El mundo de Wumpus

Inteligencia Artificial
Universidad de Las Palmas de Gran Canaria

Fecha: 17 de febrero de 2.003

Javier Honorio Casado Fernández
jhcasado[AT]gmail[DOT]com

José Ángel Montelongo Reyes
ja.montelongo[AT]gmail[DOT]com

Resumen

En el siguiente documento se describe el problema del *BuscaTesoros* en el mundo de *Wumpus*, cuáles han sido las diferentes decisiones de diseño que se han tomado así como las restricciones que se han tenido en cuenta y que ampliaciones se han contemplado. También se detallan los diferentes módulos que forman el programa: *Wumpus.lsp, BuscaTesoros.lsp, Entorno.lsp, util.lsp y main.lsp*. Así como una pequeña guía de usuario y una descripción de los resultados obtenidos para cada una de las diferentes heurísticas empleadas.

1 Presentación del Problema

En los sucesivos apartados se describe la representación y solución que hemos desarrollado para resolver el mundo del *Wumpus*. Cualas han sido los diferentes ficheros que se han escrito en *LISP*, así como su estructura de diseño. También se añade un apartado con los diferentes resultados que se han obtenido para diferentes ficheros de configuración.

2 Análisis del Problema

El problema que hemos de resolver es el de un agente que se mueve en un entorno simulado. Este agente esta dotado de una serie de sensores, detallados en la Tabla 1, que le permiten obtener información del entorno. También dispone de una serie de acciones que le permiten interactuar con el entorno, descritas en Tabla 2. El entorno por el que el agente a de moverse representa una cueva de la que no conoce nada a priori. Dicha cueva está plagada de precipicios y de un *monstruo*, el *Wumpus*, que en caso de encontrarse con el agente lo destruirá.

El agente tiene como objetivo recorrer la cueva, buscar un tesoro, cogerlo y abandonarla usando el menor número de turnos posibles, evitando caer por alguno de los precipicios o ser destruido por el *Wumpus*. Para ello, sólo puede hacer uso de los sensores descritos en la Tabla 1 y de las acciones descritas en la Tabla 2.

Sensor	Descripción
Brillo	*Devuelve cierto si el agente se encuentra en una casilla que contiene el Tesoro.*
MalOlor	*Devuelve cierto si el agente se encuentra en una casilla contigua (4 vecinos) al Wumpus.*
Brisa	*Devuelve cierto si el agente se encuentra en una casilla contigua (4 vecinos) a un precipicio.*
Golpe	*Devuelve cierto si enfrente del agente se encuentra un muro.*
Grito	*Devuelve cierto si el agente escucha un grito de un Wumpus al morir.*
Salida*	*Devuelve cierto si el agente se encuentra en una casilla de salida.*

***Tabla 1**. Sensores del agente*.

Acciones	Descripción
Avanzar	*El agente avanza una posición.*
Saltar	*El agente sale de la cueva.*
GirarDer	*El agente efectúa un giro a la derecha.*
GirarIzq	*El agente efectúa un giro a la izquierda.*
RecogerTesoro	*El agente recoge el tesoro que se en cuentra en su mima casilla.*
DisparарFlecha	*El agente dispara una flecha.*

***Tabla 2**. Acciones del agente.*

2.1 Añadidos

Con respecto a la propuesta inicial, hemos añadido un sensor más, para darle más juego al problema. Este sensor es el de *Salida*, que devuelve cierto si el agente se encuentra en una casilla de salida. Gracias a este sensor podemos experimentar con un agente que puede comenzar en cualquier parte de la cueva y, aparte de buscar el tesoro, también deberá encontrar una salida. Esta opción también nos permite colocar más de una salida de la cueva, así como permitir que varios agentes entren en la cueva desde varias posiciones diferentes y evaluar su comportamiento.

