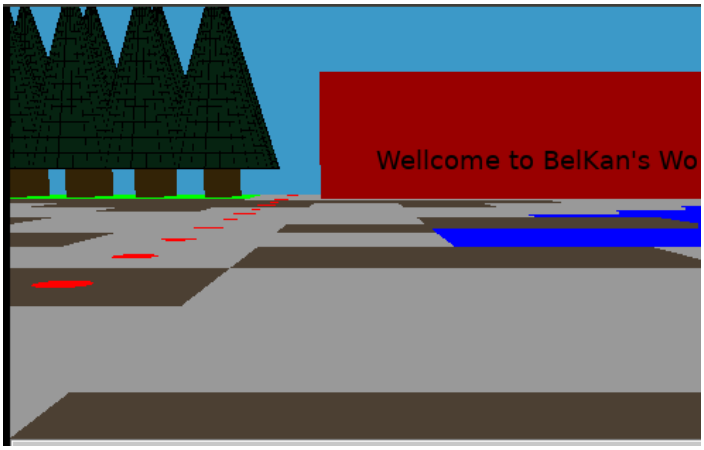
INTELIGENCIA ARTIFICIAL

E.T.S. de Ingenierías Informática y de Telecomunicación

Práctica 2



Agentes Reactivos/Deliberativos: los extraños mundos de BelKan

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL
UNIVERSIDAD DE GRANADA
Curso 2020-2021

1. Introducción

La segunda práctica de la asignatura ***Inteligencia Artificial*** consiste en el diseño e implementación de un agente reactivo/deliberativo, capaz de percibir el ambiente y actuar considerando una representación de las consecuencias de sus acciones y siguiendo un proceso de búsqueda. Se trabajará con un simulador software. Para ello, se proporciona al alumno un entorno de programación, junto con el software necesario para simular el entorno.

En esta práctica se diseñará e implementará un agente reactivo y deliberativo basado en los ejemplos del libro *Stuart Russell, Peter Norvig, "Inteligencia Artificial: Un enfoque Moderno", Prentice Hall, Segunda Edición, 2004*. El simulador que utilizaremos fue inicialmente desarrollado por el profesor Tsung-Che Chiang de la NTNU (National Taiwan Normal University of Science and Technology), pero la versión sobre la que se va a trabajar ha sido desarrollada por los profesores de la asignatura.

Originalmente, el simulador estaba orientado a experimentar con comportamientos en aspiradoras inteligentes. Las aspiradoras inteligentes son robots de uso doméstico que disponen de sensores de suciedad, un aspirador y motores para moverse por el espacio (ver Figura 1). Cuando una aspiradora inteligente se encuentra en funcionamiento, esta recorre toda la dependencia o habitación donde se encuentra, detectando y succionando suciedad hasta que, o bien termina de recorrer la dependencia, o bien aplica algún otro criterio de parada (batería baja, tiempo límite, etc.).

Figura 1: Aspiradora inteligente

Este tipo de robots es un ejemplo comercial más de máquinas que implementan técnicas de Inteligencia Artificial y, más concretamente, de Teoría de Agentes. En su versión más

simple (y también más barata), una aspiradora inteligente presenta un comportamiento *reactivo* puro: busca suciedad, la limpia, se mueve, detecta suciedad, la limpia, se mueve, y continúa con este ciclo hasta que se cumple alguna condición de parada. Otras versiones más sofisticadas permiten al robot *recordar* mediante el uso de representaciones icónicas como mapas, lo cual permite que el aparato ahorre energía y sea más eficiente en su trabajo. Finalmente, las aspiradoras más elaboradas (y más caras) pueden, además de todo lo anterior, planificar su trabajo de modo que se pueda limpiar la suciedad en el menor tiempo posible y de la forma más eficiente. Son capaces de detectar su nivel de batería y volver automáticamente al cargador cuando esta se encuentre a un nivel bajo. Estas últimas pueden ser catalogadas como *agentes deliberativos*.

En esta práctica, centraremos nuestros esfuerzos en implementar el comportamiento de un "personaje virtual" asumiendo un comportamiento reactivo y deliberativo. Utilizaremos las técnicas estudiadas en los temas 2 y 3 de la asignatura para el diseño de agentes reactivos y deliberativos.

2. Los extraños mundos de BelKan

En esta práctica tomamos como punto de partida el mundo de las aventuras gráficas de los juegos de ordenador para intentar construir sobre él personajes virtuales que manifiesten comportamientos propios e inteligentes dentro del juego. Intentamos situarnos en un problema habitual en el desarrollo de juegos para ordenador y vamos a jugar a diseñar personajes que interactúen de forma autónoma usando agentes reactivos/deliberativos.

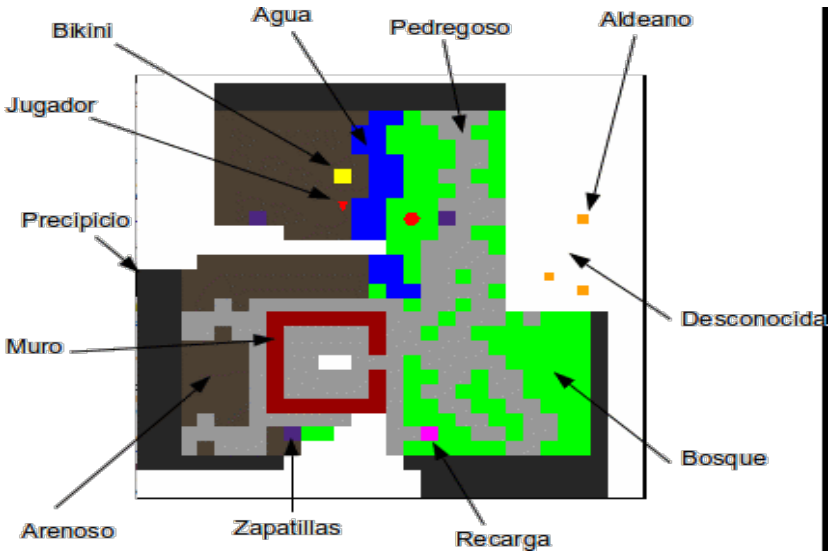
2.1. El escenario de juego

Este juego se desarrolla sobre un mapa cuadrado bidimensional discreto que contiene como máximo 100 filas y 100 columnas. El mapa representa los accidentes geográficos de la superficie de un terreno que son considerados como inmutables, es decir, los elementos dentro del mapa que no cambian durante el desarrollo del juego.

Representaremos dicha superficie usando una matriz donde la primera componente representa la fila y la segunda representa la columna dentro de nuestro mapa. Como ejemplo usaremos un mapa de tamaño 100x100 de caracteres. Fijaremos sobre este mapa las siguientes consideraciones:

- La casilla superior izquierda del mapa es la casilla [0][0].
- La casilla inferior derecha del mapa es la casilla [99][99].

