

# 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 2021-2022



# 1. Introducción

La primera práctica de la asignatura **Inteligencia Artificial** consiste en el diseño e implementación de un agente reactivo capaz de percibir el ambiente y actuar de acuerdo a reglas simples predefinidas.

## 1.1. Presentación

El objetivo de esta práctica es dar a conocer a los estudiantes de la asignatura modelos de IA simples como son los agentes reactivos. Este tipo de agentes son usados con cierta frecuencia como solución a problemas reales y también tienen una componente más lúdica que será la que se explotará en esta práctica, como es el dotar de un comportamiento “inteligente” a un agente software dentro de un juego de ordenador. Así, 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”, Prentice Hall, Segunda Edición, 2004 sobre un simulador. 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 en 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 se encuentra en funcionamiento, esta recorre toda la dependencia o habitación donde se encuentra, detectando y succionando suciedad hasta que, o bien termina de recorrer la dependencia, o bien aplica algún otro criterio de parada (batería baja, tiempo límite, etc.). Algunos enlaces que muestran el uso de este tipo de robots son los siguientes:

1. <https://www.youtube.com/watch?v=ioOnZywItdQ&t=10s>
2. [https://www.youtube.com/watch?v=3\\_pU7ayBRLI](https://www.youtube.com/watch?v=3_pU7ayBRLI)



Figura 1: Aspiradora inteligente

Este tipo de robots es un ejemplo comercial más de máquinas que implementan técnicas de Inteligencia Artificial y, más concretamente, de Teoría de Agentes. En su versión más simple (y también más barata), una aspiradora inteligente presenta un comportamiento *reactivo* puro: busca suciedad, 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.

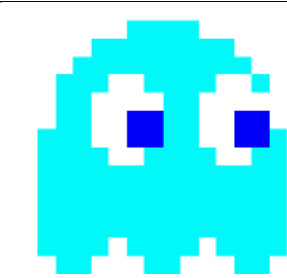
## 1.2. Personajes virtuales en juegos de ordenador

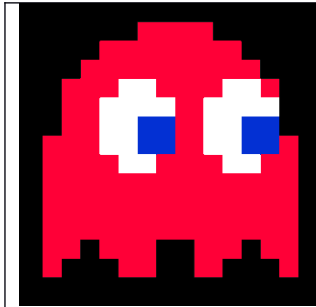
La Inteligencia Artificial tiene un papel importante en el desarrollo de los videojuegos y cobra una relevancia muy especial al definir el comportamiento de aquellos personajes que intervienen en el juego. El comportamiento de estos personajes debe ser lo más parecido al rol que representaría ese personaje 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, jajajaja. Es un juego antiguo, pero que tenéis la posibilidad de jugar “online” en <https://www.google.com/logos/2010/pacman10-i.html>.

## Departamento de Ciencias de la Computación e Inteligencia Artificial

Si habéis jugado alguna vez a este mítico juego, seguro que no reparasteis 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.

	<p>El fantasma amarillo se hace llamar <b>Clyde</b>, 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.</p> <p>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.</p> <p>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.</p>
	<p>El fantasma azul se hace llamar <b>Inky</b>. Su verdadero nombre, Bashful (en inglés, tímido), hace ver que sus movimientos son, también, bastante irregulares.</p> <p>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.</p>
	<p><b>Pinky</b> 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.</p> <p>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.</p>



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 <http://www.destructoid.com/blinky-inky-pinky-and-clyde-a-small-onomastic-study-108669.phtml>, aunque podemos encontrar otras versiones que dan una explicación algo diferente de los comportamientos distintos de los fantasmas, como por ejemplo en <https://jandresglezrt.wordpress.com/2015/04/09/los-fantasmas-enemigos-de-pac-man/>. En cualquier caso, lo que 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

En esta práctica tomamos como punto de partida el mundo de una supuesta aventura gráfica 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.

