UNIVERSIDAD COMPLUTENSE MADRID

Examen Extraordinaria de Sistemas Inteligentes

29 de junio de 2023

Instrucciones generales para realizar el examen:

- **Pon tu nombre en todas las hojas del examen**.
- Escribe con letra clara y sin faltas de ortografía
- No se permiten dispositivos electrónicos salvo una calculadora
- **Contesta a las preguntas en el hueco que se proporciona**. Puedes apoyarte en tantas hojas en sucio adicionales como quieras, pero las respuestas deben ir en el examen para evitar que se pierdan. Las hojas de respuesta pueden estar escritas por las dos caras y podéis utilizar el reverso del enunciado del examen en caso de que hiciese falta.
- El tiempo para la realización del examen es de **2.5h**.

Problema 1 (2,5 puntos)

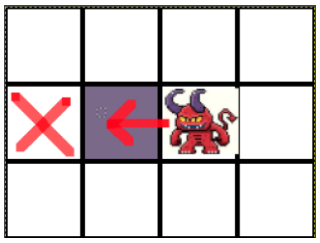
El mundo de los Wumpus es un problema de juguete muy conocido en inteligencia artificial ya que se ha usado típicamente como campo de pruebas de diferentes algoritmos. El mundo consiste en una cuadrícula de tamaña NxM cualquiera que puede contener diferentes elementos en cada casilla.

- **Aventurero**: El aventurero entra en el mundo desde la casilla de entrada y sale por la casilla de salida. Las acciones del mismo son moverse por el escenario y disparar una flecha en cualquiera de las 4 direcciones (no diagonales). La flecha por simplicidad es de velocidad instantánea. El aventurero puede morir si:
  - Cae por un agujero
  - Un Wumpu se lo come (está en la misma casilla que el Wumpu)
  - El aventurero puede tener más de una vida. Si muere vuelve a la casilla de salida mientras tenga vidas disponibles.
- **Wumpus**: Es un monstruo. En el problema clásico este no se mueve y solo se le percibe por su hedor, pero en nuestra versión del problema el Wumpu tiene movilidad.
- **Tesoros**: Son puntos para el aventurero. Si sale con más tesoros recogidos, su puntuación será mayor.
- **Pozos**: Si el aventurero cae en un pozo, pierde una vida. En algunas versiones del juego son detectables por la presencia de viento cerca del mismo.
- Casilla de **entrada** y de **salida**.
- Paredes intransitables para el Wumpu o para el aventurero

A veces puede contener otros elementos, pero normalmente estos son los elementos mínimos.

Queremos programar el comportamiento de un Wumpu, este se puede describir como sigue:

- El Wumpu se mueve una casilla por turno. En la simulación del mundo se mueven a la vez por lo que no sabe la dirección a la que se moverá el aventurero. Cada uno toma una acción y estas se ejecutan todas a la vez.
- El Wumpu tiene una visión de 1 casilla por lo que, si ve al aventurero en la casilla adyacente, sabrá que está ahí.
- Si el aventurero está mas de una casilla debe usar su sentido del olfato. El aventurero cuando pasa por una casilla deja un rastro oloroso que puede durar un cierto tiempo. El Wumpu es capaz de detectar el rastro oloroso a 3 casillas de distancia. Si detecta diferentes rastros olorosos irá hacia aquel más intenso. La intensidad puede ser porque el rastro sea más reciente o porque esté más cerca. El olor atraviesa paredes.
- Si el Wumpu no detecta ni por visión ni por olor al aventurero, se moverá aleatoriamente.
- Si el Wumpu entra en una casilla de tesoro desplaza este en la dirección de movimiento, ocupando el Wumpu la posición del tesoro
- Si un Wumpu cae en un agujero, no muere, pero pierde tres turnos en salir del agujero. Es decir que, si un Wumpu cae en un agujero, en los dos siguientes turnos no se moverá y en el siguiente turno podrá colocarse en cualquier casilla adyacente. El Wumpu tiene una inteligencia limitada y no percibe como peligroso al agujero, así que a nivel de percepción obvia su existencia, por lo que no puede esquivarlo, pero si detecta cuando se ha caído.
- El Wumpu muere si recibe el impacto de una flecha.
- El Wumpu se come al aventurero si cae en su casilla
- Para evitar que el Aventurero intente esquivarle constantemente, el Wumpu tiene una habilidad especial que es la embestida. Este movimiento especial solo lo puede realizar el Wumpu si el aventurero está en una casilla adyacente. Si realiza este movimiento en la siguiente casilla donde se mueva el Wumpu en el siguiente turno, podrá noquear al aventurero impidiendo que este se mueva en el siguiente turno. Este movimiento no lo puede ejecutar siempre. Tiene un tiempo de espera de 4 turnos. Podemos ver un ejemplo en la siguiente imagen donde el Wumpu se mueve a la casilla gris con acometida y aturde al aventurero si este está en la casilla marcada con una x roja.



