

# INTELIGENCIA ARTIFICIAL

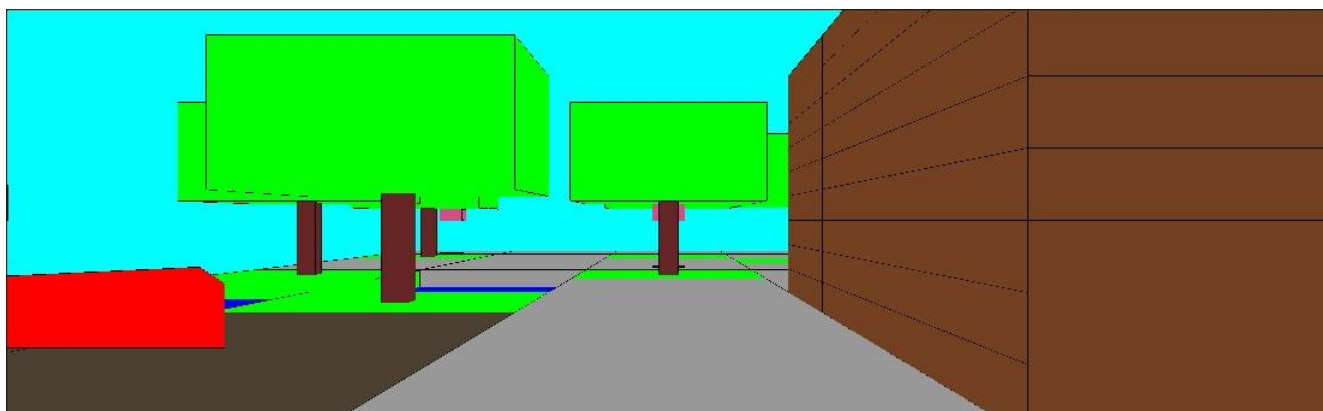
E.T.S. de Ingenierías Informática y de Telecomunicación

## Práctica 2

### Agentes Reactivos

(y algo de deliberativos)

(Los extraños mundos de BelKan)



DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

UNIVERSIDAD DE GRANADA

Curso 2015-2016

## 1. Introducción

### 1.1. Motivación

La segunda 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.

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:

- [http://www.youtube.com/watch?v=C1mVaje\\_BUM](http://www.youtube.com/watch?v=C1mVaje_BUM)
- [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 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*).

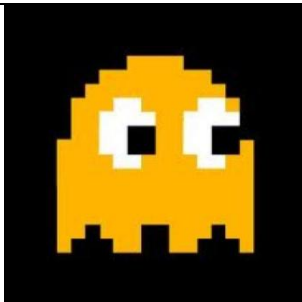
En esta práctica, centraremos nuestros esfuerzos en implementar el comportamiento de un “personaje virtual en una aventura gráfica” (un juego de ordenador) asumiendo un comportamiento reactivo y permitiendo ciertos comportamientos deliberativos para algunas fases del juego. Utilizaremos las técnicas estudiadas en los temas 2 y 3 de la asignatura para el diseño de agentes reactivos y deliberativos.

## 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.



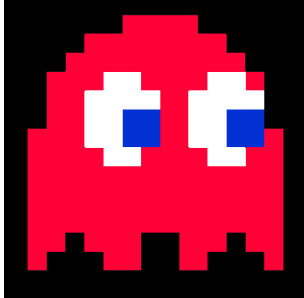
Si habéis jugado alguna vez a este mítico juego, seguro que no reparasteis en el mecanismo que permitía a los fantasmas perseguir o huir (en función de la situación del juego) del personaje principal. Bueno, pues el comportamiento de dichos fantasmas está regido por un agente básicamente reactivo. Como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





ugr

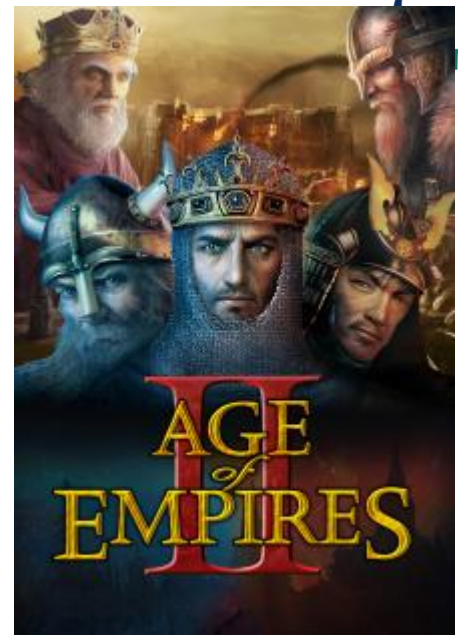
Universidad de Granada

Departamento de Ciencias de la Computación  
e Inteligencia Artificial

9

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 (opcionalmente, tendrán también cierta capacidad de deliberación).



## 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.

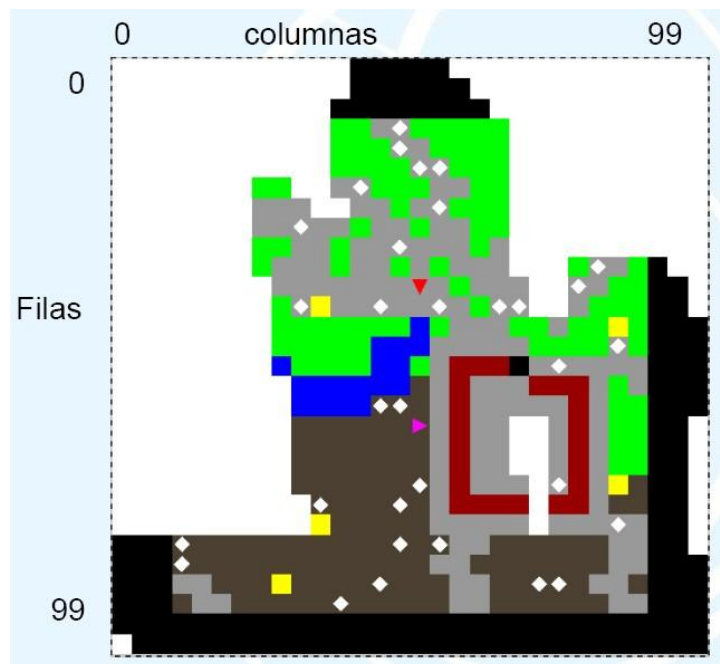
Representaremos dicha superficie usando una matriz de tamaño 100x100 de caracteres, donde la primera componente representa la fila y la segunda representa la columna dentro de nuestro mapa. Fijaremos sobre este mapa las siguientes consideraciones:

