Agente no hay camino, se hace camino al explorar.

Un acercamiento al aprendizaje por refuerzo y al algoritmo Q-Learning

Alejandro José Muñoz Aranda
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
alemunara@alum.es

Mario Ruano Fernández
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
mruano@us.es

***Resumen*— Este trabajo tiene por objetivo conocer y analizar el rendimiento de dos algoritmos de aprendizaje por refuerzo, basados en el algoritmo Q-Learning, aplicados a un problema de inteligencia artificial de descubrimiento de caminos en tableros.**

Se ha llevado a cabo la implementación previa de diferentes clases en Python para representar la realidad del problema, el algoritmo Q-Learning y una variante de este, el cual entrena al agente con exploración y explotación. También se ha creado una interfaz de usuario para facilitar el estudio experimental y la personalización de los parámetros.

Tras el estudio, se ha llegado a la conclusión de que basarse en el conocimiento del agente en fases tempranas del entrenamiento no es del todo óptimo, pudiendo llegar el algoritmo a ser incapaz de terminar su ejecución. Por esta razón, se antoja compleja la labor de encontrar un equilibrio entre los parámetros que faciliten un periodo previo de exploración suficiente para poder explotar el conocimiento que se va adquiriendo en cada episodio en sucesivos entrenamientos.

Palabras Clave — Inteligencia Artificial, Aprendizaje Automático, Aprendizaje por refuerzo, Q-Learning.

I. INTRODUCCIÓN

La Inteligencia Artificial (IA) es la combinación de algoritmos con el objetivo de crear máquinas que presenten capacidades similares o cercanas a las del ser humano. Es un área de conocimiento que evoluciona, en constante mejora, y que forma parte de nuestro día a día cada vez con un mayor protagonismo [1].

Una de las ramas de la IA es el Machine Learning (ML), que tiene como objetivo conseguir que una máquina sea capaz de aprender por si misma a partir de una serie de patrones algorítmicos y una fuente de datos. Dentro del ámbito del ML existen muchos tipos de aprendizaje. En este trabajo se lleva cabo un acercamiento al aprendizaje por refuerzo, inspirado en la psicología conductista y que tiene como propósito determinar qué acciones debe realizar un "agente inteligente" para maximizar algún tipo de recompensa [2].

Existen diversos tipos de algoritmos de aprendizaje por refuerzo como Q-Learning, SARSA o Dynamic Programming. Aquí se ha trabajado con el primero de ellos. El algoritmo Q-Learning fue introducido por Watkins en 1989 y su objetivo es aprender qué acción realizar bajo diferentes circunstancias para maximizar una recompensa [3].

El problema que se presenta en este estudio, el cual se integra en la asignatura de Inteligencia Artificial de la titulación de Grado en Ingeniería del Software de la Universidad de Sevilla, consiste en encontrar el camino más corto entre dos casillas de un tablero. Esto implica trabajar con una serie de recompensas a lo largo del recorrido trazado. El camino finalmente escogido por el agente variará según las características del entorno que se le plantee, así como de una serie de parámetros.

Para resolver este problema, se ha realizado un estudio teórico previo, el cual se basa en un problema similar [4], en el que el tablero pasa a ser una casa, las casillas, habitaciones, y la condición de moverse de una habitación a otra viene dada por la existencia de una puerta entre ambas. No solo se ha realizado un estudio de este trabajo previo, sino que también se ha llevado a cabo una labor de investigación sobre la teoría, con el objetivo de conocer más sobre el Aprendizaje por refuerzo y el algoritmo Q-Learning. Estos resultados se reflejan en el Anexo I: marco teórico.

Por último, se han ejecutado las distintas fases propuestas, a través de las cuales se ha implementado el algoritmo Q-Learning y una variante de este, así como la integración de todo esto con una interfaz gráfica de usuario, para poder realizar un estudio experimental comparativo de manera más usable.

En la sección Preliminares se abordan una serie de conceptos teóricos básicos con el fin de entender el entorno del problema y el lenguaje utilizado. En la sección Metodología se presentan los procesos seguidos en cada una de las fases previas al estudio experimental. En la sección Resultados se exponen las pruebas realizadas, junto con las comparativas y análisis de ambos algoritmos. Por último, en la sección Conclusiones se recogen las valoraciones generales del trabajo.

II. PRELIMINARES

A. Aprendizaje por refuerzo

El Aprendizaje Automático o Machine Learning (ML) es una de las áreas que comprende la Inteligencia Artificial (IA). Los algoritmos usados en ML tienen como objetivo que las computadoras aprendan a tomar decisiones por ellas mismas, sin la necesidad de ser programadas explícitamente.

Podemos clasificar los algoritmos de ML dependiendo de las necesidades del problema o del contexto y entorno donde se desenvuelven. Las tres áreas principales son: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo 13[5].

En el aprendizaje supervisado, se genera un modelo predictivo basado en datos de entrada y salida. Sin embargo, en el aprendizaje no supervisado el modelo predictivo solo está basado en datos de entrada y estos no están clasificados. Por último, el aprendizaje por refuerzo consiste en un modelo de acción-recompensa, en el que un agente inteligente se ve recompensado (o penalizado) según la acción que realice y el entorno en el que se encuentre [6].

En este trabajo nos centramos en el aprendizaje por refuerzo y en concreto en el algoritmo Q-Learning.

B. Q-Learning

Q-Learning es un algoritmo "off-policy", es decir, aprende a partir de cualquier tipo de acción que lleve a cabo el agente. No es necesario seguir una política a priori. El objetivo de este algoritmo es aprender la mejor política que maximice la recompensa final.

Para llevar a cabo el estudio experimental, se ha utilizado dicho algoritmo y una variante de este, con el objetivo de realizar comparativas en el rendimiento.

A continuación, se definen algunos conceptos claves y se aclara la terminología utilizada en este trabajo, con el objetivo de entender este algoritmo y la metodología que se desarrolla:

- Agente: es el cerebro que toma las decisiones y aprende de ellas.
- Entorno: dominio en el que el agente existe y con el que interactúa.
- Estado: conjunto de todos los estados que definen el entorno.
- Acción: interacción del agente con el entorno, lo que se traduce en una recompensa o penalización como retroalimentación.
- Recompensa: es una señal representada por un número real que la política del entorno otorga al agente al pasar de un estado a otro.
- Política: norma que representa al entorno y cómo este está construido y modelado.

C. Programación orientada a objetos con Python

Python es uno de los lenguajes de programación más populares y extendidos en la actualidad. También es uno de los lenguajes más usados en el campo del Big Data o la Inteligencia Artificial.

En este trabajo se ha utilizado Python y el paradigma de la programación orientada a objetos para modular y resolver los problemas presentados, implementando los algoritmos de entrenamiento que se presentan en la metodología seguida.

D. Librerías de Python utilizadas

A continuación, se listan las librerías utilizadas en este trabajo, necesarias para el correcto funcionamiento de la ejecución de este, y una breve justificación de su uso:

- Numpy (*numpy*): se utiliza para la manipulación de matrices.
- Random (*random*): se utiliza para la generación de números aleatorios y para la elección aleatoria de elementos de una lista.
- Matplotlib (*pyplot*): se utiliza para creación de gráficas.
- Graphviz (*graphviz*): se utiliza para generar grafos. Se ha usado para representar el tablero.
- Tkinter (*tkinter*): se utiliza para crear la interfaz gráfica de usuario.
- PIL (*PIL*): se utiliza para leer imágenes png que se muestran en la interfaz de usuario.
- Timeit (*default_timer*): se utiliza para contabilizar la duración de la ejecución del algoritmo de entrenamiento.

E. Tutorial paso a paso

Para el desarrollo de este trabajo, principalmente para la primera fase, se ha seguido un trabajo práctico paso a paso en el que se explica el algoritmo Q-Learning [4].

Este ha sido el punto de partida de la implementación de código de este trabajo.

F. Características técnicas del equipo informático

La ejecución del estudio experimental y la toma de mediciones se ha realizado en un equipo informático con las siguientes características técnicas:

- Intel Core i7 3770 CPU 3.40GHz
- 16 GB RAM
- Windows 10 Home
- Python 3.7.6

III. METODOLOGÍA

En este apartado se presenta la implementación que se ha llevado a cabo para resolver el problema y las decisiones de diseño tomadas.

De manera general, se ha seguido una metodología de fases incrementales, en las que se han realizado diferentes implementaciones de dos algoritmos de entrenamiento, hasta concluir con un estudio experimental y comparativo del funcionamiento y rendimiento de estos.

Por último, se ha implementado una interfaz gráfica de usuario para permitir que los parámetros de ambos algoritmos sean personalizables y se puedan obtener tableros aleatorios de diferentes tamaños.

A. Fase 1

En esta primera fase, se ha realizado un acercamiento al tipo de problema con el que se va a tratar a lo largo de todas las demás fases de este trabajo.