Se pide:

1. ¿Qué tipo de agente es el Wumpus? Reactivo simple, reactivo basado en modelo, deliberativo basado en objetivo... Razona la respuesta. (0.5 p)

Es un agente reactivo basado en modelo. No puede ser reactivo simple porque por ejemplo cuando cae en un agujero debe saber cuántos turnos está en el antes de salir. Por tanto, tiene que recordar que está en el estado “dentro del agujero”. Algo parecido sucede con la Embestida.

2. Modela el agente con el formalismo que consideres más apropiado. (1p)

Usaremos como formalismo la máquina de estados. Las variables de entrada o percepción serían:

- Visión de 1 casilla de distancia.
- Olfato de 3 casillas de distancia.
- Cae en el agujero.
- Cae en aventurero

- Cae en Tesoro
- Flecha

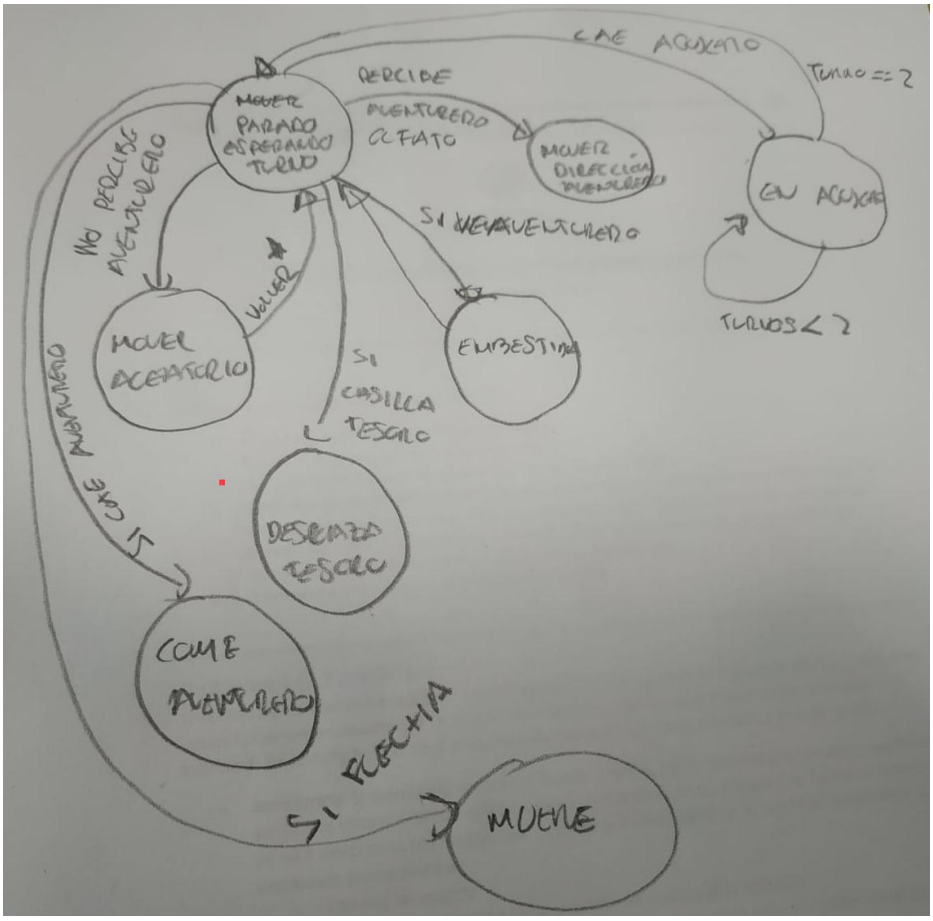
Actuadores:

- Desplazar tesoro
- Comer Aventurero
- Embestida
- MoverseDirección

Variables:

- Turnos en el agujero.

Creamos un estado denominado: Parado Esperando turno, del que parten la mayoría de las transiciones. Esto simplifica la máquina de estados en un sistema donde el agente no está todo el tiempo moviéndose si no que realiza una acción cuando el entorno se lo permite. El Wumpu podría modelarse entero como un agente reactivo simple y buena prueba de ello es que las acciones no tienen memoria y siempre vuelven al estado parado. Excepto la acción En Agujero, que tiene una transición sobre sí misma porque debe recordar que está en ese agujero para seguir esperando hasta completar los turnos de penalización. Así pues, salvo ese estado el resto de las transiciones de vuelta desde los estados actuadores al estado de parado esperando turno son transiciones sin condición y no se añaden para simplificar el gráfico (sólo lo hemos añadido en Mover aleatorio con un *)



3. Imagina que hay varios Wumpus en el juego y estos necesitan comer fruta cada 20 turnos para no morir. La fruta es un elemento que se genera aleatoriamente en el mapa. Los Wumpus no se comunican directamente entre ellos, pero pueden detectarse unos a otros en base a su rastro de olor. El objetivo de los Wumpus es que al menos uno de ellos se coma al aventurero ¿Qué tipo de colaboración realizan? (0.5p)

Sus objetivos son compatibles, los recursos podrían ser insuficientes ya que la fruta la necesitan para sobrevivir individualmente. La habilidad para capturar al aventurero aparentemente es completa, por lo que sería Obstrucción.

4. Partiendo de la idea anterior de que se comunican usando el olor, explica que tipo de comunicación están realizando. (0.5p)

La comunicación que realizan es de tipo señal. No tiene un destinatario claro, son marcas que se van dejando en el entorno y que puede leer cualquier agente.

Problema 2 (2.5 p)

Continuando con el mundo anterior de los Wumpus, ahora queremos modelar el comportamiento del aventurero. El agente aventurero debe maximizar su recompensa obtenida antes de salir del escenario, sin que muera el agente. Cada tesoro que recoja sumará 1 punto. No sabe dónde están los tesoros a priori, así que tiene que explorar el mapa para encontrarlos. El aventurero debe poder adaptarse a cualquier configuración del mapa. El aventurero puede ver a 2 casillas de distancia al Wumpu y a los tesoros, pero no puede ver los agujeros. Estos sólo pueden intuirlos de forma que si está en una casilla adyacente a uno de ellos detectará una brisa, pero no sabe en cuál de las 4 casillas adyacentes se encuentra el agujero.

En cuanto a disparar la flecha, si el agente dispara una flecha no se mueve. Tiene un número de flechas limitado que varía en función del mapa. Del mapa sabemos inicialmente solamente su tamaño y donde empezamos.

Se pide:

1. ¿Qué tipo de agente es? Razona la respuesta (0.75P)

Agentes que aprende. Ya que debe aprender a resolver el mapa que no conoce. Debe por tanto primero aprender el entorno y actuar en consecuencia una vez ha aprendido como es el entorno y que acciones debe realizar para maximizar su objetivo.