Teniendo en cuenta las consideraciones anteriores, diremos que un elemento móvil dentro del mapa va hacia el NORTE, si en su movimiento se decrementa el valor de la fila. Extendiendo este convenio, irá al SUR si incrementa su valor en la fila, irá al ESTE si incrementa su valor en las columnas, y por último irá al OESTE si decrementa su valor en columnas.



Los elementos permanentes en el terreno son los siguientes:

- ***Árboles*** *o* ***Bosque***, codificado con el carácter '**B**' y se representan en el mapa como casillas de color verde.
- ***Agua***, codificado con el carácter '**A**' y tiene asociado el color azul.
- ***Precipicios***, codificado con el carácter '**P**' y tiene asociado el color negro. Estas casillas se consideran intransitables.
- ***Suelo Pedregoso***, codificado con el carácter '**S**' y tiene asociado el color gris.

- ***Suelo Arenoso***, codificado con el carácter '**T**' y tiene asociado el color marrón.

- ***Muros***, codificado con el carácter '**M**' y son rojo oscuro.

- ***Bikini***, codificado con el carácter '**K**' y se muestra en amarillo. Esta es una casilla especial que cuando el jugador pasa por ella adquiere el objeto "bikini". Este objeto le permite reducir el consumo de batería en sus desplazamientos por el agua.

- ***Zapatillas***, codificado con el carácter '**D**' y son moradas. Esta es una casilla especial y al igual que la anterior, el jugador adquiere, en este caso el objeto "zapatillas" simplemente al posicionarse sobre ella. Las "Zapatillas" le permiten al jugador reducir el consumo de batería en los bosques.

- ***Recarga***, codificado con el carácter **'X'** y de color rosa. Esta casilla especial permite al jugador cargar su batería. Por cada instante de simulación aumenta en 10 el nivel de su batería. Cuando la batería alcanza su nivel máximo de carga (3000), estar en esta casilla no incrementa la carga.

- ***Casilla aún desconocida*** se codifica con el carácter **'?'** y se muestra en blanco (representa la parte del mundo que aún no has explorado).

Todos los mundos que usaremos en esta práctica son cerrados, ya que en todos se verifica que las tres últimas filas visibles al Norte son precipicios, y los mismo pasa con las tres últimas filas/columnas del Sur, Este y Oeste. Esto no quiere decir que no pueda haber más precipicios en el resto del mapa.

Sobre esta superficie pueden existir elementos que tienen la capacidad de moverse por sí mismos. Los elementos que aquí consideraremos son:

- ***Jugador***, se codifica con el carácter '**j**' y se muestra como un triángulo rojo. Éste es nuestro personaje, sólo habrá un jugador a la vez.

- ***Aldeano***, se codifica con el carácter '**a**' y se muestra como un cuadrado naranja. Son habitantes anónimos del mundo que se desplazan a través del mapa sin un

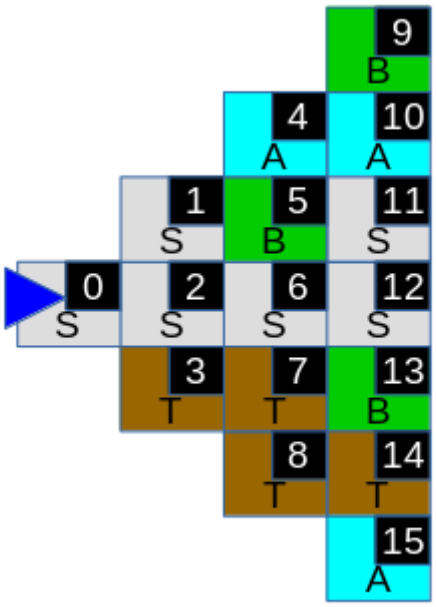
cometido específico, simplemente intentan molestarnos en nuestros movimientos. Son sólo molestos, no son peligrosos.

2.2. El protagonista

Obviamente el protagonista de la historia es el elemento llamado jugador y debe llevar a cabo su objetivo que consiste en ir desde su ubicación actual en el terreno (origen) a una o varias casillas especificadas en el mapa (destino).

2.2.1. Propiedades del agente jugador

El personaje del jugador en el simulador viene representado en forma de un triángulo rojo. Cuenta con un sistema visual que le permite observar las cosas que se le aparecen dentro de un campo de visión. Dicho campo de visión se representa usando dos vectores de caracteres de tamaño 16. El primero de ellos lo denominaremos terreno y es capaz de observar los elementos inmóviles del terreno. El segundo de ellos, que llamaremos superficie, es capaz de mostrarnos qué objetos móviles se encuentran en nuestra visión (es decir, aldeanos). Para entender cómo funciona este sistema pondremos el siguiente ejemplo: Suponed que el vector terreno contiene **SSSTABSTTBASSBTA**, su representación real sobre un plano será la siguiente:



El primer carácter (posición 0) representa el tipo de terreno sobre el que se encuentra nuestro personaje. El tercer carácter (posición 2) es justo el tipo de terreno que tengo justo delante de mí. Los caracteres de posiciones 4, 5, 6, 7 y 8 representan lo que está delante de mí, pero con una casilla más de profundidad y apareciendo de izquierda a derecha. Por último, los caracteres de posiciones de la 9 a la 15 son aquellos que están a tres casillas viéndolos de izquierda a derecha. La figura anterior representa las posiciones del vector en su distribución bidimensional (los números), y el carácter y su representación por colores cómo quedaría en un mapa.

De igual manera se estructura el vector **superficie** pero, en este caso, indicando que objetos móviles se encuentran en cada una de esas casillas. Los valores posibles en este sensor serán: **'a'** para indicar que hay un aldeano, **'j'** para indicar la posición del agente y **'_'** si la casilla está desocupada.

El personaje cuenta con sensores que miden éstas y otras cuestiones:

- **Sensor de choque (`colision`)**: Es una variable booleana que tomará el valor verdadero en caso de que la última acción del jugador haya ocasionado un choque.
- **Sensor de vida (`reset`)**: Es una variable booleana que toma el valor de verdad en caso de que la última acción del jugador le haya llevado a morir.
- **Sensores de posición (`posF`, `posC`)**: Devuelve la posición de fila y columna que ocupa el jugador.
- **Sensor de orientación (`sentido`)**: Devuelve la orientación del jugador.
- **Sensor del número de destinos activos (`num_destinos`)**: Indica el número de casillas destino que están activas y está vinculado al siguiente sensor que indica exactamente qué casillas en concreto son los destinos a alcanzar.

- **Sensor de la ubicación de los destinos activos** (**`destino`**): es un vector de enteros que representa las coordenadas de todos los destinos activos. Están codificados de la forma *fila* primero y después la *columna*. Obviamente su valor está vinculado al sensor anterior **`num_destinos`**. Si **`num_destinos`** vale 3 y **`destino`** tiene la secuencia 45 12 78 23 11 19, los destinos activos son tres, los representados por los 6 valores. Así, las casillas objetivo son: fila 45 y columna 12, fila 78 y columna 23, fila 11 y columna 19.

