

# INTELIGENCIA ARTIFICIAL

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

## Práctica 1

### Agentes Reactivos

(Los extraños mundos de BelKan)



DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

UNIVERSIDAD DE GRANADA

Curso 2016-2017

## 1. Introducción

### 1.1. Motivación

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. Se trabajará con un simulador software. Para ello, se proporciona al alumno un entorno de programación, junto con el software necesario para simular el entorno. En esta práctica, se diseñará e implementará un agente reactivo basado en los ejemplos del libro *Stuart Russell, Peter Norvig, “Inteligencia Artificial: Un enfoque Moderno”, Prentice Hall, Segunda Edición, 2004*. El simulador que utilizaremos fue inicialmente desarrollado por el profesor Tsung-Che Chiang de la NTNU (Norwegian University of Science and Technology, Trondheim), pero la versión sobre la que se va a trabajar ha sido desarrollada por los profesores de la asignatura<sup>1</sup>.

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. [http://www.youtube.com/watch?v=C1mVaje\\_BUM](http://www.youtube.com/watch?v=C1mVaje_BUM)
2. [http://www.youtube.com/watch?v=dJSc\\_EKfTsw](http://www.youtube.com/watch?v=dJSc_EKfTsw)



**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

---

<sup>1</sup>Agradecemos el trabajo realizado por José Ángel Segura en el desarrollo del entorno de simulación.

también más barata), una aspiradora inteligente presenta un comportamiento reactivo puro: Busca suciedad, la limpia, se mueve, detecta suciedad, la limpia, se mueve, y continúa con este ciclo hasta que se cumple alguna condición de parada. Otras versiones más sofisticadas permiten al robot *recordar* mediante el uso de representaciones icónicas como mapas, lo cual permite que el aparato ahorre energía y sea más eficiente en su trabajo. Finalmente, las aspiradoras más elaboradas (y más caras) pueden, además de todo lo anterior, planificar su trabajo de modo que se pueda limpiar la suciedad en el menor tiempo posible y de la forma más eficiente. Son capaces de detectar su nivel de batería y volver automáticamente al cargador cuando esta se encuentre a un nivel bajo. Estas últimas pueden ser catalogadas como *agentes deliberativos* (se estudiarán en el tema 3 de la asignatura *Búsqueda en espacios de estados*).

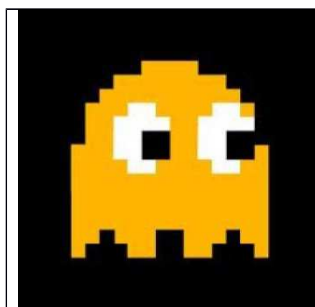
En esta práctica, centraremos nuestros esfuerzos en implementar el comportamiento de un “personaje virtual en una aventura gráfica” (un juego de ordenador) asumiendo un comportamiento reactivo. Utilizaremos **las técnicas estudiadas en el tema 2 de la asignatura** para el diseño de agentes reactivos.

## 1.2. Personajes virtuales en juegos de ordenador

Como todos sabéis, la Inteligencia Artificial tiene un papel importante en el desarrollo de los actuales juegos para consolas. Su papel cobra una relevancia muy especial al definir el comportamiento de aquellos personajes que intervienen en el juego, ya que dicho comportamiento 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 ya alguno si sabrá de quiénes hablamos.



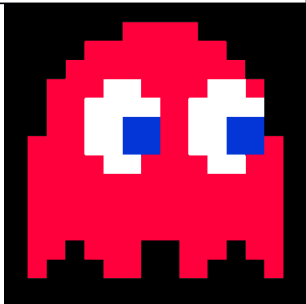
Si habéis jugado alguna vez a este mítico juego, seguro que no reparasteis en el mecanismo que permitía a los fantasmas perseguir o huir (en función de la situación del juego) del personaje principal. Bueno, pues el comportamiento de dichos fantasmas está regido por un agente básicamente reactivo. Como 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.