Otro cambio que nos ha parecido interesante es de permitir la existencia de varios tesoros en la cueva, permitiendo que los agentes, en el caso de existir varios exploren la cueva y compitan hasta llenar un tope de tesoros que pueden cargar. Los dos cambios se ajustan perfectamente al planteamiento inicial del problema, dado que sólo debemos colocar un tesoro y la salida en la posición (1, 1) para encontrarnos con el problema inicialmente descrito.

2.2 Restricciones

En principio no tomamos ninguna restricción con respecto a como deben de estar colocados los precipicios o como ha de moverse el

Wumpus. En el caso del monstruo suponemos que posee las mismas acciones para desplazarse que el agente, es decir, puede girar a la derecha, a la izquierda y avanzar. Con respecto a los sensores, el habitante de la cueva posee un único *sensor* que le informa de que es lo que hay justo enfrente de donde él está situado. Otro dato a tener en cuenta con respecto al movimiento del *Wumpus* es que no se ve afectado por la presencia de un precipicio, es decir, pasa por encima de ellos sin caer al vacío. También apuntar que el movimiento de *Wumpus* debería estar limitado, es decir, no sería recomendable que el *Wumpus*, realizara un movimiento en cada turno, puesto que cuenta con ventaja de movimientos con respecto al agente y posee un instinto que le informa de que es lo que hay en cualquier posición de la cueva.

Con respecto al agente, es él quien debe encargarse de de gestionar cuanto es el máximo tiempo que puede estar en la cueva. Esto es necesario dado que en el caso de que el agente se encuentre con un problema irresoluble, o que no sea capaz de resolver, debería, superado cierto número de turnos, abandonar la cueva aunque no haya cumplido su objetivo.

3 Decisiones de Diseño

Para desarrollar el diseño necesario que nos ha permitido representar y resolver el problema hemos optado por un análisis modular, tomando la solución clásica para resolver los antiguos juegos de ordenador, es decir, tenemos un mapa y sobre este mapa dibujamos los diferentes personajes, *sprites*. Para ello primero hemos identificando las diferentes entidades:

- ***Buscatesoros***, representación del agente *Buscatesoros*
- **Wumpus**, representación del monstruo.
- ***Entorno***, representación del entorno, contiene el mapa y todos los actores: agentes y *BuscaTesoros*.
- ***Util***, funciones útiles, utilizadas por las diferentes entidades.

Posteriormente hemos añadido las diferentes relaciones, reflejadas en el diagrama 1, que poseen cada una de las entidades.

Dado que tanto el agente buscatesoros como el *Wumpus* comparten atributos y métodos, lo ideal hubiese sido crear una clase padre, que podríamos denominar *agente*, de la que heredasen ambas entidades. No hemos optado por esta opción dado que *LISP* es un lenguaje eminentemente funcional, no orientado a objetos como *ADA* o *Java*, y aunque posee clases, no nos ha parecido oportuno decantarnos por esta opción. Por tanto hemos optado por un diseño estructurado clásico.