2. Explica que técnica utilizarías en el modelado de este agente y explica por qué. No hace falta implementarlo solo describir por encima como lo harías (1p)

Es un problema típico de aprendizaje por refuerzo. Entorno desconocido y una recompensa futura que obtener (los tesoros). Lo modelaría de la siguiente forma:

Hay una primera representación más o menos obvia que sería asignar el espacio de estados del agente como una matriz que representa el mundo dividido por casillas (asumo que se conoce la dimensión del mundo)

Para construir la table Q necesitamos saber también las acciones que puede realizar el agente. En este caso son mover en las 4 direcciones (4 acciones) y disparar la flecha también en las 4 direcciones. Así que la matriz a aprender tendría MxN filas siendo MxN la dimensión del mundo y 8 columnas.

En cuanto a la función de recompensa, recibiríamos 1 punto de recompensa por cada vez que recoge un tesoro y por ejemplo 10 puntos si consigue salir del tablero. Tendría -10 punto por cada vez que cae en un agujero. Cómo los Wumpus se mueven, con esta representación no podríamos aprender a esquivarlos.

Hay otra representación más interesante y compacta y es coger la percepción del agente (las 12 casillas que ve y las 4 casillas de detección de la brisa) y generar todas las combinaciones de estados posibles de si ve o no ve a un wumpu o detecta o no detecta la brisa y las 8 acciones que se pueden realizar. La recompensa en este caso sería 10 si llega a la salida (Asumimos que la puede ver), -10 si cae en un agujero y -1 si se lo come un Wumpu. Llegar a la salida serían 10 puntos, matar a un wumpu te daría 1 punto y coger un tesoro también te podría dar 1 punto. Podríamos dar más puntos al tesoro si queremos que priorice la recogida de tesoros a matar a los Wumpus.

3. Imagina que modelas el agente usando trazas creados por un usuario donde se guardaría el estado del mundo y la acción tomada. Se pretende que el agente imite el comportamiento del que generó las trazas. Describe una métrica que evalúe si el agente se está comportando correctamente (0.75p)

Para ver si un agente se comporta igual que el usuario hay que grabar una partida del usuario con un entorno idéntico al que luego se le va a dar al agente /Mismo mapa) y con un comportamiento similar de todo el sistema. Y evaluar si en entornos de ejecución similares se han realizado la misma acción. Podríamos utilizar por ejemplo KNN con K = 1. O bien, una alternativa es usar la distancia de edición para comprobar las trazas de acciones que han realizado el agente y el humano.

Problema 3 (2.5 p)

Disponemos de 500 partidas del juego del Wumpus donde usuarios humanos han controlado al aventurero y tomado las decisiones oportunas en cada caso. De cada partida tenemos todos los movimientos que ha realizado el agente. Toda esta información está almacenada en un dataset con los siguientes campos de entrada:

- Lo que percibe visualmente en cada una de las 12 casillas de rango de visión que tiene (recuerda que son 2 casillas de rango de visión en cada dirección, más diagonales)
- Si percibe viento en las 4 casillas de rango de percepción de viento
- El número de vidas que le queda
- Número de flechas que le quedan
- El número de tesoros que lleva acumulado
- El número de Wumpus que ha matado (asumimos que puede haber varios)
- La acción que realizó el humano en el siguiente turno.
- Si en esa partida el jugador humano consiguió llegar a la salida con al menos 1 vida.
- La recompensa final de tesoros que en esa partida el humano consiguió.

Recordemos las acciones que puede realizar el aventurero. Moverse en las 4 direcciones y disparar en las 4 direcciones. El número de flechas y de vidas es limitado.

Se pide:

1. ¿Qué trasformaciones harías a los datos para poder procesarlos con un **Árbol de Decisión** que permitiese al agente aprender un comportamiento? Razona la respuesta (0.5p)

Lo primero es identificar de los datos del dataset cuál sería la clase. En este caso “La acción que realizó el humano en el siguiente turno”. El resto serían potencialmente datos de entrada.