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,

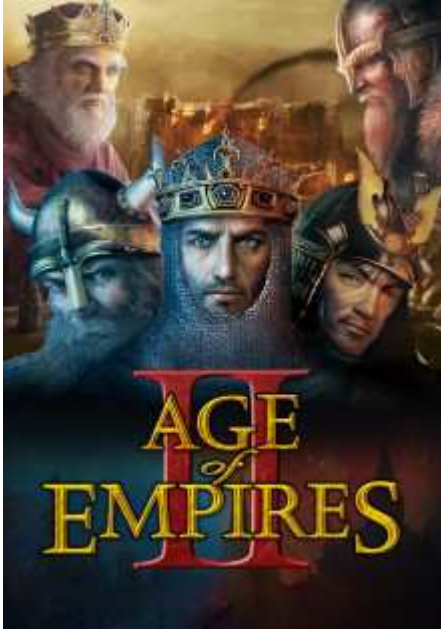
	veremos que Clyde en realidad no persigue a Pacman. Quizás no sienta una especial aversión hacia él y no sea tan vengativo como los demás fantasmas, o, tal vez, sea tan estúpido que se pierde dando vueltas por los pasillos.
	<p>El fantasma azul se hace llamar <b>Inky</b>. Su verdadero nombre, Bashful (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>
	<p>El fantasma rojo se llama <b>Blinky</b> 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.</p> <p>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).</p>

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.

Este es sólo un ejemplo concreto, pero en muchos juegos clásicos, y aún más en los juegos más modernos, la presencia de Inteligencia Artificial en dichos programas proviene de la definición de agentes reactivos (puramente reactivos o que mezclan lo reactivo con lo deliberativo).

También estamos acostumbrados, y por eso uno puede llegar a pensar, que los agentes reactivos estén vinculados con el comportamiento de los personajes secundarios de los juegos. Pero no es así, especialmente en los juegos de estrategia en tiempo real. Un juego que es algo antiguo, pero es claro ejemplo de juegos en los que los agentes reactivos toman el papel principal, es en el juego “*Age of Empires*” desarrollado en un principio por *Ensemble Studios*, más tarde por *Skybox Labs* y publicado por *Microsoft Game Studios*. El primer título apareció en 1997. Desde entonces, se han lanzado otros dos títulos y seis expansiones más. Todos ellos cuentan con dos modos principales de

juego (mapa aleatorio y campaña) y tratan sobre eventos históricos diferentes. La última versión del juego, llamado “*Age of Empires III: The Asian Dynasties*” (<http://www.ageofempires3.com/>), fue lanzado en 2007 y de él aparecieron posteriormente dos expansiones más del juego. Las críticas alabaron a Age of Empires respecto a su enfoque histórico y jugabilidad. Su *inteligencia artificial* en el juego lucha sin ventajas o "trampas", lo que no sucede en otros juegos de estrategia.



No contaremos en que consiste el juego, pero sí que diremos que el jugador humano debe manejar una serie de personajes indicándole las acciones que deben realizar. Muchas de esas acciones son exactas: seleccionamos un personaje en concreto y le damos una orden “Construir una casa aquí dónde te digo”. La parte reactiva/deliberativa viene en el movimiento del personaje a través del mapa para llegar al sitio donde le he dicho que levante la construcción. No es estrictamente deliberativa, ya que puede no conocer qué tipo de terreno puede encontrarse durante su recorrido, si habrá obstáculos o enemigos... Por consiguiente, necesita combinar la parte deliberativa con la reactiva para superar esos inconvenientes.

Si se juega con asiduidad al juego, terminamos dándonos cuenta que la IA es muy mejorable y pondremos 2 ejemplos para aquellos que conozcan el juego. El tipo de personaje básico se llama “aldeano” y es el encargado de la logística (construir, encontrar y recolectar la materia prima...). Lo

habitual es tomar a un grupo de aldeanos y decirles, por ejemplo, que corten árboles. Mientras hay árboles cerca, siguen cortando. En el momento en que no los tienen cerca, se paran y no hacen nada. Sería razonable que tuvieran la curiosidad de moverse y buscar nuevos árboles que cortar. Esto, desde el punto de vista de la “jugabilidad”, es un inconveniente, ya que si te despistas un poco es normal ver a un grupo de aldeanos mirándose unos a otros, en el mejor de los casos, o, en el peor de los casos, dispersados por el mapa sin saber tú dónde han terminado exactamente. El segundo ejemplo, y más importante a la hora de jugar, tiene que ver con el desplazamiento de las unidades guerreras por el mapa. Lo habitual es tomar un grupo de soldados e indicarles que vayan a un punto concreto del mapa. Bien, el grupo emprende el camino y va a donde le has dicho, y nada les distrae de esta acción, así que es posible que durante el trayecto un grupo de soldados enemigos te ataquen sin que ellos se defiendan. De hecho, una estrategia muy simple para ganar al jugador no humano es situarse en un punto intermedio de esa trayectoria y atacarles ahí. Los rivales no se defenderán. Claramente, no es un comportamiento inteligente y debería mejorarse incluyendo comportamientos reactivos que implementen la lógica más básica: “si quiero llegar a un sitio, tengo que mantenerme vivo para llegar”.





