Memoria Práctica IA

Adrián Portillo Sánchez

13 de abril de 2015

1. Mapa

Para esta práctica de Inteligencia Artificial, pensé que la mejor forma de optimizar la recogida de trufas sería ir creando un mapa que fuera guardando los muros y mejores caminos para así evitar chocar, y no quedarse perdido en caminos sin salida, o habitaciones con salidas pequeñas.
Para crear este mapa utilicé una matriz 17x17 donde todos los bordes serán igual a -1, esto es así ya que al no saber ni la posición ni la orientación inicial, guardaría todos los elementos independientemente de dónde empiece y hacia dónde comience orientado, además, en los çasos peores", en los que comience en un borde o esquina, mejoraría la recogida de trufas, ya que el robot nunca se saldrá de la matriz, y se ahorraría chocar contra toda una pared o incluso dos, ya que los mapas son de tamaño 8x8, y teniendo en cuenta la posición inicial, tendría que haber 7 elementos a partir de esta, así que si empieza en la posición [8,8] podrá evitar chocar contra algunos muros en los mejores casos.

En las casillas de la matriz incluyo:

0: Terreno no explorado, por defecto inicializa todo el mapa a 0. (En un principio también iba a tener el valor 1 para terreno explorado, pero tras plantear el movimiento decidí dejar otro valor).

-1: Terreno no pasable, tras chocar cambia la casilla con la que ha chocado a -1. (En un principio iba a ser 2001, ya que al haber 2000 turnos nunca llegaría a dicho tamaño y me ahorraría una comprobación, pero en casos realistas el algoritmo debería poder guardar más de 2000 turnos y el mapa perdería legibilidad).

n: Último turno en el que se recorrió la casilla (En principio iba a ser el último turno en el que extrajo, hablaré sobre esto más adelante).

```
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0 -1 -1 -1 -1 -1 -1 -1  0  0  0  0  0 -1
-1  0  0 -1 971 967 964 962 960 958 955 953 -1  0  0  0 -1
-1  0  0 -1 973  -1  -1  -1  -1  -1  -1 950 -1  0  0  0 -1
-1  0  0 -1 975  -1 837 833 818 942 944 948 -1  0  0  0 -1
-1  0  0 -1 977  -1 839 830 820 938  -1 900 -1  0  0  0 -1
-1  0  0 -1 980  -1 841 828 824 935  -1 903 -1  0  0  0 -1
-1  0  0 -1 982  -1 844 927 930 933  -1 905 -1  0  0  0 -1
-1  0  0 -1 984  -1  -1 924  -1  -1  -1 908 -1  0  0  0 -1
-1  0  0 -1 987 989 991 993 995 997 999 911 -1  0  0  0 -1
-1  0  0  0  -1  -1  -1  -1  -1  -1  -1  -1  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