Sin embargo, las dos últimas entradas nos dan una idea de cómo de buena fue la decisión del humano y recordemos que en este ejemplo no se pide imitar a un humano, si no aprender a que el agente resuelva el juego en base a lo que los humanos han hecho. Por lo que estos atributos en verdad nos están indicando que trazas son buenas para nuestro agente y cuáles no. Hay por tanto dos estrategias, uno incorporarlos como datos de entrada y que el modelo se dé cuenta que ciertos ejemplos quizás no son muy buenos, o usar esta info para limpiar el dataset y quedarnos con aquellas trazas que han dado buenos resultados al jugador humano (que son indicativo de que el jugador sabe jugar al juego y que su partida ha sido buena)

Los árboles de decisión en principio puede trabajar con valores categóricos. Así que no es necesario hacer One hot encoding salvo que la implementación del mismo lo requiera por exigencias de dicha implementación. Pero a priori, los árboles de comportamiento en general no lo necesitan (SKITLearn por ejemplo si pero nuevamente es un detalle de implementación de ese algoritmo). Por lo tanto, no haría falta hacerlo. Si alguien en el examen lo dice puede ser por la experiencia con la práctica y por tanto no se penaliza, pero es importante tenerlo en cuenta. Tampoco hace falta normalizar los datos a priori.

2. ¿Qué configuración de capas de entrada, capas de salida y función de activación elegirías si lo quisiéramos modelar con un **Perceptrón Multicapa**? Razona la respuesta (0.5p)

Si asumimos que quitamos la clase y los dos últimos atributos, tendríamos 20 atributos de entrada (12 de visión, 4 de brisa, y 4 del resto), pero aquí sí que necesitamos codificarlo con one-hot-enconding por lo que podrían ser más en función del número de categorías que tuviésemos en las 12 casillas de visión y las 4 de viento. En principio las de viento son sí o no por lo que serían 4. En las 12 de visión deberíamos poder ver la salida, a los Wumpus y los tesoros. Por lo que tendría 4 posibles valores: nada, Wumpu, tesoro, salida. Necesitaríamos 4 variables por cada atributo, con lo cual tendríamos 48 entradas de percepción y 12 del resto, un total de 60. La función de activación podría ser cualquiera en las capas ocultas y en la de salida Softmax.

3. Queremos crear un modelo que prediga únicamente lo que pasará en el siguiente turno dado el entorno antes descrito y la acción a ejecutar. El resultado puede ser (nada, pierde una vida, pierde la partida o consigue un tesoro)

Tenemos dos modelos que han sido entrenados con este dataset. Sus resultados se muestran a modo resumen en las dos siguientes matrices de confusión. Calcula el **accuracy** de cada modelo (0.5p). De los modelos propuestos y viendo sus resultados ¿Con cuál método te quedarías? Razona la respuesta (0.5p)

Árbol decisión	Nada reales	Pierde vida reales	Pierde partida Reales	Consigue tesoro reales
Nadas Modelo	90	2	5	0
Pierde vida Modelo	1	40	5	0
Pierde partida modelo	1	2	10	0
Consigue tesoro modelo	1	2	5	20

Perceptrón Multicapa	Nada reales	Pierde vida reales	Pierde partida reales	Consigue tesoro
Nada Modelo	85	1	0	5
Pierde vida Modelo	1	45	5	0
Pierde partida modelo	2	0	20	0
Consigue tesoro modelo	5	0	0	15

El accuracy del árbol de decisión es: total => 184 ejemplos. Correctamente clasificados 90+40+10+20 = 160 160/184 = 0.869.

El Accuracy del perceptrón multicapa es: 165 / 184 = 0.8913

El perceptrón es más preciso, pero es un algoritmo de caja negra. El árbol de decisión es de caja blanca. Si la diferencia de precisión de un 4% es poco significativa, me quedaría con el árbol.

Pero analizando la matriz de confusión, el perceptrón es más preciso en prever situaciones más peligrosas (perder una vida, o perder la partida) por lo que podríamos elegir el perceptrón si lo que queremos es maximizar el número de partidas donde el aventurero no muera o pierda pocas vidas.