A grandes rasgos, se tiene un tablero de *n* filas y *m* columnas, con *nm* casillas numeradas de manera única. Dada una casilla como estado inicial de partida, se pretende alcanzar una casilla objetivo mediante una secuencia de acciones, movimientos entre casillas, tratando de que el camino sea practicable y lo más corto posible.

Para modelar el problema, se ha implementado la clase *Board* (Tablero). Esta clase está constituida por el número de filas (*n*), el número de columnas (*m*), la casilla inicial (*initial_state*) y la casilla objetivo (*goal_state*).

Además, para representar las casillas se ha decidido utilizar dos diccionarios. A través de ellos se dispone de bidireccionalidad entre las claves y los valores. De este modo, uno de los diccionarios (*map_squares*) relaciona como clave la coordenada del tablero, mediante una tupla, con el número de casilla como valor, mientras que el otro diccionario (*map_states*) contiene la misma relación, pero a la inversa.

Por último, en esta clase también se encuentra la matriz de recompensas (*R*), a través de la cual se irá optimizando el "cerebro" del agente en cada episodio de entrenamiento.

En la fase 1 se plantea resolver un problema concreto, con una configuración de tablero dada de antemano. Por este motivo se ha decidido generar dos tipos de tableros:

- Tableros aleatorios: pensados para la implementación con interfaz gráfica de usuario. El usuario solo tiene que facilitar el número de filas y de columnas y la disposición de las casillas se genera de manera aleatoria.
- Tableros personalizados: para casos en los que se debe tener una configuración concreta, como en la fase 1 y fase 3. Se pasa una matriz que representa la disposición de las casillas.

Procedimiento init_R
Entrada:
- Un tablero B

Salidas:
- Una matriz de recompensas R

Algoritmo:
1. Para cada i de cero a tamaño(B)
   a. Para cada i de cero a tamaño(B)
      i. Si is_neighbor(casilla(i), casilla(j)):
         1. Si j es B.casilla_objetivo
            a. R[i,j] = 100
         2. Si no
            a. R[i,j] = 0
      ii. Entonces, si i = j e i es B.casilla_objetivo
         1. R[i,j] = 100
      iii. Si no
         1. R[i,j] = -100

Pseudocódigo 1. Procedimiento init_R

El cálculo de la matriz *R* se realiza a partir del tablero dado con el método *init_R*. Este consiste en ir recorriendo, mediante un bucle, las posiciones de dicha matriz, inicializada a 0, considerando que cada fila de la matriz corresponde a una casilla numerada del tablero. De la misma manera, se considera que las columnas corresponden a la acción de moverse desde dicha casilla a la que indica el índice de columna. Se evalúa si la columna determinada es casilla vecina o no. Esto es lo mismo que decir que si, desde una casilla, se puede realizar la acción de desplazarse hacia otra casilla.

En el Pseudocódigo 1 se observa cómo queda implementada la política de recompensas de la matriz *R*:

- La casilla destino es vecina y objetivo: 100
- La casilla destino es vecina: 0
- La casilla destino no es vecina: -100 (se utiliza dicho valor en lugar de -1, ya que, más adelante, se emplean penalizaciones con valor de -10, por lo que la representación de la no vecindad debe quedar especificada con un valor aún más pequeño, en este caso se ha elegido el valor -100).

Para este cometido de determinar la vecindad de una casilla se implementa un método auxiliar (*is_neighbor*) en la clase *Board*. Este calcula la distancia euclídea entre dos casillas dadas y determina, mediante una expresión booleana, si son casillas vecinas. Teniendo en cuenta las tuplas del diccionario *map_states* como coordenadas, para que dos casillas sean vecinas y se pueda transitar de una a otra, la distancia entre ambas deberá ser 1 (casillas horizontales y verticales) o raíz cuadrada de 2 (casillas diagonales). Cualquier otra distancia significa que no son casillas vecinas.

Por último, para representar el algoritmo Q-Learning, se ha implementado la clase *QLearningAlgorithm*. Esta clase recibe un tablero (*board*), una matriz de recompensas *R* y dos parámetros: el número de entrenamientos a realizar (*n_training*) y el valor de *gamma* (en el rango de 0 a 1).

Procedimiento training
Entrada:
- Un tablero B
- Una matriz de recompensas R
- Una matriz a cero Q
- El número de t entrenamientos
- El parámetro de aprendizaje gamma g

Salidas:
- Una matriz Q entrenada

Algoritmo:
1. Para cada i de cero a t
   a. Estado_actual = aleatorio(casillas)
   b. Mientras Verdad
      i. Acciones_posibles = []
      ii. Acciones = R[Estado_actual]
      iii. Para cada i desde cero a tamaño(B)
         1. Si Acciones[i] >= 0:
            a. Acciones_posibles[i] = Acciones[i]
      iv. Estado_siguiente = aleatorio(Acciones_posibles)
      v. Acción = Estado_siguiente
      vi. Q[Estado_actual, Acción] = R[Estado_actual, Acción] + g * máximo(Q[Estado_siguiente])
      vii. Estado_actual = Estado_siguiente
      viii. Si Estado_actual es B.casilla_objetivo
         1. Terminar ejecución

Pseudocódigo 2. Procedimiento training de QLearningAlgorithm

Dicha clase también cuenta internamente con la matriz *Q*, la cual representa al "cerebro" del agente, y que será el objeto de optimización tras los entrenamientos ejecutados por el

algoritmo de aprendizaje que se implementa en el método *training*.

El parámetro de aprendizaje *gamma*, también conocido como factor de descuento, determina la forma en la que se realizarán los entrenamientos y las actualizaciones de *Q*. Cuando *gamma* toma valores cercanos a 0, entonces se van a considerar recompensas inmediatas a la posición del agente. Por el contrario, si *gamma* toma valores que tienden a 1, entonces la optimización de *Q* tenderá a recompensas mayores a futuro.

Como se ha indicado anteriormente, a través del método *training* de *QLearningAlgorithm* se lleva a cabo el entrenamiento de la matriz *Q*. En este algoritmo descrito en el Pseudocódigo 2 se utiliza la expresión matemática (1) para actualizar dicha matriz:

$$Q(estado\_actual, acción) = R(\text{estado\_actual, acción}) + \gamma * max(Q(estado\_siguiente)) \quad (1)$$

Al terminar el entrenamiento, con el método *get_path*, que también está implementado en la clase *QLearningAlgorithm*, se consigue la secuencia de casillas que sigue el agente desde la casilla inicial a la casilla objetivo.

Procedimiento get_path
Entrada:
- Un tablero B
- Una matriz Q entrenada
- Una lista path de casillas con B.casilla_inicial

Salidas:
- Una lista path con la secuencia de casillas a seguir

Algoritmo:
1. Estado_actual = B.casilla_inicial
2. Mientras Estado_actual no sea B.casilla_objetivo
   a. Acciones = Q[Estado_actual]
   b. Para cada i desde cero a longitud(path)
      i. Acciones[path[i]] = 0
   c. Maximo_valor = max(Acciones)
   d. Si Maximo_valor <= 0
      i. Terminar la ejecución
   e. Estado_actual = aleatorio(donde(Acciones es Maximo_valor))
   f. Añadir(Estado_actual, path)
3. Retornar path

Pseudocódigo 3. Procedimiento get_path de QLearningAlgorithm

Como se observa en el Pseudocódigo 3 en el punto 2.d., si el valor máximo que indica qué acción realizar en un estado actual es 0, significa que toda la fila de la matriz *Q* para dicho estado está a 0. Esto quiere decir que se obtendrá un resultado no realista y una representación inexacta de las características del problema.

Por esta razón, tal como se observa en el punto 2.d. del pseudocódigo, se ha evaluado dicha cuestión con una condición. En caso de que el valor máximo de la fila sea 0, se terminará la ejecución del algoritmo que obtiene el camino, al tiempo que se informa al usuario por la interfaz gráfica.

De esta forma se consigue implementar todo lo necesario para dar solución al enunciado propuesto para la fase 1.

B. Fase 2

La metodología seguida en esta segunda fase ha sido similar a la descrita en el apartado de la fase 1.

Se ha utilizado el paradigma de la programación orientada a objetos y se ha implementado una nueva clase con nombre *QLearningVariantAlgorith*, la cual ha heredado de la clase *QLearningAlgorithm*. De esta forma se ha podido modificar el algoritmo de entrenamiento originario, quedando el resto de los atributos y métodos de la misma forma.

Esta clase introduce dos nuevos parámetros, *epsilon* y *alpha*, con valores entre 0 y 1, los cuales entran en juego variando el funcionamiento del algoritmo Q-Learning.

Por cada episodio de un entrenamiento, el algoritmo genera un número aleatorio *k*, comprendido entre 0 y 1, el cual se compara con el valor de *epsilon*. Si *k* es menor que *epsilon*, entonces el funcionamiento del algoritmo no cambia, actuando de la misma manera se implementa en el método *training* de la clase *QLearningAlgorithm*. En el caso de *k* sea mayor que *epsilon*, entonces el siguiente estado (casilla hacia la que se moverá el agente) no será elegido de forma aleatoria entre las posibles acciones a realizar, sino que se tomará el máximo valor en *Q* para el estado actual, es decir, el valor máximo de la fila que corresponde a la casilla actual en la que se encuentra el agente antes de realizar el movimiento.

