Universidad Rey Juan Carlos

PROYECTO PACMAN: APRENDIZAJE REFORZADO

Roxana Nicoleta Aanei
Ingeniería Robótica Software
ETSIT
URJC
Fuenlabrada, Madrid
rn.aanei.2018@alumnos.urjc.es

Juan Carlos Manzanares Serrano
Ingeniería Robótica Software
ETSIT
URJC
Fuenlabrada, Madrid
jc.manzanares.2018@alumnos.urjc.es

Marina Gil Pensado
Ingeniería Robótica Software
ETSIT
URJC
Fuenlabrada, Madrid
m.gilp.2018@alumnos.urjc.es

Cristian Sánchez Rodríguez
Ingeniería Robótica Software
ETSIT
URJC
Fuenlabrada, Madrid
c.sanchezro.2018@alumnos.urjc.es

***Abstract*—**

**El propósito principal de esta memoria es resolver un problema utilizando alguna técnica de Aprendizaje Automático aprendida en la asignatura.
La propuesta que tenemos es el juego Pacman, resolviendo su funcionamiento utilizando Aprendizaje por Refuerzo.**

***Index Terms*—**

Planteamiento del problema, Búsqueda de información, Elaboración de Hipótesis, Experimentación, Análisis y Conclusiones finales.

I. Introducción

En este documento se va reflejar toda la información relacionada con el problema que se va a plantear, que en este caso es una versión propia del Pacman.

La propuesta que teníamos inicialmente era el juego Snake, un juego cuya base es una serpiente que va creciendo de tamaño a medida que encuentra recompensas. Para abordar dicha idea al completo, nos resultaba un poco costoso y lioso de entender, por lo que adaptamos todo lo que llevábamos al juego Pacman, que sigue el mismo funcionamiento de obtener recompensas pero sin aumentar su tamaño.

Para resolver cualquiera de las dos variantes, nos hemos basado en Aprendizaje por Refuerzo.

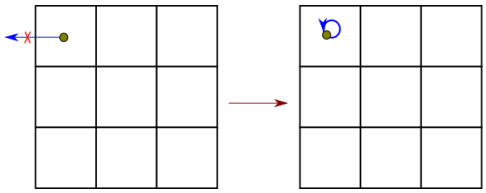
II. Planteamiento del Problema

Para resolver el problema, hemos abordado diferentes sub-problemas para ir poco a poco escalándolo hasta llegar a una versión final.

A. Pacman sin paredes. 1 Comida

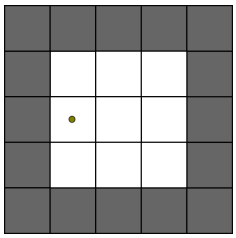
Sistema simple en el cual el Pacman tiene que llegar hasta la recompensa / comida por el camino más óptimo dentro de un entorno cerrado.

Además, en los estados extremos (”paredes”), el Pacman solo puede tomar acciones que le conduzcan de nuevo al entorno, salir fuera supone una acción que le dejaría en el mismo estado en el que se encuentra.



B. Pacman con paredes. 1 Comida

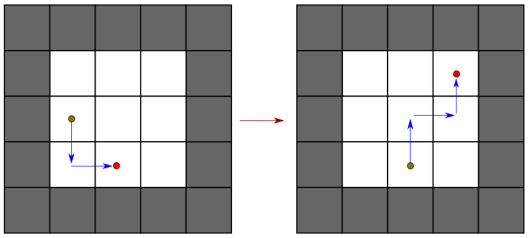
En este segundo caso, una vez realizado el apartado anterior habría que generar las paredes del entorno, zonas a las que el Pacman puede acceder pero le suponen una recompensa muy negativa.
Al realizar esto, el Pacman ya tendría opciones de perder ya que podría chocarse contra ellas y finalizar su recorrido.