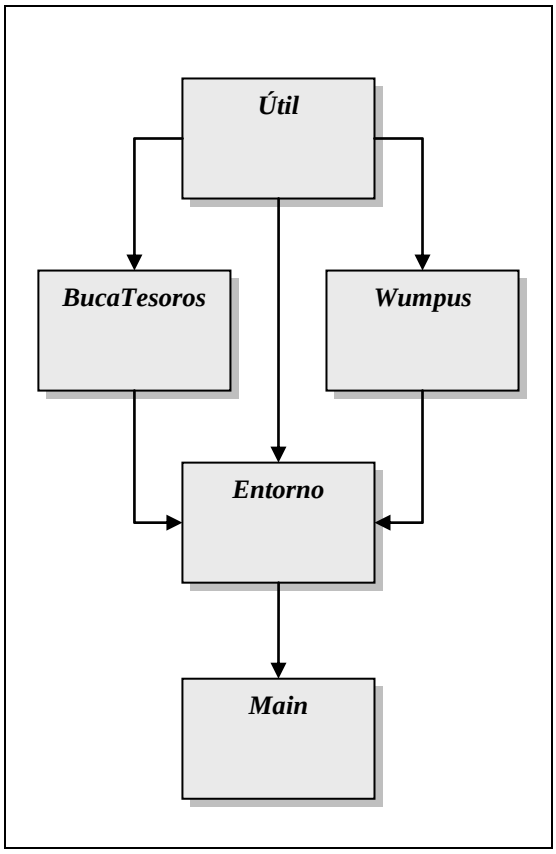


Diagrama 1. *Diagrama de la estructura del programa.*

Un aspecto que ha sido determinante a la hora de plantearnos el diseño ha sido la necesidad de utilizar un enfoque preparado para contemplar las posibles extensiones que se le pueden aplicar al problema, es decir, hemos desarrollado el diseño pensando en la existencia de *n* agentes *BuscaTesoros*, *n* monstruos *Wumpus* pululando por la cueva y *n* tesoros situados dispersamente. Esta opción complica el diseño y la implantación puesto que tenemos que controlar más entidades, *n* entidades de cada tipo. Para resolverlo utilizamos listas que contienen los diferentes actores y objetos. Esto nos obliga a incluir una lista para almacenar todos los *Wumpus*, otra para los diferentes agentes *BuscaTesoros* y otra para los tesoros. También hemos preparado el diseño para que se puedan añadir extensiones que cambien la estrategia utilizada por el agente para explorar o

salir de la cueva o el comportamiento del *Wumpos*.

```
; buble principal

 > Leer los datos desde un
   fichero

 Loop

   > Dibujar mapa y mostrar la
     información de los agentes

   > Generar movimientos de los
     agentes y del Wumpus

   > Actualizar el estado del
     entorno

EndLoop
```

Código 1. *Fragmento de pseudocódigo que ilustra el bucle principal del programa.*

Debido a que el problema representa un sistema que se desarrolla en tiempo real, hemos tenido que tener en cuenta que las acciones de los diferentes agentes no modifiquen el estado del entorno. Esto lo solucionamos utilizando listas que contienen los cambios realizados sobre los agentes, es decir, los cambios que provoca un agente o un *Wumpus* se guardan en una lista temporal y se actualizan al final del turno. De esta manera evitamos engañar a los sensores, que siempre reciben la información del entorno es ese instante de tiempo.

Tampoco nos hemos olvidado del usuario a la hora de desarrollar el diseño de la interfaz, la ejecución del programa debe de ser sencilla y permitir realizar cambios en los mapas de manera clara, rápida y cómoda. Esto nos lleva a utilizar ficheros de configuración que contengan la descripción de los agentes, de los *Wumpus,* del tesoro y de la estructura del mapa. Para poder ejecutar el programa hemos desarrollado una única función situado dentro del módulo principal que ejecuta el programa para un fichero de configuración determinado.

```
; descripción de las entidades

((Entidad0 (Atributo00 Valor)
           (Atributo01 Valor)
            ...
           (Atributo0N Valor))

 (Entidad1 (Atributo11 Valor)
           (Atributo10 Valor)
            ...
           (Atributo1N Valor))

  ...

 (EntidadN (AtributoN1 Valor)
           (AtributoN0 Valor)
            ...
           (AtributoNN Valor))
```

Código 2. *Fragmento de pseudocódigo que ilustra la estructura que poseen las diferentes entidades en el fichero de configuración.*

```
; ------------------------------
; Leyenda Mapa
; ------------------------------
;
;     M = Muro
;       = Casilla Libre
;     P = Precipicio
;     S = Salida
;

; descripción del mapa

 (Mapa (M M M . . . . M)
       (M _ _ . . . . M)
       (M _ _ . . . . M)
       (. . . . . . . .)
       (. . . . . . . .)
       (. . . . . . . .)
       (M _ _ . . . . M)
       (M M M . . . . M)))
```

Código 3. *Fragmento de pseudocódigo que ilustra la estructura que posee el mapa en el fichero de configuración.*

3.1 Movimiento del Wumpus

El *Wumpus* puede ser configurado para que realice movimientos dentro de la cueva, teniendo en cuenta, como hemos mencionado anteriormente, que el monstruo no se ve afectado por la presencia de precipicios. Existen tres posibles configuraciones:

- *Aleatorio,* el monstruo se mueve de manera totalmente aleatoria.