Otro ejemplo donde los comportamientos reactivos y deliberativos desempeñan un papel fundamental es una saga de juegos llamada “*The Settlers*”, cuya última versión se llama “*The Settlers Online*” (<https://www.ubisoft.com/es-ES/game/the-settlers-online/>) y está concebida para jugar con el propio navegador. En el caso de la versión



“*The Settler II*”, nos encontramos con un juego de estrategia en tiempo real, como “*The Age of Empires*” pero, a diferencia de este, el jugador humano tiene mucho menos control sobre las cosas que suceden. En este caso, si deseamos levantar una construcción, no indicamos qué personaje o conjunto de personajes tienen que hacerlo; simplemente marcamos el sitio de la construcción y los personajes, en función de su disponibilidad, se encargarán de hacerlo. Alguien que está habituado a juegos como “*The Age of Empires*” tendería a desesperarse con facilidad con

este juego, pero uno debe entender que en este juego todo va más lento, ya que los procesos que se deben realizar son más realistas. Siguiendo el ejemplo anterior, cuando indicamos que se levante una construcción y tenemos los recursos suficientes para poder hacerlo, en ese momento se activa un mecanismo de coordinación de agentes deliberativos que planifican la tarea teniendo en cuenta los recursos, tanto de personajes como de materiales. Una vez se tiene el plan, es posible que se reciba un ataque, en cuyo caso los personajes no siguen trabajando, sino que van a refugiarse o huyen hasta que pase el peligro (algo muy lógico). En estos casos, se activan los comportamientos reactivos, dejando en un segundo plano el plan. Cuando termina el peligro, se vuelve a “replanificar” teniendo en cuenta los personajes que aún quedan. En este juego, todo es más laborioso, más lento, menos inmediato, pero mucho más real. En este caso, la labor del jugador humano consiste más en distribuir adecuadamente la población entre los distintos gremios de trabajadores que en estar encima de todas las acciones de los personajes.

En la línea de este último juego (obviamente, de forma mucho menos ambiciosa) se orienta esta práctica, que pasamos a describir en la siguiente sección.



## 2. Los extraños mundos de BelKan

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

## 2.1. El escenario de juego

Este juego se desarrolla sobre un mapa bidimensional discreto que contiene como máximo 100 filas y 100 columnas. El mapa representa los accidentes geográficos de la superficie de parte de un planeta semejante a la Tierra. Sus elementos son considerados inmutables; es decir, el mapa no cambia durante el desarrollo del juego.

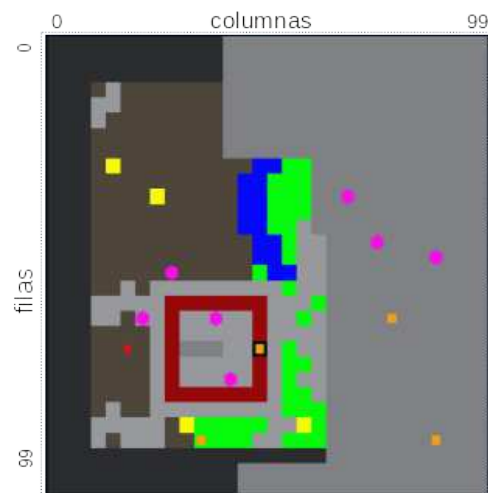
Representaremos dicha superficie usando una matriz de tamaño 100x100 de caracteres, donde la primera componente representa la fila y la segunda representa la columna dentro de nuestro mapa. Fijaremos sobre este mapa las siguientes consideraciones:

- 1 La casilla superior izquierda del mapa es la casilla [0][0].
- 2 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, ira al SUR si incrementa su valor en la fila, ira al ESTE si incrementa su valor en las columnas, y por último ira al OESTE si decrementa su valor en columnas.

Los elementos permanentes en el terreno son los siguientes:

- 1 *Árboles o Bosque*, que se codifican con el carácter '**B**' y se representan en el mapa como casillas de color verde.
- 2 *Agua*, que se codifica con el carácter '**A**' y tiene asociado el color azul.
- 3 *Precipicios*, que se codifica con el carácter '**P**' y tiene asociado el color negro.
- 4 *Suelo pedregoso*, que se codifica con el carácter '**S**' y tiene asociado el color gris.
- 5 *Suelo Arenoso*, que se codifica con el carácter '**T**' y tiene asociado el color marrón.
- 6 *Punto de Referencia o PK*, que se codifica con el carácter '**K**' y se muestra en amarillo (explicaremos su utilidad más adelante).
- 7 *Muros*, se codifican con el carácter '**M**' y son rojo oscuro.
- 8 *Puertas*, se codifican con el carácter '**D**' y son naranja con el borde negro.
- 9 *Casilla aún desconocida*, se codifica con el carácter '?' y se muestra en gris oscuro (representa la parte del mundo que aún no has explorado).