### 2.1. El escenario de juego

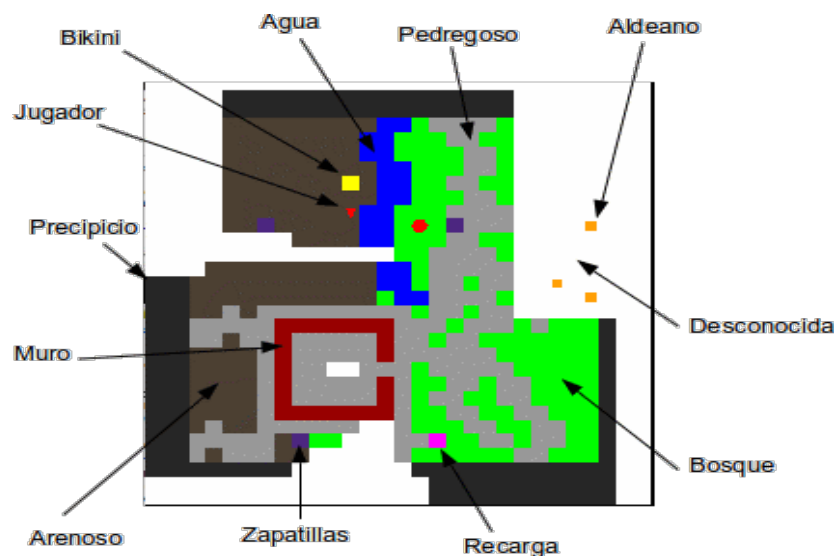
Este juego se desarrolla sobre un mapa cuadrado bidimensional discreto que contiene como máximo 100 filas y 100 columnas. El mapa representa los accidentes geográficos de la superficie de un terreno que son considerados como inmutables, es decir, los elementos dentro del mapa que no cambian durante el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

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

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

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

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



Los elementos permanentes en el terreno son los siguientes:

- **Árboles o Bosque**, codificado con el carácter '**B**' y se representan en el mapa como casillas de color verde.
- **Agua**, codificado con el carácter '**A**' y tiene asociado el color azul.
- **Precipicios**, codificado con el carácter '**P**' y tiene asociado el color negro. Si un agente se posiciona en una casilla de este tipo, sufrirá un reinicio, es decir, aparecerá 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**, codificado 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.



- **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 que cuando el jugador pasa por ella adquiere el objeto **bikini**. Este objeto le permite reducir el consumo de batería en sus desplazamientos por el agua.
- **Zapatillas**, codificado con el carácter 'D' y son moradas. Esta es una casilla especial y al igual que la anterior, el jugador adquiere, en este caso el objeto **zapatillas** simplemente al posicionarse sobre ella. Las **zapatillas** le permiten al jugador reducir el consumo de batería en los bosques.
- **Recarga**, codificado con el carácter 'X' y de color rosa. Esta casilla especial permite al jugador cargar su batería. Por cada instante de simulación aumenta en 10 el nivel de su batería. Cuando la batería alcanza su nivel máximo de carga (3000), estar en esta casilla no incrementa la carga.
- **Casilla aún desconocida** se codifica con el carácter '?' y se muestra en blanco (representa la parte del mundo que aún no has explorado).

Todos los mundos que usaremos en esta práctica son cerrados. En todos se verifica que 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:

- **Jugador**, se codifica con el carácter 'j' y se muestra como un triángulo rojo. Éste es nuestro personaje, sólo habrá un jugador a la vez.
- **Aldeano**, se codifica con el carácter 'a' y se muestra como un cuadrado naranja. Son habitantes anónimos del mundo que se desplazan a través del mapa sin un cometido específico, simplemente intentan molestarnos en nuestros movimientos. Los aldeanos pueden provocar choques o colisiones 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.
- **Lobos**: se codifica en el símbolo 'l' (le minúscula) y se muestra como un hexágono rosa. Son habitantes especiales de este mundo que, al igual que los aldeanos, deambulan por el mundo molestando. A diferencia de los anteriores, sí