- **Sensor de batería (`bateria`)**: Informa del nivel de carga de la batería. Inicialmente la batería tiene valor de 3000 y dependiendo de la acción realizada va perdiendo valor. La simulación termina cuando la carga de la batería toma el valor 0.

- **Sensor de nivel (`nivel`):** Este es un sensor que informa en qué nivel del juego se encuentra. Los valores posibles del sensor están entre 0 y 4 y tienen la siguiente interpretación:

  0 : Demo (Búsqueda en profundidad).

  1 : Plan óptimo en número de acciones.

  2 : Plan óptimo en coste (1 único objetivo).

  3 : Plan óptimo en coste (3 objetivos).

  4 : Reto (Maximizar número de misiones).

- **Sensor de tiempo consumido (`tiempo`):** Este sensor informa del tiempo acumulado que lleva consumido el agente en la toma de decisiones.

Una cuestión importante a considerar es que **el jugador tiene capacidad para llevar sólo uno de los dos objetos (zapatillas o bikini)**, de manera que cada vez que pasa por una casilla con la que consigue un objeto, automáticamente pierde el otro (si es que ya había pasado por la casilla que le daba el otro objeto), es decir, si el jugador tiene las zapatillas y para por una casilla que da el bikini, el efecto es que adquiere el bikini, pero pierde las zapatillas. Lo mismo ocurre con la situación análoga en la configuración de estos objetos.

Si se tiene un objeto y se pasa por una casilla que te ofrece ese objeto, no se produce ningún cambio, es decir, se sigue manteniendo el objeto.

2.2.2. Datos del personaje compartidos con el entorno

Dentro de la definición del agente hay una matriz llamada **`mapaResultado`** en donde se puede interactuar con el mapa. Todo cambio en esta matriz se verá reflejado automáticamente en la interfaz gráfica.

2.2.3. Acciones que puede realizar el personaje

El agente puede realizar las siguientes acciones:

- ***actFORWARD**:* le permite avanzar a la siguiente casilla del mapa siguiendo su orientación actual. Para que la operación se finalice con éxito es necesario que la casilla de destino sea transitable para nuestro personaje.
- ***actTURN_L**:* le permite mantenerse en la misma casilla y girar a la izquierda 90º teniendo en cuenta su orientación.
- ***actTURN_R**:* le permite mantenerse en la misma casilla y girar a la derecha 90º teniendo en cuenta su orientación.
- ***actIDLE**:* no hace nada.

2.2.4. Coste de las acciones

Cada acción realizada por el agente tiene un coste en tiempo de simulación y en consumo de batería. En cuanto al tiempo de simulación, todas las acciones consumen un instante independientemente de la acción que se realice y del terreno donde se encuentre el jugador. En cuanto al consumo de batería, decir que **actIDLE** consume 0 de batería y que el consumo de este recurso de las acciones **actTURN_L**, **actTURN_R** y **actFORWARD** ***depende del tipo de terreno asociado a la casilla donde se inició dicha acción***. Esto es, si el agente está en una casilla de agua (etiquetada con una ‘A’) y avanza a una casilla de terreno arenoso (etiquetada con una ‘T’), el coste en batería es el asociado a la casilla de agua, es decir, a la casilla inicial donde se produce la acción de avanzar. En las siguientes tablas se muestran los valores de consumo de energía en función de las acción a realizar, la

casilla de inicio de la acción y dependiendo del objeto que se tenga en posesión en ese momento.

actFORWARD		
Tipo de Casilla	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	200	10 (con **Bikini**)
'B'	100	15 (con **Zapatillas**)
'T'	2	2
Resto de Casillas	1	1

actTURN_L / actTURN_R		
Tipo de Casilla	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	500	5 (con **Bikini**)
'B'	3	1 (con **Zapatillas**)
'T'	2	2
Resto de Casillas	1	1

3. Objetivo de la práctica

El objetivo de esta práctica es dotar de un comportamiento inteligente a nuestro personaje usando un agente reactivo/deliberativo para definir las habilidades que le permitan alcanzar una meta concreta dentro del juego según el nivel seleccionado.

Para que sea más fácil resolver esta práctica, se han diseñado 5 niveles con dificultad incremental.

La práctica incluye un nivel 0 (Demo) que es una demostración de la forma en la que se debe implementar un algoritmo de búsqueda y cómo conectarse con el software global de la práctica. En concreto, aparece en el nivel 0 una implementación de la búsqueda en profundidad. Este nivel está incompleto y es necesario terminarlo, tal y como se describirá más adelante.

Nivel 1: Encontrar el camino con mínimo número de acciones

La misión del agente es ir a un punto destino del mapa. En este nivel nuestro agente conoce perfectamente el terreno y no hay aldeanos en el mismo. Nuestro agente no puede atravesar muros, ni precipicios, y por lo tanto, deberá esquivar estos tipos de terreno ya que en otro caso chocará (en el caso de los muros) o morirá (en caso de precipicios).

Inicialmente el agente aparecerá de forma aleatoria sobre un mapa concreto conociendo su posición a través de los sensores **`posF`** y **`posC`**, y orientación sobre el mapa a través el sensor **sentido**. El mapa es estático, es decir no hay cambios en su descripción de ningún tipo.

El objetivo del agente es crear y llevar a cabo un plan de movimientos en el mapa para llegar desde su posición inicial al destino usando el menor número de acciones.

Nivel 2: Encontrar el camino con el mínimo consumo de batería a un único objetivo

Igual que en el nivel anterior, el agente debe encontrar la secuencia de acciones que le permita llegar a una casilla del mapa, y al igual que en este último, también se conoce todo el terreno y no hay aldeanos. La única diferencia con el nivel anterior es el tipo de camino que se pide. En este caso, se pide **crear y llevar a cabo un plan de movimientos en el mapa para llegar desde su posición inicial al destino teniendo el menor consumo de batería.**

Nivel 3: Encontrar el camino con el mínimo consumo de batería a tres objetivos

Las condiciones iniciales (de posición inicial aleatoria, terreno completamente conocido y la no existencia de aldeanos) que se tienen en este nivel coinciden con el nivel anterior. La diferencia, en este caso, es que se pide **encontrar y llevar a cabo el camino óptimo en consumo de batería que permita al agente pasar por tres casillas objetivo**. Es importante destacar que se trata de encontrar un único camino (óptimo en el consumo de batería), no tres planes a cada uno de los objetivos.

Nivel 4: Agente reactivo/deliberativo

En el nivel 4 el agente no conoce el mapa ni sabe en dónde se encuentra, aunque sí su posición y orientación. El agente irá descubriendo el mapa poco a poco a través de su sistema sensorial. El agente debe planificar un camino hacia los destinos meta que se le van proponiendo (aunque no se conozca el mapa en su totalidad). Obviamente, al no conocer el mapa en su totalidad es posible que el agente planifique un camino por zonas del espacio que no le interesen (por coste o porque no sea posible pasar) y requiera tomar alguna decisión al respecto. Además, en este nivel hay aldeanos que se pasean sin un destino fijo. Un aldeano puede moverse continuamente, puede quedarse quieto, etc. Es decir, para ir de un punto a otro del mapa puede que nuestro plan inicial no funcione correctamente ya que nos crucemos en el camino con un aldeano que nos entorpezca. Los aldeanos pueden detectarse gracias a un sensor que tiene nuestro personaje, es decir, no podemos saber dónde están los aldeanos en el mapa, pero sí podemos percibirlos con nuestro sensor de superficie si estamos cerca de ellos.