C. Pacman con paredes. Varias Comidas

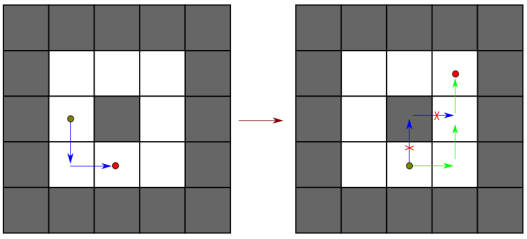
En el mismo entorno planteado en el apartado anterior, generamos *n* comidas aleatorias.

Cada vez que el Pacman llega a una recompensa, se genera la siguiente y así sucesivamente hasta un número finito *n*.



D. Pacman con paredes y obstáculos. Varias Comidas

Por último, generaríamos en el entorno diferentes bloques u obstáculos por los que el Pacman no puede avanzar y perder en caso de que lo intentase. De esta forma, generaríamos un mapa totalmente aleatorio y se irían generando comidas, añadiendo así el caso en el que el Pacman no tenga ninguna posibilidad de ganar.



III. Búsqueda de Información

Para este problema, la idea principal que surgió fue abordarlo con aprendizaje por refuerzo. Al final, cuando un ser humano se enfrenta por primera vez a un juego por el estilo, no sabe como conseguir la victoria y lo único que hace es perder. Una vez se da cuenta de como perder, empieza a explorar otras opciones hasta que en algún momento consigue llegar a la primera victoria e intenta replicarlo en el resto de partidas.

El aprendizaje por refuerzo es lo mas parecido a como jugaría una persona por lo que pensamos que sería lo mas apropiado para este problema.

Dicho método consiste principalmente en dividir el entorno en celdillas, a las cuales llamaremos estados, donde definimos las transiciones posibles y las recompensas que obtenemos a cada paso que damos.

Los estados donde tenemos obstáculos, tienen una penalización muy negativa; y las comidas, tienen una recompensa muy positiva la cual hay que alcanzar.

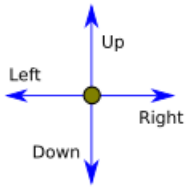
Ambos estados se consideran estados terminales, estados donde si se accede, el procedimiento ha acabado.

El objetivo del Aprendizaje por Refuerzo es extraer que acción es la más indicada en cada estado para obtener la mayor recompensa posible.

IV. Elaboración de la hipótesis.

Para abordar este problema, tendremos que diseñar un algoritmo que pueda generar todas las transiciones y recompensas de cada estado, teniendo en cuenta las 4 acciones que puede realizar (left, right, up, down). Para ello, cada estado tendrá que tener en cuenta sus vecinos.

Para las acciones de movimiento horizontal (left, right), habrá que tener en cuenta su estado siguiente y anterior y para las acciones del movimiento vertical, habrá que tener en cuenta cuántas columnas tendrá nuestro mapa de estados y hacer operaciones matemáticas simples de suma y resta.



A la hora de abordar el problema más básico, lo primero era establecer unas posiciones conocidas y comprobar el funcionamiento del agente en dicho entorno. Por ello se empezó generando posiciones concretas para el caso de un único escenario.

El siguiente objetivo consistía en generar posiciones aleatorias, tanto de la comida como del estado inicial del agente. Para ello, se debía tener en cuenta dos premisas, que el agente estuviera en una posición válida, como fuera de una pared, y que la posición inicial y final no fueran la idénticas.

Continuando con el proceso, para conseguir generar el ejemplo con varias comidas, se debía enlazar el estado final anterior, con la posición de partida nueva, teniendo en cuenta la hipótesis de que, se alcanza siempre esos estados. En caso de no ser alcanzable el estado final, el agente optará por desplazarse sin rumbo claro para evitar la penalización de acabar en un estado pared.

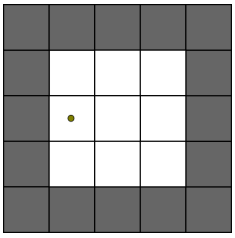
Cabe destacar que en cada entrenamiento, el agente solo tiene en cuenta cómo llegar a un único estado (la recompensa). Es decir, si en la siguiente iteración cambiamos la recompensa de lugar, el agente no sabrá llegar hasta ella sin entrenarse previamente.

