aprendizaje son necesarios varios conceptos que se van a implementar en la práctica, los cuales se van a explicar a continuación brevemente.

- FSM: consiste en la máquina de estados que modela el entorno, codificando los detalles del entorno y variables. En este caso es el propio juego a través de clases implementadas para poner en funcionamiento el juego.
- Estado(S): la máquina genera una posición aleatoria en el tablero compuesta por el número de fila (entre 0 y 7) y el número de columna (entre 0 y 14), y a partir de la cual se accederá a una celda contigua aleatoria para poder evaluar el valor de Q.
- Acción(A): la máquina, una vez ha seleccionado el estado (la delta del tablero), genera una acción aleatoria que puede ser norte, sur este u oeste, y será la dirección que tomará para evaluar la celda vecina y poder actualizar el valor de Q.
- Recompensa(r): para reforzar y fomentar el aprendizaje, existe la recompensa al llegar a la meta, con un valor fijo pero que se multiplica cuanto mejor sea el camino elegido.
- Valor Q: este valor se encarga de evaluar lo buena que es una acción para un estado y una acción aleatorios, de forma que se actualiza con el valor máximo de la celda a la que se quiere acceder hasta conseguir el mejor valor posible o haber realizado recorrido un número determinado de veces la tabla.

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max\left(Q\left(s', \forall a'\right)\right)\right)$$

1.3. Fases

Para llevar a cabo el proceso, se divide la práctica en dos métodos principales: aprendizaje y explotación. Con el primero se realizará la exploración del tablero actualizando los valores de Q y con el segundo se procederá a seguir los mejores valores Q calculados para cada celda, llegando así hasta la meta por el mejor camino.

Esta tabla que se crea para guardar los valores de Q, se guardará además en un fichero .txt de forma que la máquina accediendo al fichero pueda recorrer camino leyendo sus valores.

1.4. Requerimientos previos

Para realizar está práctica, se ha utilizado el código proporcionado para poder ejecutar el juego en Unity, en el cual se implementó la primera práctica de la asignatura que consistía en implementar agentes de búsqueda, asignado al personaje un controlador que será usado también para este algoritmo.

A partir de este controlador llamado *AbstractPathController* se ha creado la clase *QlearningMind* que hereda de ella implementando tres métodos principales: *Repath()* y *GetNextMove()* que es donde se implementará el código de el algoritmo Q-learning, llamando a nuevos métodos que se verán descritos en la memoria. Para que el jugador siga este código solo se requiere añadir este script *QlearningMind* como componente a *PathController* del personaje.

En cuanto a los parámetros, consiste en una semilla que establece un escenario aleatorio (en nuestro caso el 22), y las variables alpha y gamma que conforman el ratio o ritmo de aprendizaje y el de descuento, y cuyos valores se han asignado a 0.7 y 0.5 respectivamente.

Por último, como se mencionaba anteriormente, al ejecutar el código se genera un fichero externo llamado tableQ.txt que guarda todos los valores de Q ordenados, y se sitúa en la dirección "./Assets/Scripts/Grupo22/Qlearning/tableQ.txt" en la carpeta Assets proporcionada.

2. APRENDIZAJE CON EXPLORACIÓN 100%

Para implementar esta primera parte se ha creado el método *aprendizajeQ* a partir del cual se llama a todos los métodos necesarios de forma que cuando entramos, comenzamos un bucle para recorrer todo el tablero que solo se romperá si lo hemos recorrido un total de 100 veces cada una de las 100 celdas. Este bucle de recorrer 100 celdas dentro de las 100 veces que se ejecuta esta instrucción, a su vez sólo puede terminar si hemos dado con la meta.

Ahora bien comenzando el recorrido de las celdas, lo primero que se hace es generar tanto un estado como una acción aleatorias, asegurando que no se genera un estado en un muro o una acción que lleve a uno, y con ello, accedemos a la celda que queremos explorar, guardando el valor tanto de esta como la de la vecina a la que se va a acceder.

```
CellInfo next_cell = Get_random_cell(boardInfo);
CellInfo current_cell = next_cell;
current_action = Get_random_action(boardInfo, current_cell);
next_cell = Run_FSM(boardInfo, current_cell, current_action);
```

2.1. Método Get_Q

Devuelve el valor de Q en esta posición. Para ello recibe como parámetros la tabla en la que vamos a guardar todos los valores(un array llamado *tableQ*, declarado como global para acceder desde cualquier método), la celda actual, la acción y la información de tablero. Calculamos el índice con esta fórmula y devolvemos el valor Q obtenido.

```
float idx = current_cell.RowId * boardInfo.NumColumns +
current_cell.ColumnId;

return tableQ[(int)idx, (int)current_action];
```

