
# PRÁCTICA 2

## APRENDIZAJE POR REFUERZO

Andrés Vinagre Blanco - 100383414

Alejandro Parrado Arribas - 100383453

Curso 2019 - 2020

Grupo 83

# Índice

# 1. Descripción de los contenidos del documento.

El documento explica la **implementación de un agente automático de PacMan** usando Aprendizaje por **Refuerzo**, concretamente, el algoritmo **Q-Learning.** A lo largo del documento se explican las iteraciones que se han realizado para conseguir el agente final, así como los diferentes **atributos** usados y una **comparación** de los agentes que se han entrenado. También, se explica el proceso de entrenamiento de cada agente en los mapas.

# 2. Agentes implementados y comparación de los mismos (Fase 3).

## 2.1. Agentes implementados.

### Agente 1.

**Atributos:** distancia del PacMan al fantasma más cercano.

Para entrenar se utilizarán valores alpha =1, gamma = 0.8 y épsilon = 0.5 inicialmente.

Se ha decido poner épsilon 0.5 para que la mitad de las acciones que ejecute el PacMan sean tomadas de forma **aleatoria** y así realizar una **exploración** del entorno. Como con un solo fantasma el refuerzo no se obtiene hasta que se lo coma y acabe la partida en la primera partida se tomaran acciones aleatorias todo el tiempo, cuando empiece a **rellenarse la tabla Q**, se realizarán más acciones de la misma.

En este mapa se han jugado 10 partidas y se han obtenido los siguientes scores.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 113 | 157 | 119 | 157 | 167 | 127 | 161 | 165 | 163 | 169 |

Score medio: 149.8

En las primeras 6 partidas del **mapa 1, labAA1,** se ha tomado un valor épsilon de 0.5 para que el PacMan **explore** el entorno. Posteriormente, en las 3 partidas posteriores se ha reducido la épsilon a 0.3 para que el PacMan ejecute más **acciones dadas por la tabla Q.** En la última partida, se ha reducido alpha a 0 y épsilon a 0 con el fin de ver lo que ha aprendido el PacMan a partir de la tabla Q. Aunque toma acciones aleatorias va de forma más o menos directa al PacMan.

En el **mapa 2, labAA2,** hay dos fantasmas y tal y como tenemos nuestro agente consigue comerse al fantasma 1 (el del mapa anterior) de forma más o menos directa (con alpha 1, épsilon 0.5 y factor de descuento 0.8). Sin embargo, no se acerca al otro fantasma por lo que se ha decido incrementar épsilon a 0.7 para que interactúe más con el entorno y obtenga refuerzo si se come a este fantasma. Como no funciona aún bien se pone épsilon a 1 para que todas las acciones sean aleatorias. Tras unas ejecuciones con épsilon a 1 se ha actualizado la tabla Q y se decide reducir la épsilon de nuevo para que siga entrenando.

Detectamos que el PacMan se pierde un poco teniendo en cuenta solo la **distancia de *Manhattan*** y que en cuanto aparece un nuevo fantasma, no funciona bien la tabla Q anterior y hay que explorar de nuevo el entorno para detectar la presencia de un segundo fantasma y que la distancia de *Manhattan* comience a funcionar. Por lo que hemos pensado en incluir un nuevo atributo que aporte más información. Se realizará sobre el agente 2.

**NOTA:** nos hemos dado cuenta que no estábamos entrenando bien puesto que que con el alpha a 1 se nos perdía toda la información que tenía anteriormente la tabla Q, por lo tanto, se ha decidido entrenar siempre con **alpha 0.2 o 0.1 para preservar la información de los mapas anteriores**. La épsilon se va reduciendo a medida que suceden ejecuciones de los mapas y el factor de descuento se ha decidido considerar constante para darle más importancia al refuerzo obtenido a largo plazo.

## Agente 2.

**Atributos.**

**1. Distancia de *Manhattan*** al fantasma más cercana **discretizada**, para ello se ha realizado un proceso de discretización de rangos de distancia de *Manhattan* para dar 4 valores posibles en lugar de 18 (que es la distancia de *Manhattan* máxima en nuestro grid):

- Rango [0, 3]: se devuelve valor 0.
- Rango de (3, 6]: se devuelve valor 1.
- Rango de (6, 10]: se devuelve 2.
- Si distancia > 10: se devuelve 3.