Por ello, cada escenario nuevo que se le plantee, debe pasar por un proceso de entrenamiento, de modo que, cada comida nueva generada, equivale a un nuevo entorno para resolver.

V. Prueba de la hipótesis mediante experimentación.

Para probar nuestra hipótesis, hacemos uso de Matlab y sus funciones orientadas al aprendizaje reforzado. En primer lugar generamos las paredes, debemos identificar los estados que formarán parte de estas mediante operaciones matemáticas, siendo ”c” el numero de columnas:

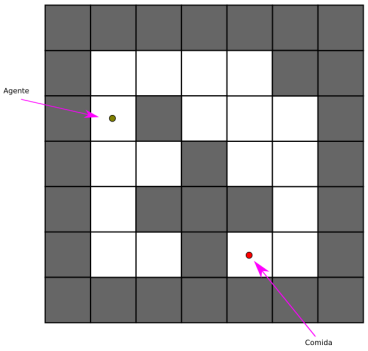
- La pared superior estará formada por los ”c” primeros estados.
- La pared inferior estará formada por los ”c” últimos estados.
- La pared lateral derecha estará formada por todos aquellos estados cuya división entre ”c” tenga un resto 0.
- La pared lateral izquierda estará formada por todos aquellos estados cuyos estados anteriores divididos entre ”c” nos da como resultado un resto 0.



El siguiente paso es generar obstáculos aleatorios que tendrán la misma funcionalidad que las paredes.

Una vez generado nuestro mapa de paredes, obstáculos y posiciones libres, determinamos la posición inicial de nuestro Pacman, teniendo en cuenta que debe estar situado en una posición libre.

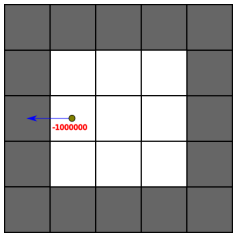
Teniendo en cuenta que realizaremos un número determinado de experimentos, por cada experimento generaremos una comida en una posición libre distinta.



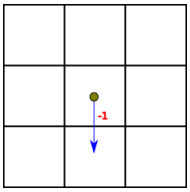
Generaremos el proceso de Markov de la siguiente manera:

- Nuestros estados terminales serán aquellos que hacen que nuestro experimento finalice, es decir, el estado de comida, o los estados correspondientes a obstáculos y paredes.
- Nuestras posibles acciones serán izquierda, derecha, arriba y abajo.

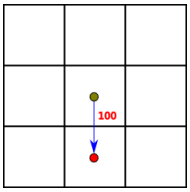
- Por cada estado, determinaremos sus transiciones y sus recompensas:
  - Si el estado actual se corresponde con una pared u obstáculo, todas las transiciones serán a ese mismo estado con recompensa 0.
  - Determinamos las transiciones y recompensas a los estados derecha, izquierda, arriba y abajo de nuestro estado actual.
  - Si alguno de estos futuros estados se corresponde con una pared, la transición a este tendrá una recompensa de -1000000 para evitar visitarlo.



  - Si un futuro estado no se corresponde con una pared o con una comida, la transición a este tendrá una recompensa de -1 por haber gastado un movimiento.



  - Si un futuro estado se corresponde con la comida (estado final) la transición a este tendrá una recompensa de 100, asegurando así que lo visitaremos.



Una vez generado nuestro proceso de Markov, debemos generar el entorno con la función de alto nivel de matlab ”rlMDPEnv”, a continuación se especifica el estado inicial en el que se encontrará el agente en la experiencia en la que se encuentre, se genera la tabla Q y una representación de la misma.

El siguiente paso es la creación del agente utilizando la representación de la tabla Q y por último ya se podría realizar el entrenamiento del agente, en nuestro caso el numero máximo de episodios será 200.