Una característica peculiar de este mundo es que es cerrado. Eso significa que no se puede salir de él, ya 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.

Sobre esta superficie existen elementos que tienen la capacidad de moverse por sí mismos o bien de ser trasladados sobre dicho mapa. Podemos clasificar a dichos elementos en dos tipos:

- *Otros personajes del juego:* A diferencia de los anteriores, estos no compiten con nosotros en el juego y tienen ya un comportamiento predefinido. Este comportamiento propio será un obstáculo para el desarrollo de nuestra tarea dentro del mapa. Los personajes genéricos del juego son los siguientes:
  - o *Aldeanos:* 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. Se codifican con el símbolo 'a'.
  - o *Lobos:* Habitantes especiales de este mundo que, al igual que los aldeanos, deambulan por el mundo molestando. A diferencia de los anteriores, si pueden resultar peligrosos. Durante el juego descubriréis en qué sentido lo son. Se codifican con el símbolo 'l' (le minúscula).
- *Los objetos,* elementos que no tienen capacidad para trasladarse por sí mismos, se codifican con caracteres entre '0' y '3'. Los jugadores pueden hacer uso de ellos durante la evolución del juego. Los diez objetos que aparecen en el juego son los siguientes:
  - o '0': Hueso
  - o '1': Bikini
  - o '2': Zapatillas
  - o '3': Llave

La utilidad de cada objeto debes descubrirla durante el juego y, en algunos casos, no es evidente. Si hay algo que parece inútil es porque no lo has usado lo suficiente como para descubrir para qué sirve o bien es que verdaderamente es inútil.

Los objetos se representan en el mapa de color rosa.

## 2.2. Nuestro Personaje

Obviamente, nuestro personaje es el protagonista de la historia y debe enfrentarse a las adversidades que le proponga el mundo y sus habitantes.

El objetivo de esta práctica es dotar de un comportamiento inteligente a este personaje usando un agente reactivo para definir las habilidades que le permitan dentro del juego, descubrir en el tanto por ciento más alto posible todos los elementos inmóviles del mapa.



### 2.2.1. Objetivo: Descubrir y orientar correctamente el mapa

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** del mapa original. El personaje se las deberá ingeniar, usando un comportamiento puramente reactivo, para ir descubriendo todo lo que hay (me refiero a la parte de 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.

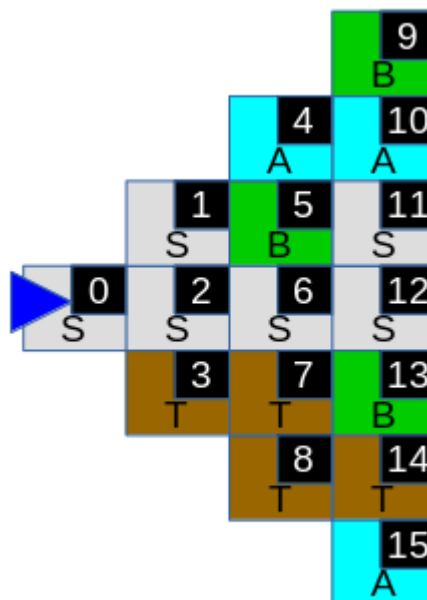
¿Cómo puedo descubrir mi posición en el mapa original? Pues la respuesta a esta pregunta son los PK o puntos de referencia (las casillas amarillas del mapa). Cada vez que nos situamos encima de una casilla PK, recibiremos cuál es la posición exacta de dicha casilla en el mapa original. Usando esta información podemos situar con completa exactitud, nuestra posición en el tablero original. Dejaremos que seas tú el que descubra la formulación necesaria para hacerlo.

Asociado a los datos que maneja nuestro personaje, hay definida una matriz de tamaño 100x100 (del mismo tamaño que el mapa original) destinada a ir guardando lo que sabemos seguro que hay en cada casilla. Esta matriz 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.

### 2.2.3. Capacidad sensorial de nuestro personaje

La siguiente pregunta a responder, después de conocer el objetivo del juego, es descubrir qué características tiene nuestro personaje. Bueno, sabemos que tiene forma de quesito. También sabemos que es rojo (esto último creo que no lo he dicho, pero ya ha quedado desvelado el secreto). Además, cuenta con un sistema visual que le permite ver las cosas que tiene delante de él. Esta 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 de mapa, es decir, el terreno. El segundo de ellos, que llamaremos ‘**superficie**’, es capaz de mostrarnos qué objetos u otros personajes se encuentran en nuestra visión. Para entender cómo funciona este sistema pondré un 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 como 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.

