INTELIGENCIA ARTIFICIAL

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

Práctica E1

Convocatoria Extraordinaria

Agentes Reactivos/Deliberativos

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL
UNIVERSIDAD DE GRANADA
Curso 2023-2024

1. Introducción

La primera práctica de la convocatoria extraordinaria 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 su entorno y de las consecuencias de sus acciones. El marco general de esta práctica extraordinaria es semejante al que se proporcionó en la práctica 2 de la convocatoria ordinaria. Cambia con respecto a esta, las cosas que se piden y la puntuación asociada a cada parte. A continuación, pasamos a hacer esa descripción general de la práctica[1].

2. Descripción

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.

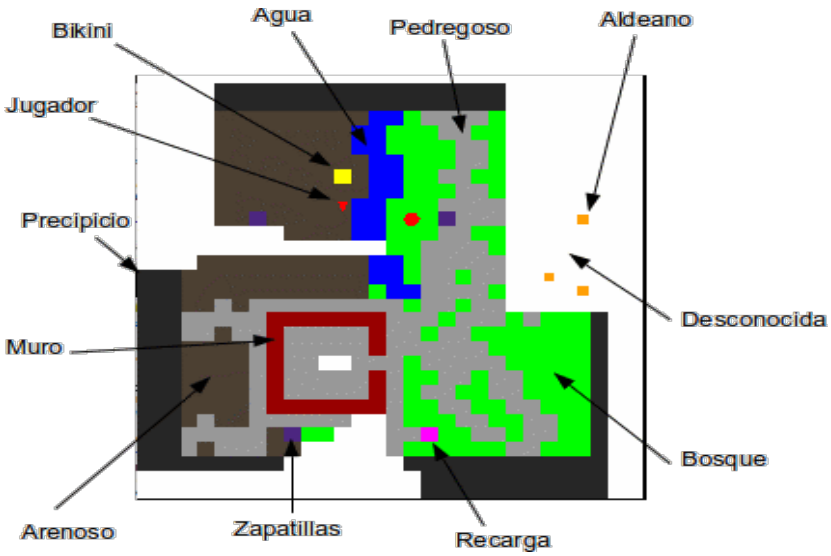
Representaremos dicha superficie usando una matriz donde la primera componente representa la fila y la segunda representa la columna dentro de nuestro mapa. Como ejemplo usaremos un mapa de tamaño 100x100 de caracteres. Fijaremos sobre este mapa las siguientes consideraciones:

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

1 Mucho de lo que aparece en esta descripción ya se había dicho en la práctica 2. Se hace para que este documento esté autocontenido y pueda ser entendido por aquellos que no intentaron hacer la versión de la convocatoria ordinaria. Se han sombreado de naranja las partes que cambian sustancialmente en cuanto a la definición del problema en esta sección 2 de descripción en relación a la de la convocatoria ordinaria.

- 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 del terreno son los siguientes:

- ***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 y caer en ella supone el final del juego.
- ***Suelo de piedra***, 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 que se quede en esta casilla sin *hacer nada*[2] 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. Este es nuestro personaje, sólo habrá un jugador sobre el mapa.
- ***Colaborador*** se codifica con el carácter **'c'** y se muestran como un triángulo naranja en el mapa general. Es un aldeano que se es receptivo a recibir órdenes del Jugador y que puede ayudar a este a agilizar tus tareas.
- ***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 Más adelante en este documento cuando se describan las acciones se verá que "no hacer nada" corresponde con la acción ***actIDLE***.

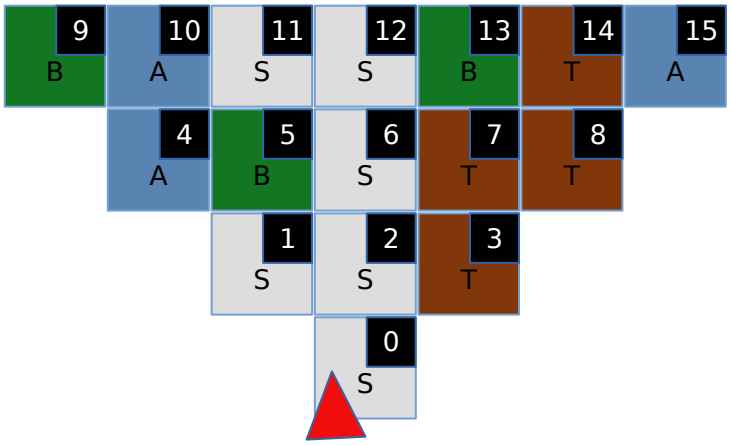
- Lobos, se codifican con el carácter **'l'** y se muestran como un círculo rosa. Son personajes que se desplazan a través del mapa tratando de obstaculizar los movimientos del agente jugador y del agente colaborador y en algunas ocasiones y bajo ciertas circunstancias no bien conocidas pueden empujarlos. Empujar a un agente consiste en desplazarlo a una casilla en la dirección en la que avanzaría el lobo. Para que un lobo pueda empujar al agente es necesario que esté en una casilla adyacente a la del agente y además encarado hacia él.

2.2. El agente jugador

Tendremos que definir comportamientos para el que llamaremos agente jugador, aunque dentro de la simulación este agente podrá involucrar en sus planes al que llamaremos agente colaborador. El objetivo de este juego será hacer desplazarse desde su ubicación actual (origen) a una casilla especificada en el mapa (destino) o bien al agente jugador o bien al agente colaborador.