Esto se ha hecho con la función denominada **"distNearGhostDisc".**

**2. Zona del fantasma más cercano:** devuelve un valor entero entre 0 y 7 correspondiente a la zona donde está situado el fantasma más cercano del PacMan. Esta zona puede ser: NORTE, SUR, ESTE, OESTE, NORESTE, NOROESTE, SURESTE Y SUROESTE. De esta manera el PacMan aprenderá a moverse hacia las zonas donde haya fantasmas. Se ha realizado con la función **getZoneNearestGhost**.

Hay un total de **32 estados** (4*8 = 32).

Para entrenarlo se seguirá el procedimiento de establecer una **alpha inicial baja (a 0.3)** ya que en las primeras ejecuciones con el entorno se están tomando **acciones aleatorias** (con probabilidad 0.6) y factor de descuento a 0.8. Para propagar el refuerzo obtenido al final de la partida con el primer fantasma.

A medida que se realizan ejecuciones sobre el mapa, se reduce el porcentaje de acciones aleatorias que se ejecutan y se reduce *alpha* para ir entrenando sobre la tabla Q que ya hay.

El mismo procedimiento se sigue en todos los mapas.

| | LABAA1 | LABAA2 | LABAA3 | LABAA4 | LABAA5 |
|---|---|---|---|---|---|
| Score medio | 157.5 | 331.4 | - | - | - |
| Score con alpha a 0 y épsilon a 0 | 183 | 383 | - | - | - |

Es importante señalar que en el **mapa 3** ha comenzado a funcionar peor debido al "túnel" en el que está metido uno de los fantasmas ya que la **distancia de *Manhattan*** que nos está calculando **no** tiene en cuenta este **muro** y anteriormente había aprendido a realizar acciones hacia la zona en la que se encuentra el fantasma más cercano. Sin duda alguna, el problema de que no entrene bien en el mapa 3 se debe la muro y a la distancia de *Manhattan*. Por eso mismo, se ha decidido incluir un atributo que solucione este problema en el **agente 4.**

## Agente 3.

Los **atributos** que tiene este agente: se tiene la **dirección del PacMan al fantasma más cercano**, es decir, si el fantasma está en el este, la dirección del PacMan es este y la dirección a comida más cercana. Para conseguir esto se ha modificado *distancer* y hacer que devuelva la dirección que debe tomar el PacMan para dar el siguiente paso y acercarse al fantasma. Con la comida la dirección devuelta es la misma. Además, se ha tenido en cuenta otro **atributo binario** que toma valor de 0 si la distancia a una comida es mayor que a la del fantasma más cercano y 1 en el caso de que la distancia a la comida sea menor que a la de el fantasma más cercano. Por lo que, hay un total de 40 estados. Se han usado las funciones **dirNearGhostWallDisc** y **dirNearFoodWallDisc**.

Este agente se ha entrenado con el mismo procedimiento que el agente 2. Se han obtenido los siguientes scores:

| | LABAA1 | LABAA2 | LABAA3 | LABAA4 | LABAA5 |
|---|---|---|---|---|---|
| Score medio | 165.56 | 348.34 | 543.45 | 657.65 | 1098.7 |
| Score con alpha a 0 y épsilon a 0 | 183 | 383 | 569 | 751 | 1236 |

**NOTA:** los mapas del 1 al 4 los entrena de forma rápida, pero para el mapa 5 se ha necesitado una hora de entrenamiento (se ha seguido el procedimiento de entrenamiento ya explicado anteriormente).

Teniendo en cuenta estos atributos explicados, ha sido el agente que mejor ha funcionado.

## Agente 4.

**Atributos:** distancia perfecta al fantasma más cercano y un atributo que indica la **zona del fantasma** (como el del agente 2). Debido a que el agente 2 ha funcionado mal cuando se ha encontrado muros en el mapa 3 con la distancia de *Manhattan*, se ha decidido usar el método de la clase *distancer* que devuelve la **distancia a un fantasma esquivando los muros**, de forma que, si hay muros de por medio, no se devolverá la distancia de *Manhattan* si no una distancia mayor. No obstante, como el máximo que puede devolver esta función *distancer* depende del mapa y de los muros que haya se ha decidido discretizar:

- **Rango [0, 2]:** se devuelve valor 0.
- **Rango de [3, 6]:** se devuelve valor 1.
- **Rango de [7, 10]:** se devuelve 2.
- Si **distancia > = 11:** se devuelve 3.

