Inteligencia Artificial

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

Práctica 1

Agentes reactivos

Departamento de Ciencias de la Computación
e Inteligencia Artificial
Universidad de Granada

Curso 2023-2024

Práctica 1: agentes reactivos

Inteligencia Artificial

Contenido:

1. Introducción

La primera práctica de la asignatura *Inteligencia Artificial* consiste en el diseño e implementación de un agente reactivo. El agente reactivo jugará el papel de un personaje de un imaginario juego de plataformas y se tendrán que definir comportamientos para realizar tareas simples que impliquen manipulación de objetos y desplazamientos sobre un mapa o terreno.

1.1. Presentación

El objetivo de esta práctica es dar a conocer a los estudiantes de la asignatura modelos simples de IA como son los agentes reactivos. Este tipo de agentes son usados con cierta frecuencia como solución a problemas reales aunque también tienen una componente más lúdica que será la que explotaremos en esta práctica. Este tipo de agente se suele utilizar para dotar de comportamiento *inteligente* a un agente software dentro de un juego de ordenador. Siguiendo esta idea, en esta práctica se diseñará e implementará un agente reactivo tomando como base algunos de los ejemplos del libro de *Stuart Russell* y *Peter Norvig*, *Inteligencia Artificial: Un enfoque Moderno*. El simulador que utilizaremos fue inicialmente desarrollado por el profesor *Tsung-Che Chiang de la 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 con aspiradoras inteligentes. Las aspiradoras inteligentes son robots de uso doméstico, de un coste aproximado entre 100 y 300 euros, que disponen de sensores de suciedad, un aspirador y motores para moverse por el espacio (ver Figura 1). Cuando una aspiradora inteligente está en funcionamiento recorre toda la dependencia o habitación donde se encuentra, detectando y succionando suciedad hasta que, o bien termina su recorrido o bien aplica algún otro criterio de parada (batería baja, tiempo límite, etc.).

Figura 1: Aspiradora inteligente

Algunos enlaces que muestran el uso de este tipo de robots son los siguientes:

- **https://www.youtube.com/watch?v=ioOnZywItdQ&t=10s**
- **https://www.youtube.com/watch?v=3_pU7ayBRLI**

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, limpia, se mueve, detecta suciedad, 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. Utilizaremos las técnicas estudiadas en el tema 2 de la asignatura para el diseño de agentes reactivos.

1.2. Personajes virtuales en juegos de ordenador

La Inteligencia Artificial tiene un papel importante en el desarrollo de los videojuegos y cobra una relevancia muy especial al definir el comportamiento de aquellos personajes que intervienen en el juego. El comportamiento de estos personajes debe ser lo más parecido al rol que representaría ese personaje en la vida real. El nivel de realismo de un juego, entre otros muchos aspectos, está relacionado con la capacidad que tienen los personajes de actuar de forma inteligente.

En concreto, los agentes puramente reactivos se vienen usando como base para el desarrollo de personajes en juegos desde los inicios de los juegos de ordenador. Si mencionamos a *Clyde*, *Inky*, *Pinky* y *Blinky*, casi nadie sabría decir quiénes son, pero si os decimos que son los nombres de los fantasmas del juego clásico PAC-MAN (*Come Cocos* en español), supongo que tampoco muchos de vosotros lo sabréis. Es un juego antiguo, pero que tenéis la posibilidad de jugar *online*.

Si habéis jugado alguna vez a este mítico juego, seguro que no reparásteis en el mecanismo que permitía a los fantasmas perseguir o huir (en función de la situación del juego) del personaje principal. Bueno, pues el comportamiento de dichos fantasmas está regido por un agente básicamente reactivo. Como curiosidad, os diré que aunque aparentemente los 4 fantasmas presentan el mismo comportamiento, en realidad no es así, cada uno de ellos tiene su propia personalidad.

El fantasma amarillo se hace llamar **Clyde**, , aunque su verdadero nombre es **Pokey** (**Otoboke** en japonés). Este nombre significa lento, aunque su velocidad no parece ser distinta de la del resto de fantasmas. Es por ello que cuando fue denominado lento, no se refería a la velocidad con la que va por el laberinto, sino a lo lento y estúpido que podría ser tomando decisiones. Si nos fijamos en las direcciones que toma en cada cruce del laberinto, veremos que **Clyde** en realidad no persigue a **Pacman**. Quizás no sienta una especial aversión hacia él y no sea tan vengativo como los demás fantasmas, o, tal vez, sea tan estúpido que se pierde dando vueltas por los pasillos.

El fantasma azul se hace llamar **Inky**. Su verdadero nombre, **Bashful** (en inglés, tímido), hace ver que sus movimientos son, también, bastante irregulares. **Inky** tiene un claro problema de inseguridad. Generalmente, evita entrar en contacto con **Pacman** debido a sus miedos y se aleja. Sin embargo, si está cerca de sus hermanos **Blinky** y **Pinky**, se siente más fuerte, aumentando su seguridad y volviéndose mucho más agresivo.