En estas situaciones deberemos integrar comportamientos reactivos y deliberativos, de manera que el agente debe encontrar un plan de navegación (un algoritmo de búsqueda) y, durante su ejecución, el agente debe considerar cómo actuar ante un posible fallo en la ejecución del plan.

En este nivel, inicialmente al agente se le proponen 3 casillas objetivo distintas. El agente tiene que pasar por cada una de ellas. La forma en la que decida hacerlo es una estrategia que depende de cada estudiante (a diferencia del nivel 3 en el que obligatoriamente hay que encontrar el camino óptimo en gasto de batería). Es importante señalar que no existe ningún sensor que informe si ya se pasó por las casillas objetivo. La única información es que cuando se completen los tres objetivos, se propondrán 3 nuevos objetivos.

A diferencia de los niveles anteriores, en este caso lo que se pide es buscar **una estrategia reactivo/deliberativa que le permita al agente pasar por el mayor número de casillas propuestas como objetivo** por el entorno. Para obtener el máximo número de objetivos se dispone de 3000 instantes de simulación, 3000 como valor inicial de la batería y 300 segundos en tiempo de elaborar los planes. La simulación termina cuando se agota alguno de estos 3 recursos.

4. El Software

Para la realización de la práctica se proporciona al alumno una implementación tanto del entorno simulado del mundo en donde se mueve nuestro personaje como de la estructura básica del agente reactivo/deliberativo.

4.1. Instalación

Se proporcionan versiones del software para el sistema operativo Linux y para el sistema operativo Windows. Dicho software se puede encontrar en la parte de la práctica 2 en la plataforma de PRADO. La versión de Windows se ha desarrollado usando el entorno de desarrollo **CodeBlocks**, así que incluye un archivo 'practica2.cbp' para facilitar el trabajo. La versión de Linux, sin embargo, no viene preparada para usar este entorno de desarrollo, si bien se incluye un archivo 'install.sh' para cargar las librerías necesarias y compilar el programa compatible con las distribuciones de ubuntu. Para otras distribuciones de linux es necesario cambiar lo que respecta al comando que instala paquetes y a cómo se llama ese paquete dentro en esa distribución. La lista de bibliotecas necesarias son: ***freeglut3 freeglut3-dev libjpeg-dev libopenmi-dev openmpi-bin openmpi-doc libxmu-dev libxi-dev cmake libboost-all-dev***.

El proceso de instalación es muy simple:

1. En la plataforma PRADO se accede dentro de la asignatura a la parte donde se encuentra el material de la práctica 2 y se elige entre el software para windows y para linux.
2. Se descarga en la carpeta de trabajo.
3. Se descomprime generando una nueva carpeta llamada 'practica2' y accedemos a dicha carpeta.
4. **Para los usuarios de Windows**, se hace doble click en el archivo 'practica2.cbp' (previamente hay que tener instalada la versión 17.12 de CodeBlocks https://sourceforge.net/projects/codeblocks/files/Binaries/17.12/Windows/codeblocks-17.12mingw-setup.exe/download), se selecciona la acción 'Build and Run' y se ejecuta el entorno de simulación de la práctica.

5. **Para los usuarios de Linux (**ubuntu**)** existe un fichero 'install.sh' que instala los paquetes necesarios y compila el código. Luego se ejecuta 'make' para compilar cada vez que se edite el código.

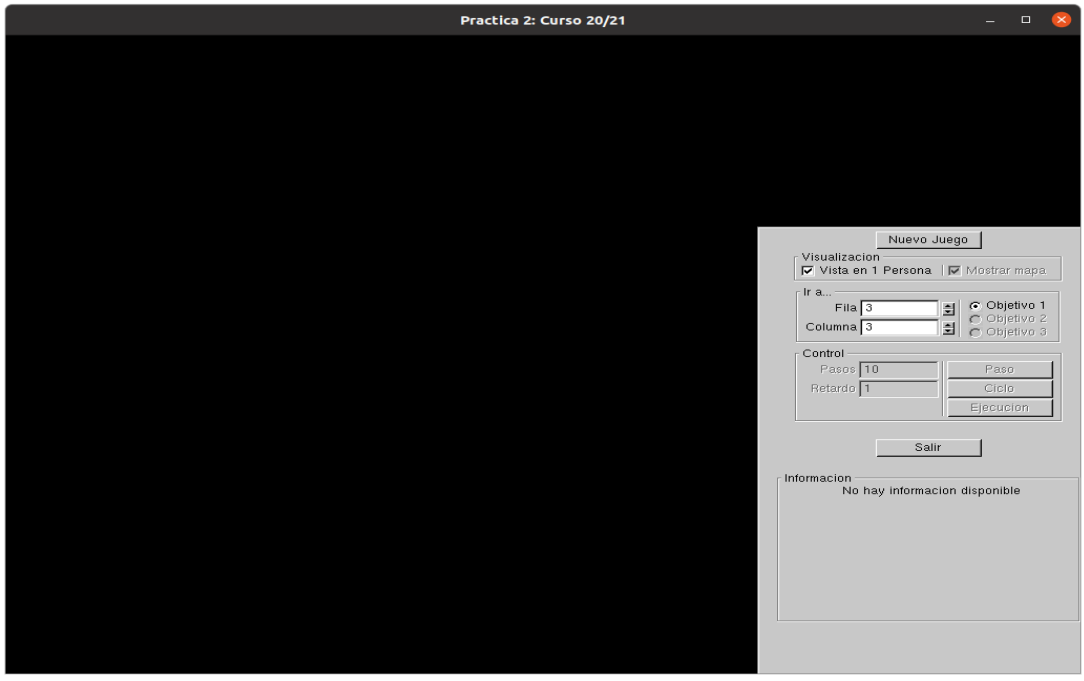
4.2. Funcionamiento del Programa

Existen dos ficheros ejecutables: Belkan y BelkanSG. El primero corresponde al simulador con interfaz gráfica, mientras que el segundo es un simulador en modo *batch* sin interfaz. La segunda versión sin entorno gráfico se ofrece para poder hacer tareas de "debugger" o de depuración de errores.