VI. Representación.

Después de entrenar el agente, obtenemos un vector de observaciones u estados por los que ha pasado, aumentándolo en cada simulación.

Cada uno de esos estados, será un argumento a pasar a nuestra función de dibujo, junto con la recompensa actual, la dimensión del entorno y las paredes u obstáculos que nos podemos encontrar en este.

Para dibujar el entorno primero pre-visualizamos una matriz correspondiente a las casillas del mapa del agente ordenadas numéricamente.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Una vez tenemos el entorno y las casillas / estados enumerados, podemos acceder a cualquiera de ellos obteniendo su fila y su columna (usando la función *find()* y buscando la casilla correspondiente).

[row, col] = find(env == position)

El problema principal que teníamos era que las filas y columnas, no representaban exactamente (X, Y) en coordenadas, si no que salían ”volteadas”.

Por ejemplo, para sacar la posición del estado 1, su fila y columna son [1, 1], lo cual se representa de la siguiente manera:

En rojo aparecen las coordenadas que nos dan [fila, columna] y en verde, dónde está realmente el estado 1.

No hemos encontrado ninguna regla para poder solventar este problema haciendo una simple suma o resta. Las dos propuestas que encontramos son las siguientes:

- Voltear el mapa:

  Cambiando el orden de las casillas, conseguimos un efecto similar.

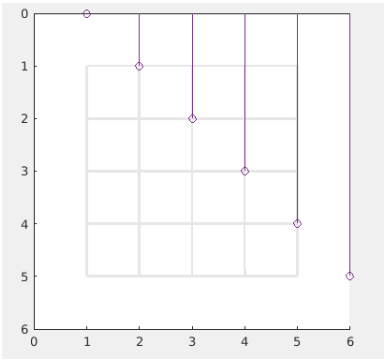
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

  Si ahora queremos [fila, columna] del estado 1, su posición será [1, 4], lo que corresponde con las coordenadas que queremos realmente.

  Esta modificación sirve tanto para las posiciones del Pacman y recompensa como para las paredes y obstáculos si se dibujan con *plot()*.
  No obstante, queríamos utilizar la función *fill()* para rellenar todo el cuadro correspondiente a paredes y obstáculos. Aunque las posiciones fueran correctas,no conseguíamos colocar de manera adecuada los obstáculos aleatorios del entorno.

- Cambiar el orden del eje Y:

  La otra solución que hemos encontrado es decrementar el eje Y desde 0 hasta el número de filas + 2.



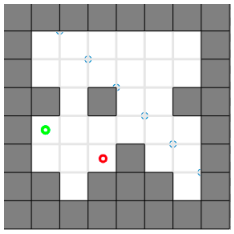
  Esta solución es mucho mas sencilla de implementar y va acorde con el modelo inicial del entorno que teníamos.
  El único fallo que tiene, es que aparecen unas líneas azules que no hemos sido capaces de ocultar.

Para dibujar todos los elementos del entorno, obteniendo su fila y su columna, se pueden representar de manera sencilla utilizando *plot()*.

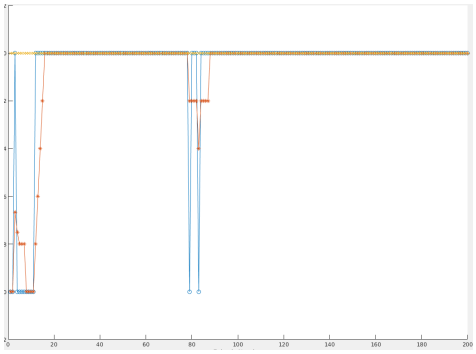
Para los bloques y paredes, hemos utilizado *fill()*, función a la cual se le pasan los vértices del polígono que queremos rellenar y representar.

VII. Análisis de los resultados.

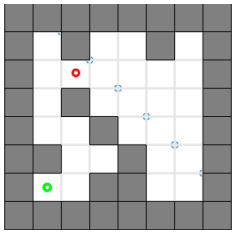
Se obtienen diversos resultados según la dificultad del agente para encontrar la comida.