Pinky es el fantasma rosa, también llamado **Speedy** (en inglés, rápido). Este fantasma es bastante metódico y, aunque tampoco es denominado rápido por su velocidad, es muy veloz tomando buenas decisiones. El fantasma rosa camina por el laberinto analizando y pensando bien todos sus movimientos, trazando en un mapa los caminos más cortos hasta **Pacman**. Se lleva muy bien con su hermano **Blinky**, con el que le encanta ponerse de acuerdo para realizar encerronas y trampas. Sin duda, es el más inteligente.

El fantasma rojo se llama **Blinky** y es conocido como **Shadow** (en inglés, sombra). Es el más agresivo de los cuatro fantasmas y su nombre viene dado porque casi siempre es la sombra de **Pacman**, persiguiéndolo incansablemente. Es posible que **Blinky** tenga alguna oculta razón por la que odia tanto a **Pacman**, ya que conforme pasa el tiempo, **Blinky** entra en un estado de furia insostenible en la que se vuelve muy agresivo y es conocido con el nombre de **Cruise Elroy** (el crucero **Elroy**).

La información anterior se ha obtenido de www.destructoid.com aunque podemos encontrar otras versiones que dan una explicación algo diferente de los comportamientos distintos de los fantasmas, como por ejemplo en jandresglezrt.wordpress.com. En cualquier caso, está claro es que tienen comportamientos diferentes y, por consiguiente, se definieron agentes reactivos específicos para cada uno de ellos.

2. Los extraños mundos de BelKan

Vamos a denominar *extraños mundos de Belkan* a un mundo virtual que se toma como punto de referencia para la construcción de un posible videojuego. Sobre él se intentan definir personajes virtuales que manifiesten comportamientos propios e inteligentes dentro del juego. Intentamos situarnos en una situación habitual en el desarrollo de juegos para ordenador y vamos a jugar a diseñar personajes que interactúen de forma autónoma usando agentes reactivos.

2.1. El escenario de juego

El juego se desarrolla sobre un tablero formado por celdas de igual tamaño. Desde un punto de vista lógico el mapa representa los accidentes geográficos de la superficie del terreno en que se desarrolla el juego. Esta superficie no cambia a lo largo del tiempo. De esta forma, la información sobre el terreno puede representarse mediante una matriz cuadrada con un número máximo de 100 filas y columnas. Cada celda de la matriz se identifica por el par de índices que indican su fila y columna respectivamente.

Consideremos un ejemplo de terreno de 100 filas y 100 columnas. Las características del terreno representado por cada celda se codifica mediante un carácter. Tendremos en cuenta las siguientes consideraciones:

- La casilla superior izquierda se identifica mediante los índices $(0, 0)$.
- La casilla inferior derecha tiene como índices $(99, 99)$.

- El agente se mueve hacia el norte si el índice de la fila de destino es menor que el de la fila actual. Es decir, si la posición actual es (i, j) la de destino será $(-1, j)$.
- El agente se mueve hacia el sur si el índice de la fila de destino es mayor que el de la fila de partida.
- En los movimientos al este y oeste lo que cambia es el índice de la columna, aumentando y disminuyendo respectivamente.

Las celdas representan diferentes tipos de terreno o contenido, usando el siguiente código definido mediante un carácter:

- **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. Si un agente se posiciona en una casilla de este tipo, sufrirá un reinicio, es decir, aparecerá siempre con orientación norte en una nueva casilla del mapa seleccionada al azar y perderá los objetos que pudiera haber conseguido.
- **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**, codificados con el carácter **M** y son rojo oscuro. El agente no puede avanzar por este tipo de casillas. Si lo intenta, se producirá un choque, es decir, se activará un sensor indicando que colisionó con algo y se quedará en la misma casilla desde la que intentó el movimiento. Los muros por tanto se consideran casillas intransitables.
- **Posicionamiento**, codificado con el carácter **G** y se muestra en celeste. Esta casilla activa los sensores de orientación y posición del agente, ofreciendo la posición y orientación exacta en ese momento de dicho agente.
- **Bikini**, codificado con el carácter **K** y se muestra en amarillo. Esta es una casilla especial en la que el jugador que pasa por ella adquiere el objeto bikini. Este objeto le permite reducir el consumo de batería en sus desplazamientos por el agua.
- **Zapatillas**, codificadas 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 (5000), 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 estos tipos de terreno y elementos se aprecian en la Fig. 2. Todos los mundos que usaremos en esta práctica son cerrados. **En todos se verificaque las tres últimas filas visibles al Norte son precipicios**, y lo 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:

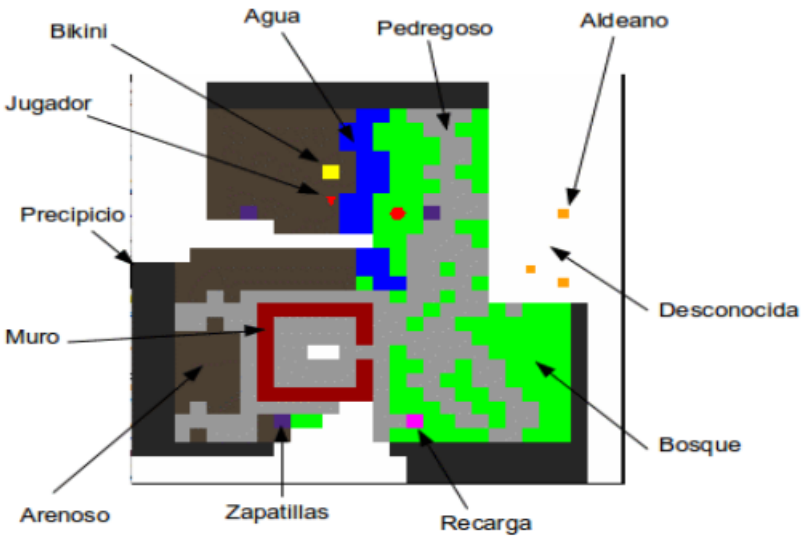

Figura 2: Representación gráfica del terreno

- **Jugador**, se codifica con el carácter **j** y se muestra como un triángulo rojo. Este es el agente al que debemos definir el comportamiento.
- **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. Los aldeanos pueden provocar choques con el jugador. Los choques suponen la activación del sensor de colisión y que el jugador no haya podido realizar el movimiento, quedándose en la misma casilla. Por consiguiente, una casilla ocupada por un aldeano es una casilla intransitable para el agente.
- **Lobos**: se codifican con el símbolo **l** (ele minúscula) y se muestran con hexágonos de color rosa. Son habitantes especiales de este mundo que, al igual que los aldeanos, deambulan por el mundo molestando. A diferencia de los anteriores, sí pueden resultar peligrosos. Chocar con ellos o que ellos te choquen supondrá ser reiniciado en el mapa, es decir, aparecer en otro punto desconocido del mapa con orientación norte y perdiendo los objetos que se hayan conseguido.

2.2. Características del agente

El protagonista de la historia y de esta práctica es el agente llamado jugador que debe llevar a cabo su objetivo, que consiste en descubrir la mayor parte posible del mapa en el que se encuentra ubicado.

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 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 se muestra en la Fig. 3.

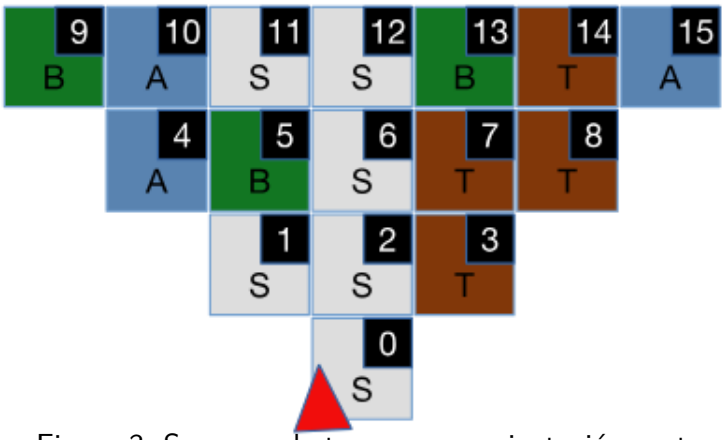

Figura 3: Sensores de terreno con orientación norte

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 qué 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 jugador, **l** para representar la posición de un lobo y _ si la casilla está desocupada.

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. Sin embargo, 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 mostrada en la Fig. 4.

Figura 4: Sensores de terreno con orientación sureste

El personaje también 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 una situación de reinicio.
- **Sensores de posición** (**posF**, **posC**): devuelve la posición de fila y columna que ocupa el jugador. Cuando este sensor no está funcionando bien devuelve los valores -1 en estos sensores.
- **Sensor de orientación** (**sentido**): devuelve la orientación del jugador. Este sensor toma 8 valores: norte, noreste, este, sureste, sur, suroeste, oeste y noroeste.
- **Sensor de batería** (**batería**): informa del nivel de carga de la batería. Inicialmente la batería tiene valor de 5000 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. La descripción detallada de los 4 niveles se puede consultar en la sección 3. Los valores posibles del sensor están entre 0 y 3 y tienen la siguiente interpretación

  0 : básico (todo es conocido).
  1 : sin sensor de posición ni de orientación.
  2 : con aldeanos y lobos.
  3 : sin sensor de posición y sensor de terreno defectuoso.

- **Sensor de tiempo consumido (tiempo):** este sensor informa del tiempo acumulado que lleva consumido el agente en la toma de decisiones desde el inicio de la simulación.