- *Territorial,* el monstruo se mueve de forma aleatoria en torno a su posición inicial.

- *Asesino,* el monstruo perseguirá al agente que esta situado a una determinada distancia.

De las posibilidades anteriores en la versión actual sólo se ha implementado la primera de ellas. También hay que tener en cuenta que todos los posibles movimientos se ven afectados por una variable que mide la probabilidad de que el monstruo realice un movimiento.

3.2 Movimiento del BuscaTesoros

El *BuscaTesoros* distingue dos fases de movimientos: una de *Exploración* donde el agente empezará a explorar el mapa hasta cumplir uno de los objetivos, y otra de *Salida*, donde el agente tiene como objetivo abandonar la cueva en el menor tiempo posible. En la fase de *exploración*, el agente puede ir rellenado un mapa sobre las posiciones por las que va pasando. Este mapa es muy útil para abandonar la cueva. Las posibles configuraciones son:

- Usar mapa durante la fase de exploración:
  Sí/No.

- Usar mapa durante la fase de salida:
  Si/No.

- Método de exploración:
  Aleatorio.
  Aleatorio usando A*

- Método de Salida:
  Distancia mínima.
  Distancia mínima usando A*.
  Aleatorio.

De las posibles configuraciones no se han implementado las que utilizan el algoritmo A* quedando estas para una futura extensión.

4 Implementación. Codificación

4.1 Guía de estilo

La guía que hemos utilizado para desarrollar el código en *LISP* se describe en los siguientes puntos:

- Las estructuras se nombran de la siguiente forma:
  StructNombreEstructura

- Las funciones se escriben con la primera letra en mayúscula y las siguientes en minúscula usando el guión – para separar. Para que no exista coincidencia de nombres entre funciones de diferentes módulos éstas poseen al principio de su declaración un identificador propio del módulo, igual que se implementa en *OpenGL*, donde cada función está precedida de la palabra *gl.* Por ejemplo:
  IdMódulo-Función

- El nombre de todas las variables se escribe en minúscula usando mayúscula para separar diferentes palabras. Ejemplo:
  variableContadora

4.2 Módulos

El programa ha sido desarrollado usando cinco módulos diferentes. Cada módulo opera sobre una estructura de datos, salvo los módulos *Útil,* que contiene diferentes funciones de utilidad y *Main* que sólo contiene el bucle principal descrito en Codigo 1. A continuación de detallan todos los módulos:

4.2.1 Módulo Útil

Este módulo está contenido en el fichero *util.lsp* que contiene diversas funciones de utilidad utilizadas por los restantes módulos. Se divide en dos partes principales, una destinada a tratamiento de listas, y otra destinada a la generación de movimientos para los agentes.

4.2.2 Módulo Wumpus

En este módulo (*wumpus.lsp*) están implementadas las funciones encargadas de generar y gestionar los movimientos del *Wumpus*. Todas las funciones que contiene el fichero trabajan sobre la estructura *StructWumpus* descrita en código 4. La función principal que genera el movimiento para cada turno es:

Wumpus-Movimiento (…)