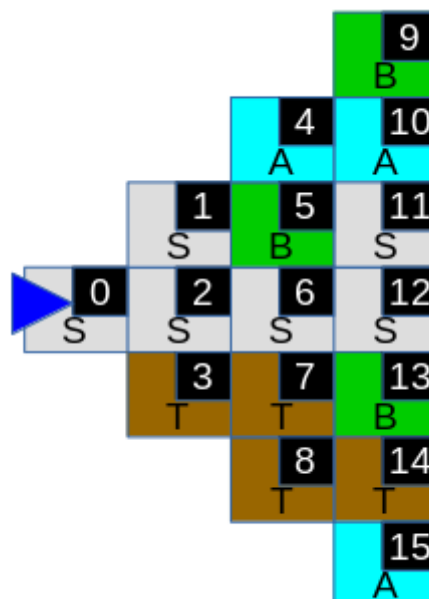
pueden resultar peligrosos. Chocar con ellos o que ellos te choquen supondrá ser reiniciado en el mapa, es decir, aparecer en otro punto desconocido del mapa perdiendo los objetos que se hayan conseguido.

## 2.2. El protagonista

Obviamente el protagonista de la historia es el ente llamado jugador y 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 **superficie**, es capaz de mostrarnos qué objetos móviles se encuentran en nuestra visión (es decir, aldeanos). Para entender cómo funciona este sistema pondremos el siguiente ejemplo: Suponed que el vector terreno contiene **SSSTABSTTBASSBTA**, su representación real sobre un plano será la siguiente:







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

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

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

- **Sensor de choque (colision):** Es una variable booleana que tomará el valor verdadero en caso de que la última acción del jugador haya ocasionado un choque.
- **Sensor de vida (reset):** Es una variable booleana que toma el valor de verdad en caso de que la última acción del jugador le haya llevado a morir.
- **Sensores de posición (posF, posC):** Devuelve la posición de fila y columna que ocupa el jugador. 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 4 valores, del 0 al 3, con la siguiente interpretación, 0 indica orientación norte, 1 orientación este, 2 orientación sur y 3 orientación oeste.
- **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 : Básico (Todo es conocido).
  - 1 : Sin sensor de posición.
  - 2 : Sin sensor de orientación.



3 : Con aldeanos.

4 : Con lobos.

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

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 personaje compartidos con el entorno

Dentro de la definición del agente hay una matriz llamada **mapaResultado** en donde se puede interactuar con el mapa. Todo cambio en esta matriz se verá reflejado automáticamente en la interfaz gráfica. Esta matriz refleja el conocimiento que tiene el agente sobre el mapa donde se encuentra moviéndose y por consiguiente, 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 personaje

El agente puede realizar las siguientes acciones:

**actFORWARD:** le permite avanzar a la siguiente casilla del mapa siguiendo su orientación actual. Para que la operación se finalice con éxito es necesario que la casilla de destino sea transitable para nuestro personaje.

**actTURN\_L:** le permite mantenerse en la misma casilla y girar a la izquierda 90° teniendo en cuenta su orientación.

**actTURN\_R:** le permite mantenerse en la misma casilla y girar a la derecha 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\_L**, **actTURN\_R** y **actFORWARD** *depende del tipo de terreno asociado a la casilla donde se inició dicha acción*. Esto es, si el agente está en una casilla de agua (etiquetada con una 'A') y avanza a una casilla de terreno arenoso (etiquetada con una 'T'), el coste en batería es el asociado a la casilla de agua, es decir, a la casilla inicial donde se produce la acción de avanzar. En las siguientes tablas se muestran los valores de consumo de energía en función de la acción a realizar, la casilla de inicio de la acción y dependiendo del objeto que se tenga en posesión en ese momento.

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

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



### 3. Objetivo de la práctica

Nuestro personaje aparecerá de forma aleatoria sobre un mundo de BelKan concreto, que no cambiará durante el juego, sin conocer su posición, aunque sabiendo que **su orientación inicial siempre coincide con el norte** (para niveles de juego superiores o iguales al 2) del mapa original. El personaje se las deberá ingeniar, usando un comportamiento puramente reactivo, para ir descubriendo todos los elementos inmóviles 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.