- 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, 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:

- *Árboles o Bosque*, que se codifican con el carácter '**B**' y se representan en el mapa como casillas de color verde.
- *Agua*, que se codifica con el carácter '**A**' y tiene asociado el color azul.
- *Precipicios*, que se codifica con el carácter '**P**' y tiene asociado el color negro.
- *Suelo pedregoso*, que se codifica con el carácter '**S**' y tiene asociado el color gris.
- *Suelo Arenoso*, que se codifica con el carácter '**T**' y tiene asociado el color marrón.
- *Punto de Referencia o PK*, que se codifica con el carácter '**K**' y se muestra en amarillo (explicaremos su utilidad más adelante).
- *Muros*, se codifican con el carácter '**M**' y son rojo oscuro.
- *Puertas*, se codifican con el carácter '**D**' y son gris oscuro.
- *Casilla aún desconocida*, se codifica con el carácter '?' y se muestra en blanco (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<sup>1</sup>. Podemos clasificar a dichos elementos en tres tipos:

- *Otros personajes jugadores*: Otros compañeros que estarán jugando con nosotros sobre el mismo mapa y vendrán codificados con los caracteres que van de la 'a' a la 'd'. Estos valores representan cómo es su orientación con respecto a la que yo tengo en el mapa. Así, 'a' significa que su dirección es contraria a la mía, 'b' que viene desde la derecha, 'c' que tiene la misma dirección que yo y 'd' que viene desde mi izquierda. En el tablero de juego, viene representados como triángulos rosas.
- *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. También puede tratarse de personajes que hay que localizar para conseguir puntos. Los personajes genéricos del juego son los siguientes:
  - *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.
  - *Osos*: Habitantes especiales de este mundo que, al igual que los aldeanos, deambulan por el mundo molestando. A diferencia de los anteriores, si son peligrosos. Durante el juego descubriréis en qué sentido lo son.

También hay habitantes con nombre propio dentro del juego, que son los siguientes:

- *La princesa*, habitual personaje de los juegos de fantasía. La distinguirás por su bonito atuendo completamente rosa.
- *El príncipe*, otro clásico personaje de los cuentos. Lo distinguirás por su atuendo de magnífico guerrero de camisa blanca y elegante y pantalón rosa (nosotros ponemos el dibujo, vosotros la imaginación para ver la parte magnífica del atuendo).
- *La bruja*, que no podía faltar estando los dos anteriores. En este caso viste un bonito conjunto camisa/pantalón en un clásico color negro.
- *Leonardo Di Caprio*, últimamente en todas las sopas y también deseamos incluirlo en esta. Viste conjunto marinerito con camisa azul y pantalones blancos.
- *El profe de IA*, el presupuesto de la práctica no daba para contratar a ningún personaje más y nos hemos tenido que incluir como personajes virtuales en nuestro propio juego. Su labor será vigilar que los jugadores no se intercambien información durante la duración de la práctica. Su vestuario es camaleónico para no ser detectada su presencia en el escenario. Así que no te fíes, mira bien porque está aunque creas que no. A veces se materializa y, en ese caso, viste con una elegantísima camisa amarilla y un pantalón azul.

---

<sup>1</sup> Los personajes se mueven cuando se juega siguiendo el juego remoto que se explicará más adelante, y que será con el que se evaluará la práctica. Sin embargo, los personajes no se mueven cuando se trabaja con un mapa estático.





Lo que caracteriza a estos personajes es que les gusta que les lleven cosas. Si les llevas objetos, te recompensarán con puntos dependiendo de lo que lleves y a qué personaje se lo entregues. Ojito con lo que llevas al profe de IA, es incorruptible. Yo no le daría cualquier cosa, tiene reacciones imprevisibles. Bueno, que os voy a contar que no sepáis ya.

Los personajes se codifican con caracteres desde la 'e' hasta la 'z'. Debes averiguar qué carácter corresponde con que personaje mientras juegas. En el mapa se representa un personaje mediante un rombo de color naranja.

- *Los objetos*, elementos que no tienen capacidad para trasladarse por sí mismos, se codifican con caracteres entre '0' y '9'. Los jugadores pueden hacer uso de ellos durante la evolución del juego. Los diez objetos que aparecen en el juego son los siguientes:
  - *Bikinis*
  - *Manzanas*
  - *Zapatillas*
  - *Palas*
  - *Rosas*
  - *Algoritmos*
  - *Oscars (de los de la academia de Hollywood)*
  - *Piedras*
  - *Lingotes de Oro*
  - *Llaves*

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.

Los objetos se representan en el mapa como rombos de color blanco.

## 2.2. Nuestro Personaje

Obviamente, nuestro personaje es el protagonista de la historia y debe enfrentarse a las adversidades que le planteen el resto de participantes en el juego.

Nuestro personaje tuvo que hacer algo malo en su juego anterior, ya que en el mundo de BelKan tiene forma de quesito y eso no es que sea ni feo ni guapo, sino que es raro. Yo sospecho, sólo sospecho, que viene de alguna versión no comercializada del “*Trivial*”. A propósito, ¿recuerdas a qué categoría se asociaba el quesito rojo en este juego? Es oír Trivial y te salen las preguntas de forma inconsciente.