Los atributos se obtienen con **distNearGhostWallDisc** y **getZoneNearestGhost.**

Se ha ido entrenando en **los mapas LABAA1, LABAA2** con un valor de alpha de 0.2 y una épsilon de 0.6 que posteriormente se ha ido reduciendo. Para **el mapa 3,** se ha decidido emplear un *alpha* de 0.1 para así ser conservadores y preservar lo aprendido con anterioridad y una épsilon de 0.6, que se ha ido reduciendo a lo largo de las ejecuciones a 0.5, 0.4 y finalmente a 0.0.

Para el **mapa 4** se ha entrenado de forma similar al 3, pero se han tenido que realizar más ejecuciones. Además, con una épsilon alto se ha explorado el entorno ya que hay estados que no se habían contemplado antes. El *alpha* se ha mantenido a 0.1. Cada tres ejecuciones se ha ido reduciendo el valor de épsilon para así trabajar con los valores que hay en la tabla Q.

Los resultados obtenidos son:

| | LABAA1 | LABAA2 | LABAA3 | LABAA4 | LABAA5 |
|---|---|---|---|---|---|
| Score medio | 143.5 | 341.8 | 487.6 | 583.88 | - |
| Score con alpha a 0 y épsilon a 0 | 183 | 383 | 569 | 571 | - |

> **NOTA:** en el mapa 4 no se come comida ya que al aprender y no tener atributos relacionados con ella, el PacMan no ha aprendido a comérsela.

En el **mapa 5,** el PacMan no consigue completar el mapa bien con estos atributos propuestos ya que aparecen puntos de comida que no se están teniendo en cuenta en los atributos seleccionados, por lo tanto, en el mapa 5 no sabe muy bien cómo generalizar y al comerse comida se produce refuerzo que no se sabe a que se debe ya que no se considera en los atributos usados y no se sabe cómo aprender con esta información.

## Agente 5.

Con el objetivo de mejorar el agente 4 incluir atributos para contemplar en los estados la existencia de comida en los mapas se ha añadido al agente 4 el siguiente atributo:

- **Distancia perfecta a la comida más cercana discretizada**: se usa el método de la clase *distancer* para obtener la distancia a la comida y al igual que en el agente anterior se ha tenido que discretizar para proporcionar un rango de valores fijo (es el mismo rango que al fantasma más cercano ya que ha funcionado bien).

Tras diferentes entrenamientos similares a los del agente anterior se ha obtenido los siguientes resultados:

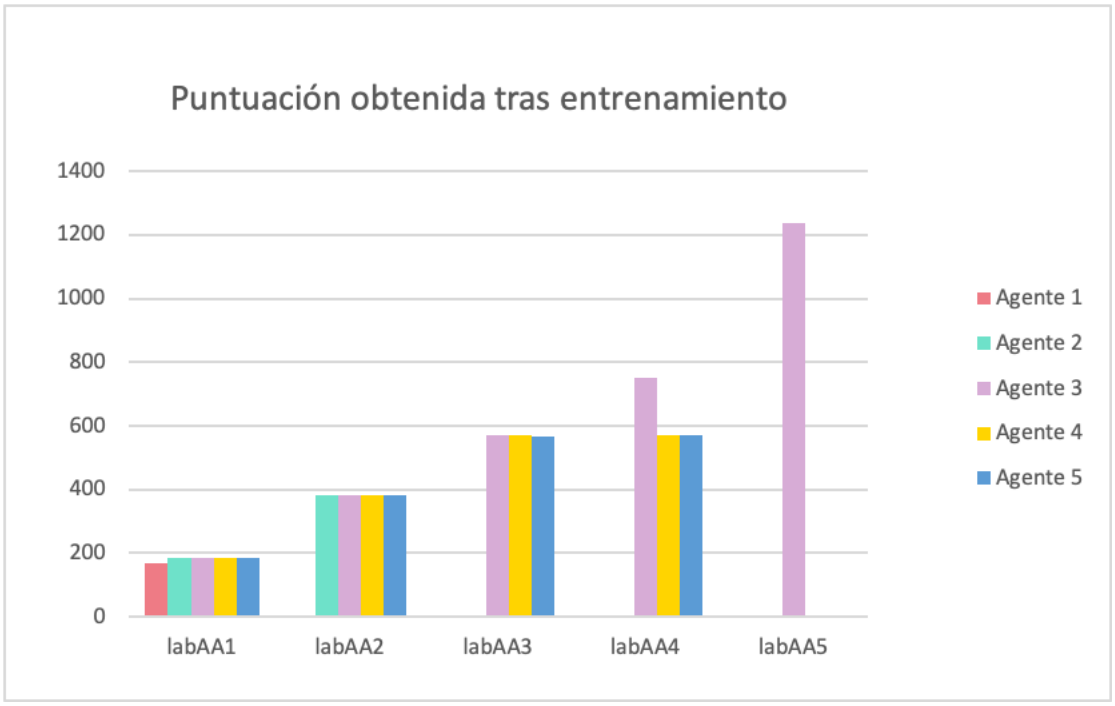
| | LABAA1 | LABAA2 | LABAA3 | LABAA4 | LABAA5 |
|---|---|---|---|---|---|
| Score medio | 130.96 | 337.8 | 479.75 | 567.34 | - |
| Score con alpha a 0 y épsilon a 0 | 183 | 381 | 567 | 569 | - |