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 **agentes**, es capaz de mostrarnos qué objetos móviles se encuentran en nuestra visión (es decir, aldeanos, colaboradores y lobos). 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:

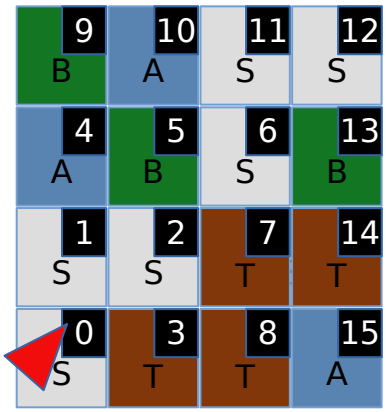


El primer carácter (posición 0) representa el tipo de terreno sobre el que se encuentra nuestro personaje. El tercer carácter (posición 2) es justo el tipo de terreno que tengo justo delante de mí. Los caracteres de las posiciones 4, 5, 6, 7 y 8 representan lo que está delante de mí, pero con una casilla 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 **agentes,** 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, **'c'** para indicar que hay un colaborador, **'l'** para indicar la posición de un lobo, **'j'** para indicar la posición del agente y **'_'** si la casilla está desocupada.

La detección de un colaborador en el sensor de agentes indica que dicho agente puede recibir órdenes del jugador. Si no se da esta condición, el jugador no puede influir sobre él.

La interpretación de los sensores con respecto al mapa real es la anterior cuando el jugador se encuentra orientado en las posiciones **norte**, **este**, **sur** y **oeste**. Cuando su orientación es **noreste**, **sureste**, **suroeste** y **noroeste** la representación cambia ligeramente, de manera que si el vector terreno contiene **SSSTABSTTBASSBTA**, su representación real sobre un plano será la siguiente:



El agente cuenta además con los sensores siguientes:

- **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 o bien el colaborador haya chocado contra un muro o el jugador haya recibido un empujón de un lobo.
- **Sensor de vida (`reset`)**: Es una variable booleana que toma el valor de verdad si una de estas circunstancias sucede: el agente colaborador choca contra otro agente, en cuyo caso el agente colaborador se reinicia en otro punto del mapa o alguno de los dos agentes cae por un precipicio en cuyo caso la simulación termina.
- **Sensores de posición del jugador (`posF`, `posC`)**: Devuelve la posición de fila y columna que ocupa el agente jugador.
- **Sensor de orientación del jugador (`sentido`)**: Devuelve la orientación del agente jugador.
- **Sensores de posición del colaborador (CLB`posF`, CLB`posC`)**: Devuelve la posición de fila y columna que ocupa el agente colaborador.
- **Sensor de orientación del colaborador (CLB`sentido`)**: Devuelve la orientación del agente colaborador.
- **Sensor de la ubicación de la casilla objetivo** (`destinoF`, `destinoC`): Mantiene la información de la casilla que se tiene que alcanzar. En la primera

se describe la coordenada de la fila y en la segunda la coordenada de la columna.

- **Sensor de llegada al destino del colaborador (CLBgoal):** El sensor se activa cuando es el colaborador el que llega a la casilla destino.
- **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 : Anchura para el agente jugador.

  1 : Dijkstra para el agente jugador.

  2: Escalada máxima pendiente para el agente colaborador.

  3 : A* para el de menor coste de los dos.

  4 : Reto (Maximizar puntuación por 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 **cada agente 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 pasa 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. Todo esto es aplicable a todos los agentes que intervienen en el juego.

Importante resaltar que el agente jugador no tiene acceso a ninguna información sobre el sistema sensorial del agente colaborador que no sea la que ya se ha explicado en la

descripción de sus sensores. El estudiante debe encargarse de usar apropiadamente todo el sistema sensorial del agente jugador para detectar y tratar de evitar situaciones peligrosas en las que se podría encontrar el agente colaborador.

2.2.2. Datos del agente jugador 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 agente jugador

El agente puede realizar las siguientes acciones:

- ***actWALK:*** le permite al jugador 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 él.
- ***actRUN***: el agente jugador avanza dos casillas siguiendo su orientación actual. Para que la acción finalice con éxito es necesario que tanto la casilla final como la casilla intermedia sean transitables para el agente colaborador.
- ***actTURN_L:*** le permite mantenerse en la misma casilla y girar a la izquierda 90º teniendo en cuenta su orientación.
- ***actTURN_SR:*** le permite mantenerse en la misma casilla y girar a la derecha 45º teniendo en cuenta su orientación.
- ***actWHEREIS:*** pone en los sensores ***posF***, ***posC*** y ***sentido*** la fila, la columna y la orientación actual del agente jugador y en **CLBposF**, **CLBposC** y **CLBsentido** la información del agente colaborador.
- ***act_CLB_WALK:*** provoca que el colaborador avance a la siguiente casilla del mapa siguiendo su orientación actual. Para que la acción finalice con éxito es necesario que la casilla de destino sea transitable para el agente colaborador. Además, en el momento de recibir la orden, el colaborador debe ser visible en el sensor de agentes del jugador.