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:

- a) Descubrir en el tanto por ciento más alto posible todos los elementos inmóviles del mapa.
- b) Realizar tareas tales como recoger objetos del mundo y llevárselos a los personajes (o molestar al resto de jugadores para que no consigan hacerlo).



Pasamos a describir con algo más de detalle cada uno de estos objetivos.

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

En relación al primero de los objetivos, que es el más importante en esta práctica, vamos a describir cómo empieza el juego. Nuestro personaje aparecerá de forma aleatoria sobre un mundo de BelKan concreto, que no cambiará durante el juego, sin conocer su posición, ni su orientación sobre el mapa original. En esta parte del juego, nuestro 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 y mi orientación exacta 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. Combinando dos de estas casillas, podemos situar con completa exactitud, nuestra posición y orientación 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.

### **2.2.2. Objetivos Secundarios**

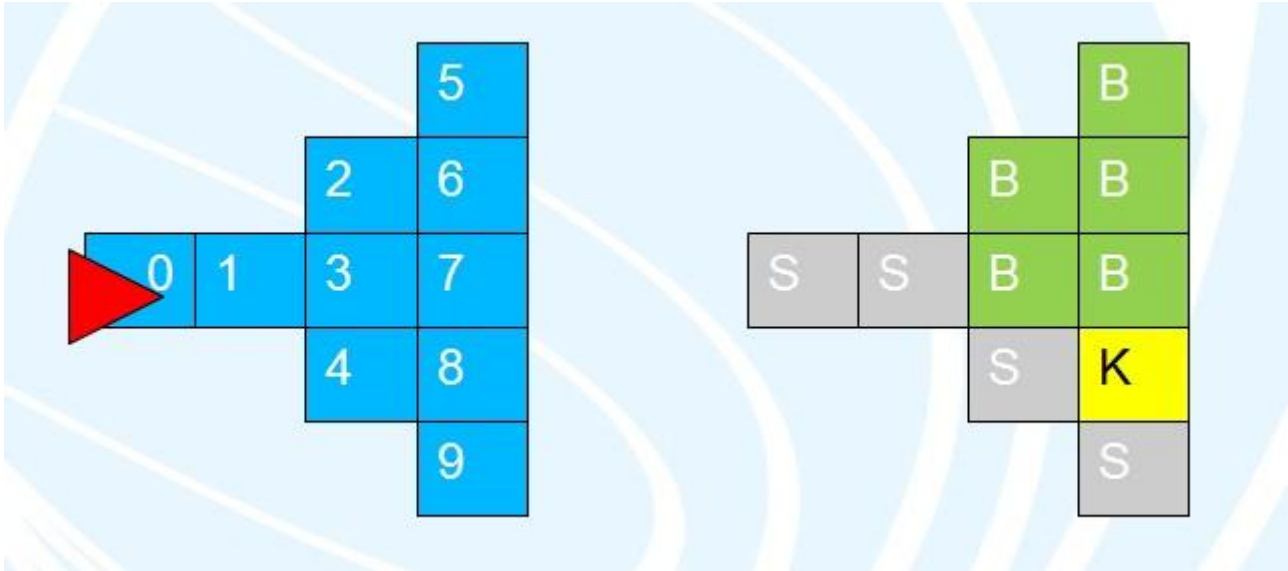
Como en la vida, durante este juego también hay tiempo para hacer otras cosas. Obviamente, durante todo el juego debemos tener presente que el objetivo principal es el anterior pero, en ocasiones, nos podríamos desviar para explorar un poco otras posibilidades del mundo que nos rodea. Además, es necesario saber salvar los peligros que nos presenta este mundo para completar la tarea anterior.

Sabremos que hemos obtenido un objetivo secundario cuando recibas puntuación por hacer algo. Hay una tarea muy clara, llevar objetos a los personajes, pero hay otras menos obvias que tendrás que ir descubriendo a medida que mejores tu personaje.

### **2.2.3. Capacidad sensorial de nuestro personaje**

La siguiente pregunta a responder, después de conocer los objetivos 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 10. El primero de ellos lo denominaremos 'VISTA\_' y es capaz de observar los elementos inmóviles de mapa, es decir, el terreno. El segundo de ellos, que llamaremos 'SURFACE\_', es capaz de mostrarnos qué objetos móviles se encuentran en

nuestra visión. Para entender cómo funciona este sistema pondré un ejemplo. Suponed que el vector VISTA\_ contiene SSBBSBBBKS, su representación real sobre un plano será la siguiente:



El primer carácter (posición 0) representa el tipo de terreno sobre el que me encuentro. El segundo carácter (posición 1) es justo el tipo de terreno que tengo justo delante de mí. Los caracteres de posiciones 2, 3 y 4 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 5 a la 9 son aquellos que están a tres casillas viéndolos de izquierda a derecha. Las figuras anteriores representan las posiciones del vector en su distribución bidimensional (la de más a la izquierda) y el carácter y su representación por colores como quedaría en un mapa (la de más a la derecha).

De igual manera se estructura el vector SURFACE\_ pero, en este caso, indicando que objetos móviles se encuentran en cada una de esas casillas.

El personaje también tiene un sensor que simula la capacidad de oír. En las acciones de movimiento cuando no se ha podido completar ese movimiento, así como en el resto de acciones, el personaje recibe información a través de un sensor llamado 'MENSAJE\_' que en lenguaje natural le informa si dicha acción se ha realizado con éxito. Este sensor es importante, ya que en algunas ocasiones es la única manera de verificar que la acción se ha completado. Por consiguiente, es necesario tener esto en cuenta durante la construcción del agente.