```
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1
-1  0  0  0  0  0  0  0  0  0  0  0 -1 -1 -1 -1 -1
-1  0  0  0  0  0  0  0 -1 944 946 948 951 954 957 960 964 -1
-1  0  0  0  0  0  0  0  0  -1 937  -1  -1  -1 983 980 966 -1
-1  0  0  0  0  0  0  0 -1 931 934  -1   0  -1 986 976 968 -1
-1  0  0  0  0  0  0  0 -1 928 924  -1  -1  -1 989 974 971 -1
-1  0  0  0  0  0  0  0  0  -1 921 918 915 913 991 888 891 -1
-1  0  0  0  0  0  0  0 -1 777 857 854 852 910 995 998 894 -1
-1  0  0  0  0  0  0  0 -1 774 859  -1 848 908  -1 999 896 -1
-1  0  0  0  0  0  0  0 -1 772 862 864 867 906 903 901 899 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1  -1  -1  -1  -1  -1  -1  -1 -1
```

Estas imágenes muestran el mapa creado para el turno 1000 en 2 ejemplos. Estos mapas los crea una función ShowMap(), que generalmente está comentada en el código, y que muestra el mapa en cada paso del agente. La primera imagen muestra la matriz creada en un mapa donde se comienza en el centro del mapa (concretamente el agentmap), mientras que la segunda muestra la matriz para un mapa donde se comienza en una de las esquinas (concretamente el mapa1).

Como se puede observar, todos los elementos de la matriz rondan números entre el 850 y el 990, esto es así porque una vez ha tomado un mapa, se crea un camino y va dejando en ese camino los turnos, de forma que recorre toda la matriz en una cantidad relativamente pequeña de turnos, cosa que ocurre por la lógica que explicaré ahora.

2. Movimiento

El movimiento del robot es relativamente sencillo, este, entre opciones de equivalente valor, se mueve hacia delante, ya que así no perderá turnos girando. En el momento en el que choca, gira por defecto a la izquierda (he preferido no hacerlo aleatorio para así no dejar nada al azar) y guarda un -1 en la casilla con la que ha chocado. Entre opciones donde las posiciones contiguas al robot tienen d istintos valores, el robot elegirá girar en la dirección o seguir hacia delante, siempre mirando hacia la posición donde se encuentra el menor valor, siguiendo una preferencia de pasar por terreno no explorado sobre pasar por terreno explorado, y de pasar por una posición donde pasó hace más tiempo por una posición más reciente, así asegura, recorrer siempre las posiciones más lejanas en el tiempo y reduce las posibilidades de perder trufas por dejarse posiciones descuidadas, aunque siempre perderá alguna.

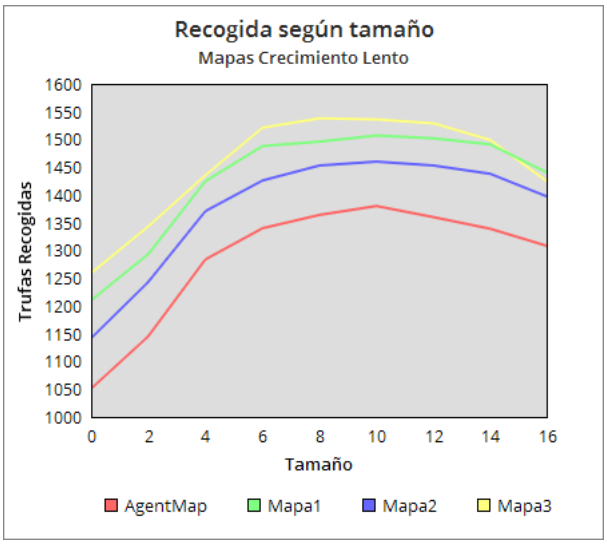
Esta elección la realiza la función getDirection(), que se utilizará cuando no ha chocado y coincide con la dirección en la que se está dirigiendo para seguir adelante y cambiar la posición del robot en la matriz, o si son distintos elige la dirección a la que girar.

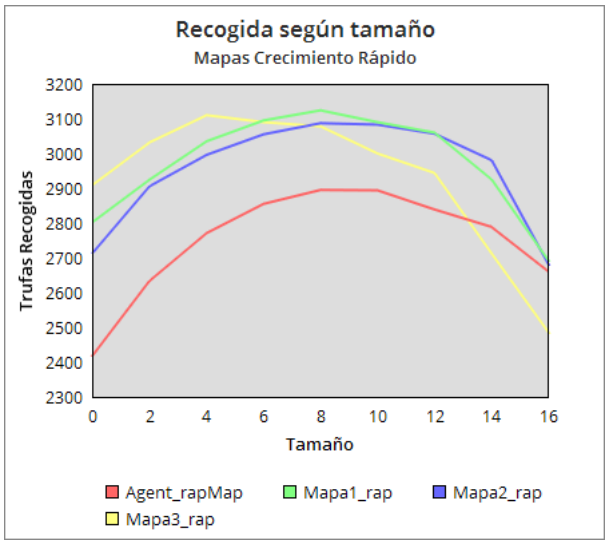
Mi programa le asigna a cada dirección un entero en una variable direction, siendo arriba igual a 0, izquierda a 1, abajo a 2, y derecha a 3. Todo esto relativo a la orientación inicial del robot. Así cada vez que gira a la izquierda sumaría 1 a la variable, y girar a la derecha restaría 1. Así para decidir si girar a la derecha realizaría la comprobación: (direction==0 && getDirection()==3) y la comprobación: ((direction - getDirection()==0), si ninguna de estas se cumple giraría a la izquierda por defecto.