- ***act_CLB_TURN_SR:*** provoca que el agente colaborador permanezca en la misma casilla pero que cambie su orientación 45º a la derecha. En el momento de recibir la orden, el colaborador debe ser visible en el sensor de agentes del jugador.
- ***act_CLB_STOP***: ordena al agente colaborador que pare. Esta acción es importante, ya que el agente colaborador en cada iteración aplica la última orden que recibió del agente jugador. Por ejemplo, si el agente jugador le ordenó act_CLB_TURN_SR para girar, en el siguiente instante el agente colaborador seguirá aplicando esta acción de giro, y lo seguirá haciendo hasta recibir otra orden del agente jugador. Solo cuando reciba esta orden act_CLB_STOP no aplicará ninguna acción. ***Sólo existe un caso especial en el que el colaborar se detendrá (es decir, se aplicará automáticamente la acción act_CLB_STOP sin que el jugador si lo ordene) y esto es cuando sea él el que se posicione en una casilla destino.***
- ***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 tanto la acción **actIDLE** como la acción **act_CLB_STOP** consume **0** de batería. La acción **actWHEREIS** consume **200** de batería. ***El consumo asociado al resto de las acciones depende del tipo de terreno a donde el agente inició la acción***. Así en las acciones **actTURN_L**, **actTURN_SR**, **actWALK** y **actRUN** dependen de la casilla de inicio del agente jugador, mientras que las acciones **act_CLB_WALK y act_CLB_TURN_SR** dependen de la casilla de inicio del agente colaborador. Esto es, si el agente jugador 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. Las acciones que implican el movimiento del colaborador también se tendrán en cuenta para el cálculo del consumo de batería. Así, en cada iteración se restará de la batería del jugador el consumo que provoca el movimiento del agente jugador y el movimiento del agente colaborador.

En las siguientes tablas se muestran los valores de consumo de energía en función de la acción a realizar, la casilla de inicio de la acción y dependiendo del objeto que se tenga en posesión en ese momento.

actWALK / act_CLB_WALK (jugador y colaborador)		
Casilla Inicial	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	100	10 (con **Bikini**)
'B'	50	15 (con **Zapatillas**)
'T'	2	2
Resto de Casillas	1	1

actRUN (jugador)		
Casilla Inicial	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	150	15 (con **Bikini**)
'B'	75	25 (con **Zapatillas**)
'T'	3	3
Resto de Casillas	1	1

actTURN_L (jugador)		
Casilla Inicial	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	30	5 (con **Bikini**)
'B'	7	1 (con **Zapatillas**)
'T'	2	2
Resto de Casillas	1	1

actTURN_SR / act_CLB_TURN_SR (jugador y colaborador)		
Casilla Inicial	**Gasto Normal Batería**	**Gasto Reducido Batería**
'A'	10	2 (con **Bikini**)
'B'	5	1 (con **Zapatillas**)
'T'	1	1
Resto de Casillas	1	1

Una última consideración sobre las tablas y los consumos de batería. Para que se produzcan los consumos reducidos de energía en el movimiento del colaborador es necesario que este haya pasado previamente por una casilla de bikini o zapatillas. Que el jugador tenga el bikini, no implica que lo tenga el colaborador y viceversa. Cada agente lo debe conseguir pasando por encima de la casilla correspondiente.

3. Objetivo de la práctica

El objetivo de esta práctica es dotar de un comportamiento inteligente al agente jugador usando un agente reactivo/deliberativo para definir las habilidades que le permitan alcanzar los objetivos que se establecen en los niveles definidos. Los niveles han sido diseñados para que vayan incrementando en dificultad, empezando por un nivel 0 que está muy dirigido para ayudar al estudiante a iniciar la práctica.

Nivel 0: Encontrar el camino con el mínimo número de acciones que lleve al agente JUGADOR a una casilla objetivo

La construcción de este nivel irá guiada por el tutorial que ya se usó en la práctica 2 de la convocatoria ordinaria y que está disponible con el material de la asignatura en PRADO. Dicho tutorial ayudará al estudiante a empezar con el desarrollo de la práctica[3], detectar los elementos que son relevantes en el diseño de cada algoritmo y esperamos que agilice y facilite la elaboración del resto de los niveles. Así, que el primer paso (después de leer este guion) para empezar con la resolución de la práctica será seguir dicho tutorial.

El objetivo que se persigue en este nivel es encontrar una secuencia de acciones para llevar al agente jugador desde la posición donde aparece en el mapa a la posición establecida como destino. Las condiciones iniciales para la realización de esta búsqueda son de información completa, es decir, se conoce todo el mapa (a través de ***mapaResultado***) y el sistema sensorial del jugador funciona correctamente y por tanto puede acceder tanto a su posición real (a través de ***posF***, ***posC*** y ***sentido***) como a la posición del agente colaborador (con ***CLBposF***, ***CLBposC***, ***CLBsentido***). En este nivel, el agente colaborador debe ser percibido por el agente jugador como un mero obstáculo del camino, ya que **el plan no puede incluir ninguna orden que implique un movimiento del colaborador**.

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

3 Esto solo lo tendrán que hacer los estudiantes que no lo hicieron en durante la convocatoria ordinaria. Los que ya lo hicieran, pueden usar directamente la implementación desarrollada en aquel momento. Esto es aplicable al resto de implementaciones que el estudiante hiciera durante la convocatoria ordinaria y SIEMPRE QUE SEAN DE AUTORÍA PROPIA.

Nivel 1: Encontrar el camino con el mínimo consumo de batería usando el ALGORITMO DE DIJKSTRA (coste uniforme) que lleve al agente JUGADOR a una casilla objetivo

Ahora el agente jugador 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 ni lobos. Las diferencias con el nivel anterior son: el tipo de camino que se pide (el mínimo en consumo de batería) y que el plan solo puede contener acciones para el agente jugador (ninguna para el agente colaborador).

El objetivo es construir y ejecutar con éxito un plan (con solo acciones para el agente jugador) para llevar al agente jugador desde su posición inicial al destino con el menor consumo de batería posible (usando el algoritmo de Dijsktra).

Nivel 2: Aplicar el algoritmo de ESCALADA MÁXIMA PENDIENTE para encontrar un camino que lleve al agente COLABORADOR a una casilla objetivo

Para este nivel trabajaremos con el método de escalada de máxima pendiente. Haremos uso de una heurística ya definida llamada en el código HeuristicaParaNivel2 basada en las distancias de manhattan de los dos agentes a la casilla objetivo. Es obligatorio usar esa heurística.