#### 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 datos propios de la clase se utilizan para determinar las posesiones en cada momento del personaje. El primero llamado 'EN\_USO\_' es de tipo carácter y puede tomar los valores del '0' al '9' o el valor '-'. Este último valor indica que no tiene nada. Hay que entender que 'EN\_USO\_' tiene un sentido en función del objeto, si es una manzana significa que lo tiene en la mano,



y si son unas zapatillas se supone que las tiene puestas. Los objetos almacenados en la mochila se representan mediante un vector de caracteres, que puede tomar cada componente el mismo rango de valores descrito para 'EN\_USO\_'. El nombre de dicho vector 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 puede sufrir accidentes durante su aventura por este mundo, ya que, en algunos momentos, su integridad física puede verse afectada. Inicialmente, nuestro personaje tiene 5 unidades de vida. Estas unidades las puede ir perdiendo debido a accidentes, ya sea por mala suerte, ya sea por imprudencia. Esta información viene definida en el dato 'SALUD\_'. Cuando un personaje consume sus 5 unidades de vida, muere. La muerte en este juego se llama reinicio y viene controlado por el dato 'REINICIADO\_'. No os pongáis tristes, ya que morir en el juego implica que se vuelve a reaparecer en el mismo mundo, con una nueva posición y orientación (ambas desconocidas) y se pierden las posesiones que se tuvieran en el momento del reinicio. El alumno debe ser hábil para que esa secuencia de aventura vivida desde el inicio hasta su reinicio sea productiva y se pueda aprovechar en sus sucesivas etapas de vida.

Dos elementos más se proporcionan como información al personaje y están relacionados con la puntuación que se va obteniendo en el juego. El primero de ellos se llama 'PUNTUACION\_' y sólo nos informa de los puntos que vamos acumulando durante el juego. Es simplemente un dato informativo y, en principio, no útil para el diseño del agente. Hay otra puntuación, más importante, que esta que se indica, que se produce cuando el juego termina, y está relacionada con el objetivo fundamental del juego. Como ya se comentó con anterioridad, mide el porcentaje de mapa descubierto por el personaje que coincide en posición y orientación con el mapa original. Pasamos a explicar esto con más detenimiento.

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

Dentro de la definición del agente, hay tres matrices declaradas como públicas a las que puede acceder el entorno. Dos de ellas, llamadas 'mapa\_entorno\_' y 'mapa\_objetos\_', son matrices de caracteres de tamaño 200x200. Estas dos matrices llevan la información de lo que voy recorriendo en el mundo, y sobre ellas, de forma automática, se va escribiendo la información proveniente de 'VISTA\_', en la primera de las matrices, y 'SURFACE\_' en la segunda. Su tamaño es del doble del tamaño del mapa original, y la razón es que ya que desconocemos nuestra posición en el mundo original, construimos una matriz lo suficientemente grande para no tener que controlar si me salgo de la matriz. Obviamente, para ello debo situarme en el centro de dichas matrices y establecer una orientación inicial. Estos valores iniciales vienen definidos en los datos 'x\_', 'y\_', 'orientacion\_' y, como he dicho, hacen referencia a la ubicación de nuestro personaje en estas dos matrices, no en la matriz original.

Hay una tercera matriz, 'matriz\_solucion\_', también de caracteres y ésta del mismo tamaño que el mapa original. Esta matriz es la que se envía al final del juego para ser comparada con el mapa original. Por consiguiente, cada vez que se sepa seguro que una casilla del mapa original contiene un tipo de terreno, se anotará en esta matriz. Muy importante: la matriz se envía automáticamente en cuanto se recibe la señal de fin de juego; por consiguiente, se debe ir modificando durante todo el tiempo que dure el juego.

Obviamente, para escribir en esta matriz, es necesario estar seguro de estar con la misma orientación que el mapa original y de que mis coordenadas 'x\_' e 'y\_' coincidan también con la posición en el mapa original. Así, antes de escribir en ella, es necesario orientarse. Esta debe ser la primera tarea de nuestro personaje.





### 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:
  - *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.
  - *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.
- No hacer nada:
  - *actIDLE*, pues como su nombre indica, no hace nada.
- Seis acciones asociadas con la manipulación de objetos:
  - *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 donde esté el objeto.
  - *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.
  - *actPUSH* le permite guardar un objeto en la mochila. Para ello, debe tener un objeto en uso y la mochila no debe estar llena.
  - *actPOP*, la operación contraria, permite recuperar el primer objeto de los que están en la mochila y dicho objeto pasa a estar en uso.
  - *actGIVE* es la operación mediante la que se entrega un objeto a uno de los personajes especiales del juego. Dejamos que seáis vosotros lo que averigüéis las condiciones.
  - *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.

### 2.2.7. Las acciones manejadas desde el teclado

Para ayudar a entender el funcionamiento del juego, durante la fase de desarrollo, se ha habilitado el uso del teclado para invocar las acciones del personaje, que de esta forma, el alumno puede ponerse en



distintas situaciones con las que se encontrará su personaje y estudiar las posibilidades que tiene para solventar cada una de ellas.

La relación de teclas y acciones se describe en la siguiente tabla:

<b>actFORWARD</b>	E
<b>actTURN_L</b>	S
<b>actTURN_R</b>	F
<b>actPICKUP</b>	H
<b>actPUTDOWN</b>	N
<b>actPUSH</b>	J
<b>actPOP</b>	M
<b>actGIVE</b>	D
<b>actTHROW</b>	Barra Espaciadora

### 3. Objetivo de la práctica

La práctica actual tiene como objetivo diseñar e implementar comportamientos que nuestro personaje se comporte como un agente reactivo. Dichos comportamientos deben permitirle, al menos, dar respuesta al objetivo principal del juego, determinar el contenido del mapa original así como su orientación correcta. Adicionalmente, el personaje intentará obtener la máxima puntuación posible resolviendo misiones que vayan surgiendo a lo largo de la evolución del juego. Es importante tener en cuenta que todos los objetivos se deben cumplir en una simulación de 20000 pasos.