Esta es la función que utiliza el módulo *Entorno* para generar el movimiento del *Wumpus.*

```
(defstruct StructWumpus

  ; variables de configuración
  nombre                  ; nombre del Wumpus
  posicion                ; posición del Wumpus (x y)
  (forma 'W)              ; pinta del BuscaTesoros
  (direccion NORTE)       ; Norte, Sur, Oesto, Este
  (movimiento 0.15)       ; probalidad de realizar un movimiento (0.0
                          ; => no se mueve nunca, 1.0 => se mueve
                          ; siempre)
  (instinto 'Aleatorio)   ; indica como de comporta el Wumpus. Valores:
                          ; Aleatorio, Territorial, Asesino
  (radioOido 4)           ; indica el número de casillas en las cuales
                          ; el Wumpus es capaz de oír al BuscaTesoros

  ; variables de estado
  (estado 'Vivo))         ; Valores: Muerto, Vivo
```

Código 3. *Fragmento de código en LISP donde se describe la estructura del Wumpus.*

4.2.3 Módulo Entorno

Módulo contenido dentro del fichero *entorno.lsp*. El módulo se divide en varias partes encargadas de realizar diferentes tareas:

- Gestión de los ficheros de configuración, funciones destinadas a leer los ficheros con las diferentes configuraciones de mapas, agentes y *Wumpus.*

- Gestión de las acciones y estados de los agentes, la mayor parte del código esta destina a gestionar los estados de los agentes, así como del *Wumpus*.

- Dibujar el mapa y mostrar el estado de los atributos más interesantes de todos los agentes y Wumpus que participan en la simulación.

4.2.4 Módulo BuscaTesoros

Dentro del fichero BucaTesoroes.lsp se encuentra la implementación del módulo más denso encargado de gestionar los diferentes movimientos que puede realizar el agente. El módulo se divide en varias partes:

- Sensores, grupos de funciones encargadas de gestionar los diferentes sensores de los que está dotado el agente, descritas en la Tabla 1

- Acciones, funciones que implementan las diferentes acciones que es capaz de realizar el agente, descritas en la Tabla 2.

- Gestión del mapa que el agente va generando con ayuda de los sensores.

- Gestión de las diferentes acciones que puede desarrollar el agente dentro de la cueva y de su estado.

La función principal de este módulo, que se encarga de desarrollar las diferentes acciones del agente, es:

BT-Movimiento (…)

Su estructura es:

; generamos las tareas a realizar
(BT-Generar-Tareas …)

; miramos los sensores
(BT-Mirar-Sensores …)

; ejecutamos la acción de la lista de tareas
(BT-Ejecutar-Movimiento …)

; actualiza los objetivos del agente
(BT-Actualizar-Objetivos …))