¿Cómo puedo descubrir mi posición en el mapa original? 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, devolviéndote 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 son: ***freeglut3 freeglut3-dev libjpeg-dev libopenmi-dev openmpi-bin openmpi-doc libxmu-dev libxi-dev cmake libboost-all-dev***.

El proceso de instalación es muy simple 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. Leer detenidamente las instrucciones que se proporcionan en ese repositorio para conseguir una correcta configuración del software.

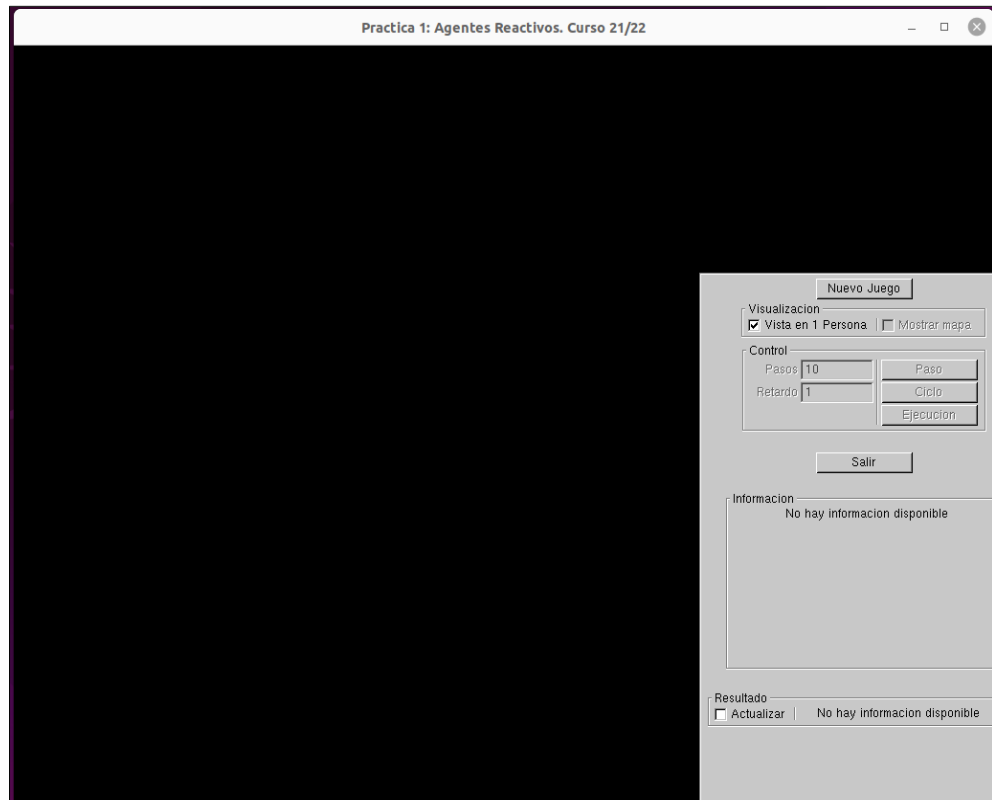
## 4.2. Funcionamiento del Programa

Existen dos ficheros ejecutables: ***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 para poder hacer tareas de “debugger” o de depuración de errores.

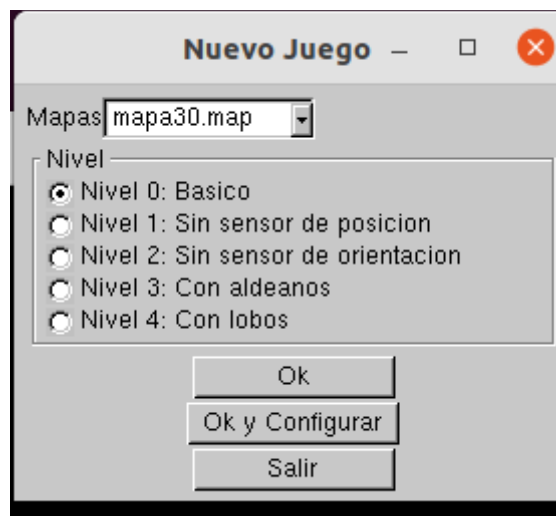
Empezamos describiendo la versión con entorno gráfico.