### 4. El Software

#### 4.1. Instalación

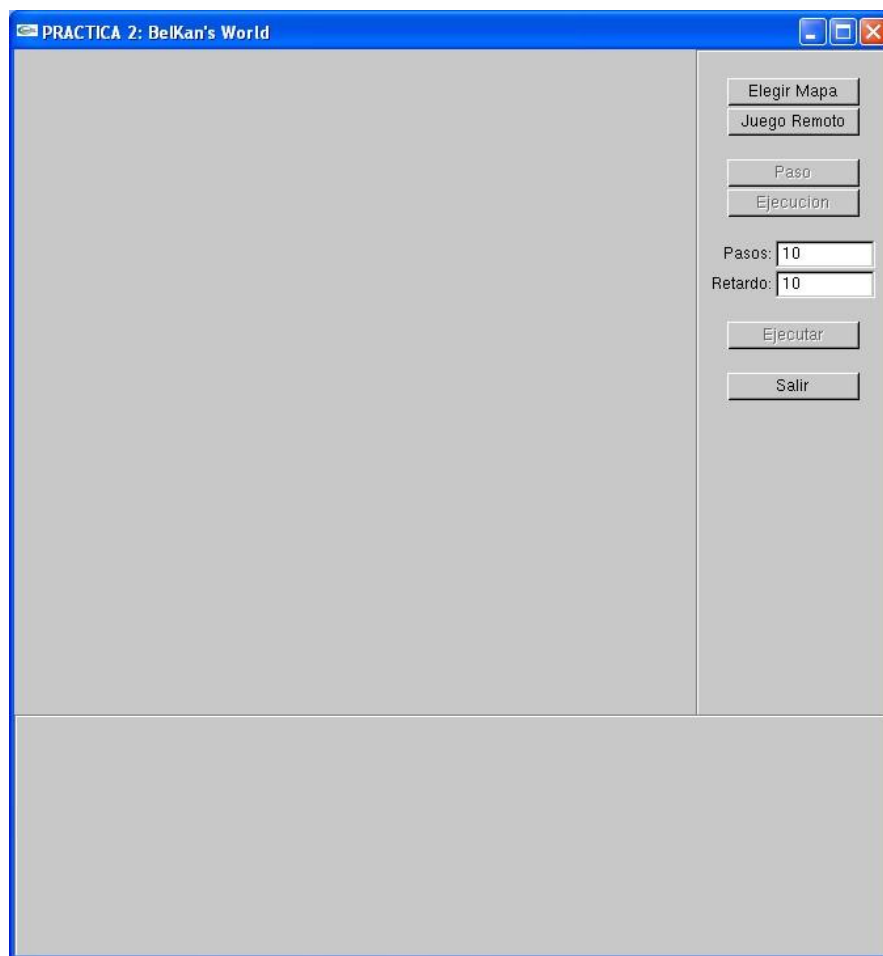
Para la realización de la práctica se proporciona al alumno una implementación tanto del entorno simulado del mundo en donde se mueve nuestro personaje como de la estructura básica del agente reactivo (sin la base de conocimiento, que es lo que se pide que se complete).

Se proporcionan versiones del software para el sistema operativo Linux y para el sistema operativo Windows. Dicho software se puede encontrar en el apartado de “Material de la Asignatura” dentro de la plataforma docente de DECSAI, <http://decsai.ugr.es> . La versión de Windows se ha desarrollado usando el entorno de desarrollo **CodeBlocks**, así que incluye un archivo ‘practica2.cbp’ para facilitar el trabajo. La versión de Linux, sin embargo, no viene preparada para usar este entorno de desarrollo, si bien se incluye un archivo ‘makefile’ para compilar el programa.

El proceso de instalación es muy simple:



1. Se accede a la sección de Material de la Asignatura y se selecciona el archivo comprimido 'practica2.zip' para la versión de Windows o 'practica2.tgz' para la versión de Linux.
2. Se descarga en la carpeta de trabajo.
3. Se descomprime generando una nueva carpeta llamada 'practica2' y accedemos a dicha carpeta.
4. Para los usuarios de Windows, se hace doble click en el archivo 'practica2.cbp' (previamente hay que tener instalado CodeBlocks <http://www.codeblocks.org/>), se selecciona la acción 'Build and Run' y se ejecuta el entorno de simulación de la práctica.
5. Para los usuarios de Linux, simplemente se ejecuta la orden 'make' para compilar y la orden './BelKan\_Client' para ejecutar el entorno.



Hay dos elementos adicionales que debes tener en cuenta si deseas instalar este software en tu ordenador personal o portátil cuando éste trabaja bajo sistema operativo Linux (esto es aplicable a los Mac también). Estos dos elementos son: la librería gráfica y el sistema de conexión por sockets. Para el primero de ellos, es necesario tener instalada la librería 'freeglut3'. Para saber cómo instalarla, poned en un navegador "freeglut 3 "+ Sistema Operativo (por ejemplo, los usuario de Ubuntu deben buscar "freeglut3 Ubuntu").

Los que tengáis instalado Ubuntu podéis seguir los pasos indicados en la página a la que se puede acceder mediante el enlace <http://ubuntuforums.org/showthread.php?t=345177>. Para el resto de



sistemas operativos, buscad una página semejante a la anterior que se adapte a vuestra versión de Linux. Lo mismo pueden hacer los usuarios de Mac.

A pesar de tener instalada la librería y de que no dé errores de compilación, esto no asegura que (la parte gráfica del software) vaya a funcionar perfectamente. Sería largo de explicar la razón, pero aquí os daré una explicación breve sacada de Wikipedia:

*“Direct3D y OpenGL son interfaces para la programación de aplicaciones (API acrónimo del inglés Application Programming Interfaces) competitivas que pueden ser usadas en aplicaciones para representar gráficos por computadora (o también llamado computación gráfica) en 2D y 3D, tomando como ventaja de la aceleración por hardware cuando se dispone de esta. Las unidades de procesamiento gráfico (GPU acrónimo del inglés Graphics Processing Unit) modernas, pueden implementar una versión particular de una o ambas de estas interfaces.”*

Y de aquí proviene la razón de que no os funcione bien del todo. Si vuestra tarjeta gráfica está diseñada específicamente para Direct3D (que es la que usan la mayoría de los juegos de ordenador), trabajará con OpenGL en modo simulación y, a veces, esa simulación no es tan buena. Un síntoma de esto suelen ser los problemas de refresco de ventanas (no se ven los botones, sólo actualiza una ventana o parte de una ventana y el resto no,...). Esto último tiene poca solución. A veces se puede corregir instalando una máquina virtual y ejecutando un sistema operativo distinto (como Windows), aunque eso no lo podemos garantizar. Lo que sí podemos garantizar es que funciona razonablemente bien en los ordenadores de los laboratorios de prácticas, así que sería conveniente para los que tengáis problemas de este tipo que aprovechéis bien el tiempo en las sesiones de prácticas.