2.2. Método Get_Reward

Para la celda a la que queremos acceder, observamos si recibe alguna recompensa o penalización. Si el agente trata de atravesar un muro recibirá una penalización de -1; si llega a la meta en cambio, recibirá una recompensa de 100l.

```
if(!next_cell.Walkable){
                reward = -1;
            }

            for(int i = 0; i <goals.Length; i++){
                if(next_cell == goals[i]){
                    reward = 100;
                }
            }
```

2.3. Método Get_maxQ

Para poder actualizar el valor de Q es necesario saber cual es el máximo valor Q de la celda a la que queremos acceder, para ello sabiendo su posición en la tabla, miramos el valor guardado para ese estado y esa acción y lo devolvemos como parámetro.

```
float idx = next_cell.RowId * boardInfo.NumColumns +
next_cell.ColumnId;
            float maxQ = 0;
            for(int i = 0; i <boardInfo.NumColumns; i++){
                if(tableQ[(int)idx, i] > maxQ){
                    maxQ = tableQ[(int)idx,i];
                }
            }
            return maxQ;
```

2.4. Método Update_Rule.

Ahora que tenemos todas las variables necesarias, podemos calcular el nuevo valor de Q con la fórmula que presentábamos más arriba, haciendo uso de el mayor valor Q de la celda vecina, alpha, gamma y el valor actual de Q del estado y acción actuales.

```
return Q=(1-alpha)*current_Q + alpha*(reward + gamma * next_Qmax);
```

2.5. Método Update_tableQ

Una vez calculado este nuevo valor Q, lo metemos en la tabla, actualizando el valor anterior, cambiándolo por este nuevo. Necesitaremos la tabla, la celda en la que estamos, la acción, la nueva Q y board info para índices.

```
float idx = current_cell.RowId * boardInfo.NumColumns +
current_cell.ColumnId;

tableQ[(int)idx, (int)current_action] = next_Q;
```

2.6. Evalute_Stop

Tal y como se introducía anteriormente, el bucle de recorrer las celdas a partir de la primera aleatoria generada, solo se rompe si hemos llegado al máximo de iteraciones (100), es decir, hemos recorrido todas las celdas, o bien si hemos llegado a la meta. Con este método comprobamos si ha sucedido alguna de estas dos cosas y si es así devolvemos true.

```
if(iter == N_ITER_MAX || next_cell == goals[0]){
   return true;
}else{
   return false;
}
```

2.7. Fichero

Como decíamos, ahora toda esta tabla con los valores de Q, se guarda en un fichero. Para ello hemos situado en cada fila del fichero una celda, numeradas por filas de 0 al 7 y por columnas de la A a la M. De esta forma para la celda XY podemos ver 4 valores en orden separados por barras que muestran los valores para la acción norte, este, sur y oeste en este orden.

3. EXPLORACIÓN (TABLA YA APRENDIDA)

Para esta segunda parte, se ha creado el método explotar que devuelve la dirección que debe seguir el personaje en función de la celda en la que se encuentra. Para ello se guardan la posición actual y la siguiente y se comprueba para la celda actual cual es el mejor vecino al que ir en función del valor de Q guardado en la tabla.

3.1. Get_Best_Action

En este método, para escoger el mejor valor de Q, inicializamos una variable a menos infinito de forma que mientras recorremos todos los valores de esa celda, si encuentra uno mayor se actualiza y se guarda la acción que se está mirando en la tabla, para así devolver la acción con mejor Q y que el personaje se mueva en esa dirección.

```
 for (int i = 0; i < 4; i++)
{
       if (tableQ[(int)idx, i] > bestQ)
       {
              bestQ = tableQ[idx, i];
              best_action = i;
        }
}
```

4. MOVER AL PERSONAJE

Para implementar tanto el aprendizaje como la explotación se ha usado en el método *GetNextMove()*, heredado de *AbstractPathMind*, una condición de forma que si no existe ningún fichero se crea la tabla de valores Q y se accede al método *aprendizajeQ()* que asignará los valores y creará el fichero, y si ya existe una tabla, lo que hace es ir al método *explotar()* para saber qué pasos debe dar el personaje.

5. RESULTADOS OBTENIDOS

Con todo esto, hemos logrado el objetivo de la práctica, haciendo uso del aprendizaje reforzado para conseguir que el personaje llegue a la meta por el camino más óptimo, se cual sea la disposición del escenario, y guardando la tabla de valores Q en el fichero “tableQ.txt”.
Por otro lado, según los valores de alpha y gamma o el número de veces que exploramos la tabla, los resultados pueden variar.