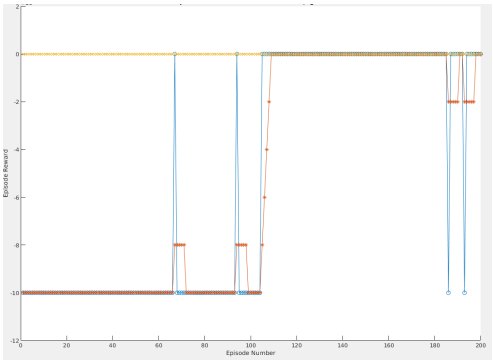
En este caso, vemos como alcanzar la comida es relativamente sencillo y requiere de pocos pasos para lograrse, esto se puede ver reflejado en el entrenamiento:

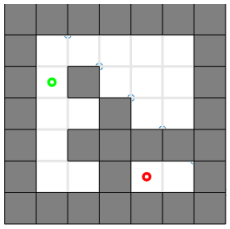


Como podemos observar en la gráfica, nuestro agente tarda pocas iteraciones en encontrar el camino más rápido para llegar a la comida ya que esta se encuentra en linea recta y sin ningún obstáculo de por medio.

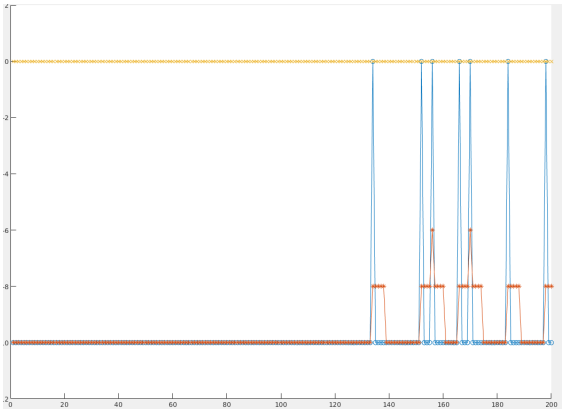


En este mapa el Pacman tiene que recorrer un camino mas complejo para llegar a la comida. Durante el entrenamiento, el agente necesita más episodios para encontrar el camino:





En este último caso, la comida está encerrada por obstáculos y no se puede acceder a ella, con lo que nuestro agente comenzará a moverse sin rumbo lógico evitando las paredes, porque es la mayor recompensa posible que puede obtener, esto se puede observar en la siguiente gráfica:



VIII. Conclusiones.

En conclusión, a través del aprendizaje por refuerzo, se consigue generar un agente capaz de tomar decisiones, en un entorno concreto, que esta determinado por una serie de reglas, y con el fin de resolver un problema con éxito.

En este caso, nuestro agente es un Pacman, y el problema a resolver es la obtención de comida evitando obstáculos.

Dentro de las posibles mejoras, para optimizar el proceso de aprendizaje y extender la funcionalidad, se podría ampliar la percepción del agente, de forma que pudiera obtener más información del entorno (p.ej información de la vecindad).

Esto permitiría generar entornos más complejos, lo que evitaría la necesidad de entrenar al agente por cada comida generada, o dicho de otro modo, permitiría resolver problemas más completos y genéricos.

Cabe destacar que se puede llegar a resolver el problema del juego ”Snake”, partiendo de la base expuesta en este artículo. Aplicando una extensión de la funcionalidad y definiendo las reglas pertinentes.
No obstante, según el orden que hemos seguido para entrenar nuestro agente Pacman, llegaría un punto en el que todos los estados del mapa serían estados terminales.
Es decir, tenemos que tener en cuenta su movimiento y la ”cola” que lleva detrás para no chocarse consigo mismo. Con lo cual, a cada paso que diera el agente, tendríamos que reordenar las recompensas, creando un nuevo proceso de decisión de Markov.

Estos últimos detalles, tendrían que solventarse de una manera distinta a la que se plantea aquí.