#### 2.2.4. Otros elementos del personaje

Además del sistema sensorial descrito anteriormente, nuestro personaje tiene la capacidad de coger objetos del mundo. Así que podemos suponer que tiene una mano (o algo parecido) para realizar esta operación. También, supondremos que tiene una mochila con la capacidad de almacenar hasta 4 objetos recogidos en el mundo. Dos sensores permiten determinar los objetos que posee en cada momento del personaje. El primero llamado '**objetoActivo**' es de tipo carácter y puede tomar los valores del '0' al '3' o el valor '-'. Este último valor indica que no tiene nada. Hay que entender que **objetoActivo** tiene un sentido diferente en función del tipo de objeto del que se trate, así, si son las llaves significa que las tiene en la mano, y si son unas zapatillas se supone que las tiene puestas en sus pies. De los objetos almacenados en la mochila, el sensor nos informa de objeto que será el siguiente en salir de ella. El nombre de dicho sensor es '**mochila**'. La mochila funciona como una cola, es decir, cada vez que meto un objeto en ella, se coloca al final y, cada vez que saco objeto, lo toma del principio siguiendo la filosofía de "primero en entrar, primero en salir".

Debemos indicar que nuestro personaje sufre un deterioro paulatino que puede verse aumentado por ciertos accidentes que puede sufrir durante su aventura, ya que, en algunos momentos, su integridad física puede verse afectada. Inicialmente, nuestro personaje empieza con 1000 unidades de vida. En cada ciclo del juego, pierde una de estas unidades. Puede perder unidades de vida adicionales, ya sea por el infortunio, ya sea por imprudencia. Esta información viene definida en el sensor '**vida**'. Cuando un personaje consume sus unidades de vida, muere. La muerte en este juego se llama reinicio y viene controlado por el dato '**reset**'. No os pongáis tristes, ya que morir en el juego implica que se vuelve a reaparecer en el mismo mundo, con una nueva posición desconocidas (aunque con orientación norte) y se pierden las posesiones que se tuvieran en el momento del reinicio y se vuelve a empezar con 1000 unidades de vida. El alumno debe ser hábil para que esa secuencia de la aventura vivida entre reinicios sea productiva y se pueda aprovechar en sus sucesivas etapas de vida.

#### 2.2.5. Datos del personaje compartidos con el entorno

El agente comparte con el entorno una matriz llamada '**mapaResultado**'. Esta matriz es siempre de tamaño 100x100 y es donde se tiene que poner la parte del mapa explorado para su evaluación por parte de simulador. La primera componente de la matriz es la fila y la segunda la columna.

Esta matriz debe ser usada por el estudiante sólo para incluir los valores del mapa cuando esté seguro de que ya se ubican correctamente. Para las situaciones donde aún no está seguro, el estudiante deberá definir otras matrices auxiliares para almacenar temporalmente el terreno reconocido.

El proceso de evaluación del mapa es el último proceso que se realiza justo después de realizar todas las iteraciones de simulación.

### 2.2.6. Las acciones que puede realizar el personaje

Nuestro personaje puede realizar 10 acciones distintas durante el juego. Las acciones las podemos considerar de tres clases:

- Tres acciones asociadas a operaciones de movimiento sobre el mundo:
  - o **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. El alumno debe averiguar que casillas son transitables y bajo qué condiciones.
  - o **actTURN\_L** le permite mantenerse en la misma casilla y girar a la izquierda 90° teniendo en cuenta su orientación.
  - o **actTURN\_R** le permite mantenerse en la misma casilla y girar a la derecha 90° teniendo en cuenta su orientación.
- No hacer nada:
  - o **actIDLE**, pues como su nombre indica, no hace nada.
- Seis acciones asociadas con la manipulación de objetos:
  - o **actPICKUP** le permite coger un objeto que está en el mundo. Para que la operación se realice con éxito es necesario que el personaje no tenga nada en uso y que esté situado en una casilla adyacente a la del objeto y que esté orientado hacia el objeto.
  - o **actPUTDOWN** es la acción contraria y, por consiguiente, le permite dejar un objeto en el mundo. Las condiciones no son simétricas a las anteriores y dejamos que seáis vosotros los que lo averigüéis.
  - o **actPUSH** le permite guardar un objeto en la mochila. Para ello, debe tener un objeto en uso y la mochila no debe estar llena.
  - o **actPOP**, la operación contraria, permite recuperar el primer objeto de los que están en la mochila y dicho objeto pasa a estar en uso.
  - o **actGIVE** es la operación mediante la que se entrega un objeto a uno de los personajes especiales del juego. Dejamos que seáis vosotros los que averigüéis las condiciones.