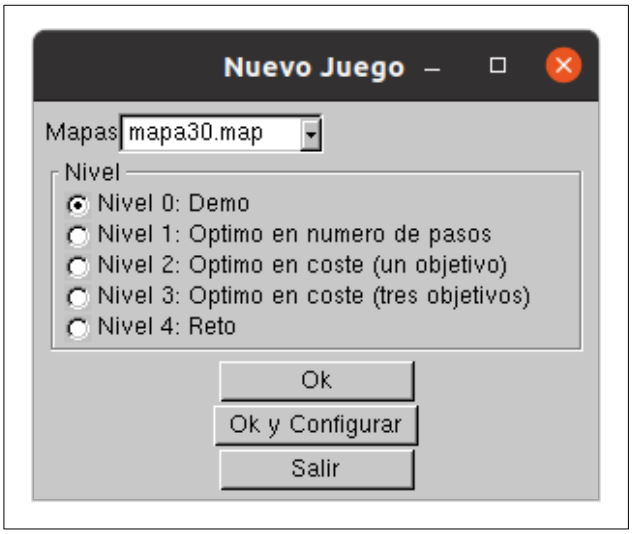
Empezamos describiendo la versión con entorno gráfico.

4.2.1. Interfaz gráfica

Para ejecutar el simulador con interfaz hay que escribir "***./Belkan***" en linux. En Windows, para ejecutarlo dentro del entorno de CodeBlocks, pulsar "**build and run**".



Al arrancar el programa nos mostrará una ventana que es la ventana principal. Para iniciar el programa se debe elegir el botón **Nuevo Juego** que abriría la siguiente ventana:



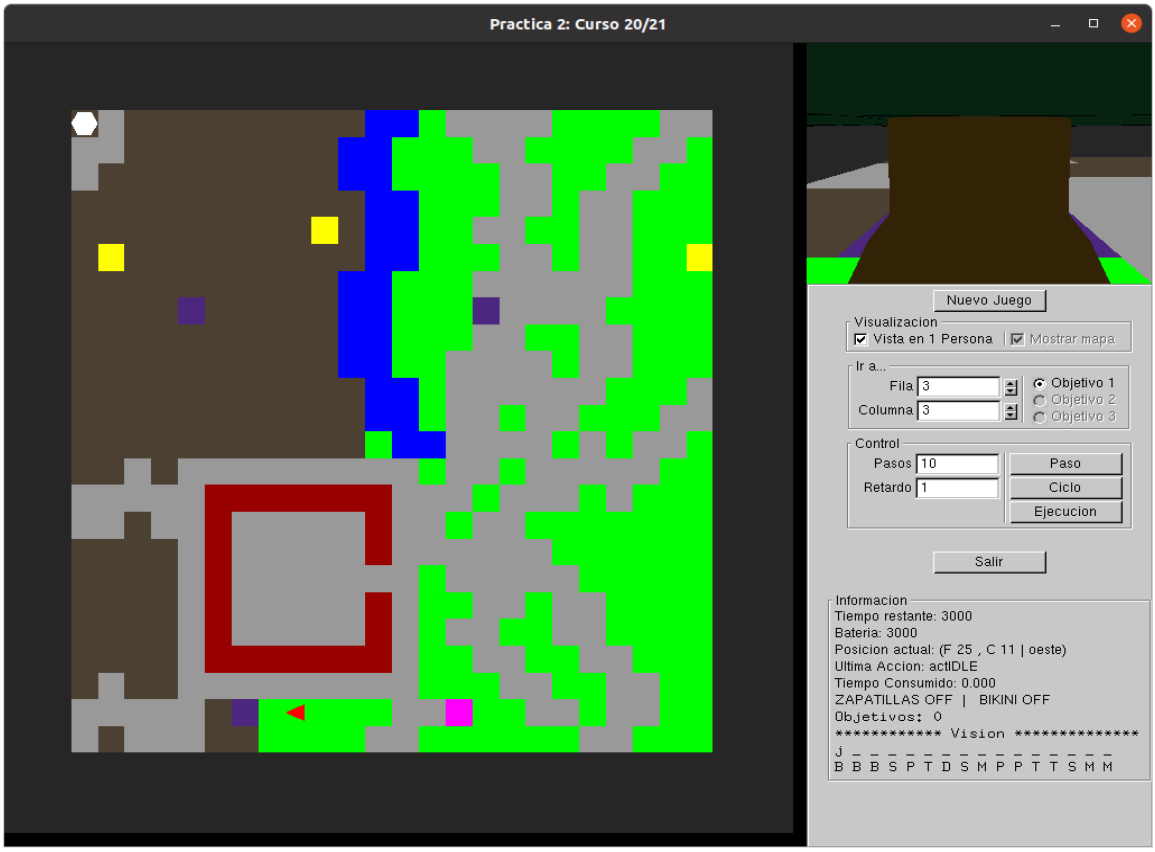
En esta nueva ventana se puede elegir el mapa con el cual trabajar (debe estar dentro de la carpeta “mapas”) y el nivel deseado. En la versión que se proporciona al estudiante, los niveles del 1 a 4 están sin implementar. Obviamente, el objetivo es ir poco a poco ofreciendo en el software la funcionalidad que se propone en cada nivel. El único que se encuentra implementado a medias es el nivel 0 que corresponde con la Demo.

Seleccionaremos el **Nivel 0: Demo** fijando como mapa **mapa30.map**, y presionaremos el botón de **Ok**.

La ventana principal se actualizará y podremos entonces distinguir tres partes: la izquierda que está destinada a dibujar el mapa del mundo, la superior derecha que mostrará una visión del mapa desde el punto de vista del agente, y la inferior derecha que contiene los botones que controlan el simulador e información sobre los valores de los sensores.

Los botones **Paso**, **Ciclo** y **Ejecucion** son las tres variantes que permite el simulador para avanzar en la simulación. El primero de ellos, **Paso**, invoca al agente que se haya definido y devuelve la siguiente acción. El botón **Ciclo** realiza el número que se indica en el campo **Pasos** con el retardo que se especifica en el campo **Retardo**. Por último el botón **Ejecucion** realiza una ejecución completa de la simulación. Indicar que estando activa esta última, si se pulsa el botón **Paso**, se puede detener su ejecución completa.

El último botón que podemos encontrar es **Salir** que cierra la aplicación.



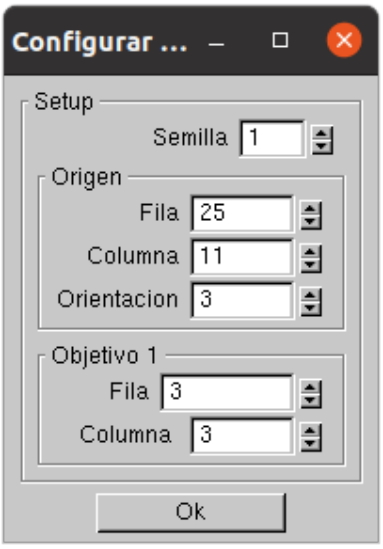
Dentro del grupo de actuadores denominados "Visualizacion", podemos decidir si deseamos que se refresque o no la visión en primera persona del agente activando o desactivando la opción **Vista en 1 Persona**. Solo en el último nivel, se nos permite cambiar **Mostrar mapa**. Esta opción permite ver el mapa que lleva reconocido el agente frente a la visión completa del mapa.