Como se puede observar en el Pseudocódigo 4 en el punto 1.c.vii. tras cada movimiento se actualiza el parámetro *epsilon*. En dicha actualización entra en juego el parámetro *alpha*, el cual puede considerarse un factor de descuento de *epsilon*.

Procedimiento training "variado"
Entrada:
- Un tablero B
- Una matriz de recompensas R
- Una matriz a cero Q
- El número de t entrenamientos
- El parámetro de aprendizaje gamma g
- El parámetro epsilon E
- El parámetro alfa a

Salidas:
- Una matriz Q entrenada

Algoritmo:
1. Para cada i de cero a t
   a. Estado_actual = aleatorio(casillas)
   b. e = E
   c. Mientras Verdad
      i. Acciones_posibles = []
      ii. Acciones = R[Estado_actual]
      iii. Para cada i desde cero a tamaño(B)
         1. Si Acciones[i] >= 0:
            a. Acciones_posibles[i] = Acciones[i]
      iv. k = aleatorio()
      v. Si k > e:
         1. Maximo_valor = max(Q[Estado_actual])
         2. Estado_siguiente=aleatorio(donde(Q[Estado_actual] sea Maximo_valor))
      vi. Si no
         1. Estado_siguiente = aleatorio(Acciones_posibles)
      vii. e = e * a
      viii. Acción = Estado_siguiente
      ix. Q[Estado_actual, Acción] = R[Estado_actual, Acción] + g * máximo(Q[Estado_siguiente])
      x. Estado_actual = Estado_siguiente
      xi. Si Estado_actual es B.casilla_objetivo
         1. Terminar ejecución

Pseudocódigo 4. Procedimiento training de QLearningVariantAlgorithm

Por lo general, *epsilon* va decreciendo cada vez que se multiplica por *alpha*. Por este motivo, la probabilidad de que *k* sea mayor que *epsilon* cada vez es mayor.

Al terminar el entrenamiento, antes de comenzar uno nuevo, el parámetro *epsilon* vuelve a tomar su valor originario. Esto puede observarse en el Pseudocódigo 4 en el punto 1.b.

De esta manera se ha resuelto la fase 2. En este punto, se cuenta con todo lo necesario para poder ejecutar la fase 3 y el estudio experimental comparativo para ambos algoritmos.

C. *Fase 3*

Esta fase corresponde a la realización del estudio experimental en el que se comparan ambos algoritmos de entrenamiento que se han implementado previamente en las fases anteriores.

Antes de poner en práctica todo el trabajo previo, se necesita configurar un entorno en el que aplicar los algoritmos. Para ello, se ha creado un tablero de mayores dimensiones (diez filas y diez columnas) y se han establecido recompensas intermedias y penalizaciones. De esta forma, el escenario por el que se mueve el agente adquiere más complejidad de la que tiene un tablero simple que se basa solo en el criterio de vecindad entre casillas.

En la Fig. 1. se puede observar la configuración del tablero y la disposición de sus cien casillas.

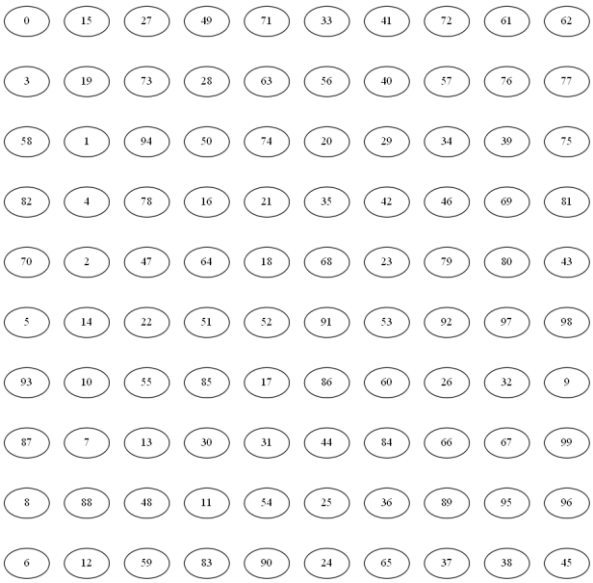

Fig. 1. Tablero utilizado en la fase 3.

Por último, se establece una configuración concreta de recompensas, quedando la matriz R de la siguiente forma:

- Transitar a una casilla vecina: 0
- Transitar a una casilla no vecina: -100
- Transitar a la casilla objetivo: 100
- Transitar de casilla 0 a 19: 30
- Transitar de casilla 19 a 58: 30
- Transitar de casilla 70 a 5: 30
- Transitar de casilla 10 a 13: 30
- Transitar de casilla 90 a 24: 80
- Transitar de casilla 38 a 96: 30
- Transitar de casilla 27 a 49: 30
- Transitar de casilla 33 a 40: 30
- Transitar de casilla 34 a 69: 30
- Transitar de casilla 43 a 98: 30
- Transitar de casilla 18 a 91: -10
- Transitar de casilla 58 a 3: -10
- Transitar de casilla 9 a 99: -10
- Transitar de casilla 32 a 99: -10
- Transitar de casilla 67 a 99: -10

Finalmente se va a considerar la casilla inicial como la casilla 0 y la final como la 99. Por tanto, basándonos en la configuración de la matriz *R*, se observa que el acceso a la casilla 99 está bastante restringido, tal como se puede ver en la Fig. 2.

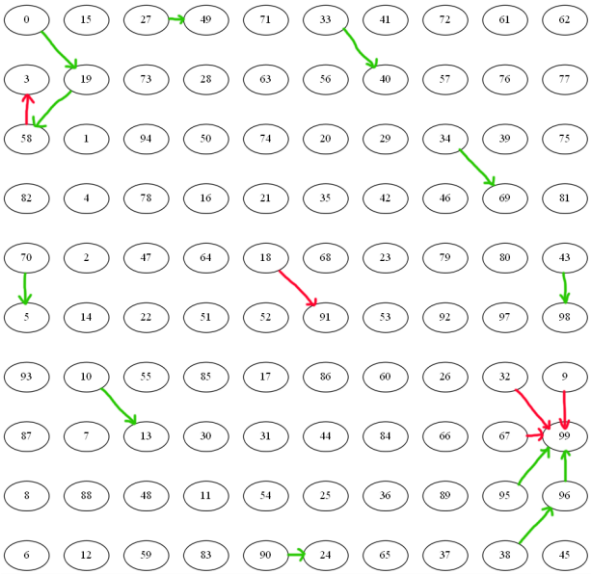

Fig. 2. Tablero utilizado en la fase 3 con las recompensas intermedias en verde y penalizaciones en rojo.

Con esta configuración se ha llevado a cabo el estudio experimental comparativo de algoritmos de entrenamiento.

D. *Integración con interfaz gráfica de usuario*

Para llevar a cabo la integración de los algoritmos y los problemas planteados con una interfaz gráfica, de manera que los usuarios puedan personalizar los parámetros, se ha utilizado la librería Tkinter.

A continuación, se presentan las diferentes vistas que se han creado, explicando brevemente su funcionalidad.

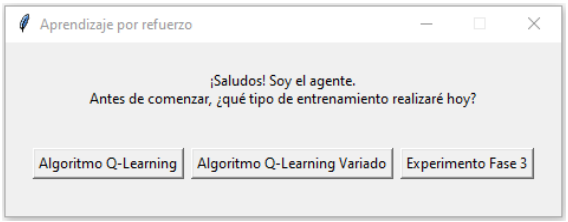

Fig. 3. Vista de inicio de la interfaz de usuario.

En la Fig. 3. se selecciona el tipo de algoritmo que se aplicará para el entrenamiento del agente, o bien se puede seleccionar el experimento de la fase 3.

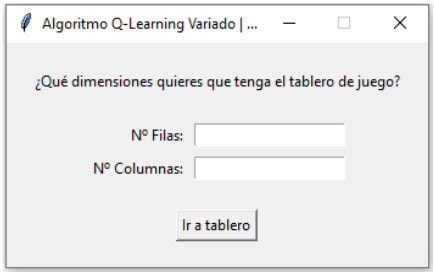

Fig. 4. Vista de personalización de tablero.