No obstante, es importante señalar que a pesar de haber tenido en cuenta el atributo de distancia a la comida más cercana **no ha sido suficiente para aprender y generalizar en el mapa 5**, ya que es el mapa que más puntos de comida tiene y solo con la distancia, el agente PacMan no tiene información de en qué dirección se encuentra la comida.

Se ha pensado en incorporar en el agente 5 un atributo similar al de la distancia que informe sobre la zona de la comida más cercana, pero el rango de estados sería de 512 (8*4*4*4) y se excede bastante ya que lo recomendado es 200 filas, por lo que no se ha incluido.

## 2.2. Comparación de los agentes.

Como en la **práctica 1 no se consiguió un modelo** que funcionara bien no se han comparado los agentes obtenidos con aprendizaje por refuerzo con los obtenidos por aprendizaje supervisado.




Como se observa en la gráfica, la **puntuación de los agentes** en el primer mapa es la misma para todos excepto para el agente 1, donde la puntuación es más baja. Vemos que ha medida que se han probado mapas hay agentes que obtienen puntuación de 0 ya que **los atributos que se han usado para construirlos no son suficientes para representar de forma general estos mapas** y, por lo tanto, no pueden generalizarse. **Los agentes 3, 4 y 5 han funcionado bien hasta el mapa 4** pero en el mapa 5 el 4 y el 5 no lo han completado. El **agente 4** no lo ha completado debido a que en los atributos no se tenía en **cuenta los puntos de comida** y como el agente obtiene refuerzo cuando se come una comida el agente se "pierde" ya que sus atributos **no están representando** porqué se ha conseguido un refuerzo.

Además, el **agente 5,** a pesar de tener la **distancia a la comida más cercana**, esta información no ha sido suficiente para completar de forma satisfactoria el mapa 5 puesto que a pesar de decirle la distancia no se indicaba en qué zona estaba la comida por lo que **el PacMan no sabía a qué direcciones tenía que ir para acercarse** (podría haberse solucionado añadiendo un atributo pero la tabla Q hubiera tenido 512 estados, tal y como hemos comentado anteriormente en el agente 5).

El **éxito en el mapa 5** se ha obtenido solo con un agente, el **agente 3,** esto es porque se ha tenido en cuenta la **dirección a tomar por el PacMan para ir a por la comida más cercana o para comerse al fantasma más cercano**. Asimismo, se ve que en el mapa 4 gracias a este atributo de comida el score es bastante más alto que otros agentes.

## 3. Justificación del conjunto de atributos final elegido y su rango utilizado para la definición de los estados.

Los **atributos** que se han elegido son aquellos que conforman el **agente 3:**

- **Dirección que toma el PacMan para ir al fantasma más cercano** (toma valores 0,1,2 o 3).
- **Dirección que toma el PacMan para ir a la comida más cercana** (0, 1, 2, 3 y 4).
- **Atributo binario con valor 1 si la distancia a una comida es mayor a la distancia a un fantasma y 0 en caso de que la distancia sea menor a la del fantasma.**

Por lo que el rango de combinaciones posibles que pueden darse es de 40. Por lo que hay 40 estados, de modo que la tabla Q tiene 40 filas, una para cada estado. En lugar de devolver la dirección como un string se ha decidido de volver valores enteros.

## 4. Descripción de la función de refuerzo final empleada

La función de refuerzo utilizada tiene tres casos:

- **200 de refuerzo si el PacMan se come a un fantasma.**
- **100 de refuerzo si el PacMan se come un punto de comida.**
- 0 en cualquier otro estado.

De esta forma, como el objetivo de la tarea de Aprendizaje Automático es **comerse a todos los fantasmas** se da más refuerzo a comer fantasmas que a comer puntos de comida, el algoritmo puede decidir dependiendo del refuerzo obtenido en los estados si es mejor que vaya a por un punto de comida o un fantasma puesto que siempre se va a maximizar el refuerzo.