Podemos tomar otras opciones desde la ventana principal del simulador. Se puede observar que agrupado bajo el nombre "Ir a .." tenemos una serie de campos que son configurables. Asociados a **Fila** y **Columna** el simulador informa de las coordenadas de fila y columna en la que se encuentra el objetivo. Ya que en esta práctica se contempla la posibilidad de ir a solo una casilla destino o a tres casillas destinos distintas, en función del nivel elegido, se activará solo el marcador **Objetivo 1** o los tres marcadores **Objetivo 1**, **Objetivo2** y **Objetivo3**. En los niveles 0, 1 y 2, solo habrá un objetivo, por tanto los valores de **Fila** y **Columna** se corresponderán con ese único objetivo. En los otros niveles, **Fila** y **Columna** mantendrán las coordenadas del objetivo que se encuentre marcado entre los tres

Objetivos. Existe la posibilidad de cambiar el destino desde esta ventana. Si se sitúa el ratón sobre el mapa en la casilla que se desea que sea destino y se pulsa el botón derecho, automáticamente el destino que en ese momento se encuentre activo se cambiará en el mapa y en los campos **Fila** y **Columna**.

No es esta la única forma de poder cambiar la configuración, pero sí lo es sin reiniciar la simulación. Para otra opción que sí requiere una reiniciación debemos volver a pulsar el botón de **Nuevo Juego** y ahora en lugar de dar **Ok**, pulsamos el botón **Ok y Configurar**. Nos aparecerá la siguiente ventana que nos permite cambiar los parámetros de la simulación.

Los parámetros modificables son la semilla del generador de números aleatorios, la posición y orientación inicial del agente y las posiciones de los objetivos. En el caso de haber seleccionado el nivel 3 o 4, esta ventana se extiende con las opciones para poder modificar las casillas objetivo 2 y 3.



Para finalizar con la descripción del entorno gráfico, bajo el título de Informacion, se recogen los valores en cada instante de algunos de los sensores del agente así como de alguna información adicional. En concreto, se informa de:

- **Tiempo restante**: cantidad de ciclos de simulación que quedan para terminar.

- **Bateria**: cantidad de batería que le queda al agente.
- **Posicion actual**: se indica la fila, la columna y orientación del agente.
- **Ultima Accion**: indica cuál fue la última acción que realizó el agente.
- **Tiempo Consumido**: expreso en segundos la cantidad acumulada de tiempo que ha utilizado el agente en tomar las decisiones hasta este momento de la simulación.
- **ZAPATILLAS/BIKINI**: la palabra **ON** asociada a estos dos objetos indica estar en su posición en ese momento. La palabra **OFF** indicaría lo contrario. Se recuerda que el agente no puede tener simultáneamente estos dos objetos.
- **Objetivos**: indica el número de casillas objetivo alcanzadas hasta el momento.

Bajo el texto ********** **Vision** ********** se indican los valores de los que informan los sensores de terreno y superficie en este instante con la interpretación que ya se indicó anteriormente en este documento.

4.2.2. Sistema *batch*

Se incluye con el software la posibilidad de crear un ejecutable sin interfaz gráfica para dar la posibilidad de realizar las operaciones de depuración de errores con mayor facilidad, ya que las librerías gráficas incluyen programación basada en eventos que alteran el normal funcionamiento de las herramientas como el conocido debugger **ddd s**i se usa linux o el propio depurador incluido en el entorno de programación CodeBlocks. En la versión linux se generan automáticamente los dos ejecutables Belkan y BelkanSG. En el caso de windows, se incluye un proyecto de codeblocks ***belkanSG.cbp*** para construir este segundo ejecutable. Tanto en Windows como en Linux, y tanto la versión gráfica como la versión sin gráficos, hacen uso los archivos que describen el comportamiento del agente, por esta razón, su uso principal será para rastrear errores en vuestro propio código.

Ya que no tiene versión gráfica, cuando se usa BelkanSG es necesario pasarle todos los parámetros en la línea de comandos para que funcione correctamente. Una descripción de su sintaxis para su invocación sería la siguiente:

./BelkanSG <mapa> <semilla> <nivel> <fila> <col> <ori> <filO$_1$> <colO$_1$> ...

donde

<**mapa**> es el camino y nombre del mapa que se desea usaremos

<**semilla**> es un número entero con el que se inicia el generador de números aleatorios

<**nivel**> es un número entero entre 0 y 4 indicando que nivel se quiere ejecutar

<**fila**> fila inicial donde empezará el agente en la simulación

<**col**> columna inicial donde empezará el agente en la simulación

<**ori**> un número entre 0 y 3 indicando la orientación con la que empezará el agente, siendo 0=norte, 1=este, 2=sur, 3=oeste.

<**filO$_i$**> fila de la casilla del objetivo i-ésimo

<**colO$_i$**> columna de la casilla del objetivo i-ésimo.

Por ejemplo podemos ejecutar **./BelkanSG mapas/mapa30.map 1 0 4 5 1 3 20 5 12** lo cual utilizará el mapa llamado **mapa30.map** indicado con una semilla **1** en el nivel **0** situando al agente en la posición de fila **4** y columna **5**, con orientación este (**1**). El agente debe ir a la casilla objetivo de fila **3** y columna **20** y después a la casilla de fila **5** y columna **12**. En caso de no indicar destinos suficientes, el simulador los elegirá al azar. Si se proponen más destinos de los necesarios, como ocurre en esta invocación ya que el nivel 0 sólo llega al primer destino, los no necesarios se ignoran.

Al finalizar la ejecución nos ofrece los siguientes datos de la simulación:

- los instantes de simulación consumidos,

- el tiempo consumido acumulado requerido por el agente,
- el nivel final de la batería,
- el número de colisiones que hemos sufrido,
- si la simulación terminó porque murió el agente,
- y la cantidad de destinos alcanzados.