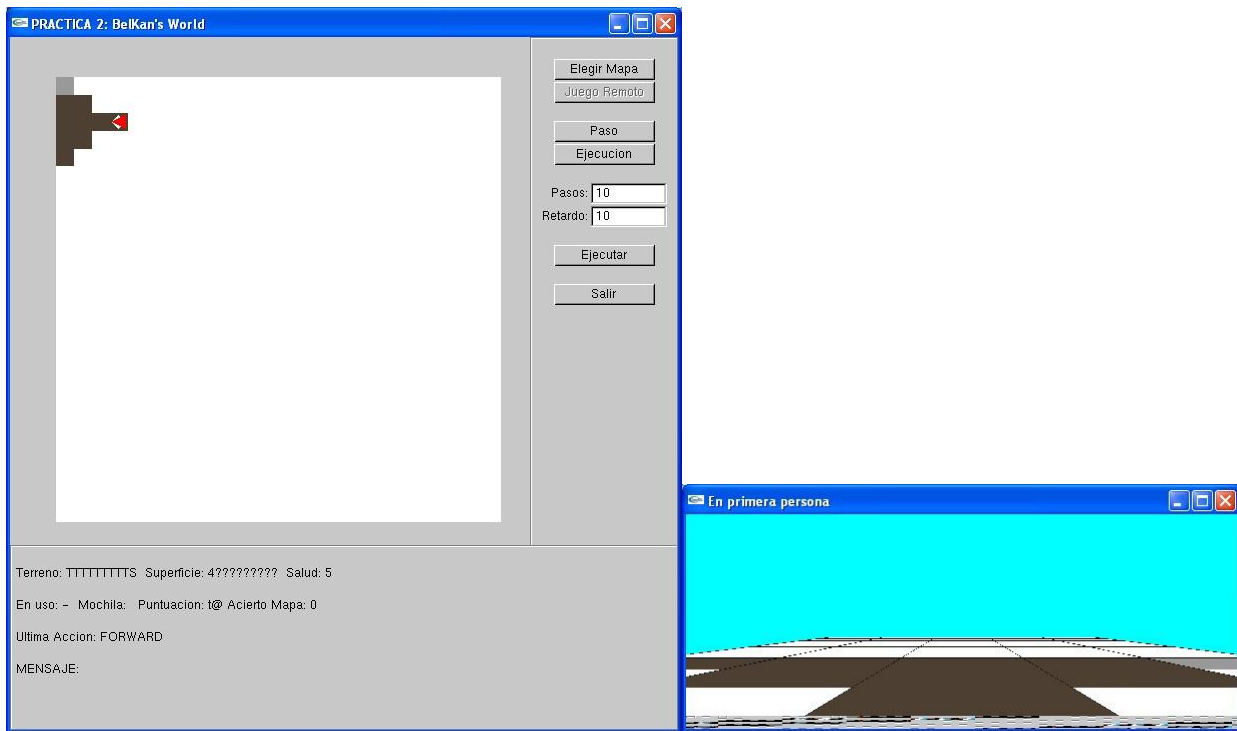
El otro elemento del que hablábamos es la biblioteca de sockets que permite la comunicación entre equipos y que aquí se usará para conectarnos con un servidor. Aunque también hay distintas versiones, en principio esto no debería suponer ningún problema, ya que en la construcción del software para Linux y Windows se contemplan las peculiaridades de cada una de ellas. Sin embargo, la continua evolución de los sistemas operativos y la posibilidad de cambios en las bibliotecas no las podemos prever. Durante las clases de prácticas intentaremos resolver todos los problemas que os vayan surgiendo y que esté en nuestra mano hacerlo. En este sentido, también os pedimos que, si alguien ha tenido un problema relativo a esto que os estamos contando y lo ha podido resolver, que nos lo haga saber para poder comunicárselo a vuestros compañeros y facilitarles también la tarea.

## **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 con el fondo completamente azul que mostrará una visión del mapa desde el punto de vista de nuestro personaje.

En la ventana principal, podremos distinguir tres partes: la central y más grande que está destinada a dibujar la parte del mapa del mundo que hemos recorrido; la inferior, que mostrará los valores de los sensores de nuestro personaje; y la de la parte derecha, que contiene los botones que controlan el simulador, que puede trabajar en local (o de forma aislada) o en remoto (con personajes diseñados por otros compañeros). Para usar la parte de trabajo local, 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, 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’.

Aparte de esos botones, os recuerdo que durante la fase de desarrollo se puede usar el teclado para activar directamente cada una de las acciones del personaje. Para ello, consultad los detalles en la sección 2.2.7.

El modo remoto, al cual se accede mediante el botón ‘Juego Remoto’, presenta el mismo comportamiento que el modo local, siendo la única diferencia la forma de inicio. Tras pulsar dicho botón, se nos pedirá que introduzcamos un identificador de usuario. Aquí debemos introducir nuestro DNI o número de pasaporte, se pulsa ‘Aceptar’ y nos aparecerá una ventana semejante a la anterior.

Este modo remoto sólo será accesible durante las sesiones de prácticas y sólo para los alumnos que están en ese momento en clase. En este modo, los personajes definidos por vosotros pueden compiten entre sí en el mismo escenario (es el equivalente en un juego de ordenador al modo ‘on line’).