Una cuestión importante es que en la implementación del método de escalada se tendrá en cuenta el siguiente criterio de desempate en el caso que dos o más hijas obtengan la misma valoración se seleccionará como mejor el primero que aparezca antes en esta lista: `act_IDLE`, `act_CLB_WALK`, `act_CLB_TURN_R`, `act_CLB_STOP`, `actWALK`, `actRUN`, `actTURN_L`, `actTURN_SR`.

En el caso que el método de escalada se quede en un óptimo local y no encuentre un camino solución a la casilla objetivo, se usará el algoritmo de dijkstra que se definió en el nivel anterior, y en este caso, será el agente JUGADOR el que llegue al objetivo.

El objetivo es usar el método de escalada de máxima pendiente tomando como heurística la que se incluye en el software para construir y ejecutar

con éxito un plan que debe llevar al agente colaborar desde su posición inicial al destino. Si el método de escalada de máxima pendiente no encuentra solución, se usará el algoritmo de Dijkstra definido en el nivel 1 y será entonces el agente jugador el que debe llegar a la casilla objetivo.

Nivel 3: Encontrar el camino con el mínimo consumo de batería usando el ALGORITMO A* que lleve a uno de los dos (JUGADOR o COLABORADOR) a una casilla objetivo

En este nivel, el agente que debe alcanzar la casilla objetivo es aquel de los dos cuyo plan (que ha de ser óptimo) implique un menor consumo de batería. Para la construcción de dicho plan será necesario utilizar el algoritmo A*. Como se requiere que la solución sea óptima, el estudiante tiene que utilizar una heurística que sea admisible para este problema. En esta aproximación, al igual que en la anterior, se debe tener en cuenta que el consumo de batería depende del agente que lo realice, de la casilla donde empiece dicha acción y de si se está o no en posesión de los objetos que permiten reducir el consumo de batería.

El objetivo del agente jugador es crear y llevar a cabo un plan de acciones en el mapa para hacer que llegue a la posición destino el agente que menor cantidad de batería consuma para conseguirlo (usando el algoritmo A* con una heurística admisible).

Nivel 4: Reto (Maximizar la puntuación en los objetivos alcanzados)

El agente no conoce el mapa ni su posición, ni su orientación, ni la posición y orientación del colaborador (estos sensores solo funcionarán cuando se usa la acción **actWHEREIS**). En este nivel se le irán proponiendo al agente casillas objetivo que deberá ir alcanzando. El agente debe ir proponiendo planes para alcanzar (él o el agente colaborador) 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 por zonas del espacio que no interesen (por coste o porque no sea posible pasar) y requiera tomar alguna decisión al respecto. Además, en este nivel hay aldeanos y lobos que se pasean sin un destino fijo y se pueden convertir en obstáculos que impidan la consecución del plan. En consecuencia, será necesario replanificar. Los aldeanos y los lobos pueden

detectarse gracias al **sensor agentes** del agente jugador, es decir, no podemos saber dónde están los aldeanos en el mapa, pero sí podemos percibirlos con nuestro sensor si estamos cerca de ellos.

En estas situaciones deberemos integrar comportamientos reactivos y deliberativos, de manera que el agente debe encontrar una secuencia de acciones que terminen ubicando al agente jugador o al agente colaborador en la casilla destino. Dicho plan debe obtenerse por el algoritmo de búsqueda que el estudiante considere más apropiado para este problema, y, que durante la ejecución de dicho plan, el agente sepa cómo actuar ante un posible fallo que impida la consecución del plan.

Cada vez que se consiga llegar a la casilla objetivo se considerará que la misión está completada, se computará con una puntuación la consecución de esa misión por el jugador, se generará una nueva casilla objetivo y el agente tiene que repetir el proceso de intentar completar la nueva misión. La puntuación que se obtiene por la consecución de la misión depende del agente (el jugador o el colaborador) que se sitúe primero en la casilla destino. Si es el agente jugador, se obtendrá un punto de valoración. Si es el agente colaborador, se obtendrán 10 puntos. Hay que recordar que el agente colaborador se detiene cuando llega a una casilla destino.

Cada simulación empezará con 3000 instantes de simulación, 3000 puntos de batería y con un tiempo máximo de 300 segundos para elaborar los planes. La simulación termina cuando alguno de estos valores llega a cero.

El objetivo es definir un comportamiento reactivo/deliberativo que permita a un agente jugador y a un agente colaborador obtener la mayor puntuación posible por la consecución de misiones durante la duración de una simulación. Parten de un mapa desconocido, con posiciones desconocidas donde puede haber aldeanos y lobos. La puntuación asignada por la consecución de una misión depende de qué agente fue el primero en llegar a la casilla objetivo (1 punto agente jugador, 10 puntos agente colaborador).

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 proporciona sólo versión para el sistema operativo Linux y se puede encontrar en https://github.com/ugr-ccia-IA/practicaE1. La versión del software está preparada para ser usada para la distribución de UBUNTU, aunque es fácil de adaptar para cualquier otra distribución con pequeños cambios. En la versión proporcionada se incluye un archivo 'install.sh' para cargar las librerías necesarias y compilar el programa. 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*** (son las mismas que se necesitaban en la práctica 1).

El proceso de instalación es muy simple y consiste en seguir las instrucciones que se proporcionan en el repositorio de GitHub (https://github.com/ugr-ccia-IA/practicaE1) para acceder y trabajar con el software. Leer detenidamente las instrucciones que se proporcionan en ese repositorio para conseguir una correcta configuración del software.