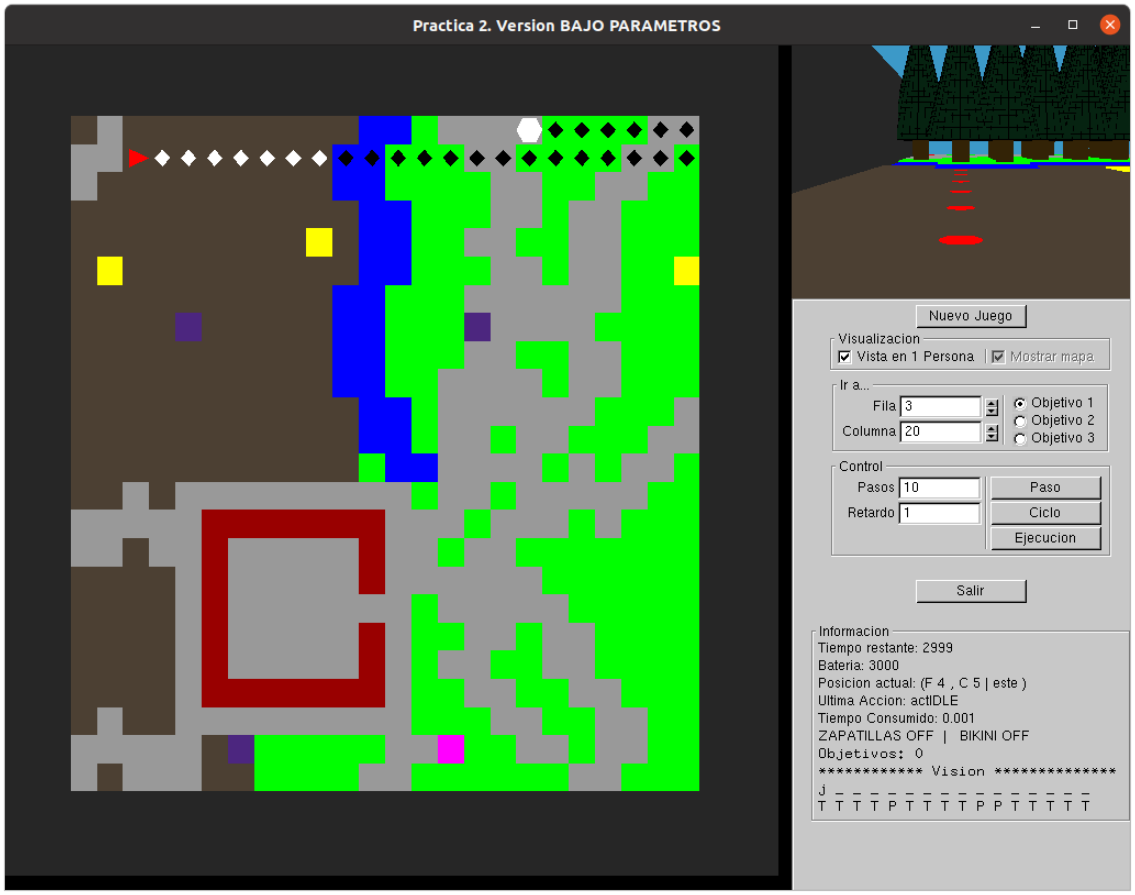
4.2.3. Sistema *batch* y entorno gráfico

Se incluye una tercera posibilidad de ejecución que consiste en combinar el modelo batch, para poder indicarle las condiciones de la simulación, con visualizar el comportamiento real del agente levantando el entorno gráfico. La forma de invocar esta opción es igual: usar los mismos parámetros en el mismo orden con el mismo significado que se describen en la versión *batch* pero aplicado sobre Belkan, es decir,

./Belkan <mapa> <semilla> <nivel> <fila> <col> <ori> <filO$_1$> <colO$_1$> ...

Algo a destacar cuando se toma esta opción para ejecutar el software es que la simulación queda detenida tras la ejecución de la primera acción del agente. Así, por ejemplo, para los niveles del 0 al 3, la simulación empezará aplicando el algoritmo de búsqueda y se detendrá cuando encuentre un camino hacia el destino. Como ejemplo, si se invoca usando el siguiente comando

./Belkan mapas/mapa30.map 1 0 4 5 1 3 20 5 12



el entorno gráfico se detendría como muestra la imagen, indicando con rombos blancos y negros las casillas por las que pasa el camino que lleva al objetivo, marcado con un círculo blanco.

4.3. Descripción del Software

De todos los archivos que se proporcionan para compilar el programa, el alumno solo puede modificar 2 de ellos, los archivos ***'jugador.hpp'*** y ***'jugador.cpp'*** que se incluyen en la carpeta Comportamiento_Jugador. Estos archivos contendrán los comportamientos implementados para nuestro agente deliberativo/reactivo. Además, dentro de estos dos archivos, no se puede eliminar nada de lo que hay originalmente, únicamente se puede añadir. Pasamos a ver con un poco más de detalle cada uno de estos archivos.

Empecemos con el archivo ***'jugador.hpp'.*** Lo primero que nos encontramos es con la definición del tipo de dato **estado**. Un estado es un registro con 3 campos enteros que indican una fila, una columna y una orientación.

Vemos que aquí se declara la clase ComportamientoJugador. De los métodos implementados en la parte pública vamos a destacar 4 de ellos:

- El constructor usado en el nivel 4. Aquí se tendrán que inicializar las variables de estado que se consideren necesarias para resolver el nivel 4 de la práctica.

  ComportamientoJugador(unsigned int size) : Comportamiento(size)

- El constructor usado en los niveles 0, 1, 2 o 3. Aquí se tendrán que inicializar las variables de estado que se consideren necesarias para resolver los niveles 1, 2, 3 de la práctica.

  ComportamientoJugador(std::vector< std::vector< unsigned char> > mapaR)

- El método que describe el comportamiento del agente y que debe ser modificado para ir incorporándole la resolución de los distintos niveles de la prácticas.

  Action think(Sensores sensores)

- Un método que permite pintar sobre la parte gráfica del plan el camino encontrado.

  void VisualizaPlan(const estado &st, const list<Action> &plan)

Si nos vamos a la parte inferior del código vemos la declaración de los datos privados de la clase. Tiene declaradas las siguientes tres variables de estado:

- `actual`: de tipo **estado** y mantendrá la información sobre las coordenadas y orientación del agente en cada momento.
- `objetivos`: una lista de estados y mantendrá los objetivos que actualmente están activos. En el caso de los niveles del 0 al 2, sólo habrá un objetivo, en los dos últimos niveles habrá 3 objetivos activos.

- `plan`: lista de acciones que determinan un plan de movimientos. Se inicializa en una lista vacía.

El estudiante puede agregar tantas variables de estado como crea convenientes.

Como ya vimos, existe una variable **mapaResultado** que se incluye a partir del fichero ***'comportamiento.hpp'*** en donde se encuentra el mapa. En los niveles del 0 al 3 esta variable se utiliza básicamente como si fuera de sólo lectura, mientras que en el nivel 4 la matriz viene inicializada con '?' y se debe ir completando a medida que se descubre el mapa (al llenarla se irá automáticamente dibujando en la interfaz gráfica).

En el archivo ***'jugador.cpp'*** se describe el comportamiento del agente. En este archivo podemos añadir tantas funciones como necesitemos, pero las únicas funciones que se pueden modificar, y sólo para añadir código, son ***'pathFinding'*** y ***'think'***. El método ***'think'*** se dedica a definir las reglas que regirán el comportamiento del agente y devuelve un valor de tipo ***'Action'***. Este es un tipo de dato enumerado declarado en ***'comportamiento.hpp'*** que puede tomar los valores {**actFORWARD**, **actTURN_L**, **actTURN_R**, **actIDLE**} como ya hemos comentado anteriormente.

5. Método de evaluación y entrega de prácticas

5.1. Entrega de prácticas

Se pide desarrollar un programa (modificando el código de los ficheros del simulador 'jugador.cpp' y 'jugador.hpp') con el comportamiento requerido para el agente. Estos dos ficheros deberán entregarse en la plataforma PRADO de la asignatura, en un fichero ZIP, que no contenga carpetas, de nombre practica2.zip. Este archivo ZIP deberá contener sólo el código fuente de estos dos ficheros con la solución del alumno así como un fichero de documentación en formato PDF que describa el comportamiento implementado con un máximo de 5 páginas.