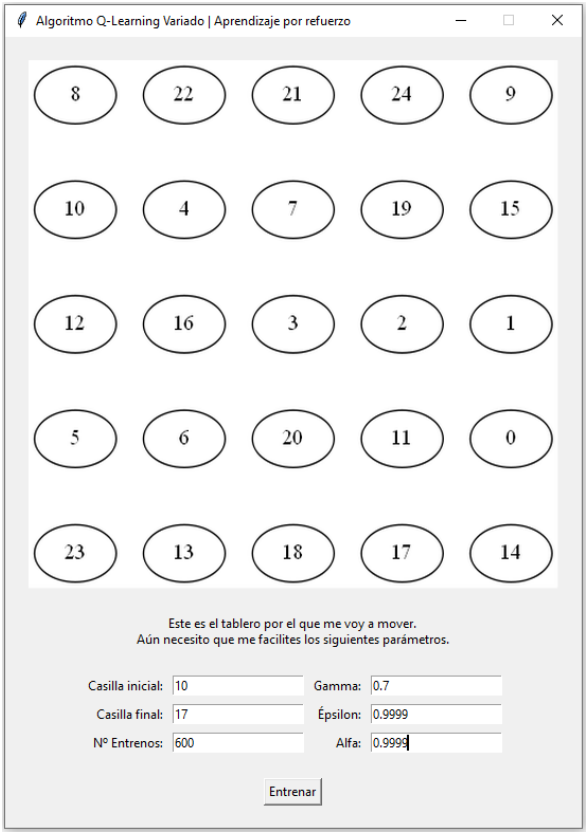

Fig. 5. Vista de especificación de parámetros del problema.

Una vez que el usuario selecciona alguno de los dos tipos de algoritmos, accede a una vista como la de la Fig. 4. donde personaliza el número de filas y de columnas que tendrá el tablero para, posteriormente, llegar la vista de la Fig. 5. donde termina de especificar el resto de los parámetros necesarios para llevar a cabo el entrenamiento del agente.

Tras finalizar los entrenamientos, se puede obtener el camino resultante, como en la vista de la Fig. 6. y la gráfica de rendimiento, como en la vista de la Fig. 7.

En la siguiente sección se explican los experimentos realizados y las comparativas de aplicar ambos algoritmos al problema de la fase 3.

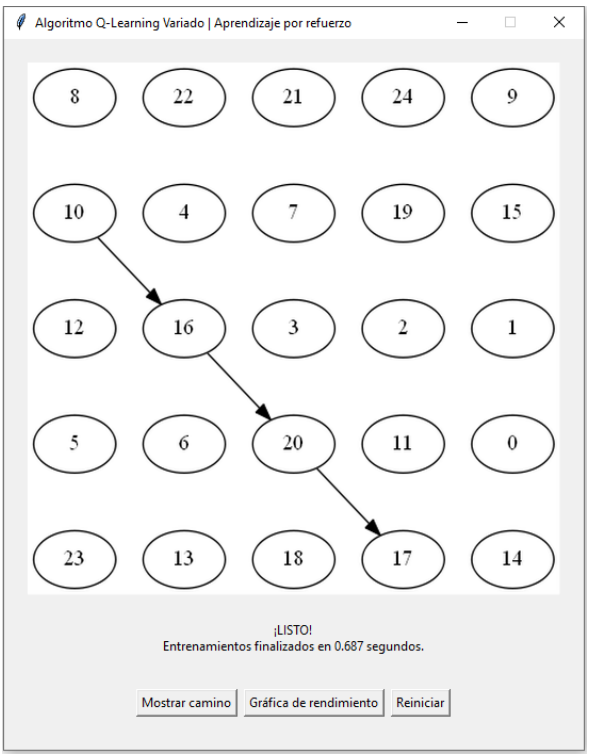

Fig. 6. Vista con el camino solución.

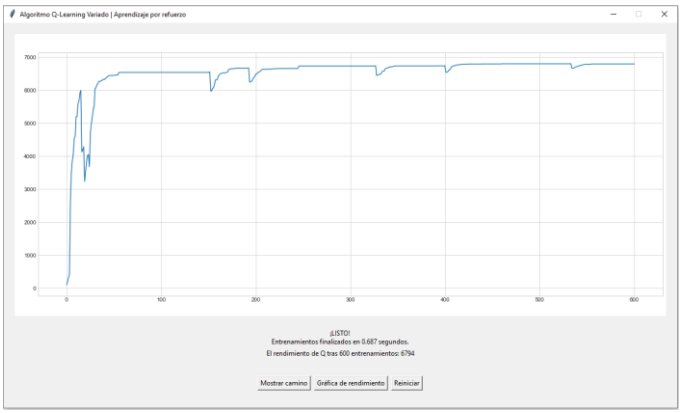

Fig. 7. Vista de gráfica de rendimiento de un proceso de entrenamiento.

IV. Resultados

Los experimentos realizados han consistido en la ejecución de ambos algoritmos para un entorno de problema conocido, con una configuración previa del tablero como la que se expone en el apartado de la fase 3. En concreto, un tablero de tamaño 10x10 con casilla inicial 0 y casilla objetivo 99.

El objetivo ha sido medir el rendimiento de la matriz *Q* entrenada para una configuración determinada de parámetros y comparar el comportamiento de ambos algoritmos, comparando también el coste temporal de las distintas ejecuciones. Para cada configuración, se recogen los resultados en tablas que se adjunta en el Anexo II: Tablas de resultados.

Para calcular el rendimiento de cada entrenamiento, se suman todos los elementos de la matriz *Q*, se divide por el mayor de ellos y se multiplica por 100, tal como se indica en la expresión matemática (2).

$$(\Sigma Q(n,m) / max\{Q(n,m)\}) * 100 \qquad (2)$$

En primer lugar, se ha tratado de aproximar el error que se comete en la medición del rendimiento. Cuando el valor del rendimiento está en torno a 10.000, existe una desviación de ±500 puntos. Según va aumentando el rendimiento, también aumenta la desviación, llegando a alcanzarse una desviación de ±1000 para valores alrededor de los 20.000 puntos de rendimiento. Aunque estas desviaciones solo tienen lugar en los primeros entrenamientos de una misma ejecución, según van progresando y acumulando entrenamientos, el rendimiento se va estabilizando y la desviación también disminuye.

Se ha comparado el rendimiento para diferentes valores de los parámetros *n_training*, *gamma*, *epsilon* y *alpha*.

A. *El impacto de gamma*

Antes de comenzar las comparativas entre ambos algoritmos se ha analizado el impacto en los resultados que genera la variación del parámetro *gamma*, también conocido como factor de aprendizaje. Como se indicó anteriormente, para valores cercanos a 0, *gamma* hace que el entrenamiento tienda a optimizar para recompensas inmediatas, mientras que para valores cercanos a 1 se optimiza para recompensas mayores a futuro.

Esto puede observarse en las Tablas 1, 2, 3 y 4, frente a las Tablas 5, 6, 7, y 8 en el caso del algoritmo Q-Learning. Para un valor medio de *gamma* (0.5), el rendimiento es bastante más bajo, oscilando entre los 7500-8000 puntos, frente al rendimiento para un *gamma* cercano a 1 (0.85) que oscila entre los 30000-32000 puntos. Además, vemos que, para el caso de 1500 entrenamientos y *gamma* medio, aunque el algoritmo consigue encontrar un camino para la configuración de la Tabla 3, tal como se refleja en la Fig. 8., no es óptimo, dado que se realizan rodeos hasta alcanzar la casilla objetivo. Para *gamma* con valor de 0.85, sí se obtiene el camino óptimo.

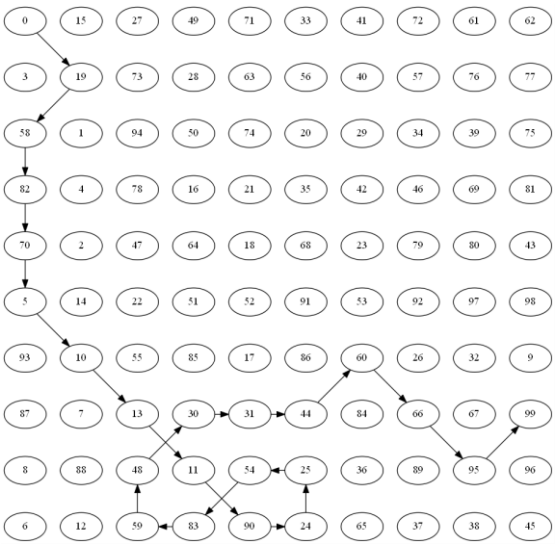

Fig. 8. Camino resuelto con Q-Learning, 1500 entrenamientos y gamma a 0.5

Este comportamiento es similar en el caso del algoritmo Q-Learning variado.

Otro aspecto a tener en cuenta es la variación del rendimiento respecto al incremento del número de entrenamientos y los cambios en *gamma*. En las Tablas 9, 10, 11 y 12 se tienen distintas configuraciones en el número de entrenamientos para Q-Learning variado y con *gamma* medio (0.5). Por otro lado, en las Tablas 13, 14, 15 y 16, se aumenta *gamma* a un valor superior (0.85). Como puede verse en la Fig. 9., para ambas situaciones del factor de aprendizaje, aunque el rendimiento parece decrecer conforme aumenta el número de entrenamientos, la caída en el rendimiento es menor que para una configuración experimental similar en el algoritmo Q-Learning, tal como se observa en la Fig. 10.