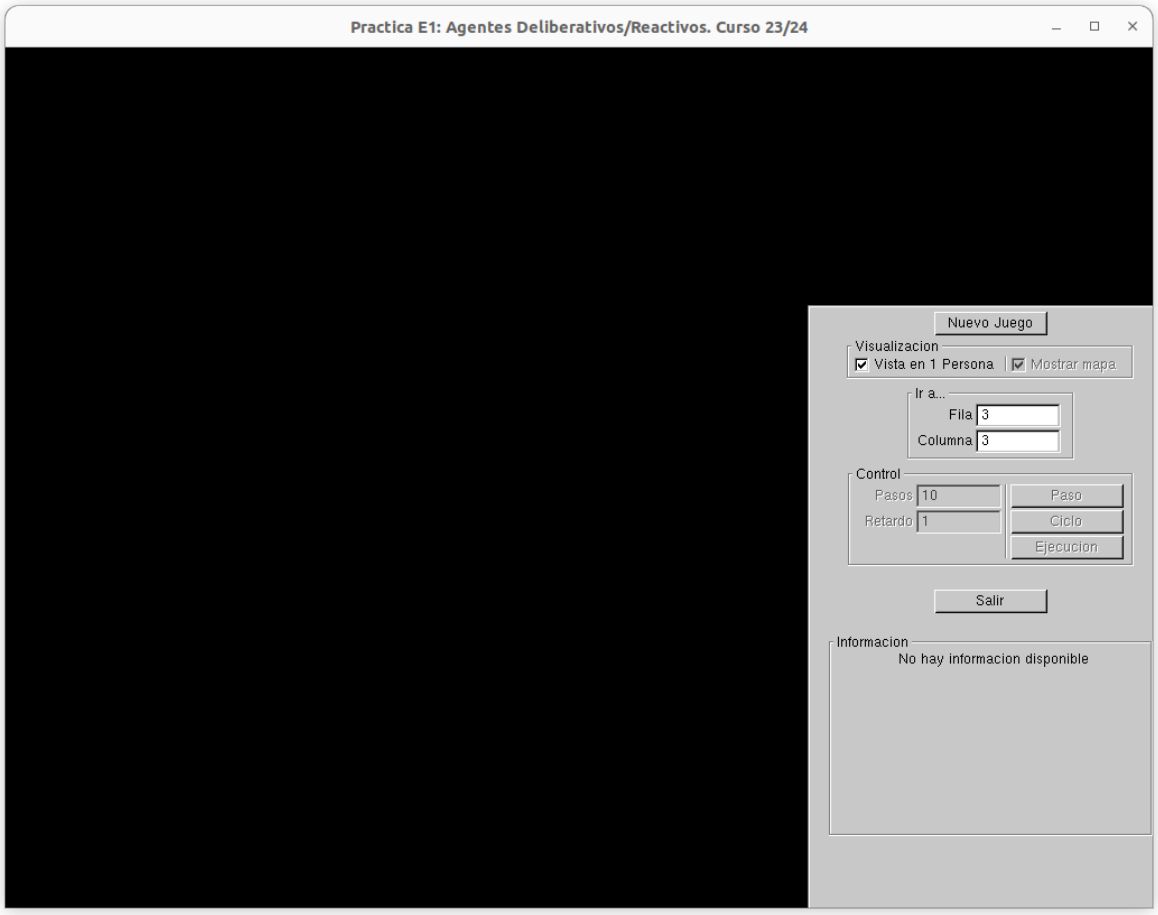
4.2. Funcionamiento del Programa

Existen dos ficheros ejecutables: practicaE1 y practicaE1SG. 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

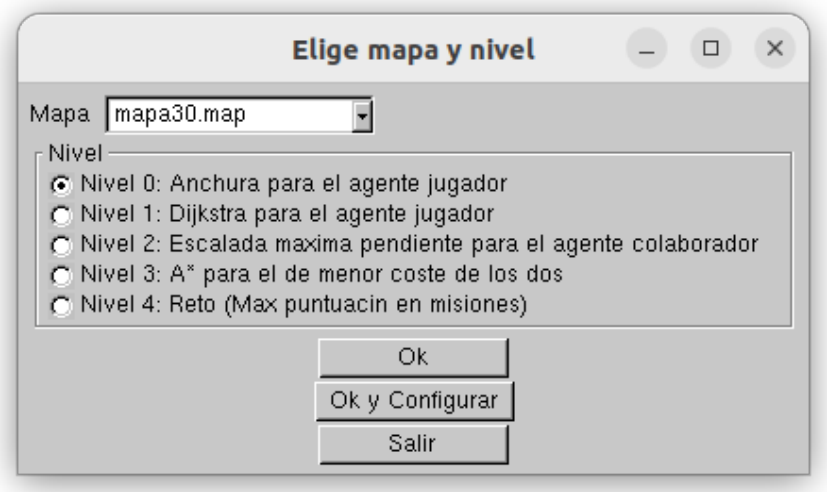
Para ejecutar el simulador con interfaz hay que escribir "***./practicaE1***".



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 3 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: Anchura para el agente jugador** fijando como mapa **mapa30.map**, y presionaremos el botón de **Ok**.