No se evaluarán aquellas prácticas que contengan ficheros ejecutables o virus.

5.2. Cuestionario de autoevaluación

Tras la entrega de la práctica se habilitará un plazo de 2 días para que los estudiantes realicen un proceso de autoevaluación de su trabajo. Para ello se suministrará un documento en el que se pedirá al estudiante que responda una serie de preguntas sobre cómo realizó la práctica, sobre algunas cuestiones de diseño y que ponga a prueba su software a partir de una serie de configuraciones iniciales que se le propondrán. El objetivo es determinar si se alcanza el grado de satisfacción para considerar los niveles presentados por el estudiantes superados.

5.3. Método de evaluación

En el método de evaluación se asocia una valoración máxima en puntos a cada uno de los niveles que se piden en esta práctica, siendo la distribución de puntos y requisitos para obtener dichos puntos los que se describen a continuación:

Nivel	Puntuación	Requisito
0	0	Terminar el nivel de demo
1	2	Tener correcto el nivel 0
2	3	Tener correcto el nivel 1 y 0

3	2	Tener correcto el nivel 2, 1 y 0
4	3	Tener correcto el nivel 2, 1 y 0 (no es necesario el nivel 3)

Nivel 0: Demo

En la versión que se entrega este nivel está sin terminar del todo. Lo que está implementado es el algoritmo de búsqueda usado (búsqueda en profundidad) y el agente devuelve el plan para llegar al objetivo, pero falta que el agente lleve a cabo el plan y se pueda ver en el entorno gráfico cómo alcanza la casilla objetivo.

La forma de hacerlo es mediante un comportamiento reactivo que va tomando una a una las acciones del plan en el orden correcto, va verificando que la acción se puede aplicar (no cae en una casilla precipicio o no tiene delante una casilla con un muro) y avanza sobre el mapa hasta llegar a la casilla objetivo.

Este comportamiento reactivo será común para los niveles del 1 al 3, donde el mapa es completamente conocido y no hay nada externo y dinámico que impida la consecución del plan. En el tutorial que se entrega con el material de la práctica se describe la forma en la que se resuelve este problema y se incorpora al software. Por tanto, este nivel se puede terminar de forma simple tras ver el tutorial.

Nivel 1: Encontrar el plan con el mínimo número de acciones

La única diferencia entre este nivel y el anterior, es el algoritmo de búsqueda utilizado. En este caso se pide que el plan que se obtenga tenga el mínimo número de acciones que permitan llevar al agente desde su posición actual hasta la casilla objetivo. Sabemos que una implementación correcta del algoritmo de búsqueda en anchura nos devuelve justo este plan. Así que se superará este nivel si se consigue esa implementación correcta.

Nivel 2: Encontrar el plan de mínimo consumo de batería.

Al igual que en el nivel anterior, este se consigue superar si se implementa un algoritmo de búsqueda que devuelva el plan de menor consumo de batería. En este caso los algoritmos candidatos serán, el algoritmo de costo uniforme (o Dijsktra) o una implementación del algoritmo A* usando una heurística admisible.

Nivel 3: Encontrar el plan de mínimo consumo de batería pasando por 3 casillas objetivo.

Aquí se pretende al igual que en el nivel anterior encontrar un plan de mínimo consumo de batería, pero en este caso, el camino debe ser óptimo pasando obligatoriamente por 3 casillas. Aunque el algoritmo que se implemente aquí pueda ser igual que el algoritmo usado en el nivel 2 (costo uniforme o A*) debe ser adaptado convenientemente para que sea capaz de encontrar el plan óptimo.

Algunos comentarios sobre este nivel. Primero, no es necesario tenerlo resuelto para poder abordar el nivel 4. Segundo, el espacio de búsqueda de soluciones es muy alto incluso con mapas de pocas casillas, por eso recomendamos que en la experimentación se pongan los objetivos cerca entre sí y cerca del agente.

Para considerar superado este nivel es necesario que los planes que se obtengan sean óptimos.

Nivel 4: Reto.

Este es el nivel más especial y el que se plantea como un juego. En este nivel se pueden usar los algoritmos de búsqueda implementados para los niveles anteriores o se puede definir un algoritmo de búsqueda nuevo exclusivo para este nivel. La idea es definir una estrategia que permita conseguir el máximo número de objetivos posible antes de agotar alguno de estos tres recursos: el número máximo de instantes de simulación que es 3000, la batería que es también de 3000 y el tiempo acumulado de pensar por el agente que es de 300 segundos.

Para valorar este nivel se realizarán pruebas sobre distintos mapas con distintas configuraciones de posiciones iniciales y de listas de objetivos. En base al resultado de esas pruebas se otorgará una calificación entre 0 y 3 puntos.

La nota final se calculará sumando los puntos obtenidos en cada nivel, teniendo en cuenta las restricciones descritas anteriormente relativas a que para considerar algunos de los niveles superiores, deben estar resueltos los niveles inferiores.

5.4. Fechas Importantes

La fecha tope para la entrega será el **LUNES 10 DE MAYO DE 2021** antes de las **23:00 horas** y desde el 11 DE MAYO estará disponible la entrega para el cuestionario de autoevaluación hasta **EL JUEVES 13 DE MAYO** a las **23:00 horas**.

5.5. Observaciones Finales

Esta **práctica es INDIVIDUAL**. El profesorado para asegurar la originalidad de cada una de las entregas, someterá a estas a un procedimiento de detección de copias. En el caso de detectar prácticas copiadas, los involucrados (tanto el que se copió como el que se ha dejado copiar) tendrán suspensa la asignatura. Por esta razón, recomendamos que en ningún caso se intercambie código entre los alumnos. No servirá como justificación del parecido entre dos prácticas el argumento *"es que la hemos hecho juntos y por eso son tan parecidas"*, o "como estudiamos juntos, pues...", ya que como se ha dicho antes, **las prácticas son INDIVIDUALES**.

Como se ha comentado previamente, el objetivo de la defensa de prácticas es evaluar la capacidad del alumno para enfrentarse a este problema. Por consiguiente, se asume que todo el código que aparece en su práctica ha sido introducido por él por alguna razón y que dicho alumno domina perfectamente el código que entrega. Así, si durante cualquier momento del proceso de defensa el alumno no justifica adecuadamente algo de lo que aparece en su código, la práctica se considerará copiada y tendrá suspensa la asignatura. Por esta razón, aconsejamos que el alumno no incluya nada en su código que no sea capaz de explicar qué misión cumple dentro de su práctica y que revise el código con anterioridad a la defensa de prácticas.

Por último, las prácticas presentadas en tiempo y forma, pero no defendidas por el alumno, se considerarán como no entregadas y el alumno obtendrá la calificación de 0. El supuesto anterior se aplica a aquellas prácticas no involucradas en un proceso de copia. En este último caso, el alumno tendrá suspensa la asignatura.