Este documento recoge la memoria de la práctica final de Aprendizaje Automático, basada en aprendizaje por refuerzo, y en él se pretende reflejar todos los pasos realizados para conseguir un agente, que mediante una serie de pautas dadas por Q-learning, aprenda de manera automática cual es el movimiento más conveniente en cada estado.

1. Definición de estados: donde se estudiará cuáles son los atributos necesarios para la definición de un estado en el Pac-man. Por lo tanto estudiaremos varias versiones, las evaluaremos y llegaremos a una definición de estado después de haber hecho un estudio sobre todas las versiones.
2. Tratamiento de los datos: procesamiento de los datos para una correcta definición de estados.
3. Descripción del código: donde veremos todas las clases y métodos creados para la correcta implementación de nuestro agente. Se explicará todo de manera detallada y clara.
4. Descripción de asignación de estado empleada: en esta parte se verá el porqué de los atributos elegidos para los estados además de empezar a comentar posibles mejoras para nuestro agente.
5. Descripción final de nuestro agente: pequeña descripción de los parámetros usados para conseguir nuestro agente. En este apartado también veremos los distintos agentes creados, viendo los parámetros que han variado entre unos y otros.
6. Descripción y análisis de los resultados producidos por el agente final implementado tras la evaluación, correspondientes a la fase 4 de la memoria.
7. Conclusiones sobre la práctica.

# DEFINICIÓN DE ESTADOS

Realmente esta ha sido una de las partes que más problemas nos ha dado, ya que es bastante difícil encontrar atributos que den información real a Pac-man del mundo. Había que tener en cuenta los posibles valores de los atributos, dado que cuantos más valores posibles, más estados reales se podían crear.

Hay que destacar que hay estados poco probables o imposibles, más adelante veremos algunos ejemplos.

## Versión 1

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|l|l|l|l|}

\hline

\multicolumn{4}{|c|}{\textbf{Atributos}} \\ \hline

NorteDisponible & SurDisponible & EsteDisponible & OesteDisponible \\ \hline

\end{tabular}

\end{table}

En esta primera versión quisimos representar los estados simplemente con las acciones disponibles, aplicando combinatoria básica sacamos que el número de estados posibles es 24. Tener un número pequeño de estados puede suponer ventaja pero no en este caso, ya que como apreciamos no recogemos información para favorecer al Q-learning dado que todos los atributos que recogemos trabajan sobre el presente, es decir, facilitan información al estado actual. En este caso solo aportan los movimientos posibles.

Por lo mencionado en el párrafo anterior, descartamos dicha versión e hicimos una nueva.

## Versión 2

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|l|l|l|l|l|}

\hline

\multicolumn{5}{|c|}{\textbf{Atributos}} \\ \hline

NorteDisponible & SurDisponible & EsteDisponible & OesteDisponible & DireccionFantasmaCerano \\ \hline

\end{tabular}

\end{table}

Como se aprecia en la tabla, añadimos un nuevo atributo que representa hacia donde tiene que tirar nuestro Pac-man para avanzar hacia el fantasma más cercano. Por lo tanto, dicho atributo tiene cuatro posibles valores, lo cual hace que aumente el número de estados hasta 64 (24\*4).

Gracias a este atributo, Pac-man conoce un poco más el “futuro” y no solo le aportamos información del estado actual. A pesar de esto, decidimos seguir modificando los estados para seguir realizando pruebas.

## Versión 3

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|l|l|ll}

\hline

\multicolumn{4}{|c|}{\textbf{Atributos}} \\ \hline

NorteDisponible & SurDisponible & \multicolumn{1}{l|}{EsteDisponible} & \multicolumn{1}{l|}{OesteDisponible} \\ \hline

DireccionFantasmaCerano & DistanciaDiscretizada & & \\ \cline{1-2}

\end{tabular}

\end{table}

En esta tercera versión, añadimos la distancia discretizada al fantasma más cercano. Esto a nuestro criterio podría aportar algo de valor, pasamos de tener 64 estados a tener 256. En el tratamiento de los datos explicaremos qué realizamos con dicho atributo.