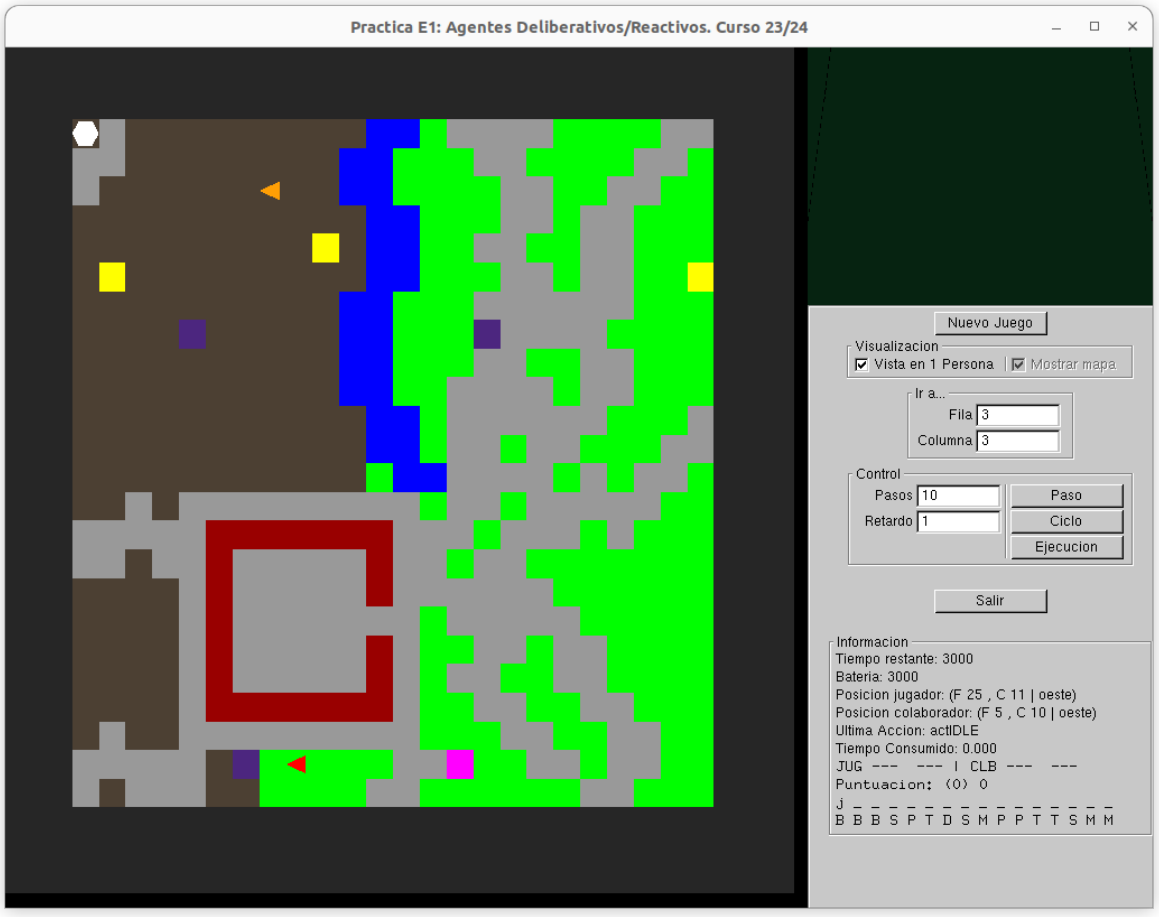


La ventana principal se actualizará y podremos entonces distinguir tres partes: la izquierda que está destinada a dibujar el mapa del mundo, la superior derecha que mostrará una visión del mapa desde el punto de vista del agente, y la inferior derecha que contiene los botones que controlan el simulador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

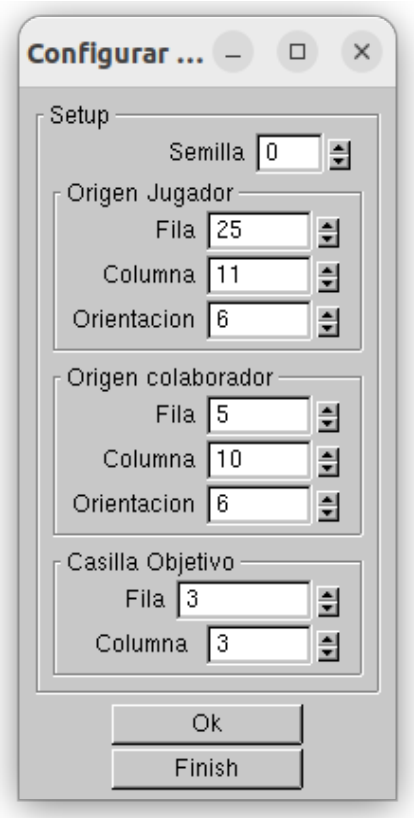
El último botón que podemos encontrar es **Salir** que cierra la aplicación.

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.



Se puede observar que bajo el nombre "Ir a .." tenemos la **Fila** y **Columna** de la casilla objetivo. 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 jugador, la posición y orientación inicial del agente colaborador y el objetivo.

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.
- **Batería**: cantidad de batería que le queda al agente.
- **Posición jugador**: indica fila, columna y orientación del agente jugador.
- **Posición colaborador**: indica fila, columna y orientación del agente colaborador.
- **Última Acción**: 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.
- **JUG:** indicará que objeto tiene el agente jugador. Si no tiene ninguno aparecerá con eel valor "---". Di igual forma tras la palabra SON: se indicará el objeto en posesión del agente colaborador. Se recuerda que cada agente solo puede tener un objeto.
- **Puntuación**: indica el número de misiones conseguidas hasta el momento (entre paréntesis) y la puntuación por esas misiones conseguidas en el nivel 3.