Durante el juego el jugador tiene la capacidad de coleccionar dos objetos (zapatillas o bikini) que le permitirán reducir el consumo de batería al transitar sobre algunos tipos de terreno. Más adelante se describe cómo afecta la posesión de estos objetos en el consumo.

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. Está matriz refleja el conocimiento que tiene el agente sobre el mapa donde se encuentra moviéndose. El objetivo será hacer que el contenido de este mapa coincida lo más posible con el mapa original del terreno.

2.2.3. Acciones que puede realizar el agente jugador

El agente puede realizar las siguientes acciones:

- **actWALK**: le permite avanzar a la siguiente casilla del mapa siguiendo su orientación actual (la que en un instante anterior se encontraba en la casilla 2 de su línea de visión). Para que la operación se finalice con éxito es necesario que la casilla de destino sea transitable para nuestro personaje.

- **actRUN**: le permite avanzar dos casillas del mapa siguiendo su orientación actual, es decir, en la que sería la casilla 6 en su línea de visión en el instante anterior. Para que la operación se finalice con éxito es necesario tanto que la casilla destino como la anterior al destino sean transitables, es decir, visto desde el instante anterior a esta acción, las casillas 2 y 6 de su línea de visión deben ser transitables. En el caso de muro o aldeano, el agente provoca una colisión y se queda en la casilla origen sin avanzar. En el caso de precipio o lobo, se produce un reinicio del agente jugador en otro punto del mapa. En el caso de que esta acción se realice con éxito, sólo se consigue el posicionamiento, el bikini, las zapatillas o la recarga si estas coinciden con la casilla final en la que termina el agente al concluir la acción. Si entre la casilla de inicio de esta acción y la de final había alguna de esas casillas especiales, no se considerará que se estuvo en esa casilla.
- **actTURN_SR**: le permite mantenerse en la misma casilla y girar a la derecha $45°$ teniendo en cuenta su orientación.
- **actTURN_L**: le permite mantenerse en la misma casilla y girar a la izquierda $90°$ 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_SR**, **actTURN_L** y **actWALK** 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 la acción a realizar, la casilla de inicio de la acción y dependiendo del objeto que se tenga en posesión en ese momento.

actWALK

tipo de casilla	*gasto normal batería*	*gasto reducido batería*
A	50	5 (con bikini)
B	25	5 (con zapatillas)
T	2	2
resto de casillas	1	1

actRUN

tipo de casilla	*gasto normal batería*	*gasto reducido batería*
A	500	10 (con bikini)
B	250	15 (con zapatillas)
T	3	3
resto de casillas	1	1

actTURN_SR

tipo de casilla	*gasto normal batería*	*gasto reducido batería*
A	50	5 (con bikini)
B	10	1 (con zapatillas)
T	2	2
resto de casillas	1	1

actTURN_L

tipo de casilla	*gasto normal batería*	*gasto reducido batería*
A	50	5 (con bikini)
B	20	1 (con zapatillas)
T	2	2
resto de casillas	1	1

3. Objetivo de la práctica

El agente aparecerá de forma aleatoria sobre un mundo concreto, cuyo terreno no cambiará durante el juego. El personaje, usando un comportamiento puramente reactivo, deberá ir descubriendo cada una de las casillas que componen el terreno del mapa, es decir, qué casilla tiene agua, qué casilla es un muro... y, además, deberá determinar su posición y orientación exacta sobre el mapa original.

Se van a considerar los siguientes 4 niveles de funcionamiento:

- **Nivel 0**: el sistema sensorial funciona correctamente y en todo momento el agente conoce su posición y su orientación a través de los sensores correspondientes.
- **Nivel 1**: a diferencia del nivel anterior, los sensores de posición y el de orientación del agente no funcionan, así que los sensores **posF** y **posC** devuelven los valores -1 y el sensor de orientación devuelve siempre el valor **norte**. Sí se conoce que en el momento en el que empieza la simulación (instante 0) el agente tiene siempre orientación **norte**.
- **Nivel 2**: sensorialmente igual que el nivel 1 y también conociéndose en el instante 0 que la orientación es norte, pero en este nivel aparecen los aldeanos y los lobos. Los lobos son agentes que entorpecen los movimientos y a diferencia de los aldeanos, una colisión con ellos provoca que el agente sea reiniciado en el mapa. Ser reiniciado implica aparecer en una nueva casilla del mapa desconocida, aunque con orientación norte y perdiendo las zapatillas y el bikini si los llevaba.
- **Nivel 3**: al igual que en el nivel anterior, los sensores de posición y de orientación no funcionan. A diferencia del nivel anterior, **no se conoce** la orientación del agente en en inicio de la simulación (instante 0) y además en el sensor de terreno no aparece la información de lo que se en las casillas 6, 11, 12 y 13.