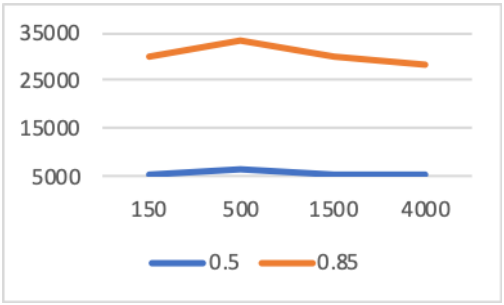

Fig. 9. Rendimientos para Q-Learning variado con gamma a 0.5 y 0.85.

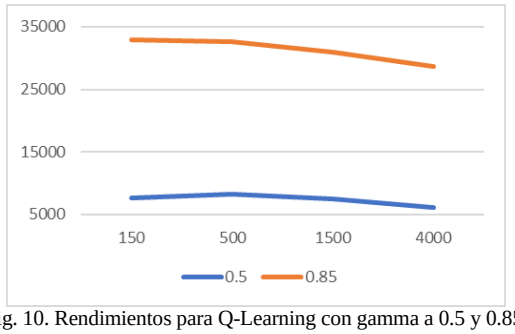

Fig. 10. Rendimientos para Q-Learning con gamma a 0.5 y 0.85.

En definitiva, el valor de *gamma* influye directamente en el rendimiento de ambos algoritmos y con un impacto considerable, siendo necesario un valor cercano a 1 para que el agente no se pierda o consiga optimizar el camino, como en el caso de la Fig. 11. La gran diferencia de puntuación es debida a que *gamma* es un factor de reducción. Cuando se tiene valores pequeños incrementa muy poco los valores que se van actualizando en la matriz *Q*, en contraposición con valores altos, donde el crecimiento se dispara, debido a que la reducción de la recompensa es mucho menor.

En las Tablas 17 y 18 se recogen configuraciones para *gamma* en torno a 0.2. Esto es insuficiente para dar un camino óptimo y el agente tiende a perderse, como ocurre en la configuración de la Tabla 19.

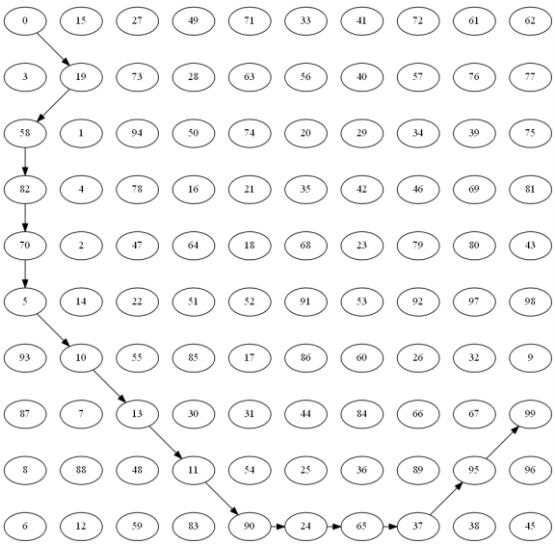

Fig. 11. Camino resulto con Q-Learning, 4000 entrenamientos y gamm a 0.85

Por último, se han localizado puntos de estabilización del rendimiento a partir de los 5000-6000 entrenamientos. Esto puede verse reflejado en las gráficas de las Fig. 12. y 13.

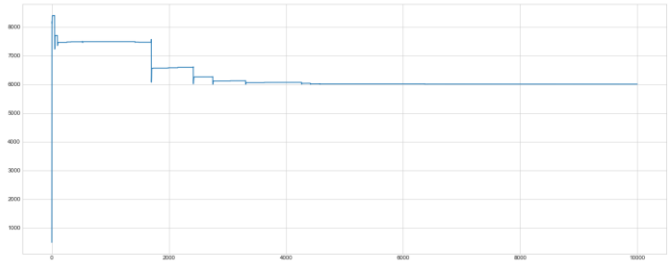

Fig. 12. Rendimiento para Q-Learning con gamma a 0.5 y 10000 entrenos.

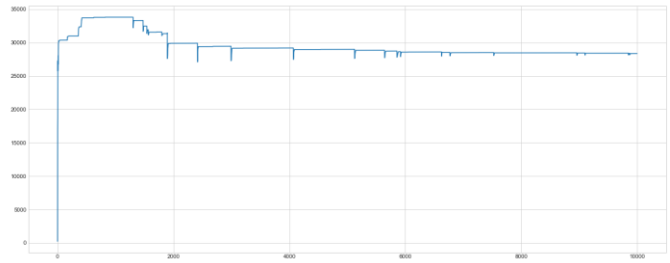

Fig. 13. Rendimiento para Q-Learning con gamma a 0.85 y 10000 entrenos.

Estas gráficas de rendimiento han sido obtenidas para configuraciones recogidas en las Tablas 21, 22, 23 y 24.

B. *Coste temporal de la ejecución*

Otro de los aspectos para tener en cuenta en las mediciones ha sido el tiempo de duración de la ejecución de los algoritmos. En concreto, se ha medido el tiempo que tarda en completarse le método *training*.

Tal y como puede observarse en la Fig. 14., no hay grandes diferencias en el consumo de tiempo cuando el factor de aprendizaje *gamma* cambia. Por otro lado, es lógico que, al aumentar el número de entrenamientos, el algoritmo tarde más terminar.

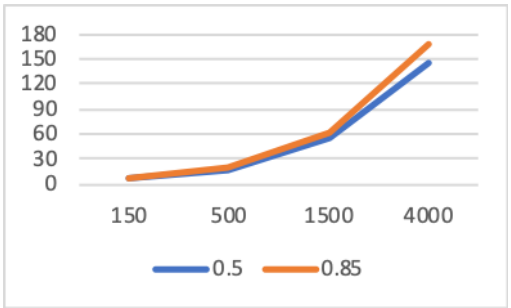

Fig. 14. Tiempos de ejecución, en segundos, para Q-Learning con gamma a 0.5 y 0.85.

C. *Épsilon y alfa: exploración y explotación*

A la hora de especificar los parámetros para ejecutar varias veces el algoritmo Q-Learning variado, se ha detectado que el algoritmo es incapaz de terminar si la matriz *Q* no tiene suficiente información y se comienza, por mera probabilidad en aumento, con la fase de explotación. Por ejemplo, esto pasa para una configuración como la de la Tabla 20.

Aquí entran en juego dos momentos o fases en la ejecución. Que ocurra una u otra, como ya se comentó con anterioridad, dependerá de *epsilon* y un valor aleatorio *k*, el cual también está comprendido entre 0 y 1.

Para el caso del tablero que se ha utilizado en este trabajo experimental, se requiere de una disminución del valor de *epsilon* lo suficientemente lenta como para que la exploración previa sea capaz de facilitar una matriz *Q* óptima para comenzar una fase de explotación.

Por este motivo, se ha detectado que tanto *epsilon* como *alfa* deben tender a 1 necesariamente al inicio de la ejecución, para que el algoritmo sea capaz de finalizar y para que los resultados devueltos puedan dar lugar a un camino, aunque no necesariamente debe ser el óptimo que, como se ha visto anteriormente, depende de otros parámetros. Se recuerda que *alfa* es el factor de reducción de *epsilon*.

Los valores utilizados en este trabajo han sido de 0.9999 tanto para e*psilon* como para *alfa*.

D. *Comparando Q-Learning con Q-Learning variado*

Después de tomar las mediciones que se registran en las tablas del Anexo II: Tablas de resultados y de valorar y analizar los aspectos relativos a los parámetros que entran en juego a la hora de ejecutar los diferentes algoritmos, se ha pasado a un análisis comparativo de ambos.

Para una misma configuración en cuanto al número de entrenamientos y variando únicamente el valor de *gamma*, se contempla bastante similitud entre el rendimiento de Q-Learning y Q-Learning variado. Aunque ya se ha indicado que en ambos casos los rendimientos suelen decrecer antes de estabilizarse, esta caída parece más acuciada en el caso de un *gamma* alto (0.85), tal como se observa en la Fig. 15.

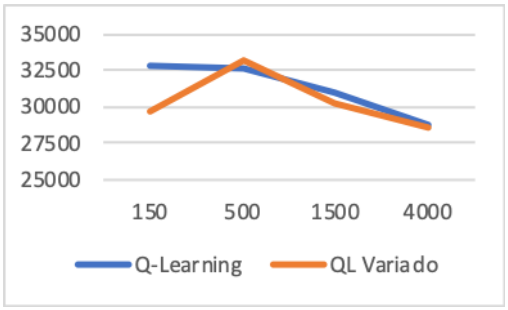

Fig. 15. Rendimientos de ambos algoritmos con gamma a 0.85.

También se puede contemplar que, a diferencia de lo que ocurre con un *gamma* medio, como se ve en la Fig. 16., el rendimiento es mucho más similar entre los algoritmos.