4. Queremos crear un sistema que permita aprender a jugar al juego de los Wumpus pero sin necesidad de tener un dataset. La idea es que con un simulador y grabando las imágenes de cada uno de los turnos, el sistema pueda aprender lo que está sucediendo en el juego y construir por detrás algún sistema razonador. Explica brevemente como lo implementarías sin entrar en detalles de bajo nivel. Se valorará positivamente en la puntuación que se planteen diferentes alternativas (0.5 p).

Lo que queremos es construir un sistema de machine learning que analice la entrada como una imagen. A priori una red convolucional que procese la imagen y aprenda a detectar los diferentes tipos de parámetros sería lo más intuitivo a probar y permitir que esa red extraiga las características. También podríamos crear un autoencoder.

Se podría intentar añadir algún elemento recurrente debido a que los frames de los juegos son muy parecidos y dependen uno de otros. Habría que realizar diferentes pruebas a ver cuál funciona mejor, pero estas serían a priori las primeras aproximaciones para probar.

Problema 04 (2,5p)

Queremos utilizar A* para resolver el siguiente problema dentro del mundo de los Wumpus.

Queremos encontrar la forma más optima de llegar de la entrada a la salida, pero recogiendo todos los tesoros posibles. Es decir, el algoritmo debe trazar la ruta más corta que pase por todos los tesoros. La ubicación de los tesoros en este caso es conocida y para abrir los cofres de tesoro hay diferente coste (la complejidad para forzar la cerradura del cofre no es siempre la misma). Hay tres tipos de cofre, el cofre de coste 1 unidad de tiempo, el de dos unidades de tiempo y un cofre que tiene un coste aleatorio con una probabilidad desconocida por cofre. Para poder abrir un cofre de tesoro el aventurero debe estar sobre la casilla del cofre y después ejecutar la acción de abrir el cofre.

En esta versión del problema no hay Wumpus o son estáticos. Por lo tanto, lo único que hay son zonas por donde no puedes transitar (paredes, agujeros, Wumpus, etc) y zonas de coste de movimiento que puede ser diferente de 1 unidad. Un ejemplo de configuración del tablero es la siguiente:

Donde:

- El cofre de color normal tiene un coste de apertura de 1
- El azul de coste de apertura de 2
- El rojo de coste de apertura aleatorio.
- Casilla negra intransitable
- Casilla azul coste de tránsito de 2

El agente conoce la ubicación de los cofres y su tipo, así como la ubicación de la salida y la entrada.

Se pide:

1. Formaliza el problema como un problema de búsqueda. (1p)

Valores para almacenar

- Posición X e Y del Aventurero (casilla)
- Lista con los tesoros disponibles que guarde su posición y si han sido o no recogidos.

Estado inicial

Aventurero.Posición = Entrada.Posición

Todos los tesoros marcados como no recogidos.

Estado final

Todos los tesoros marcados como recogidos

Aventurero.Posición == Salida.Posición

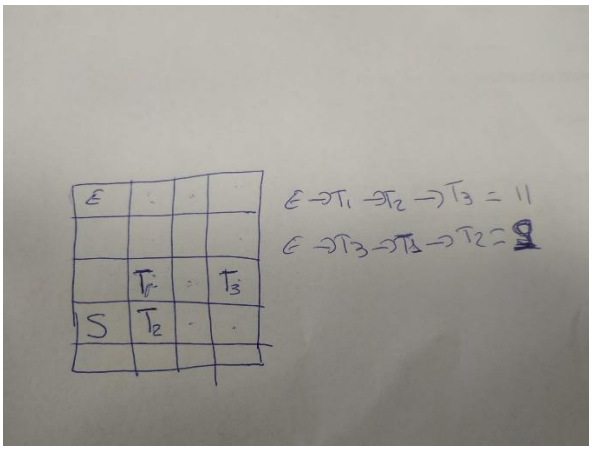
Operadores

- Mover izquierda
- Mover derecha
- Mover arriba
- Mover Abajo
- Abrir cofre.