¿Cómo puedo descubrir mi posición en el mapa original en los niveles donde no funciona correctamente el sensor de orientación y los de posición? Pues la respuesta a esta pregunta son los puntos de posicionamiento (las casillas celestes del mapa codificadas con el símbolo **G**). Cada vez que nos situamos encima de una casilla de posicionamiento, en el sensor **posF** aparecerá la fila exacta en la que me encuentro y de igual manera en el sensor **posC** la columna exacta y en el sensor sentido la orientación actual. Usando esta información podemos situar con completa exactitud nuestra posición en el tablero original.

Asociado a los datos que maneja nuestro personaje, hay definida una matriz del mismo tamaño que el mapa original, destinada a ir guardando lo que sabemos seguro que hay en cada casilla. Esta matriz (**mapaResultado**) tendrá que ir siendo actualizada durante el desarrollo del juego. Cuando el juego termina, automáticamente se envía el contenido de dicha matriz para su evaluación, devolviendo el tanto por ciento de semejanza con respecto al mapa original.

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.

4.1. Instalación

Se proporciona sólo versión para el sistema operativo **Linux** y se puede encontrar en https://github.com/ugr-ccia-IA/practica1. 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 es:

- 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 y consiste en seguir las instrucciones que se proporcionan en el repositorio de **GitHub** (`https://github.com/ugr-ccia-IA/practica1`) para acceder y trabajar con el software. Por favor, leed 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: **practica1** y **practica1SG**. 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 una ejecución más rápida de la simulación. Además, con esta segunda opción, se pueden realizar tareas de *debugger* o de depuración de errores de forma más fácil. Empezamos describiendo la versión con entorno gráfico.

4.2.1. Interfaz gráfica

Para ejecutar el simulador con interfaz hay que ejecutar el comando:

```
> ./practica1
```

Aparecerá la ventana principal de la aplicación, ver Fig. 5.

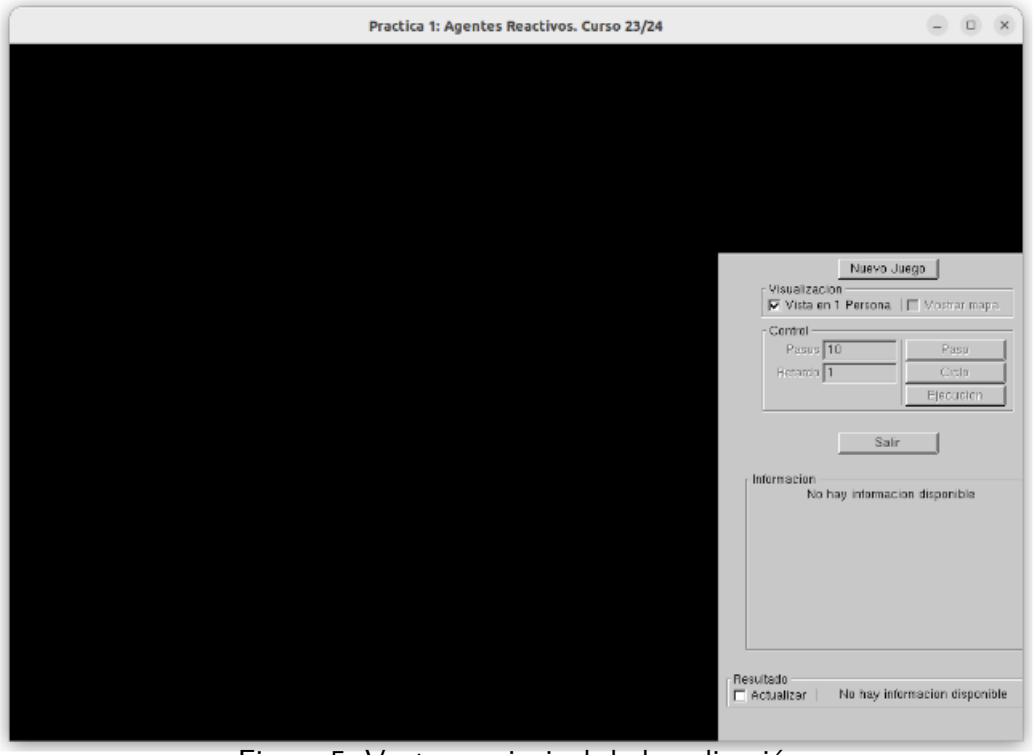

Figura 5: Ventana principal de la aplicación

Para iniciar la simulación se debe elegir el botón **Nuevo Juego** que abriría la siguiente ventana que se muestra en la Fig. 6.

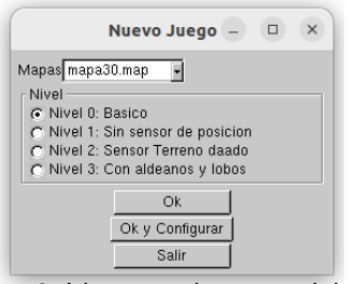

Figura 6: Ventana de inicio del juego

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 0 al 3 están sin implementar. Obviamente, el objetivo es ir poco a poco ofreciendo un comportamiento reactivo que permita obtener un buen funcionamiento en los distintos niveles de dificultad que se presentan.