Haciendo un análisis teórico fue cuando realmente nos dimos cuenta de que este atributo era irrelevante y solo iba a ser usado para calcular el refuerzo obtenido en cada paso (de esto se hablará más adelante).

## Versión 4

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|l|l|ll}

\hline

\multicolumn{4}{|c|}{\textbf{Atributos}} \\ \hline

NorteDisponible & SurDisponible & \multicolumn{1}{l|}{EsteDisponible} & \multicolumn{1}{l|}{OesteDisponible} \\ \hline

DireccionFantasmaCerano & FantasmasVivos & & \\ \cline{1-2}

\end{tabular}

\end{table}

Cuarta y penúltima versión, añadimos como atributo en cada estado el número de fantasmas vivos en ese instante. Con esto pensábamos que ganábamos información, pasamos de tener 56 estados (versión 2) a tener 256 de nuevo.

Tras razonar, pensamos que estábamos mezclando de nuevo atributos necesarios para calcular el refuerzo y atributos necesarios para representar los estados, por lo tanto decidimos enviar un email al profesor de prácticas José Carlos González para que nos sacase de dudas. Su respuesta fue clara, no hacía falta meter dicho atributo en los estados y por lo tanto procedimos a borrarlo.

## Versión final

La versión final, como ya se ha ido vaticinando, es la versión 2.

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|l|l|l|l|l|}

\hline

\multicolumn{5}{|c|}{\textbf{Atributos}} \\ \hline

NorteDisponible & SurDisponible & EsteDisponible & OesteDisponible & DireccionFantasmaCerano \\ \hline

\end{tabular}

\end{table}

En esta versión recogemos los atributos que consideramos necesarios como mínimo para representar un estado. Toda la práctica desarrollada a partir de este punto parte de esta definición de estados.

# TRATAMIENTO DE LOS DATOS

Para la versión 3 de la definición de estados tuvimos que discretizar los datos, es decir, que el atributo solo tuviese cuatro posibles valores. A pesar de que la distancia al fantasma cercano puede ir desde 0 hasta un número finito, discretizamos dicho atributo dando como posible valor un número natural comprendido entre 0 y 5 (ambos excluidos).

Mediante bucles if y unos rangos creados a nuestro criterio en Python realizamos dicha discretización, que a pesar de haber sido descartada como atributo, la usaremos en la función de refuerzo.

# Descripción de asignación de estado empleada

En esta parte del documento vamos a ver el porqué de los atributos elegidos para nuestros estados.

Todos nuestros estados tienen que aportar información a Pac-man sobre el conocimiento del mundo, es decir, conocer de manera verídica la situación actual en la que se encuentra. Para ello lo primero que pensamos que era totalmente necesario, era saber hacia dónde podía tirar Pac-man, esto se traduce en conocer qué movimientos hay disponibles en cada estado. Gracias a esto conseguimos nuestros cuatro primeros atributos, cada uno con dos posibles valores (Disponible o no disponible), alcanzando un total de 24 = 16 estados posibles.

Ahora nuestro Pac-man conoce los movimientos disponibles que tiene en cada momento , pero no tiene ningún conocimiento que le indique en cierta medida hacia dónde ir, es decir, algo que le sirva de guía para conseguir lograr su objetivo, que es comerse a todos los fantasmas. Por lo tanto, llegamos a la conclusión de que podíamos añadir un atributo que indicase hacia donde tiene que tirar Pac-Man para acercarse al fantasma más cercano.

Dicho atributo puede tener cuatro posibles valores (Norte, Sur, Este y Oeste), así que sumando estas posibilidades a todos los atributos anteriores, obtenemos 24\*4 = 24\*22 = 26 = 64 estados.

Este es el número mínimo de estados que hemos conseguido definir para implementar correctamente Q-learning. Tener el mínimo número de estados supone ciertas ventajas y desventajas.