### 4.2.1. Interfaz gráfica

Para ejecutar el simulador con interfaz hay que escribir “***./practica1***”.



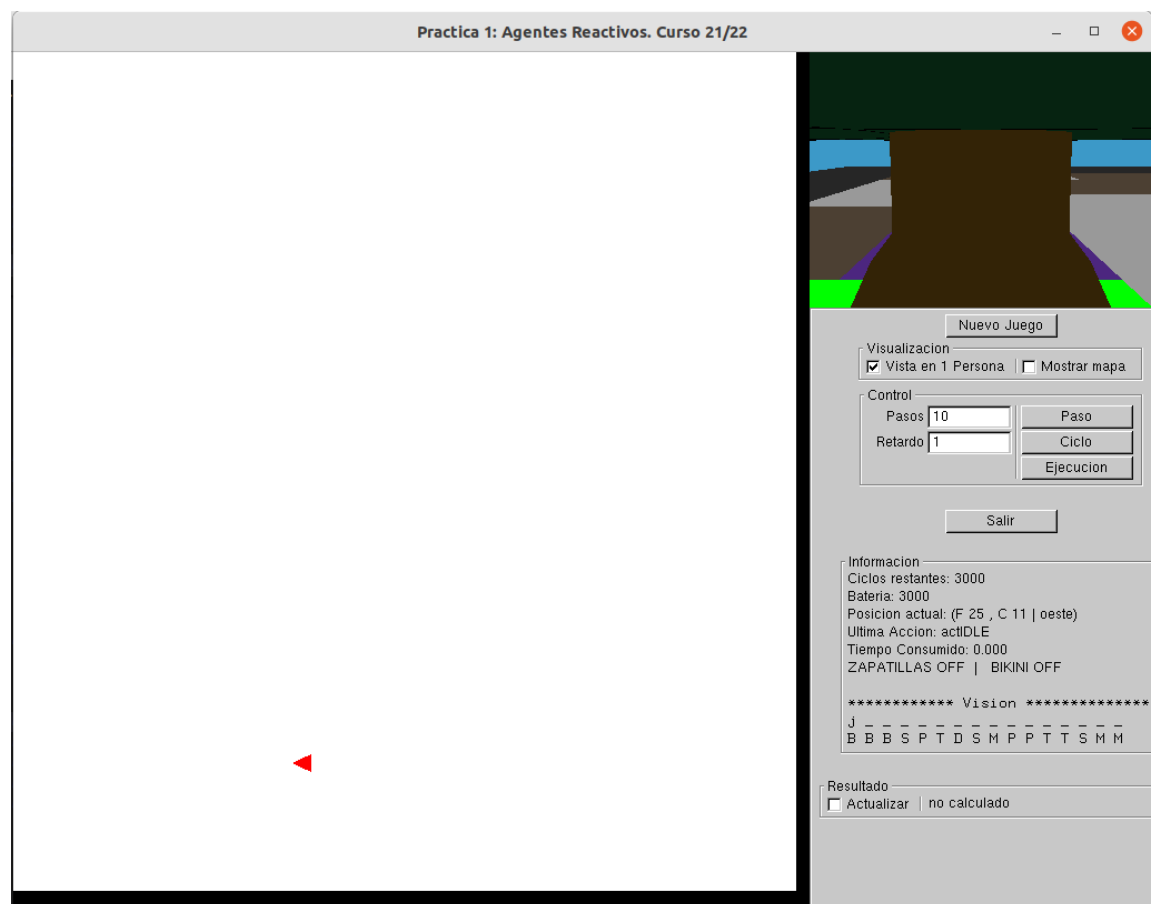
Al arrancar el programa nos mostrará la ventana principal. Para iniciar el programa se debe elegir el botón **Nuevo Juego** que abriría la siguiente ventana:



## Departamento de Ciencias de la Computación e Inteligencia Artificial

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 4 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.