Seleccionaremos el **nivel 0: Basico** 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 y en el que inicialmente aparecerá vacía, ya que no conocemos aún nada de él, 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 (ver Fig. 7).

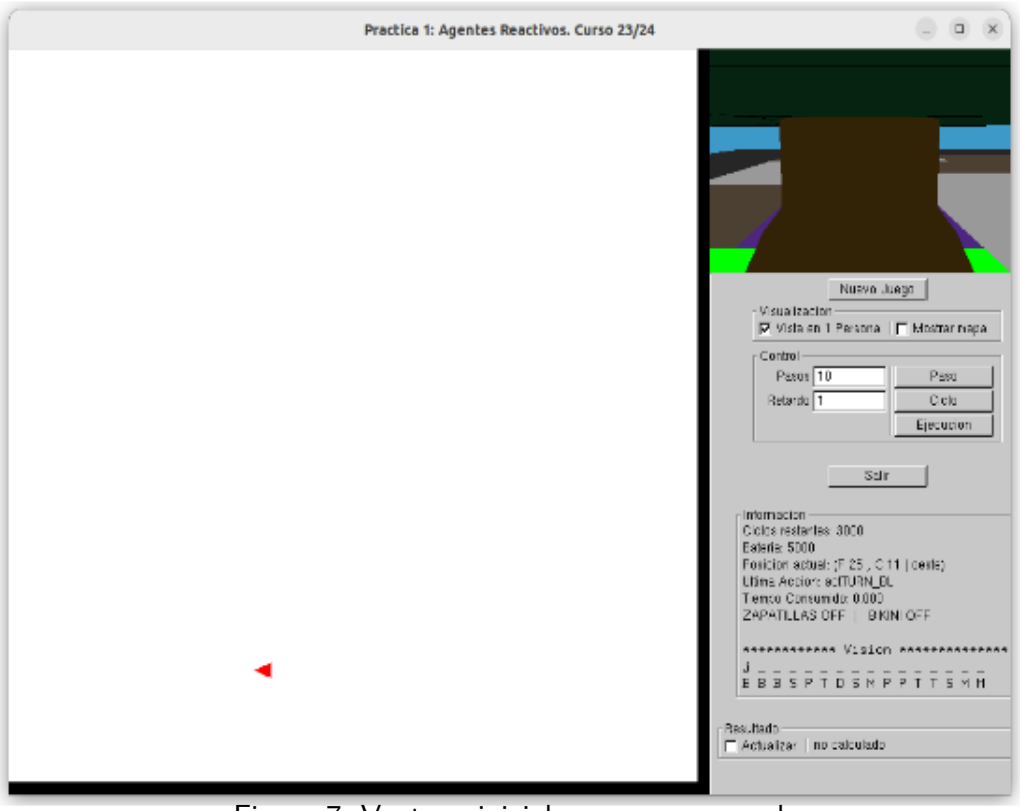

Figura 7: Ventana inicial con mapa cargado

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 de acciones 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**. La opción **Mostrar mapa** permite ver el mapa que lleva reconocido el agente frente a la visión completa del mapa. Dentro del grupo de actuadores denominados **Resultado**, podemos decidir si queremos ver o no la evolución en el porcentaje de semejanza entre el mapa que vamos descubriendo y el original. Se da la opción de no actualizar, ya que consume tiempo ese cálculo y ralentiza la simulación.

Podemos cambiar las condiciones iniciales del juego si después de dar al botón de **Nuevo Juego** y seleccionar un mapa, pulsamos el botón **Ok** y **Configurar**. En este caso, nos aparecerá la siguiente ventana (Fig. 8) que nos permite cambiar los parámetros de la simulación.

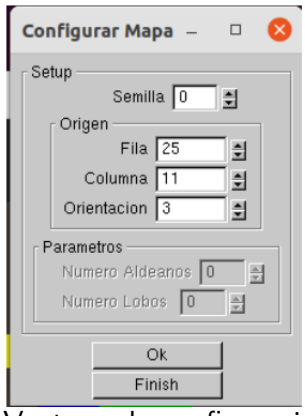

Figura 8: Ventana de configuración de mapa

Los parámetros modificables son la semilla del generador de números aleatorios, la posición y orientación inicial del agente y el número de aldeanos (sólo tiene sentido a partir del nivel 2) y lobos (sólo tiene sentido en el nivel 3) a considerar en el juego. Para fijar el cambio, es necesario pulsar el botón **Ok**. A la vez que sale el menú anterior, tenemos la posibilidad de ver sobre la ventana original el mapa que se debe descubrir, de manera que se pueda situar al jugador conociendo su situación exacta en el mapa. Una vez que esté con la configuración correcta y se haya pulsado el botón **Ok** para fijar esa configuración inicial, se debe pulsar **Finish** para ir a empezar la simulación.