2. Define una función heurística admisible (0.75) y otra no admisible (0.25p).

Buscamos el cofre más cercano al aventurero (entre aquellos no recogidos). Calcular Manhattan desde la posición del aventurero al más cercano + manhattan desde ese cofre al más cercano al primer cofre recogido + la distancia desde ese ultimo cofre al mas cercano a este y así sucesivamente más la distancia del último cofre a la salida. Más la suma de los costes de abrir un cofre que en el caso mínimo será de 1

Si sabemos el tipo, podemos sumar el coste real del cofre excepto el aleatorio que tendríamos que ponerlo a 1 que es el mínimo. NOTA: El orden en el que se recorren los puntos puede influir en la optimalidad del algoritmo. Esta función podría no ser admisible en según que configuraciones de tablero. Para estar seguro de que la heurística sea admisible no podemos asumir un orden a pesar de que lo que hemos planteado sea la mejor opción para la mayoría de las situaciones. Imaginemos este ejemplo



Por lo que en este caso sería no admisible. Para garantizar que fuese admisible la heurística tendría que calcular la distancia más corta que hay entre cada elemento y el elemento mas cercano a este. Por ejemplo, en este ejemplo, el elemento más cercano a E es T1 = 3. El más cercano a T1 es T2 que es igual a 1. De T2 es igual a 1 (que es T1) Y a T3 es 2 (que es igual a t1) y el más cercano a la salida es T2 que sería 1 = 3+1+1+2 + 1= 8. Esta si sería admisible.

3. Describe los 6 primeros descendentes que expandirá A* usando la heurística admisible que has definido.

En este caso el estado está compuesto por la posición X,Y = (1,0) (asumimos coordenada 0,0 en la esquina superior derecha)

Y tres cofres con la siguiente info: (0,1,False) (3,2,False) (3,0,False) => [(1,0), (0,1,False) (3,2,False) (3,0,False)] Pero los valores que cambian son los booleanos así que por simplificar nuestro estado será lo siguiente: [(1,0), False, False, False]

Recordemos que la heurística es la distancia a cada uno de los tesoros, desde el elemento más cercano. El tesoro más cercano a la posición del agente recibirá como distancia la distancia desde el agente, el resto se calculará la distancia desde el tesoro más cercano. Para calcular la distancia a la salida se calculará desde el tesoro más cercano no recogido.

TA = Tesoro amarillo.

TR = Tesoro rojo

TV = tesoro verde.

Distancia a TA (desde E) = 2, Distancia a TR (desde TA) = 2, Distancia a TV (desde TA/TR) = 4, Distancia a S (desde TV) = 2. La suma es 10 + Coste de abrir los cofres que es 2+1+1 (el coste del aleatorio es mínimo 1) = 14.

La tabla quedaría así:

ID	0	1	2	3	4	5	**6**	7	8	9	10	11
Estado	1,0 0,0,0	0,0 0,0,0	2,0 0,0,0	1,1 0,0,0	0,1 0,0,0	1,2 0,0,0	**0,0 0,0,0**	1,2 1,0,0	1,1 1,0,0	0,0 1,0,0	1,2 1,0,0	1,0 1,0,0
G	0	2	1	1	2	3	**4**	3	4	5	6	5
H	14	13	15	13	12	14	**13**	11	10	10	9	11
Padre	-	0	0	0	3	3	**4**	4	7	7	8	8
Open	T1	T2	T2	T2	T3	T3	Peor	T4	T5	T5	T6	T6
Close	T1	T6		T2	T2		que 0	T4	T5			

Donde ID es un identificador para luego saber cuál es el padre, Open es el instante de tiempo T que se metió el nodo en la lista de abierta, close es el instante de tiempo T que se metió el nodo en cerrada, G el coste acumulado, H la heurística antes descrita y el padre el id del padre que generó los sucesores.

En T6 tocaría expandir los sucesores del nodo 1 que tendría una función f de 15, lo mismo que el nodo 10 y el 9. Cualquiera podría ser pero hemos elegido el que más tiempo lleva en la lista. El nodo 6 marcado en negrita es otro camino por el que hemos llegado a un nodo no cerrado, pero que en este caso el coste es mayor que el que había y no se ha actualizado su coste.