3. Recogida.

En un principio pensé que el robot extraería cada turno en el que había cambiado de posición, y dejaría en dicha casilla el turno de dicha extracción, pero luego pensé que usar la acción SNIFF, y sólo recoger si hubiera una cantidad mínima de trufas, podría optimizar la recogida de trufas y alcanzar números más altos.

El número de trufas mínimo que tiene que haber en una casilla para su extracción lo selecciné a través de un estudio de las trufas recogidas, el cual me mostró que el tamaño mínimo óptimo de recogida es 8.





Estas gráficas representan el número de trufas que recoge para distintos tamaños mínimos de recogida, la primera representa el tamaño mínimo para los mapas de crecimiento lento, mientras que la segunda representa el tamaño mínimo para mapas de crecimiento rápido. Como se puede observar, el mejor número para la mayoría de mapas y casos es 8, excepto algunas excepeciones, como es el caso del mapa3 de crecimiento rápido, que tiene su pico en 4 trufas de mínimo, lo cual es curioso, ya que este es el mayor número de trufas que he podido recoger en cualquier mapa en cualquier situación, llegando a las 3210 trufas, una cantidad de considero bastante notable, teniendo en cuenta que se producen tan solo 2000 turnos.

También intente ver si el agente podía diferenciar entre terrenos baldíos, para tratar de evitar pasar por esas casillas, pero tras meditarlo y ver resultados en distintos mapas, los terrenos baldíos suelen quedar en callejones sin salida y lugares similares (suelen quedar siempre 2 o 3 en los mapas de crecimiento rápido), por lo que como mucho ganaría 1 turno por cada vez que fuera a pasar por ahí, y no son muchas veces las que pasa, unas 5 o 6 durante toda la ejecución, por lo que preferí no complicarme en casos como ese.

Otro caso que pensé implementar era el caso de si olía pero las trufas no llegaban al tamaño mínimo, en lugar de dejar el turno de extracción, dejara un número menor al turno actual en una cantidad proporcional al número de turnos que hubieran pasado. Pensé que le restara aproximadamente un 5 % por cada trufa que hubiera ya en la casilla, pero tras probar la implementación, el programa perdía muchísimas trufas (algo así como 400) no se muy bien por qué, por lo que decidí dejarlo ya que considero que la mejora no hubiera sido tan notable incluso funcionando correctamente. Aunque pienso que implementé esto correctamente, por lo que no sé muy bien por qué provocaba una diferencia tan grande en el número de trufas recogidas.

4. Otras Mejoras Planteadas.

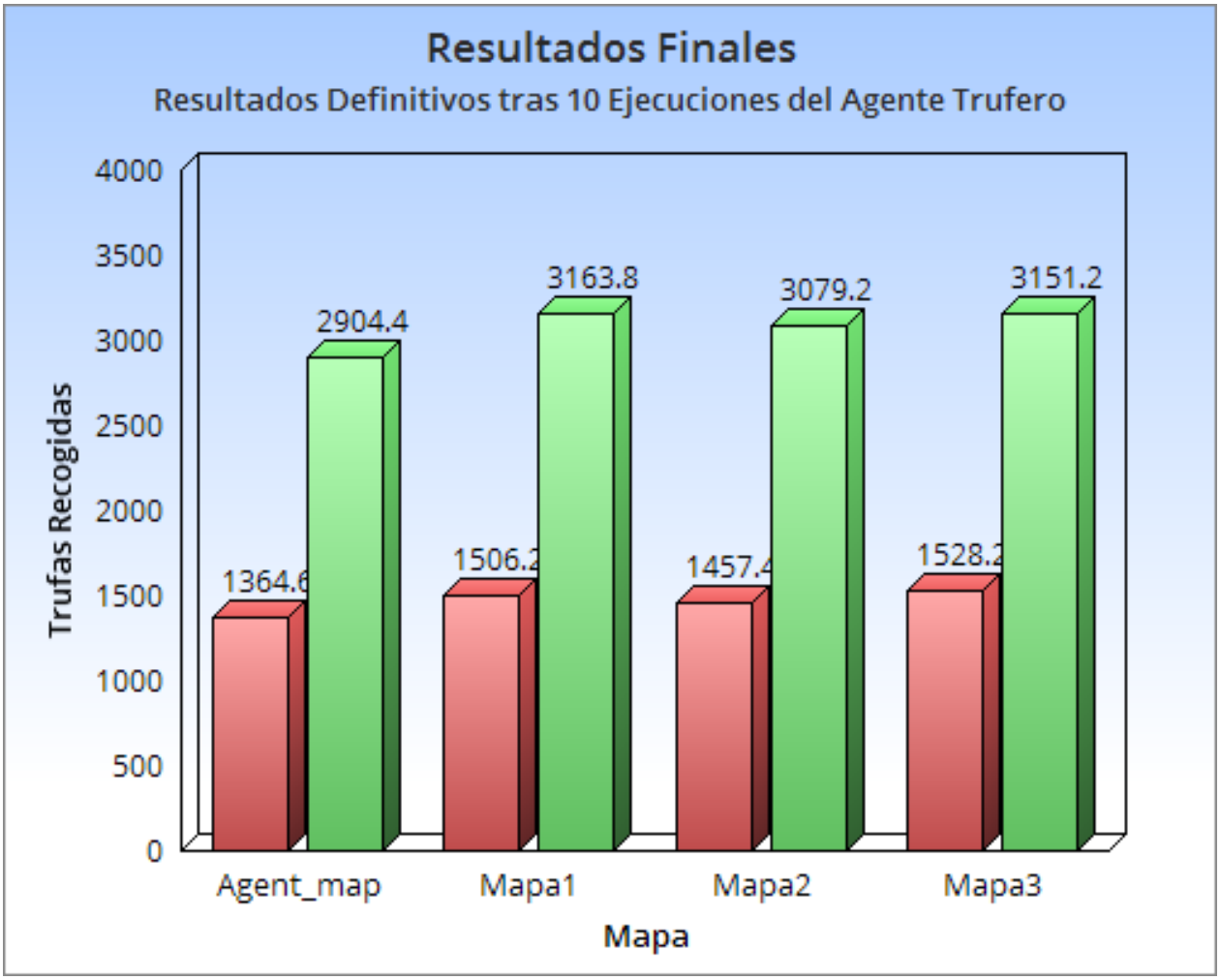