## 5. Descripción del tratamiento de los datos y pasos realizados para ello

Los datos que se han obtenido para implementar los diferentes agentes mencionados se han tenido que **discretizar con el objetivo de reducir el tamaño de la tabla Q** y así poder condensar más información con el menor número de estados posibles. En general, nuestras discretizaciones han funcionado bien sobre los agentes y se ha entrenando con agentes con la distancia sin discretizar y discretizadas y se han obtenido los mismos resultados. Sin embargo, **una discretización errónea** podría llevar a introducir no determinismo y a perder la política óptima en las tareas de aprendizaje por refuerzo, eso sí, se podría conseguir una política subóptima.

Los pasos seguidos para llevar a cabo esta **discretización se han explicado en los agentes anteriores.**

## 6. Descripción del código generado para llevar a cabo la política de comportamiento

Para llevar a cabo nuestro agente PacMan se ha **implementado código** que explicaremos a continuación por partes. Señalar que nosotros hemos trabajado con state, que es equivalente a gameState, no hemos creado un estado formado por tuplas para representar nuestro PacMan si no que en el método **computeposition** ha sido donde hemos "filtrado" state, **seleccionando aquellos atributos que nos interesaban para construir los estados y devolver la fila de la tabla Q,** equivalente a ese estado.

**NOTA: debido a que hemos modificado el método getDistance de distancer.py, es necesario ejecutar con todo nuestro código, no solo copiar el archivo busterAgents.py.**

Además, se ha usado una variable **lastState**, de forma que el entrenamiento se realiza con state y lastState. Esto es equivalente a usar state y nextState puesto que en la función getPolicy se utiliza state y no lasState, por lo tanto, **NO se está ejecutando en diferido.**

En primer lugar, tenemos las siguientes funciones implementadas:

- **ComputePosition:** recibimos un state y se seleccionan los atributos necesarios de ese state para devolver una fila de la tabla Q. La variable global **option** de la clase QLearning simplemente se utiliza para elegir el conjunto de atributos sobre el que estamos trabajando, lo que nos ha permitido no tener que borrar QLearning cada vez que queramos entrenar con estados diferentes.
- **GetAction:** se ha añadido una línea para llamar a la función **update** cada vez que la última acción no sea None y pueda realizarse. Luego se actualiza la variable *lastState* y *lastAction*. Aquí es donde se llama a la **función de refuerzo** para pasarle el valor a update.
- **Frefuerzo:** es la función de refuerzo explicada en el apartado 4 de esta memoria. Se ha usado la variable **food**, para que cada vez que desapareza una comida del mapa se de un refuerzo de **100**. Para los fantasmas, se ha usado la variable **GhostPositions** y así acceder a todas las posiciones de los fantasmas para dar un refuerzo de **200** cuando el PacMan se lo coma.
- **Final:** función auxiliar que llama al método *update* justo en el último *tick* del juego. Ha sido necesaria incluirla porque de lo contrario no se obtenía el refuerzo al comerse el último fantasma.
- **Manhattan**: función que calcula la distancia de Manhattan entre un punto origen y un punto destino.
- **distNearGhostWall**: método que calcula la distancia perfecta de origen a destino esquivando los muros. Para ello, es necesario que se llame a la función *getDistance de distancer.py.*
- **distNearGhostWallDisc:** método que **discretiza la distancia devuelta** por el método anterior ya que al no saber el número de muros esta es variable y se necesita un conjunto de estados fijos. Dicha **discretización se ha explicado en los agentes** donde se ha usado este atributo.
- **distNearFoodWall:** función similar a distNearGhostWall con la diferencia de quee devuelve la distancia a la comida más cercana esquivando muros.
- **distNearFoodWallDisc:** discretización de la función anterior de forma similar a distNearGhostWallDisc.
- **distNearGhost:** devuelve la distancia de Manhattan al fantasma más cercano al PacMan.

- **distNearGhostDisc:** devuelve la distancia de Manhattan discretizada al fantasma más cercano al PacMan. La discretización se explica en los agentes en los que se ha usado.
- **getZoneNearestGhost:** devuelve la zona del fantasma más cercano respecto al PacMan, teniendo en cuenta: norte, sur, noreste, noroeste, sureste, suroeste, este y oeste.
- **dirNearestFoodWall: devuelve la dirección que debe tomar el PacMan para ir a la comida más cercana.**
- **dirNearestFoodWallDisc:** igual que la anterior, pero devolviendo un entero por cada dirección.
- **dirNearestGhostWall**: igual que **dirNearestFoodWall** pero con fantasmas.
- **dirNearestGhostWallDisc**: igual que la anterior, pero devolviendo un entero por cada dirección.