Para finalizar con la descripción del entorno gráfico, bajo el título de **Información**, 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:

- ciclos restantes: 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: expresado 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.

Bajo el texto ********** **Vision** ********** se indican los valores de los que informan los sensores de terreno y agentes 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, Al hacer **make** se generan automáticamente los dos ejecutables: **practica1** y **practica1SG**. 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 **practica1SG** 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:

```
./practica1SG <mapa> <semilla> <nivel> <fila> <col> <ori>
```

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.
- $< fila >$: 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 7 indicando la orientación con la que empezará el agente, siendo 0=norte, 1=noreste, 2=este, 3=sureste, 4=sur, 5=suroeste, 6=oeste, 7=noroeste.

Por ejemplo podemos ejecutar

```
./practica1SG mapas/mapa30.map 1 0 4 5 1
```

lo que producirá que se use el mapa llamado **mapa30.map**, la semilla 1 en el nivel 0 y situando al agente en la posición de fila 4 y columna 5, con orientación noreste (1).

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

- instantes de simulación consumidos
- tiempo consumido acumulado requerido por el agente
- nivel final de la batería
- número de colisiones que hemos sufrido
- si la simulación terminó porque murió el agente
- porcentaje de mapa descubierto

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 iniciando 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 **practica1**, es decir:

```
./practica1 <mapa> <semilla> <nivel> <fila> <col> <ori>
```

Algo a destacar cuando se toma esta opción para ejecutar el software de esta forma

```
./practica1 mapas/mapa30.map 1 0 4 5 1
```

es que la simulación queda detenida tras la ejecución de la primera acción del agente.

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 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. Empezaremos considerando el archivo **jugador.hpp**, donde se declara la clase **ComportamientoJugador**.

```
#ifndef COMPORTAMIENTOJUGADOR_H
#define COMPORTAMIENTOJUGADOR_H

#include "comportamientos/comportamiento.hpp"
using namespace std;

class ComportamientoJugador : public Comportamiento{

  public:
    ComportamientoJugador(unsigned int size) : Comportamiento(size){
      // Constructor de la clase
      // Dar el valor inicial a las variables de estado
    }

    ComportamientoJugador(const ComportamientoJugador & comport) :
    ↪  Comportamiento(comport){}
    ~ComportamientoJugador(){}

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

  private:

  // Declarar aquí las variables de estado

};

#endif
```

Los métodos implementados en la parte pública son :

- constructor de la clase. Aquí se tendrán que inicializar las variables de estado que se consideren necesarias para resolver la práctica.

```
ComportamientoJugador(unsigned int size) :
↪  Comportamiento(size)
```

- constructor de copia:

```
ComportamientoJugador(const ComportamientoJugador &compor) :
↪  ComportamientoJugador(compor){}
```

- destructor de la clase:

```
~ComportamientoJugador(){}
```

- el método que describe el comportamiento del agente y que debe ser modificado para ir incorporándole la resolución de los distintos niveles de la práctica y cuya descripción se encuentra en el fichero **jugador.cpp**.

```
Action think(Sensores sensores);
```

- un método que establece como interacciona este agente con otros agentes en el mundo. Este método es irrelevante para esta práctica.

```
int interact (Action accion, int valor);
```

En la parte inferior del código vamos a declarar los datos privados de la clase. Para esta práctica, es la zona donde se declararán las variables de estado para resolver el problema. Recordar que las variables de estado para un agente reactivo almacenan su memoria sobre lo que lleva ya conocido sobre el mundo. En función de las cosas que se recuerden se podrán definir nuevos comportamientos del agente y por consiguiente, tener un comportamiento más inteligente o más apropiado para la resolución del problema que se propone. El estudiante puede agregar tantas variables de estado como crea conveniente.

En el archivo **jugador.cpp** se describe el comportamiento del agente.

```
#include <iostream>

using namespace std;

Action ComportamientoJugador::think(Sensores sensores) {

   Action accion = actIDLE;
   cout << "Posicion: fila " << sensores.posF << " columna " <<
   ↪  sensores.posC << " ";
   switch(sensores.sentido){
      case norte: cout << "Norte" << endl; break;
      case noreste: cout << "Noreste" << endl; break;
      case este: cout << "Este" << endl; break;
      case sureste: cout << "Sureste" << endl; break;
      case sur: cout << "Sur" << endl; break;
      case suroeste: cout << "Suroeste" << endl; break;
      case oeste: cout << "Oeste" << endl; break;
      case noroeste: cout << "Noroeste" << endl; break;
   }

   cout << "Terreno: ";
   for (int i=0; i<sensores.terreno.size(); i++)
        cout << sensores.terreno[i];
   cout << endl;

```

```
25      cout << "Superficie: ";
26      for (int i=0; i<sensores.agentes.size(); i++)
27           cout << sensores.agentes[i];
28      cout << endl;
29
30      cout << "Colisión: " << sensores.colision << endl;
31      cout << "Reset: " << sensores.reset << endl;
32      cout << "Vida: " << sensores.vida << endl;
33      cout << endl;
34
35      // Determinar el efecto de la ultima accion enviada
36      return accion;
37  }
38
39  int ComportamientoJugador::interact(Action accion, int valor) {
40      return false;
41  }
```

En concreto, el único método que hay que completar es el método **think**. En su versión inicial, **think** contiene la forma en la que se accede al valor del sistema sensorial que proporciona el agente. La información sobre los tipos de datos asociados a cada uno de los valores que proporciona el sistema sensorial se puede encontrar en el archivo **comportamiento.hpp**.