- o **actTHROW** permite lanzar un objeto contra los elementos y personajes del mundo. No todos los objetos pueden ser lanzados y no en todas las condiciones.

### 3. Objetivo de la práctica

La práctica actual tiene como objetivo diseñar e implementar un comportamiento reactivo a nuestro personaje. Dicho comportamiento debe permitirle determinar el contenido del mapa original así como su orientación correcta. Es importante tener en cuenta que partida del juego tiene 20,000 pasos.

## 4. El Software

### 4.1. Instalación

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 (sin la base de conocimiento, que es lo que se pide que se complete).

Se proporcionan versiones del software para el sistema operativo Linux. Dicho software se puede encontrar en el apartado de “Material de la Asignatura” dentro de la plataforma docente de DECSAI, <http://decsai.ugr.es>.

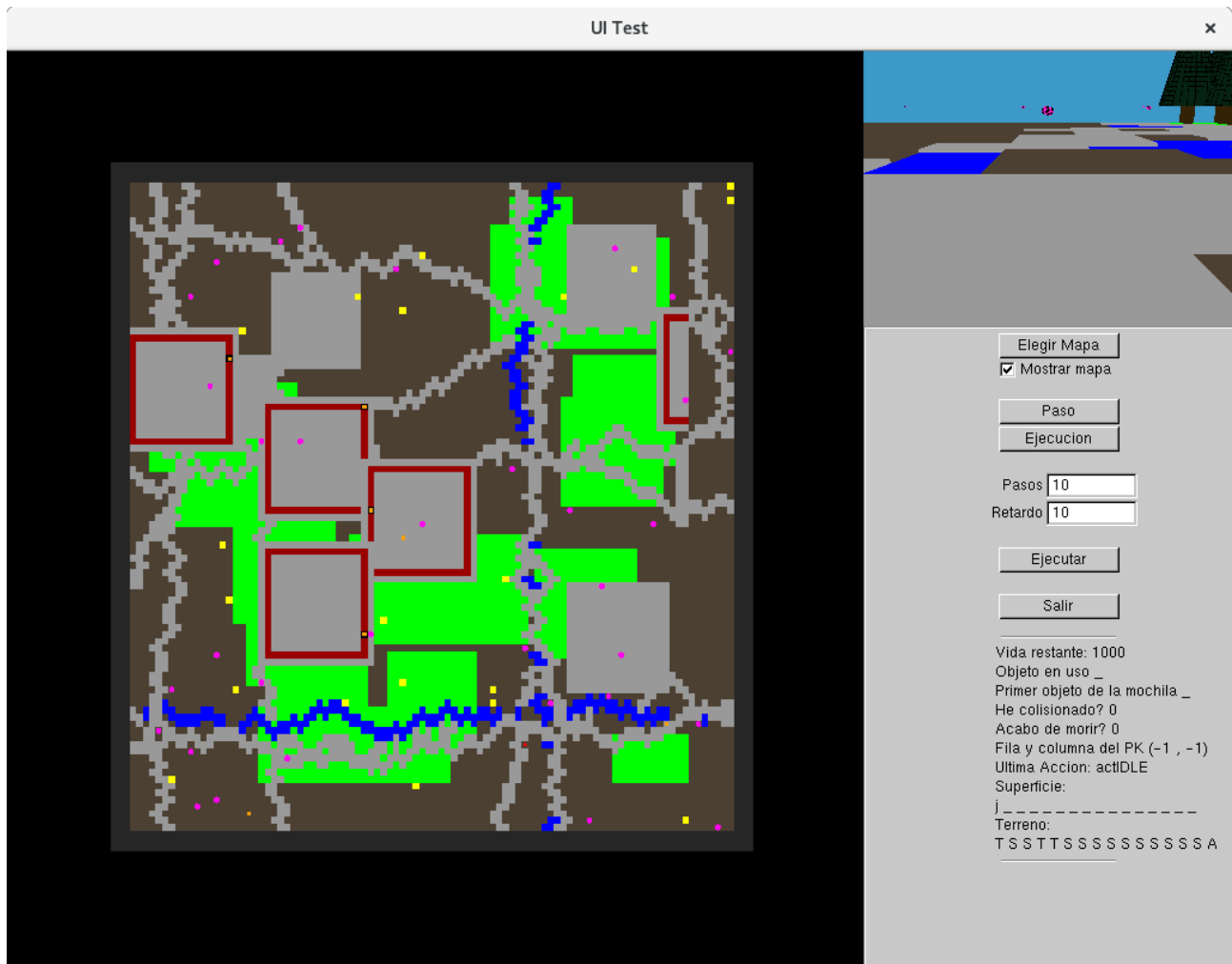
El proceso de instalación es simple:

- Se accede a la sección de Material de la Asignatura y se selecciona el archivo comprimido ‘practica1.tgz’.
- Se descarga en la carpeta de trabajo.
- Se crea una carpeta nueva y accedemos a dicha carpeta
- Se descomprime usando el comando ‘**tar zxvf ../practica1.tgz**’)
- Estando en dicha carpeta, ejecutamos el comando ‘**./install.sh**’. Esto instalará todas las bibliotecas necesarias para compilar la práctica. (Para otras distribuciones distintas a UBUNTU, es necesario modificar el install.sh y colocar como se llaman dichas bibliotecas en esa distribución)
- Una vez terminado este proceso, que puede llevarse un tiempo, se ejecuta el comando ‘**cmake .**’, que genera el fichero “**makefile**”.
- Por último, ejecutamos el comando ‘**make**’ para compilar el software.

### 4.2. Funcionamiento del Programa

Al arrancar el programa, nos mostrará una ventana como la que se muestra en la figura anterior, que es la ventana principal, y una ventana adicional a su derecha que mostrará una visión del mapa desde el punto de vista de nuestro personaje.

En la ventana principal está destinada a dibujar la parte del mapa del mundo que hemos recorrido; la de la parte derecha, que contiene los botones que controlan el simulador; y la inferior derecha, que mostrará los valores de los sensores de nuestro personaje;. Para su uso se ha de pulsar el botón '*Elegir Mapa*'. Una vez pulsado, nos aparecerá una nueva ventana donde deberemos elegir un mapa. Tras elegir el mapa, las ventanas se actualizarán y nos aparecerá algo como:



En la parte centro, se muestra lo que ve nuestro personaje siguiendo el código de colores que ya describimos anteriormente; en la parte inferior derecha, los valores actuales de los sensores, en el formato en el que son recibidos; y en la ventana del punto de vista, una proyección de lo que ve nuestro personaje en este instante.

Los botones '*Paso*', '*Ejecución*' y '*Ejecutar*' 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. Inicialmente, el software tiene implementado el comportamiento que "haya lo que haya voy hacia adelante". El botón '*Ejecución*' realiza una ejecución completa de la



simulación. En el modo local, una simulación completa implica 20.000 iteraciones. El último botón, ‘Ejecutar’ es un punto intermedio entre ‘Paso’ y ‘Ejecución’ que permite ejecutar de forma consecutiva tantas acciones como se indique en el campo ‘Pasos’. Por defecto, su valor viene fijado en ‘10’.

El botón “Mostrar mapa” alterna entre el mapa descubierto por el agente con el mapa completo a descubrir.

#### 4.2.1. Funcionamiento del Programa. Versión sin entorno Gráfico

Se incluye con el software una versión que no tiene entorno gráfico. Esta versión es muy útil para realizar la depuración de nuestro código cuando se produzcan errores. Tanto la versión con gráficos como la que no tiene gráficos, comparten todos los archivos, menos el fichero que contiene la función ‘main()’, por consiguiente, el funcionamiento el agente que implementemos debe funcionar igual en ambas versiones (tened en cuenta que hay componentes aleatorias en la ejecución de una simulación, por tanto, el funcionamiento es igual aunque el comportamiento particular en cada simulación puede ser distinta).

En la versión sin entorno gráficos se debe agregar como argumento en la llamada el fichero que contiene el mapa. Por ejemplo, si estamos en la carpeta raíz del software, invocar al programa sería de la siguiente forma:

**./BelkanSG ./mapas/mapa30.map**

#### 4.3. Descripción del Software

De todos los archivos que se proporcionan para compilar el programa, el alumno sólo puede modificar 2 de ellos, los archivos ‘jugador.cpp’ y ‘jugador.hpp’. Estos archivos se encuentran en la carpeta “Comportamientos\_Jugador” y 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. Para aclarar esto mejor, 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     }
12
13     ComportamientoJugador(const ComportamientoJugador & comport) : Comportamiento(comport){}
14     ~ComportamientoJugador(){}
15
16     Action think(Sensores sensores);
17
18     int interact(Action accion, int valor);
19
20
21     ComportamientoJugador * clone(){return new ComportamientoJugador(*this);}
22
23
24 private:
25
26 };
27
28
29 #endif
```

Pasamos a mirar el archivo '**jugador.hpp**'. En este archivo está declarada la clase '**ComportamientoJugador**'. Destacar por su relevancia para la práctica, el constructor (línea 10) y el método que define el comportamiento del agente (línea 16). El primero de ellos se usará durante la realización de la práctica para inicializar el valor de las variables de estado. La definición de dichas variables de estado se realizará después de la cláusula "private:".

El método "think" es el que contendrá el comportamiento del agente. Se puede ver que toma como entrada una variable de tipo "Sensores" y devuelve una acción. "sensores" es una variable de tipo "struct" que contiene los siguientes campos:

terreno	vector <unsigned char>
superficie	vector <unsigned char>
colision	bool
mochila	unsigned char
reset	bool
vida	int
objetoActivo	unsigned char

La cada campo codifica la información que aporta cada uno de los sensores y cuya descripción ya se hizo anteriormente.

El fichero "**jugador.cpp**" contendrá la implementación de la función "**think**" junto con todas aquellas otras funciones que el estudiante considere necesarias para definir el comportamiento del agente. Podrán definirse tantas funciones como se consideren oportunas, pero todas han de estar en este fichero.

```

1 #include "../Comportamientos_Jugador/jugador.hpp"
2 #include <iostream>
3 using namespace std;
4
5
6
7
8
9 Action ComportamientoJugador::think(Sensores sensores){
10
11     Action accion = actIDLE;
12
13     // En esta matriz de tamaño 100x100 hay que escribir el mapa solución
14     // mapaResultado[fila][columna] = lo que hay en fila columna
15
16
17     cout << "Terreno: ";
18     for (int i=0; i<sensores.terreno.size(); i++)
19         cout << sensores.terreno[i];
20     cout << endl;
21
22     cout << "Superficie: ";
23     for (int i=0; i<sensores.superficie.size(); i++)
24         cout << sensores.superficie[i];
25     cout << endl;
26
27     cout << "Colisión: " << sensores.colision << endl;
28     cout << "Mochila: " << sensores.mochila << endl;
29     cout << "Reset: " << sensores.reset << endl;
30     cout << "Vida: " << sensores.vida << endl;
31     cout << "objetoActivo: " << sensores.objetoActivo << endl;
32     cout << endl;
33
34     return accion;
35 }
36
37 int ComportamientoJugador::interact(Action accion, int valor){
38     return false;
39 }

```

En la versión que se entrega al estudiante de ‘jugador.cpp’, “think” devuelve “actIDLE”, es decir, no hacer nada, y el código sólo incluye mensajes por la terminal de los valores de los sensores.

## 5. Método de evaluación y entrega de prácticas

### 5.1. Método de evaluación

A partir del comportamiento entregado por el alumno en los ficheros ‘jugador.cpp’ y ‘jugador.hpp’ se procederá a hacer una simulación sobre un conjunto de nuevos mapas, donde cada agente compite solo (tenéis que considerar que los otros personajes definidos en el juego pueden moverse) sobre dicho mundo. La simulación constará de 20.000 iteraciones y las condiciones iniciales serán las mismas para los comportamientos definidos por cada estudiante.

La nota final que cada estudiante obtendrá será una ponderación de la media de acierto sobre el conjunto de mapas sobre el que se testeará y la complejidad del comportamiento definido que será evaluada durante el proceso de defensa.

Obviamente, las prácticas que se detecten copiadas implicarán un suspenso para el alumno en la asignatura, y la no presentación al proceso de defensa, un cero en la práctica.

## 5.2. 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 ficheros deberán entregarse mediante la plataforma web de la asignatura, en un fichero ZIP que no contenga carpetas. El archivo ZIP deberá contener sólo el código fuente de estos dos ficheros con la solución del alumno y un archivo de documentación en formato PDF describiendo (en lenguaje natural o mediante gráficos) el comportamiento definido en su agente con un máximo de 5 páginas.

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

## 5.3. Fechas Importantes

La fecha fijada para la entrega de las prácticas para cada uno de los grupos será la siguiente:

Grupo de prácticas	Fecha límite entrega
A3, B3, C3 y D2	22 de Marzo hasta las 23:00 horas
A1, B1	23 de Marzo hasta las 23:00 horas
C1	20 de Marzo hasta las 23:00 horas
A2,B2,C2,D1	21 de Marzo hasta las 23:00 horas

*Nota: Las fechas límite de entrega representan el concepto de hasta esa fecha, por consiguiente, es posible hacer la entrega antes de la fecha arriba indicada.*