Estas funciones anteriormente descritas sirven para obtener los atributos de los diferentes agentes que hemos implementado y se ha explicado en los apartados correspondientes a cada agente.

## 7. Descripción del agente final implementado

Para llegar al agente final **(agente 3)** se ha iterado sobre diferentes agentes construidos sobre diferentes conjuntos de atributos. En el **Agente 1** solo se tenía en cuenta la **distancia de *Manhattan*** al fantasma, pero nos dimos cuenta de que es insuficiente porque en un mapa en el que la distancia de *Manhattan* puede ser 18 pues el PacMan no tiene **información de a que dirección tiene que moverse.** Por eso mismo, se decidió construir el **Agente 2** formado por la distancia de Manhattan (discretizada para que no sea tan grande la tabla Q) y la zona del fantasma más cercano desde el PacMan (Sur, Este, Oeste, Norte, Noreste, Noroeste, Suroeste y Sureste). De esta manera, el PacMan ya **tiene información sobre hacia donde tiene que moverse**. Sin embargo, en el mapa 3 el agente 2 nos comenzó a fallar puesto que la **distancia de *Manhattan* no tenía en cuenta los muros**, indicando en este mapa que la distancia a un fantasma es de 2 cuando en realidad es mucho mayor ya que hay que bordear el muro para llegar al fantasma, por lo que nos empezó a flaquear lo que habíamos implementado anteriormente.

En este punto, decidimos necesario usar la clase distancer.py que tiene un método que devuelve la **distancia perfecta entre dos puntos,** es decir, te devuelve el camino más corto entre dos puntos esquivando los muros. Gracias a este método se pudo **reemplazar la distancia de *Manhattan*** del agente 2 por la distancia perfecta. Eso sí, dado que esta distancia puede tomar muchos valores porque depende del número de muros que haya entre medias de un PacMan y un fantasma se decidió **discretizarla**. Aunque discretizar conlleve perder información en nuestro caso nos ha funcionado bien ya que cuando hay muros los ha esquivado. Por lo que el **agente 4** está formado por la distancia perfecta discretizada y la dirección a la que está el fantasma más cercano del PacMan.

De forma paralela, se desarrolló el **agente 3** en el que se modifica el método distancer de la clase distancer.py para que devuelva la **dirección correcta para acercarse al fantasma** o a un punto de comida. Es decir, si el fantasma está en el este, distancer devolverá la dirección este ya que es la mejor forma de acercarse al fantasma si no hay muros, en el caso de que haya muros y no pueda irse al este, este método devuelve la dirección correcta para acercarse al fantasma o comida más cercano evitando dichos muros. También se realiza lo mismo con el punto de comida más cercano y, además, hay un atributo binario que indica si el PacMan está más cerca de la comida más cercana o del fantasma más cercano.

**Dado que los Agentes 1, 2, 4 y 5** no han tenido éxito en el mapa 5 el único agente que lo ha conseguido completar es el **Agente 3.** Por lo que, por esto mismo, es el agente final que se entregará junto con el código, así como su qtable. No obstante, señalar que **ha sido costoso entrenarlo y que ha habido que hacer bastantes iteraciones para que funcione de forma correcta.**

## 8. Conclusiones

En esta práctica creemos que **hemos visto funcionar el aprendizaje automático de forma correcta** ya que en la primera práctica nos habíamos llevado una decepción al haber probado muchos modelos y ver que no funcionaba ninguno. Sin duda alguna, lo más difícil ha sido en pensar atributos que funcionen de forma correcta y que hagan que el PacMan aprenda.

Por otro lado, también hemos tenido que **dedicarle tiempo al proceso de entrenamiento** ya que hay que encontrar los valores adecuados de épsilon y alpha para que el PacMan no "desentrene" y pierda información que ya ha aprendido en anteriores mapas o aprenda de forma errónea.

En general, la práctica nos ha parecido **bastante entretenida y ha sido satisfactorio** ver cómo lo que estamos haciendo empieza a funcionar, lo que ha supuesto un primer contacto con Aprendizaje Automático positivo. A pesar de la situación, creemos que los profesores habéis estado a la altura resolviendo todas las dudas.  Por lo que hemos acabado satisfechos con esta práctica.