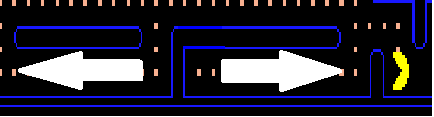
* Ventajas:

Tener menos estados supone trabajar con menos atributos y por lo tanto hay que realizar menos implementación en Python, es más básico y se pueden hacer pruebas fácilmente.

* Desventajas:

Claramente 64 estados se pueden quedar escasos para cualquier mapa medianamente grande de Pac-man, ya que en muchos casos se repetirán los estados. Esto conlleva la modificación de los Q-valores de un mismo estado varias veces en la misma partida en caso de que se repitan los estados, es decir, modifica valores que nosotros ya habíamos actualizado anteriormente en la misma partida.

En mapas en los que se repitan pocos estados nuestro agente va a aprender rápido y fácil pero va a tener problemas en ciertos mapas, como por ejemplo en este tipo de túnel.



En este caso apreciamos que las flechas recogen el mismo estado:

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|c|c|}

\hline

\multicolumn{2}{|c|}{\textbf{Estado}} \\ \hline

NorteDisponible & NO \\ \hline

SurDisponible & NO \\ \hline

EsteDisponible & SI \\ \hline

OesteDisponible & SI \\ \hline

Fantasma+cercano & OESTE \\ \hline

\end{tabular}

\end{table}

Lo que está ocurriendo es que tenemos dos “túneles” donde Pac-man se encuentra en el mismo estado pero tiene que tomar caminos contrarios para alcanzar al fantasma. Esto es una gran desventaja ya que con la definición de nuestros estados y Q-learning resulta imposible que en una misma partida pudiese pasar por los dos “túneles” realizando los movimientos correctos.

Con lo cual llegamos a la conclusión de que no hubiese estado de más añadir algún atributo que hubiese aportado información complementaria como por ejemplo el número de fantasmas vivos, con esto hubiese sabido nuestro agente qué hacer en función del número de fantasmas que se haya comido ya.

COPIA Y PEGA LA INTRODUCCION, HA SIDO CAMBIADA ENTERA.

SE TE HA OLVIDADO MENCIONAR EN REFUERZO.PY QUE SE DA REFUERZO MUY NEGATIVO CUANDO SE CHOCA CON UNA PARED

A PARTIR DE AQUÍ AÑADIR AL PUNTO CORRESPONDIENTE (MIRAR LA INTRODUCCION)

# Descripción final de nuestro agente

Previo a detallar cual ha sido nuestro agente final, vamos a ver los parámetros que pueden ser modificados y que afectan a la hora de cómo aprende nuestro agente.

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|c|}

\hline

\textbf{Parámetros a modificar} \\ \hline

Alpha \\ \hline

Gamma \\ \hline

Refuerzo \\ \hline

\end{tabular}

\end{table}

Estos parámetros pueden ser modificados fácilmente, alpha y gamma en *bustersAgents.py* y refuerzo en *refuerzo.py*.

Tras modificar dichos parámetros basta con crear la tabla Q de nuevo (con tantas filas como estados, en nuestro caso 64) y poner a ejecutar un número suficientemente grande de veces Pac-man hasta llegar a la tabla Q óptima.

Hemos probado 5 versiones distintas de agentes modificando dichos parámetros, todos sobre el mapa *finalMap* para así poder comprobar la rapidez de aprendizaje de cada agente comparándolo con los demás.

Primero vamos a ver un progreso general que se corresponde con cualquiera de los agentes probados:

\begin{figure}[H]

**\centering**

**\includegraphics**[scale=1]{images/p2.png}

\end{figure}

Como se aprecia en la imagen, vemos que en cada conjunto de partidas va mejorando y aumentando el porcentaje de victorias. En cada partida va actualizando la tabla Q y por lo tanto va mejorando hasta alcanzar la tabla Q óptima, pero mientras no alcance dicha tabla modifica en cada movimiento los Q-valores favoreciendo el aprendizaje del agente.