Bajo el texto ********** **Visión** ********** 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 **gdb.** Al hacer ***make*** se generan automáticamente los dos ejecutables ***practicaE1*** y ***practicaE1SG***. 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 **practicaE1SG** 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:

./practicaE1SG <mapa> <semilla> <nivel> <filaJ> <colJ> <oriJ> <filaS> <colS> <oriS> <filO$_i$> <colO$_i$>

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 3 indicando que nivel se quiere ejecutar
- **<filaJ>** fila inicial donde empezará el agente jugador
- **<colJ>** columna inicial donde empezará el agente jugador
- **<oriJ>** un número entre 0 y 7 indicando la orientación con la que empezará el agente jugador, siendo 0=norte, 1=noreste, 2=este, 3=sureste, 4=sur, 5=suroeste, 6=oeste y 7=noroeste.
- **<filaS>** fila inicial donde empezará el agente colaborador
- **<colS>** columna inicial donde empezará el agente colaborador
- **<oriS>** un número entre 0 y 7 indicando la orientación con la que empezará el agente colaborador, siendo 0=norte, 1=noreste, 2=este, 3=sureste, 4=sur, 5=suroeste, 6=oeste y 7=noroeste.
- **<filO$_i$>** fila de la casilla del objetivo i-ésimo
- **<colO$_i$>** columna de la casilla del objetivo i-ésimo.

Por ejemplo, podemos ejecutar **./practicaE1 mapas/mapa30.map 1 0 4 5 1 7 7 3 12 5** lo cual utilizará el mapa llamado **mapa30.map** indicado con una semilla **1** en el nivel **0** situando al agente jugador en la posición de fila **4** y columna **5**, con orientación **noreste** (**1**), situando al agente colaborador en la fila **7** y columna **7** con orientación **sureste** (3). Por estar en el nivel **0**, es el agente jugador el que debe ir a la casilla objetivo de fila **12** y columna **5**. En el caso de no indicar destinos suficientes, el simulador los elegirá al azar. Si se proponen más destinos de los necesarios, como ocurre en esta llamada 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,
- cantidad de mapa descubierto,
- 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 **practicaE1**, es decir,

./practicaE1 <mapa> <semilla> <nivel> <filaJ> <colJ> <oriJ> <filaS> <colS> <oriS> <filO$_i$> <colO$_i$>

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 un comando como este

el entorno gráfico se detendría como muestra la imagen, indicando con rombos rojos el avance del agente jugador y con rombos naranjas el avance del agente colaborador.

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.

```
#ifndef COMPORTAMIENTOJUGADOR_H
#define COMPORTAMIENTOJUGADOR_H

#include "comportamientos/comportamiento.hpp"

#include <list>

double HeuristicaParaElNivel2(const ubicacion &jugador, const ubicacion &colaborador, const ubicacion &final);


class ComportamientoJugador : public Comportamiento {
  public:
    ComportamientoJugador(unsigned int size) : Comportamiento(size) {
      // Inicializar Variables de Estado
    }
    ComportamientoJugador(std::vector< std::vector< unsigned char> > mapaR) : Comportamiento(mapaR) {
      // Inicializar Variables de Estado
    }
    ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
    ~ComportamientoJugador(){}

    Action think(Sensores sensores);
    int interact(Action accion, int valor);


  private:
    // Declarar Variables de Estado

};

#endif
```

Empecemos con el archivo ***'jugador.hpp'.*** En él se declara la clase ComportamientoJugador y se incluye la cabecera de la función heurística que se debe usar en el método de escalada. De los métodos implementados en la parte pública vamos a destacar 3 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 3 de la práctica.

  ComportamientoJugador(unsigned int size) : Comportamiento(size)

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

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

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

  Action think(Sensores sensores)

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 describirá el comportamiento del agente. Revisé el tutorial para ver como se ha de modificar dicho fichero ya que en su versión inicial, el método `think` qué es el que hay que desarrollar se encuentra vacío.

```cpp
#include "../Comportamientos_Jugador/jugador.hpp"
#include "motorlib/util.h"

#include <iostream>
#include <cmath>
#include <set>
#include <stack>

// Cabecera de la heurística que se tiene que usar en el nivel 2.
// Tiene 3 entradas que son la ubicación del jugador, del colaborador y la de la casilla destino.
double HeuristicaParaElNivel2(const ubicacion &jugador, const ubicacion &colaborador, const ubicacion &final){
    double d1 = abs(colaborador.f - final.f) + abs(colaborador.c - final.c);
    double d2 = abs(jugador.f - colaborador.f) + abs(jugador.c - colaborador.c);
    double d3 = 8 - (int) colaborador.brujula;

    if (d1 == 0)
        return d1;
    else
        return (d1*d1*8) + d3 + d2;
}

// Este es el método principal que se piden en la practica.
// Tiene como entrada la información de los sensores y devuelve la acción a realizar.
// Para ver los distintos sensores mirar fichero "comportamiento.hpp"
Action ComportamientoJugador::think(Sensores sensores)
{
    Action accion = actIDLE;

    // Incluir aquí el comportamiento del agente jugador

    return accion;
}


int ComportamientoJugador::interact(Action accion, int valor)
{
    return false;
}
```

También se encuentra la definición de la heurística que se tiene que utilizar en el nivel 2 de esta práctica. La función tiene 3 entradas que son tres parámetros de tipo `ubicacion`: el del jugador, el del colaborador y la casilla final.

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 **practicaE1.zip**. Este archivo ZIP deberá contener sólo el código fuente de estos dos ficheros con la solución del alumno.

No se evaluarán aquellas prácticas que no contengan exclusivamente estos 2 ficheros y exactamente con estos nombres.

5.2. 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 0
1	1	Tener correcto el nivel 0
2	2	Tener correcto el nivel 1 y 0
3	2	Tener correcto el nivel 2, 1 y 0
4	5	Tener correcto el nivel 3, 2, 1 y 0

Nivel 0: Anchura para el agente jugador.