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**. La opción **Mostrar mapa** permite ver el mapa que lleva reconocido el agente frente a la visión completa del mapa.

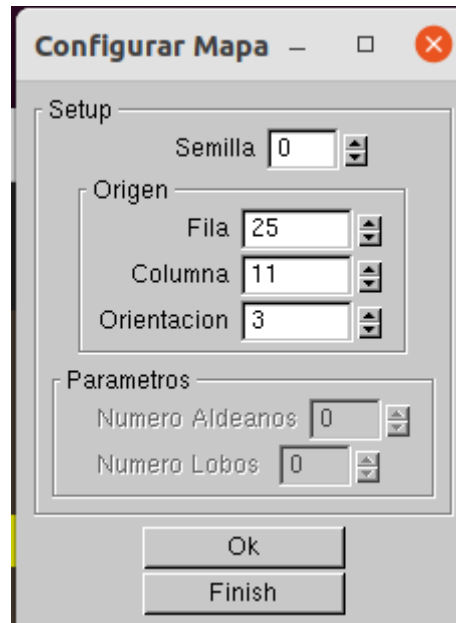
Dentro del grupo de actuadores denominados “Resultado”, podemos decidir si queremos ver o no la evolución en el porcentaje de semejanza entre el mapa que vamos descubriendo y el original. Se da la opción de no actualizar, ya que consume tiempo ese cálculo y ralentiza la simulación.

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 que nos permite cambiar los parámetros de la simulación.

Los parámetros modificables son la semilla del generador de números aleatorios, la posición y orientación inicial del agente y el número de aldeanos (sólo tiene sentido a partir del nivel 3) y lobos (sólo tiene sentido en el nivel 4) 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 **superficie** en este instante con la interpretación que ya se indicó anteriormente en este documento.

#### 4.2.2. Sistema *batch*

Se incluye con el software la posibilidad de crear un ejecutable sin interfaz gráfica para dar la posibilidad de realizar las operaciones de depuración de errores con mayor facilidad, ya que las librerías gráficas incluyen programación basada en eventos que alteran el normal funcionamiento de las herramientas como el conocido debugger **ddd**. 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:

<b>./practica1SG &lt;mapa&gt; &lt;semilla&gt; &lt;nivel&gt; &lt;fila&gt; &lt;col&gt; &lt;ori&gt;</b>
--

donde

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

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

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

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

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

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



Por ejemplo podemos ejecutar `./practica1SG mapas/mapa30.map 1 0 4 5 1` lo cual utilizará el mapa llamado **mapa30.map** indicado con una semilla **1** en el nivel **0** situando al agente en la posición de fila **4** y columna **5** con orientación **este** (**1**).

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

- los instantes de simulación consumidos,
- el tiempo consumido acumulado requerido por el agente,
- el nivel final de la batería,
- el número de colisiones que hemos sufrido,
- si la simulación terminó porque murió el agente,
- y el 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 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 *practica1*, es decir,

<code>./practica1 &lt;mapa&gt; &lt;semilla&gt; &lt;nivel&gt; &lt;fila&gt; &lt;col&gt; &lt;ori&gt;</code>
--

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.

```
1  #ifndef COMPORTAMIENTOJUGADOR_H
2  #define COMPORTAMIENTOJUGADOR_H
3
4  #include "comportamientos/comportamiento.hpp"
5  using namespace std;
6
7  class ComportamientoJugador : public Comportamiento{
8
9  public:
10     ComportamientoJugador(unsigned int size) : Comportamiento(size){
11         // Constructor de la clase
12         // Dar el valor inicial a las variables de estado
13     }
14
15     ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
16     ~ComportamientoJugador(){}
17
18     Action think(Sensores sensores);
19     int interact(Action accion, int valor);
20
21 private:
22
23     // Declarar aquí las variables de estado
24
25 };
26 #endif
```

Empecemos con el archivo **'jugador.hpp'**. Vemos que aquí se declara la clase *ComportamientoJugador*. Los métodos implementados en la parte pública son :

- El 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)**



- El constructor de copia.