Este modo será el que se utilizará para la valoración de los comportamientos definidos por el alumno, así que es importante que, durante las sesiones de prácticas, se ajusten dichos comportamientos para que se obtengan buenos resultados en modo remoto.

### 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 ‘belkan.cpp’ y ‘belkan.h’. 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. Para aclarar esto mejor, pasamos a ver con un poco más de detalle cada uno de estos archivos.

```
66     private:
67         //Variables de interaccion con el entorno grafico
68         int size_;
69
70         //SENSORES
71         char VISTA_[10];
72         char SURFACE_[10];
73         bool REINICIADO_;
74         string MENSAJE_;
75         char EN_USO_;
76         char MOCHILLA_[5];
77         char PUNTUACION_[9];
78         bool FIN_JUEGO_;
79         char SALUD_;
80
81         //Variables de estado
82         int x_, y_, orientacion_;
83         int last_accion_;
84         string role_;
85
86     };
```

Pasamos a mirar el archivo ‘belkan.h’, y nos vamos a la parte inferior del código. En este archivo está declarada la clase ‘Agent’ y, en la figura anterior, vemos la declaración de los datos privados. Las declaraciones desde la línea 71 a la 79 son los sensores y datos de comunicación con el entorno. Como se puede observar, los nombres corresponden con los que se indicaron anteriormente en este mismo documento al describir al personaje.

A partir de la línea 81 se declararán las variables de estado. Actualmente hay declaradas 5, de las que las 3 primeras son las que permiten posicionarme y orientarme en un mapa interno que se va construyendo mientras voy interactuando con el entorno. Estas variables me ubican dentro de dos matrices, que también vienen declaradas en la misma clase, pero como públicas. La razón de que su declaración sea pública es para permitir que el entorno acceda a ellas y visualice la información que conocemos hasta el momento.

```
53     void Perceive(Environment &env);
54     bool Perceive_Remote(conexion_client &Cliente, Environment &env);
55     void ActualizarInformacion(Environment *env);
56     ActionType Think();
57     void FixLastAction(Agent::ActionType x){last_accion_=x;};
58
59     char mapa_entorno_[200][200]; // mapa que muestra el tipo de terreno
60     char mapa_objetos_[200][200]; // mapa que muestra los objetos que estan encima del terreno
61     char mapa_solucion_[100][100]; // Mapa que almacena la solucion que el alumno propone
62     // Funciones de acceso a los datos
63     void GetCoord(int &fila, int &columna, int &brujula){fila=y_;columna=x_;brujula=orientacion_};
64
```

Los nombres de estas matrices son 'mapa\_entorno\_' que representa la superficie del mapa, y 'mapa\_objetos\_' que representa los objetos vistos sobre la superficie del mapa. Como se muestra en la figura anterior, hay una matriz más llamada 'mapa\_solucion\_' que será donde pondremos lo que vamos conociendo del mapa original, con su posición y orientación correcta, como ya se explicó anteriormente.

```
17 public:
18 Agent() {
19     x_ = 99;
20     y_ = 99;
21     orientacion_ = 3;
22     role_ = "PLYR";
23     last_accion_ = 3;
24     REINICIADO_ = false;
25     size_ = 200;
26     for (int i=0; i<200; i++)
27     {
28         for (int j=0; j<200; j++) {
29             mapa_entorno_[i][j] = '?';
30             mapa_objetos_[i][j] = '?';
31         }
32     }
33     for (int i=0; i<100; i++)
34     {
35         for (int j=0; j<100; j++)
36             mapa_solucion_[i][j] = '?';
37     }
```

En la parte superior de este mismo fichero, podemos encontrar el constructor de la clase Agent. En él debemos inicializar todas las variables de estado que incorporemos a nuestro agente. Podemos observar que las variables ya definidas están inicializadas aquí: las coordenadas 'x\_' e 'y\_', la orientación y las matrices ya comentadas (el carácter '?' en la inicialización indica que no se conoce el contenido de las casillas).

En el archivo 'belkan.cpp' se describe el comportamiento del agente. Las funciones definidas al principio del archivo son importantes para la gestión de las matrices anteriores. Por esa razón, todo este código se debe mantenerse tal y como está.

En este archivo, podemos añadir tantas funciones como necesitemos, pero las únicas funciones que se pueden modificar, y sólo para añadir código, son 'ActualizarInformacion' y 'Think'. Antes de explicar el código de cada una de ellas, es interesante saber que, dentro del proceso de tomar una decisión dentro de un agente reactivo, podemos considerar dos fases diferenciadas. La primera de ellas consiste en determinar cómo la última acción aplicada sobre el entorno ha afectado a mis variables de estado. Justo esa labor es la que debe realizarse en la función 'ActualizarInformacion', mientras que 'Think' asume que todas las variables de estado tienen sus valores correctos y se dedica únicamente a definir las reglas que regirán su comportamiento.

```
103 void Agent::ActualizarInformacion(Environment *env){
104     // Actualizar mi informacion interna
105     if (REINICIADO_){
106         // Lo que tengas que hacer si eres reposicionado en el juego
107     }
108
109     switch(last_accion_){
110     case 0: //avanzar
111         switch(orientacion_){
112             case 0: // norte
113                 y_--;
114                 break;
115             case 1: // este
116                 x_++;
117                 break;
118             case 2: // sur
119                 y_++;
120                 break;
121             case 3: // oeste
122                 x_--;
123                 break;
124         }
125         break;
126     case 1: // girar izq
127         orientacion_=(orientacion_+3)%4;
128         break;
129     case 2: // girar dch
130         orientacion_=(orientacion_+1)%4;
131         break;
132     }
133
134     // Comprobacion para no salirme del rango del mapa
135     bool algo_va_mal=false;
136     if (y_<0){
137     }
138     else if (y_>199){
139     }
140     if (x_<0){
141     }
142     else if (x_>199){
143     }
144
145     if (algo_va_mal){
146
147     }
148
149     PasarVectoraMapaCaracteres(y_,x_,mapa_entorno_,VISTA_,orientacion_);
150     PasarVectoraMapaCaracteres(y_,x_,mapa_objetos_,SURFACE_,orientacion_);
151
152     env->ActualizarMatrizUsuario(mapa_entorno_);
153
154 }
```