Hay dos tareas importantes a desarrollar en este nivel. Por un lado, la implementación de un método de búsqueda en anchura para encontrar el plan con el mínimo número de acciones que permitirán llevar al agente jugador a la casilla destino. Por otro lado, la implementación de la parte reactiva del comportamiento. En este nivel, donde el mapa es conocido y no hay otros agentes en el mundo (excepto el agente colaborador) el

comportamiento básico es coger la siguiente acción del plan que se está ejecutando, mandar a que se ejecute y eliminarla del plan.

El tutorial que se adjunta en la práctica irá guiando al alumno por este desarrollo del software de forma que la realización completa de dicho tutorial implicará que quede resuelto el problema que se plantea en este nivel.

En lo relativo a la calificación de este nivel en la práctica decir que aunque aporta 0 puntos a la calificación final, su correcta ejecución es imprescindible para que el resto de niveles sean considerados.

Nivel 1: Algoritmo de Dijsktra para llevar al agente jugador al destino con el mínimo consumo de batería

El objetivo de este nivel se centra en la implementación del algoritmo de búsqueda, en este caso en el de Dijsktra (también llamado Coste Uniforme), para encontrar un camino a al casilla objetivo para el agente jugador (el plan solo puede incluir acciones para este agente), con el mínimo coste de batería posible.

A diferencia del nivel anterior, en este nivel y en el siguiente, se evalúan los costes de los planes por el consumo de la batería, y en dicho consumo influye el tipo de terreno, la acción aplicada y si se está en posesión o no del objeto que permite reducir el consumo. Esto en esencia va a indicar que el concepto de estado debe contener más información que el que tenía en los niveles anteriores.

Obtener un funcionamiento completamente correcto de este nivel implicará 1 puntos más a la calificación final. Una implementación muy deficiente llevaría a obtener 0 puntos y a no evaluar los niveles superiores.

Nivel 2: Escalada máxima pendiente para llevar al agente colaborador al destino.

El objetivo es la implementación del método de escalada de máxima pendiente. Además se pide que haciendo uso de la heurística proporcionada (de nombre `HeuristicaParaElNivel2`) se use esta implementación para construir un camino que lleve al agente colaborador a la casilla destino. Si no fuera posible encontrar ese camino por quedarse el método en un óptimo local, en ese caso, se devolvería un camino

que llevara al agente jugador a la casilla destino haciendo uso del algoritmo definido en el nivel 1.

La correcta implementación del algoritmo pedido para este problema implicará 2 puntos adicionales en la notal final.

Nivel 3: Algoritmo A* para llevar a uno de los dos agentes (jugador o colaborador) al destino con el mínimo consumo de batería.

El objetivo es la implementación del algoritmo A*, y por tanto, la obtención de secuencias de acciones que traten de optimizar el coste (reducir el gasto de batería en este caso) en el problema que se plantea en esta práctica. En este nivel hay tanto que implementar correctamente este algoritmo como definir una heurística que permita asegurar que las soluciones que encuentra son óptimas.

Los planes obtenidos deben determinar cuál de los dos agentes puede ser llevado a la casilla objetivo consumiendo la menor cantidad de batería. Indicar que es necesario considerar que objeto tiene el agente jugador y que objeto tiene el agente colaborador (si es que tienen alguno), ya que cada uno de ellos lo debe de tener para poder aplicarse las reducciones de batería, y por tanto, cada uno de ellos pasar por la casilla que permite obtenerlos.

La correcta implementación del algoritmo pedido para este problema implicará 2 puntos adicionales en la nota final.

Nivel 4: Reto. Maximizar la puntuación de misiones.

Este es el nivel 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 algoritmos de búsqueda nuevos, exclusivos para este nivel. La idea es definir una estrategia que permita conseguir por un lado la máxima puntuación en la consecución de misiones y por otro maximizar la cantidad de mapa que es descubierto. Una misión consiste en llevar o bien al agente jugador o al agente colaborador a una casilla marcada como destino. La puntuación obtenida en cada misión conseguida será de 1 punto si es el agente jugador el primero en llegar a la casilla destino y 10 puntos si es el agente colaborador el que lo hace primero. Tras cada misión conseguida se genera una nueva misión, es decir, una nueva casilla

destino. El juego continúa hasta que se agota alguno de estos tres recursos: el número máximo de instantes de simulación que es 3000 inicialmente, la batería que es al inicio de 3000 y el tiempo acumulado de pensar por el agente que es de 300 segundos.

Inicialmente el mapa es desconocido, las posiciones de los agentes jugador y colaborador también lo son y además puede haber en los mapas aldeanos y lobos. Todo esto implica que la parte reactiva del comportamiento también debe modificarse para tener en cuenta la nueva situación y para incluir los comportamientos necesarios que el estudiante considera que le llevarán a obtener la máxima puntuación.

La simulación también terminará si el agente jugador o el agente colaborador caen por un precipicio. El agente colaborador se reiniciará en una nueva casilla desconocida del mapa si choca con cualquier otro agente. Los lobos solo pueden atacar al agente jugador. Su ataque consiste en desplazarlo de su casilla cuando se encuentra en una posición adyacente a la suya. No siempre el "empujón" del lobo logra desplazar de la casilla al agente jugador, por eso, el agente jugador tras el ataque de un lobo debería verificar si se produjo o no el desplazamiento.

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 5 puntos en función de la puntuación obtenida en la consecución de misiones. De forma más detallada se describe como se asignarán estos valores en el documento de información adicional llamado " Ejemplos prácticas: Práctica E1" que se adjunta como material complementaria a esta práctica.

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.

5.3. Fechas Importantes

La fecha tope para la entrega será el **LUNES 14 DE JULIO DE 2024** antes de las **23:00 horas**.

5.4. 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 por tanto suspensa, reservándose los profesores elevar a instancias superiores si la falta se considerara grave. 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.