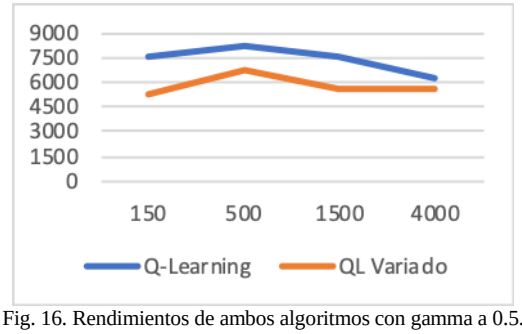

Fig. 16. Rendimientos de ambos algoritmos con gamma a 0.5.

En base a esto, y con la visión de los análisis y comparativas realizados hasta este punto, se podría pensar que no existe ninguna ventaja en utilizar Q-Learning variado para resolver la problemática expuesta. Sin embargo, si en lugar de atender a la puntuación del rendimiento de la matriz *Q*, el cual es muy similar para ambos casos, se pone el foco en el tiempo

de ejecución de los algoritmos, se tienen resultados mucho más interesantes.

Concretamente, partiendo del hecho de que Q-Learning variado pone en práctica la fase inicial de exploración para, a continuación, continuar entrenando al agente explotando el conocimiento que ya ha adquirido previamente, se puede deducir que el costo en términos de tiempo es inferior que en el caso de Q-Learning, ya que aquí siempre se lleva a cabo exploración basándose en la aleatoriedad de las acciones posibles, dada una casilla actual.

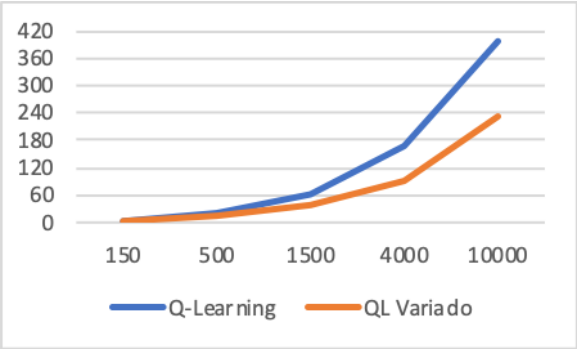

Fig. 17. Tiempos de ejecución, en segundos, para ambos algoritmos con gamma a 0.85.

En efecto, y como cierre de la labor experimental, en la Fig. 17. se puede observar una gran diferencia de tiempos cuando el número de entrenamientos comienza a crecer mucho. Aunque alrededor de los 10000 entrenamientos el tiempo de ejecución de Q-Learning variado es casi un 50% inferior al de Q-Learning, ya a partir de los 500 entrenamientos comienza a notarse el efecto de la fase de explotación, sobre la única fase de exploración que se realiza en el algoritmo Q-Learning.

V. Conclusiones

Este trabajo ha tenido por objetivo la comparativa del rendimiento y beneficios de utilizar un algoritmo de entrenamiento como Q-Learning o su variación para resolver problemas de exploración de caminos. Se han implementado y se han realizado pruebas, constatando que el funcionamiento era correcto y estaba alineado con las indicaciones teóricas.

Tras la fase de experimentación y la ejecución de diferentes configuraciones, se ha llegado a la conclusión de que el rendimiento en ambos algoritmos es bastante más alto cuando se tiene un factor de aprendizaje cercano a 1, en especial en tableros de una complejidad alta.

Aunque a priori pareciera que ambos algoritmos actúan de una forma similar, si se atiende únicamente a la variable de puntos de rendimiento, cuando el número de entrenamientos crece se ven realmente los beneficios de aplicar exploración-explotación si se pone el foco en la duración temporal del entrenamiento. Sin embargo, se puede añadir a esta idea que existe un importante problema ligado a las características de cada situación o entorno de aprendizaje: el equilibrio entre parámetros que "controlan" el paso de exploración a explotación.

Por tanto, una mejora futura o una posible vía para continuar este trabajo podría consistir en investigar el equilibrio entre el número de entrenamientos bajo exploración frente al número de entrenamientos bajo explotación.

VI. Anexo I: marco teórico

En Inteligencia Artificial, el aprendizaje por refuerzo es un campo del Aprendizaje Automático que, mediante algoritmos, consigue que un agente aprenda sobre el entorno que le rodea basándose en una estrategia de recompensas y penalizaciones. Esto no consiste en dar indicaciones a un agente inteligente de lo que tiene que hacer o de las acciones que debe realizar, sino que este experimenta libremente diferentes acciones permitidas en una política, bajo unas normas, que le llevan a distintas recompensas [7].

En el campo del Machine Learning existen otros tipos de aprendizajes como el aprendizaje supervisado o el no supervisado. En el aprendizaje supervisado, a partir de datos de entrada y salida, se crea un modelo predictivo que analiza y clasifica nuevas situaciones para obtener la salida adecuada en casos futuros, pero siempre dentro de un entorno explorado. En entornos no explorados este tipo de aprendizaje no es óptimo, ya que clasifica a partir de los datos que conoce [7]. Algunos ejemplos de algoritmos de aprendizaje supervisado son los árboles de decisión o la clasificación de Naïve Bayes.

El no Supervisado tiene carácter exploratorio y se da cuando solo se dispone de datos de entrada. Estos datos se analizan para intentar encontrar algún tipo de organización o patrón que pueda estar repitiéndose. Algunos ejemplos de algoritmos de aprendizaje no supervisado son los problemas de clustering o el perfilado (profiling).

A. *El aprendizaje por refuerzo*

A medio camino se encuentra el aprendizaje por refuerzo (RL). Algunos ejemplos de algoritmos son Q-Learning, SARSA o el método de Montecarlo [8].

El principal elemento del aprendizaje por refuerzo es el agente, que es el cerebro del algoritmo y el encargado de interactuar con el entorno para maximizar a largo plazo la recompensa obtenida. Cada problema es diferente y estará modelado de una manera distinta para poder representar el contexto y el entorno con el que el agente interactúa [9].

Por otro lado, la representación de las normas y las particularidades del entorno quedan marcadas en una política. Esta define la forma en la que el agente se puede comportar en cada momento, es decir, consiste en un mapeo a la acción que puede tomar el agente dado un estado específico. Sobre los tipos de políticas se tratará más adelante.

Cada acción que realiza el agente tiene asociada una recompensa. La función de valor indica lo bueno que es a largo plazo realizar una acción desde un estado concreto. Esta función retorna la mayor recompensa que un agente puede esperar desde un estado o realizando una acción concreta. La recompensa otorga valor a cada acción y el agente escogerá aquella acción que le otorgue más valor y no mayor recompensa, ya que son estas acciones la que le otorgan mayor recompensa a largo plazo.

Por último, el modelo del entorno imita el comportamiento del medio ambiente en el que se mueve el agente. Los modelos se utilizan para planificar las acciones. Los métodos que usan estos modelos se conocen como "model-based" y los métodos basados en la prueba y error se conocen como "model-free" [7].

En aprendizaje por refuerzo, el agente puede usar el conocimiento generado por el conjunto de acciones que ya

realizó para decidir una nueva acción futura. Esto se conoce como explotación. Pero para ello el agente primero debe descubrir el efecto de tomar determinadas acciones, lo que le genera conocimiento sobre su entorno [10].

Estas dos formas de actuar y de tomar decisiones deben estar correctamente equilibradas para conseguir un aprendizaje óptimo. Abusar de cualquiera de ellas exclusivamente lleva a resultados poco optimizados o a la incapacidad de terminar con la ejecución de los algoritmos. Encontrar el equilibrio entre la exploración de lo desconocido y explotación de los conocimientos adquiridos sigue siendo un dilema hoy en día al que no se le ha encontrado una solución definitiva [7].

Existen políticas que tienen como objetivo equilibrar la exploración y la explotación. El algoritmo epsilon-greedy o ϵ-greedy es uno de los más utilizados en aprendizaje por refuerzo, ya que su implementación es sencilla y obtiene buenos resultados. Su funcionamiento puede explicarse de manera simple. El agente escoge una acción de forma aleatoria con una probabilidad ϵ o se escoge la opción que maximiza la recompensa para el resto de las probabilidades. El principal problema de estos algoritmos greedy es que aprendizaje es lento. Esto se debe a que la minimización del "regret" es lineal. El "regret" se puede definir como el coste al que se renunciando al no seguir la política óptima en cada instante [11].

Como se ha expuesto anteriormente, otro concepto clave en aprendizaje automático son las políticas o "policy". Los algoritmos usados para el aprendizaje pueden ser [9] [12]:

- "On-policy": pretenden mejorar la política usada para tomar decisiones futuras.
- "Off-policy": la política de comportamiento está separada de la política que se quiere mejorar. El algoritmo Q-Learning se considera de este tipo.