Con toda seguridad, durante la evolución de la práctica, los estudiantes tendrán que definir procedimientos o funciones adicionales para definir mejor el comportamiento del agente. También, deberán eliminar el contenido actual que tiene el método **think** y sustituirlo por el comportamiento que desean dar al agente para resolver el problema mediante una secuencia de reglas de producción. Para que el estudiante se vaya aproximando a la forma en la que debe evolucionar su práctica, le aconsejamos que revise el tutorial que también se proporciona como material.

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 **practica1.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 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

El método de evaluación consiste en probar el funcionamiento del comportamiento basado en agentes reactivos propuesto por el estudiante sobre 3 mapas (semejantes a los proporcionados a los estudiantes). Para ello, se asocia una puntuación de hasta 10 puntos para cada uno de los 3 mapas, siendo la distribución de puntos la que se describen en la Tabla 1.

nivel	puntuación
0	2
1	3
2	2
3	3

Tabla 1: Puntuación según niveles

La fórmula que se utiliza para obtener la valoración obtenida en cada mapa será:

$$nota_1 = s_0 p_0 + s_1 p_1 + s_2 p_2 + s_3 p_3$$

donde s_i representa el grado en el que se ha descubierto el mapa en el nivel i y p_i la puntuación asociada al nivel i, que es la que se muestra en la tabla anterior. El cálculo del grado de descubrimiento del mapa se calculará usando la siguiente fórmula:

$$s_i = min\left[1.0, \frac{aciertosNetos}{totalDeCasillas \times 0.85}\right]$$

siendo

$$aciertosNetos = max[0, (aciertos - errores)]$$

donde **aciertos** es el número de casillas que se han clasificado de forma correcta, **errores** el número de casillas que no se han clasificado de forma correcta y **totalDeCasillas** el número de casillas que contiene el mapa. Las casillas a las que no se les haya asignado un tipo de terreno, no contabilizan ni en aciertos ni en errores.

Se puede observar que si $aciertosNetos \geq 0.85 \times totalDeCasillas$, se cumpole que $s_i = 1.0$; por tanto si el valor de $aciertosNetos$ es mayor o igual al $85\,\%$ del mapa entonces se conseguirían todos los puntos asociados a ese nivel.

En aquellas situaciones donde la simulación no termine de forma normal (se produzca una violación de segmento o "core"), se asignará el valor $s_i = -0.1$. La **nota final** de la práctica es la **media aritmética** de las obtenidas en los 3 mapas. En todos los niveles de dificultad que se proponen el objetivo es el mismo: intentar maximizar con el comportamiento reactivo que se defina en base a los sensores y a las variables de estado que defina el estudiante, el porcentaje de mapa descubierto por el agente. Es muy importante que dicha información de mapa descubierto se almacene de forma correcta en la matriz **mapaResultado**, ya que será está variable la que se usará para comparar la semejanza entre el mapa real y lo descubierto.

Además, las condiciones en las que se desarrollará cada ejecución o simulación son la siguiente: 3000 instantes de simulación, 5000 puntos de batería y un máximo de 300 segundos para decidir las siguiente acción. Cuando se agote alguno de ellos, la ejecución/simulación terminará.

5.4. Fechas importantes

La fecha tope para la entrega será el **DOMINGO 7 DE ABRIL DE 2024** antes de las **23:00 horas** y desde el **LUNES 8 DE ABRIL** estará disponible la entrega para el cuestionario de autoevaluación hasta el **MIÉRCOLES 10 DE ABRIL** 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 l a 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**.

Por consiguiente, se asume que todo el código nuevo que aparece en su práctica ha sido introducido por él por alguna razón y que dicho estudiante domina perfectamente el código que entrega. Durante cualquier momento del proceso de evaluación, el alumno puede ser requerido para que justifique adecuadamente algo de lo que aparece en su código. Si el alumno no responde a ese requerimiento o responde al requerimiento pero no ofrece una respuesta convincente que lo justifique, la práctica se considerará copiada y tendrá suspensa la práctica, y el profesorado se reserva la posibilidad de tomar acciones adicionales que impliquen consecuencias más graves para el estudiante. 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 que no presenten el documento de autoevaluación , 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, estarán sujetos posibles acciones adicionales con repercusiones más graves para los estudiantes.