Otras mejoras que planteé sobre el algoritmo definitivo eran, por ejemplo, comenzar a oler a partir de cierto turno, y en los primeros turnos simplemente avanzar, pero al no saber en qué punto pueden comenzar a aparecer trufas (ya que se generan de forma aleatoria), no sabía muy bien en qué turno hacerlo y si esta mejora iba a ser lo suficientemente sustancial. Sobre esta mejora hice algunas pruebas poco concluyentes, que rondaban entre los 200 y 300 turnos para comenzar a extraer, pero los resultados mejoraron poco en algunos mapas, mientras que empeoraron bastante en otros (los de crecimiento rápido), comenzando entre el turno 100 y el 200, los resultados mejoraban en todos los mapas excepto en algunos que empeoraba en unas 10 trufas, pero llega a mejorar en 100 trufas algunos mapas como el mapa3 rápido, que llega a recoger 3150, por lo que lo implementé para que comenzara a oler en el turno 150.

Otra mejora que planteé fue la de detectar mapas de crecimiento rápido, viendo la cantidad de trufas recogidas a partir de cierto turno, para ajustar las variables a los valores más óptimos para el tipo de mapa, pero como los valores para mapas de crecimiento rápido y crecimiento lento más óptimos eran fundamentalmente iguales, preferí dejar esto sin implementar.

Por último, pensé en implementar el algoritmo de forma que si el robot pasaba por dos posiciones que están a 8 casillas de distancia en horizontal o vertical, es decir, que hubiera recorrido el mapa de un extremo al otro, insertará -1 en toda la columna contigua a dichas dos posiciones, esto no lo implementé porque, a parte de parecerme una mejora poco sustancial, complicaría el algoritmo demasiado, y comprobar todos los turnos si había pasado por las posiciones a 8 casillas de distancia me parecía que reducía la eficiencia del programa, además de empeorar la legibilidad del código.

5. Resultados del agente definitivo.

En esta gráfica podremos observar los resultados del agente tanto en los mapas rápidos como en los lentos de forma comparativa, las barras rojas se corresponden a los mapas lentos y las verdes a los rápidos.



Como se puede observar en los mapas lentos los resultados rondan entre las 1300 y 1500 trufas, mientras que en los rápidos rondan entre las 2900 y 3150 trufas, además es de destacar que los resultados son algo más del doble, lo cual quiere decir que el programa se adapta bien tanto a los mapas de crecimiento lento como a los de crecimiento rápido.

No creo que se pueda mejorar mucho más la recogida de trufas, tal vez en como mucho 100 trufas en el caso de los rápidos y 50 en los lentos, ya que pienso que no pueden crecer muchas más de las que el algoritmo recoge, ya que el algoritmo no se deja ninguna casilla baldía en los mapas lentos y se deja 2 o 3 como mucho en los rápidos, por lo que las mejoras en los resultados no creo que sean mucho mayores de las que el algoritmo da, sobre todo teniendo en cuenta que sólo recoge si hay 8 trufas o más, igual siguiendo un camino más eficiente y subiendo ese umbral a un número mayor se puede optimizar a unos números mayores, pero con mi algoritmo estos son los mejores números que se pueden obtener, no sé si habrá un algoritmo reactivo mejor, o si se puede conseguir de forma deliberativa unos resultados mucho mayores. Pero en definitiva, estoy contento con mis resultados y los considero bastante notables.