Los Procesos de Decisión de Márkov (MDP) [13] son la idealización del aprendizaje por refuerzo. Describen el entorno en el que se desarrolla el aprendizaje. Un proceso de Márkov es un fenómeno aleatorio que depende del tiempo y para el cual se cumple la propiedad de Márkov, la cual viene dada por la expresión matemática (3) [14]. Esta se refiere a la propiedad de ciertos procesos estocásticos (concepto matemático que sirve para usar magnitudes aleatorias que varían con el tiempo [15]) en la que la distribución de probabilidad del valor futuro de una variable aleatoria depende únicamente de su valor presente, sin tener en cuenta el pasado de esta variable. La idea de los MDP es mapear estados y acciones en diferentes estados, y calcular así la acción óptima en cada estado [16].

$$\mathbb{P}\left[S_{t+1} \middle| S_t\right] = \mathbb{P}\left[S_{t+1} \middle| S_1, \ldots, S_t\right] \qquad (3)$$

B. Aplicaciones

El aprendizaje por refuerzo está presente más de lo que se puede llegar a pensar en el día a día de la sociedad actual y se utiliza en muchos ámbitos profesionales, aunque sigue en continuo desarrollo y evolución. Puede observarse en sistemas de navegación de robots, drones o incluso en coches autónomos y en la personalización del comercio electrónico ajustando las ofertas a cada cliente. También hay importantes aplicaciones en el campo de la gestión de recursos, como el uso de sistemas de enfriamiento, en el ámbito de los videojuegos e incluso en tratamientos médicos a medida recomendando medicamentos y dosis según las características del cuadro médico de un paciente [17].

En el mundo de los videojuegos, la IA se emplea, entre otras muchas cosas, para diseñar el comportamiento de los "Non Player Characters" (NPC). Estos son los elementos o personajes que aparecen en los videojuegos, que tienen un comportamiento determinado y que no dependen del control de un ser humano, considerándose elementos autónomos. El objetivo principal de los NPC es el de plantear un verdadero reto al jugador, que fomente e incentive el entretenimiento de las personas que consumen videojuegos. Por ello, crear una inteligencia artificial capaz de derrotar al humano o que se acerque a la capacidad resolutiva, deductiva y creativa de este es el foco principal [18]. Implementar algoritmos que permitan el aprendizaje de estos agentes conforme se repiten acciones o casuísticas, puede ayudar a optimizar comportamientos o adaptarlos a la forma de actuar de los humanos. Esto es algo que podría tener aplicación en el mundo de los videojuegos multijugador-masivos online, dado el gran flujo constante de información y datos que puede retroalimentar a los sistemas, entrenándolos y optimizándolos.

Esta aplicación de la inteligencia artificial en torno a los NPC es una entre tantas aplicaciones en el mundo de los videojuegos. Son muchas las empresas más punteras de tecnología las que están optando por invertir en investigación aplicada en el desarrollo de videojuegos. Un ejemplo de esto es la inteligencia artificial Angelina, creada por Michael Cook en 2011. Esta IA es capaz de crear juegos nuevos y originales a partir de información como imágenes o textos [18] [19].

Una de las empresas más importantes en este ámbito es DeepMind. Fundada en 2010 por los investigadores Demis Hassabis, Shane Legg y Mustafa Suleyman. Estos investigadores de IA ganaron una gran reputación en muy poco tiempo. En 2014 Google adquirió la empresa renombrándola como Google DeepMind. Esta empresa ha llevado a cabo grandes logros en el ámbito de la inteligencia artificial [20] [21] [22].

- En 2014 desarrollaron un software a partir de aprendizaje por refuerzo (Deep Q-Learning) capaz de aprender a jugar a los 57 juegos de la Atari 2600. En todos ellos se consiguió alcanzar una puntuación mayor que la media humana.
- En 2015 lanzaron el proyecto AlphaGo, en el cual consiguieron vencer por 5-0 a un profesional del juego GO, convirtiéndose en la primera IA capaz de vencer a un profesional de este juego.
- En 2019, el programa AlphaStar consiguió ganar en StarCraft II, considerado uno de los videojuegos estratégicos más complejos, a uno de los mejores jugadores de este juego por 5-0.

C. Otros algoritmos

Existen muchos tipos de algoritmos de aprendizaje por refuerzo. Este trabajo versa sobra el algoritmo Q-Learning. El desarrollo de este algoritmo (Watkins, 1989) fue muy importante en la época. Se trata de un algoritmo "off-policy", ya que siempre actualiza la matriz *Q* usando la mejor acción posible, siendo la variable γ (*gamma*), conocida como factor de descuento o factor de aprendizaje, la encargada de decidir el peso de la mejor acción. Esto es, cuanto mayor sea γ mayor

impacto se le dará al hecho de escoger la acción con mayor recompensa [3]. Además, este algoritmo, a diferencia de otros, no necesita conocer un modelo del entorno para desarrollarse [23].

SARSA es otro algoritmo (4) de aprendizaje por refuerzo similar a Q-Learning. Sin embargo, este es "on-policy". Esto quiere decir que tiene una política inicial que va actualizando al final de cada episodio. Esta diferencia no implica que ambos algoritmos no puedan utilizar la política ε-greedy para alcanzar un equilibrio entre exploración y explotación. Q-Learning, con el objeto de conseguir el mejor recorrido, puede que pase por penalizaciones peligrosas. Por otro lado, SARSA las ignorará en un principio hasta que los parámetros de exploración se reduzcan. Por tanto, SARSA es un algoritmo más seguro, pero por contra, Q-Learning es óptimo para entornos de bajo coste e iteraciones rápidas [24].

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]. \tag{4}$$

Otro algoritmo utilizado en aprendizaje por refuerzo es DYNA. Este algoritmo combina Q-Learning y Q-Planning, de forma que utiliza tanto la experiencia simulada como la real para mejorar el modelo y la política [26]. Los agentes en DYNA son capaces de planear, actuar, aprender sobre el modelo y mejorar la política de forma paralela. La idea es aprender de la experiencia, pero a la vez usar un modelo simulado para generar más experiencia simulada y aprender de una forma muy rápida.

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S,Q)
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Pseudocódigo 5. Algoritmo DYNA [25]

En ocasiones el algoritmo Q-Learning tradicional no tiene buenos resultados, por ello han surgido variantes. El algoritmo Q-Learning doble surgió debido a que el tradicional sobrevaloraba en ocasiones los valores de la acción. Para evitar esto se realizan estimaciones dobles de los valores, de forma que a partir de *QA* se obtiene *QB* y se actualiza *QA* con ese valor.

Otra variante es el Deep Reinforcement Learning, en el cual se combina el aprendizaje por refuerzo y las redes neuronales. Una de las técnicas más importantes dentro del Deep Reinforcement Learning es el Deep Q-Network (DQN). En este algoritmo se combina el Q-Learning y las redes convolucionales profundas [23]. Fue el primero en demostrar que se podía hacer una aproximación de la función *Q* usando redes neuronales y Q-Learning. Utiliza el "experience replay", que permite guardar en una base de datos las experiencias del agente y usar cada paso para más de una actualización.

VII. Anexo II: Tablas de resultados

En el presente anexo se adjuntan todas las tablas con las mediciones de los resultados de ejecutar las siguientes configuraciones:

Tabla 1. Resultados para la configuración 1

Algoritmo	Q-Learning
Entrenos	150
Gamma	0.5
Épsilon	-
Alfa	-
Rendimiento	7588
Tiempo	5.17 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 2. Resultados para la configuración 2

Algoritmo	Q-Learning
Entrenos	500
Gamma	0.5
Épsilon	-
Alfa	-
Rendimiento	8185
Tiempo	16.95 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 3. Resultados para la configuración 3

Algoritmo	Q-Learning
Entrenos	1500
Gamma	0.5
Épsilon	-
Alfa	-
Rendimiento	7467
Tiempo	54.53 segundos
Observaciones	Resuelve camino. No es óptimo, con tendencia a perderse.

Tabla 4. Resultados para la configuración 4

Algoritmo	Q-Learning
Entrenos	4000
Gamma	0.5
Épsilon	-
Alfa	-
Rendimiento	6179
Tiempo	143.88 segundos
Observaciones	Resuelve camino. No es el mejor.

Tabla 5. Resultados para la configuración 5

Algoritmo	Q-Learning
Entrenos	150
Gamma	0.85
Épsilon	-
Alfa	-
Rendimiento	32859
Tiempo	5.89 segundos
Observaciones	Resuelve camino. No es el mejor.

Tabla 6. Resultados para la configuración 6

Algoritmo	Q-Learning
Entrenos	500
Gamma	0.85
Épsilon	-
Alfa	-
Rendimiento	32655
Tiempo	19.3 segundos
Observaciones	Resuelve camino. No es el mejor.

Tabla 7. Resultados para la configuración 7

Algoritmo	Q-Learning
Entrenos	1500
Gamma	0.85
Épsilon	-
Alfa	-
Rendimiento	30991
Tiempo	61.23 segundos
Observaciones	Resuelve camino. El más óptimo.

Tabla 8. Resultados para la configuración 8

Algoritmo	Q-Learning
Entrenos	4000
Gamma	0.85
Épsilon	-
Alfa	-
Rendimiento	28703
Tiempo	165.8 segundos
Observaciones	Resuelve camino. El más óptimo.