Todas las acciones que va a realizar el agente las va almacenando en una lista de tareas. Cuando la lista de tareas está vacía, el agente vuelve a generar otra serie de tareas que lo lleven a otro punto. La estructura del agente de describe en la Código 4.

```
; variables de configuración
nombre                          ; nombre del BuscaTesoros
posicion                        ; posición del BuscaTesoros (x y)
(forma 'B)                      ; pinta del BuscaTesoros
(direccion ESTE)                ; Norte, Sur, Oeste, Este
(numFlechas 1)                  ; número de flechas disponibles
(asesino 1.0)                   ; probabilidad de que el
                                ; BuscaTesoros dispare una flecha
(valiente 0.0)                  ; probabilidad de atravesar una zona
(espontaneo 0.0)                ; probabilidad de generar una acción
                                ; espontanea
(metodoExploracion 'Aleatorio)  ; indica cual es la estrategia que
                                ; utiliza el agente para buscar
                                ; Valores: Aleatorio, AleatorioA
(mapaExploracion 'Si)           ; usa el mapa durante la exploración
(metodoSalida 'Aleatorio)       ; indica cual es método utilizado
                                ; por el agente para salir
                                ; Valores: Aleatorio,
                                ; DistanciaMinima, DistanciaMinimaA
(mapaSalir 'Si)                 ; el agente crea un mapa y lo usa
                                ; para salir de la cueva
(tesorosBuscar 1)               ; le indica al BuscaTesoros el
                                ; número de tesoros que debe buscar
                                ; para poder abandonar la cueva
(maxTurnosBusqueda 50000)       ; máximo número de turnos que se
                                ; pueden gastar en la búsqueda
(anchoMapa 15)                  ; ancho máximo del mapa
(altoMapa 15)                   ; alto máximo del mapa

; variables de estado
(estado 'Vivo)                  ; Muerto, Vivo, Fuera, Tramposo
(turnos 0)                      ; turnos utilizados para conseguir
                                ; el objetivo
(objetivo 'Explorar)            ; indica cual es el objetivo del
                                ; BuscaTesoros. Valores: Explorar,
                                ; Salir
(wumpusMuertos 0)               ; número de Wumpus que ha matado
listaTesoros                    ; lista con los tesoros recogidos
                                ; por el BuscaTesoros
listaSalidas                    ; posiciones de las posibles salidas
                                ; ((x1 y1) (x2 y2) ... (xn yn))
listaTareas                     ; lista de tareas a realizar.
                                ; Valores: GirarDerecha,
                                ; GirarIzquierda, Avanzar,
                                ; DispararFlecha, Saltar,
                                ; RecogerTesoro
(flechaDisparada nil)           ; t si acaba de disparar una flecha
(alertaBrisa 0)                 ; indicador de peligro de brisa
(pasarDeOlor 0)                 ; indica cuantos turnos pasa el
                                ; agente de mirar el sensor
(retrocediendo 0)               ; número de turnos que se pasa el
                                ; agente retrocediendo
(alertaGiros 0)                 ; número de giros consecutivos
(estadoSensores nil)            ; lista con los sensores que están
                                ; activos
mapa                           ; mapa que se va construyendo
```

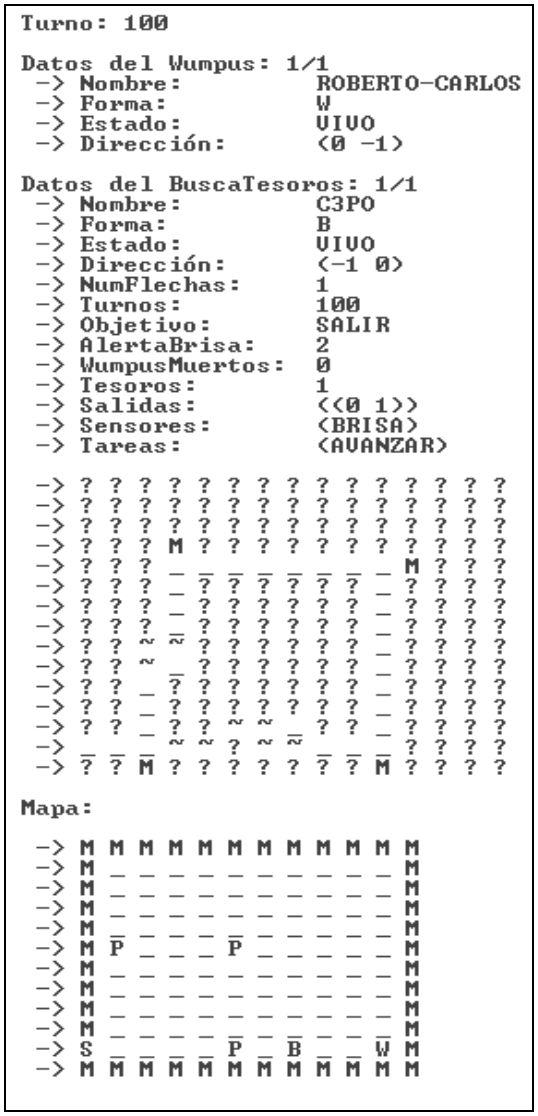
Código 4. *Fragmento de código en LISP donde se describe la estructura del agente BuscaTesoros..*

4.2.5 Módulo Main

Dentro del fichero *main.lsp* se encuentra el fichero que desarrolla el bucle principal descrito en código 1.