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

- El 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 es 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.

```
1  #include "../Comportamientos_Jugador/jugador.hpp"
2  #include <iostream>
3  using namespace std;
4
5
6
7  Action ComportamientoJugador::think(Sensores sensores){
8
9      Action accion = actIDLE;
10
11     cout << "Posicion: fila " << sensores.posF << " columna " << sensores.posC << " ";
12     switch(sensores.sentido){
13         case 0: cout << "Norte" << endl; break;
14         case 1: cout << "Este" << endl; break;
15         case 2: cout << "Sur " << endl; break;
16         case 3: cout << "Oeste" << endl; break;
17     }
18     cout << "Terreno: ";
19     for (int i=0; i<sensores.terreno.size(); i++)
20         cout << sensores.terreno[i];
21     cout << endl;
22
23     cout << "Superficie: ";
24     for (int i=0; i<sensores.superficie.size(); i++)
25         cout << sensores.superficie[i];
26     cout << endl;
27
28     cout << "Colisión: " << sensores.colision << endl;
29     cout << "Reset: " << sensores.reset << endl;
30     cout << "Vida: " << sensores.vida << endl;
31     cout << endl;
32
33
34     // Determinar el efecto de la ultima accion enviada
35     return accion;
36 }
37
38 int ComportamientoJugador::interact(Action accion, int valor){
39     return false;
40 }
```

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 así como un fichero de documentación en formato PDF que describa el comportamiento implementado con un máximo de 5 páginas.

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

### 5.2. Cuestionario de autoevaluación

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

### 5.3. Método de evaluación

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 a continuación:

Nivel	Puntuación
0	1
1	2
2	3
3	2
4	2

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

$$\text{Nota}_i = s_0 p_0 + s_1 p_1 + s_2 p_2 + s_3 p_3 + s_4 p_4$$

dónde  $s_i$  representa el porcentaje de mapa descubierto y  $p_i$  la puntuación asociada al nivel  $i$ , que es la que se muestra en la tabla anterior.

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 esta 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, 3000 puntos de batería y un máximo de 300 segundos para decidir la siguiente acción. Cuando se agote alguno de ellos, la ejecución/simulación terminará.

Los 5 niveles de dificultad son los siguientes:

**Nivel 0:** En este nivel, 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, no se conoce la posición del agente aunque sí su orientación. En este caso, los sensores posF y posC devuelven los valores -1. Estos sensores sólo ofrecerán los valores correctos de fila y columna cuando el





agente se encuentre situado en una casilla de **posicionamiento** (las etiquetadas con la letra 'G').

**Nivel 2:** Ahora no se conoce tampoco la orientación, aunque se sabe que siempre que se aparece en el mapa, se empieza con orientación **norte**. Al igual que en el nivel anterior, hay que buscar una casilla de **posicionamiento** para conseguir situarse correctamente en el mapa. En este nivel, el sensor de orientación no funciona bien y siempre devuelve que el agente está orientado al norte.

**Nivel 3:** Sensorialmente es igual que el nivel 2, pero a diferencia de este aparecen aldeanos en la simulación que pueden obstaculizar los movimientos del agente.

**Nivel 4:** Sensorialmente igual que los niveles 2 y 3, pero a diferencia del anterior, ahora pueden aparecer 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. Es decir, aparecer en una nueva casilla del mapa desconocida, aunque con orientación norte y perdiendo las zapatillas y el bikini si los llevaba.

## 5.4. Fechas Importantes

La fecha tope para la entrega será el **LUNES 4 DE ABRIL DE 2022** antes de las **23:00 horas** y desde el 5 DE ABRIL estará disponible la entrega para el cuestionario de autoevaluación hasta **EL MIÉRCOLES 7 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 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**.



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.