Tabla 9. Resultados para la configuración 9

Algoritmo	Q-Learning variado
Entrenos	150
Gamma	0.5
Épsilon	0.9999
Alfa	0.9999
Rendimiento	5238
Tiempo	3.91 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 10. Resultados para la configuración 10

Algoritmo	Q-Learning variado
Entrenos	500
Gamma	0.5
Épsilon	0.9999
Alfa	0.9999
Rendimiento	6675
Tiempo	12.51 segundos
Observaciones	Resuelve camino. No es el mejor.

Tabla 11. Resultados para la configuración 11

Algoritmo	Q-Learning variado
Entrenos	1500
Gamma	0.5
Épsilon	0.9999
Alfa	0.9999
Rendimiento	5660
Tiempo	38.18 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 12. Resultados para la configuración 12

Algoritmo	Q-Learning variado
Entrenos	4000
Gamma	0.5
Épsilon	0.9999
Alfa	0.9999
Rendimiento	5570
Tiempo	96.86 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 13. Resultados para la configuración 13

Algoritmo	Q-Learning variado
Entrenos	150
Gamma	0.85
Épsilon	0.9999
Alfa	0.9999
Rendimiento	29716
Tiempo	4.4 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 14. Resultados para la configuración 14

Algoritmo	Q-Learning variado
Entrenos	500
Gamma	0.85
Épsilon	0.9999
Alfa	0.9999
Rendimiento	33145
Tiempo	17.55 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 15. Resultados para la configuración 15

Algoritmo	Q-Learning variado
Entrenos	1500
Gamma	0.85
Épsilon	0.9999
Alfa	0.9999
Rendimiento	30270
Tiempo	37.02 segundos
Observaciones	Resuelve camino. El más óptimo.

Tabla 16. Resultados para la configuración 16

Algoritmo	Q-Learning variado
Entrenos	4000
Gamma	0.85
Épsilon	0.9999
Alfa	0.9999
Rendimiento	28532
Tiempo	94 segundos
Observaciones	Resuelve camino. El más óptimo.

Tabla 17. Resultados para la configuración 17

Algoritmo	Q-Learning
Entrenos	150
Gamma	0.25
Épsilon	-
Alfa	-
Rendimiento	2455
Tiempo	5.26 segundos
Observaciones	Resuelve camino. No es óptimo. Recorre las recompensas.

Tabla 18. Resultados para la configuración 18

Algoritmo	Q-Learning
Entrenos	1500
Gamma	0.2
Épsilon	-
Alfa	-
Rendimiento	1747
Tiempo	52.28 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 19. Resultados para la configuración 19

Algoritmo	Q-Learning
Entrenos	500
Gamma	0.15
Épsilon	-
Alfa	-
Rendimiento	1389
Tiempo	-
Observaciones	No resuelve el camino.

Tabla 20. Resultados para la configuración 20

Algoritmo	Q-Learning variado
Entrenos	150
Gamma	0.8
Épsilon	0.9999
Alfa	0.7
Rendimiento	-
Tiempo	-
Observaciones	El algoritmo no termina.

Tabla 21. Resultados para la configuración 21

Algoritmo	Q-Learning
Entrenos	10000
Gamma	0.5
Épsilon	-
Alfa	-
Rendimiento	6003
Tiempo	405.33 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 22. Resultados para la configuración 22

Algoritmo	Q-Learning
Entrenos	10000
Gamma	0.85
Épsilon	-
Alfa	-
Rendimiento	28376
Tiempo	395.52
Observaciones	Resuelve camino. El más óptimo.

Tabla 23. Resultados para la configuración 23

Algoritmo	Q-Learning variado
Entrenos	10000
Gamma	0.5
Épsilon	0.9999
Alfa	0.9999
Rendimiento	5338
Tiempo	257.15 segundos
Observaciones	Resuelve camino. No es óptimo.

Tabla 24. Resultados para la configuración 24

Algoritmo	Q-Learning variado
Entrenos	10000
Gamma	0.85
Épsilon	0.9999
Alfa	0.9999
Rendimiento	28394
Tiempo	234.83 segundos
Observaciones	Resuelve camino. El más óptimo.

REFERENCIAS

[1] Página web del blog de Iberdrola. https://www.iberdrola.com/innovacion/que-es-inteligencia-artificial. Consultada el 16/06/2020.

[2] Página web de Aprendizaje por refuerzo en Wikipedia. https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo. Consultada el 16/06/2020.

[3] Página web de Q-Learning en Wikipedia. https://es.wikipedia.org/wiki/Q-learning#Historia. Consultada el 16/06/2020.

[4] Página web de Mnemosyne_Studio. http://mnemstudio.org/path-finding-q-learning-tutorial.htm. Consultada el 16/06/2020.

[5] Página web Medium – Tipos de aprendizaje automático. https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2. Consultada el 17/06/2020.

[6] Página web de Fernando Sancho – Aprendizaje por refuerzo: algoritmo Q-Learning. http://www.cs.us.es/~fsancho/?e=109. Consultada el 16/06/2020.

[7] Página web Medium – Aprendizaje por Refuerzo: Introducción al mundo del RL. https://medium.com/aprendizaje-por-refuerzo-introducci%C3%B3n-al-mundo-del/aprendizaje-por-refuerzo-introducci%C3%B3n-al-mundo-del-rl-1fcfbaa1c87. Consultada el 16/06/2020.

[8] Página web del blog de Telefónica. https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/. Consultada el 17/06/2020.

[9] Página web INAOE – Aprendizaje por Refuerzo – Eduardo Morales. https://ccc.inaoep.mx/~emorales/Cursos/Aprendizaje2/Acetatos/refuerzo.pdf. Consultada el 16/06/2020.

[10] Página web de Rumbo Loxodromico – Aplicaciones de Aprendizaje por Refuerzo en Videojuegos. https://rastergraphics.wordpress.com/2012/05/12/aplicaciones-de-aprendizaje-por-refuerzo-en-video-juegos/. Consultada el 16/06/2020.

[11] Página web de Universidad de Valladolid – Reinforcement learning como reacción frente a anomalías en la red. https://uvadoc.uva.es/bitstream/handle/10324/33081/TFM-G934.pdf. Consultada el 16/06/2020.

[12] Página web de Analytics India Mag. https://analyticsindiamag.com/reinforcement-learning-policy/. Consultada el 17/06/2020.

[13] Página web INAOE – Procesos de Decisión de Markov – Enrique Sucar. https://ccc.inaoep.mx/~esucar/Clases-mgp/pgm15-mdp-2012.pdf. Consultada el 17/06/2020.

[14] Página web de Proceso de Markov en Wikipedia. https://es.wikipedia.org/wiki/Proceso_de_M%C3%A1rkov. Consultada el 17/06/2020.

[15] Página web de Proceso estocástico en Wikipedia. https://es.wikipedia.org/wiki/Proceso_estoc%C3%A1stico. Consultada el 17/06/2020.

[16] Página web Medium – Aprendizaje por Refuerzo: Procesos de Decisión de Markov (parte 1). https://medium.com/aprendizaje-por-refuerzo-introducci%C3%B3n-al-mundo-del/aprendizaje-por-refuerzo-procesos-de-decisi%C3%B3n-de-markov-parte-1-8a0aed1e6c59. Consultada el 17/06/2020.

[17] Página web del Instituto de Ingeniería del Conocimiento. https://www.iic.uam.es/aprendizaje-profundo-por-refuerzo/. Consultada el 18/06/2020.

[18] Página web de IAT – Inteligencia Artificial en Videojuegos. https://iat.es/tecnologias/inteligencia-artificial/videojuegos/. Consultada el 18/06/2020.

[19] Página web de la IA Angelina. http://www.gamesbyangelina.org/. Consultada el 19/06/2020.

[20] Página web de DeepMind en Wikipedia. https://es.wikipedia.org/wiki/DeepMind. Consultada el 18/06/2020.

[21] Página web de DeepMind. https://deepmind.com/about. Consultada el 19/06/2020.

[22] Página web de Planeta Chatboot – Introducción al Aprendizaje por Refuerzo. https://planetachatbot.com/introducci%C3%B3n-al-aprendizaje-por-refuerzo-f910d669d077. Consultada el 17/06/2020.

[23] Página web de la Universidad Politécnica de Madrid – Comparación de técnicas de aprendizaje por refuerzo jugando a videojuego de tenis. http://oa.upm.es/55888/1/TFM_PABLO_SAN_JOSE_BARRIOS.pdf. Consultada el 17/06/2020.

[24] Página web de StackExchange sobre la diferencia de SARSA y Q-Learning. https://stats.stackexchange.com/questions/326788/when-to-choose-sarsa-vs-q-learning. Consultada el 19/06/2020.

[25] Página web Incompleteideas. http://incompleteideas.net/book/bookdraft2017nov5.pdf. Consultada el 15/06/2020.

[26] Página web Medium – Online Planning Agent: DynaQ Algorithm and Dyna Maze Example (Sutton and Barto 2016). https://medium.com/@ranko.mosic/online-planning-agent-dyna-q-algorithm-and-dyna-maze-example-sutton-and-barto-2016-7ad84a6dc52b. Consultada el 19/06/2020.