Esta imagen fue tomada durante el aprendizaje de uno de los agentes mientras alcanzaba su tabla Q óptima. Como vemos en este fragmento, empieza ganando 15 de 50 partidas y acaba ganando 23 de 50, lo cual casi representa el 50% de las partidas jugadas.

Con esto pretendemos demostrar que nuestro agente comienza sin conocer absolutamente nada y a través de Q-learning se da el proceso de aprendizaje.

Uno de los principales problemas que tuvimos fue que nuestro Pac-Man realizaba algún tipo de ciclo, es decir, repetía dos o más movimientos de manera consecutiva hasta llegar al máximo número de movimientos establecidos.

Para solucionar esto lo que hicimos fue aplicar un refuerzo negativo en cada paso siempre y cuando en ese mismo paso no haya ganado ningún refuerzo positivo (ya sea por haberse acercado al fantasma más cercano o por habérselo comido). Gracias esto, a la larga salía de los bucles y buscaba otro camino. En la siguiente tabla cuando hacemos uso de la palabra refuerzo nos referimos a este caso particular.

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|c|l|l|l|l|l|}

\hline

\multicolumn{6}{|c|}{\textbf{Tabla de agentes con sus parámetros}} \\ \hline

Agente & \multicolumn{1}{c|}{Agente 1} & \multicolumn{1}{c|}{Agente 2} & \multicolumn{1}{c|}{Agente 3} & \multicolumn{1}{c|}{Agente 4} & \multicolumn{1}{c|}{Agente 5} \\ \hline

Alpha & 0.1 & 0.5 & 0.5 & 0.8 & 0.8 \\ \hline

Gamma & 0.2 & 0.6 & 0.2 & 1 & 0.5 \\ \hline

Refuerzo & -0.5 & -0.1 & -1 & -1 & 0 \\ \hline

\end{tabular}

\end{table}

Para cada uno de estos agentes se han realizado aprendizajes individuales entre 8 y 10 horas en el mapa *finalMap*. Tras dejar un periodo de aprendizaje considerado como suficiente, se han probado los agentes de manera individual haciendo nulos los parámetros alpha y gamma (esto desactiva la función de actualización de los Q-valores).

Los que peor han funcionado han sido el Agente 5 y el Agente 4.

* Agente 5: debido a no tener un refuerzo negativo en cada paso, su aprendizaje ha sido mucho más lento o incluso nulo en algunas partidas. Por lo tanto, o bien necesita muchas más horas para aprender o no aplicar refuerzo negativo es totalmente incompatible con nuestra implementación.
* Agente 4: creemos que es por culpa de gamma, ya que tiene el valor más alto posible y la propagación es máxima.

El agente que mejor ha funcionado ha sido el agente 2:

* Los agentes 1 y 2 comenzaron su aprendizaje en paralelo en el mismo ordenador, al cabo de 3 horas tenía mucha mejor pinta el agente 1 que el 2 pero para nuestra sorpresa, al finalizar el periodo de aprendizaje, el agente 2 resultaba mejor que el agente 1 por las pruebas realizadas con los parámetros alpha y gamma nulos.

Por lo tanto, nuestro agente final es el Agente 2.

\begin{table}[]

\centering

\caption{My caption}

\label{my-label}

\begin{tabular}{|c|r|}

\hline

\multicolumn{2}{|c|}{\textbf{Agente final}} \\ \hline

Alpha & 0.5 \\ \hline

Gamma & 0.6 \\ \hline

Refuerzo & -0.1 \\ \hline

\end{tabular}

\end{table}

Creemos que ha funcionado debido a que hemos dado unos valores intermedios a sus parámetros alpha y gamma además de dar un refuerzo negativo mínimo. Sobre este agente hemos realizado las pruebas finales obteniendo unos resultados bastante positivos, que van como mínimo desde el 60% de partidas ganadas llegando incluso al 90%.

\begin{figure}[H]

**\centering**

**\includegraphics**[scale=1]{images/fase4.png}

\end{figure}

\begin{figure}[H]

**\centering**

**\includegraphics**[scale=1]{images/fase4-1.png}

\end{figure}