4.3 Interacción con el usuario

Una vez cargado el módulo principal llamamos a la función *(Main-Run nombreFichero saltos)* que se encargará de ejecutar la simulación. *nombreFichero* contiene el nombre del fichero que con el mapa y los agentes que van a actuar. La variable *saltos* determina cada cuantos turnos se muestran en pantalla los resultados. A continuación se muestra un ejemplo de una captura.



Captura 1. *Captura del programa ejecutando mapa03.lsp.*

5 Manual de Uso

La ejecución del programa se basa en ficheros configuración. Con el programa se suministran 7 mapas: *mapa00.lsp, mapa01.lsp, mapa02.lsp, mapa03.lsp, mapa04.lsp, mapa05.lsp, mapa06.lsp.* Cada fichero contiene la configuración de los diferentes *BuscaTesoros* y *Wumpus* que participan en la simulación. Los atributos que se pueden añadir y modificar, tanto para el *Wumpus* como para el *BuscaTesoros*, están descritos en las tablas *Código 3* y *Código 4*, bajo el epígrafe *variables de configuración*. A continuación se detalla un ejemplo:

1.º) Teclear *(load "main.lsp")* para cargar todos los módulos.

2.º) Teclear *(Main-Run "mapa03.lsp" 100)* para lanzar la ejecución del bucle principal.

3.º) Para ir avanzando la ejecución de la simulación presionar *ENTER.*

4.º) Para parar la ejecución de la simulación presionar *CRLT+C.*

La ejecución del programa volcará en la ventana del simulador algo similar a los escrito en *Captura 1*. Al final de la memoria se adjuntan el código de varios ficheros de ejemplo.

6 Estudio Experimental

Como se ha comentado anteriormente, con el programa se adjuntan siete mapas con diferentes configuraciones. Al final del documento se adjuntan copias impresas de los mismos. A continuación describimos los resultados obtenidos:

- *mapa00.lsp*, mapa muy simple que el agente es capaz de resolver en un par de turnos.
- *mapa01.lsp*, ejemplo simple, sin *enemigos*.
- *mapa02.lsp*, mapa simple sin precipicios.
- *mapa03.lsp*, en este mapa existe un W*umpus* que posee una capacidad de movimiento total, lo que provoca que el agente muera en un buen número de intentos.
- *mapa04.lsp*, mapa que el agente no es capaz de resolver.
- *mapa05.lsp*, en este ejemplo el algoritmo de mínima distancia no funciona.

- *mapa06.lsp*, ejemplo con varios *Wumpus* y varios agentes. También se incluyen dos salidas.

7 Conclusiones

Dado el carácter probabilístico del problema las soluciones planteadas fallan en algunos casos, como por ejemplo si el *Wumpus* posee una capacidad de movimiento muy alta, la vida del agente estará más que comprometida. También cabe destacar que existen determinadas configuraciones que el agente no es capaz de resolver, como son el entorno simulado en el fichero *mapa04.lsp*, o si el agente tiene la mala suerte de matar el *Wumpus* en la casilla donde se encuentra el tesoro. En este caso el agente se acercará, pero al detectar al *Wumpus*, retrocederá sin poder llegar al tesoro. Para solucionar esto, existe una variable que controla la valentía del agente, lo que en algunos casos le lleva a una muerte prematura.

8 Otras Consideraciones

El trabajo actual está incompleto desde el punto de vista de que no están implementados los métodos basados en el algoritmo A*. Este apartado junto con los siguientes se reserva para una futura extensión que además contendrá:

- Más mapas y simulaciones.

- Comparaciones con los métodos desarrollados por otros compañeros.

- Análisis exhaustivo de los resultados obtenidos para las diferentes configuraciones.

- Detallado específico de los métodos implementados para resolver el problema.