En este caso, lo que hacemos es actualizar los valores de 'x\_', 'y\_' y orientación dependiendo de nuestra orientación y en función de la acción de movimiento aplicada. Adicionalmente, en las líneas de la 135 a la 155, se introduce una variable de control para avisarnos si nos estamos saliendo de los límites de la matriz.



```
165 // -----
166 Agent::ActionType Agent::Think()
167 {
168     Agent::ActionType accion = actFORWARD; // Por defecto avanza
169
170
171     // tomar accion
172
173
174     // recuerdo la ultima accion realizada
175     last_accion_ = accion;
176
177     return accion;
178
179 }
```

El método ‘Think’, como ya hemos dicho, implementa el comportamiento del agente. Debajo del comentario ‘// tomar accion’, y hasta antes del comentario ‘//recuerdo la última acción realizada’ se han incluido las reglas que rijan el comportamiento de nuestro agente. En la versión inicial, como se puede ver en la figura anterior, se ha implementado el comportamiento de avanzar siempre.

Como se puede observar, este método devuelve un valor de tipo ‘Agent::ActionType’. Este es un tipo de dato enumerado declarado en belkan.h, cuya definición aparece en la siguiente figura:

```
38 enum ActionType
39 {
40     actFORWARD, // avanzar
41     actTURN_L,  // Girar Izquierda
42     actTURN_R,  // Girar Derecha
43     actIDLE,    // No hacer nada
44     actPICKUP,  // Recoger un objeto
45     actPUTDOWN, // Soltar un objeto
46     actPUSH,    // Meter en la mochila
47     actPOP,     // Sacar de la mochila
48     actGIVE,    // Dar un objeto a un personaje
49     actTHROW    // Lanzar un objeto
50
51 };
```

#### 4.4. Algunos ejemplos para empezar a implementar el personaje

Ilustraremos, con algunos ejemplos cortos, la forma en la que se debe añadir conocimiento al agente.

##### 4.4.1. Limpiar el mapa tras un reinicio

Como ya se explicaba en la sección anterior, antes de que nuestro agente tome una decisión, es necesario actualizar el valor de las variables de estado con los nuevos valores que se reciben del entorno. Justo éste es el cometido de la función ‘ActualizarInformacion’.

Una de las situaciones que genera más distorsión visual y que puede llevar a graves errores si no se controla bien, es aquella en la que somos reiniciados en el juego. Como se contó con anterioridad, una vez que se produce esta circunstancia durante el juego, somos situados de nuevo en el mapa, pero con unas nuevas coordenadas y una nueva orientación. Por consiguiente, es como si se empezara de nuevo el juego.

Por esa razón, vamos a modificar la función ‘ActualizarInformacion’ para que limpie los mapas que va construyendo nuestro agente y nos deje como si estuviéramos en la situación inicial de juego.

El código que debe introducirse se muestra en el siguiente listado:

```
103 void Agent::ActualizarInformacion(Environment *env){
104     // Actualizar mi informacion interna
105     if (REINICIADO_){
106         // Lo que tengas que hacer si eres reposicionado en el juego
107         for (int i=0; i<200; i++){
108             for (int j=0; j<200; j++){
109                 mapa_entorno_[i][j]='?';
110                 mapa_objetos_[i][j]='?';
111             }
112             x_=99;
113             y_=99;
114             orientacion_=3;
115         }
```

En este código, se inicializan las dos matrices ‘mapa\_entorno\_’ y ‘mapa\_objetos\_’ con el carácter ‘?’, que se interpreta en el primer caso como desconocido y en el segundo caso como que no hay ningún objeto en esa posición. Obsérvese que la tercera matriz ‘mapa\_solucion\_’ no se toca, ya que es posible que durante la misma ejecución hayamos determinado ya valores para los que se conozca su verdadera ubicación en el mapa original.

Además, se les asigna a las coordenadas ‘x\_’ e ‘y\_’ la posición central de nuestras matrices (99,99) y se coloca la orientación como estaba en el constructor de la clase (en este caso, con el valor 3 que indica orientación Oeste).

#### 4.4.2. Incluyendo un comportamiento simple para evitar chocar I

En este segundo ejemplo vamos a mostrar un comportamiento simple para ilustrar la forma en la que se introduce el conocimiento en nuestro agente. Nos debemos situar en la función 'Think', que es la encargada de tomar las decisiones.

Supongamos que se desea introducir el siguiente comportamiento: “avanza mientras no encuentres obstáculos y, si encuentras un obstáculo, gira a la derecha”.

Después de haber jugado un rato con las teclas sabemos que, en principio, nuestro personaje no puede avanzar si tiene delante de él un árbol (que se codifica con una 'B'), un precipicio (codificado con la 'P'), un muro (codificado con la 'M') o una puerta (codificado con la 'D'). De momento, vamos a considerar que estos son los únicos obstáculos posibles (no es verdad, hay más y los deberéis descubrir).

Para concretar el comportamiento anterior plantearemos dos reglas:

1. *Si delante de mí hay un árbol, un precipicio, un muro o una puerta,*
  - 1.1. *Gira a la derecha.*
2. *Si no, avanza*

Para saber qué terreno tengo delante de mí, miro la componente 1 del vector 'VISTA\_'. Sabiendo esto, es fácil implementar el comportamiento anterior. En el siguiente listado se muestra el código necesario:

```
174 Agent::ActionType Agent::Think()  
175 {  
176     Agent::ActionType accion = actFORWARD; // Por defecto avanza  
177  
178  
179     // tomar accion  
180     if (VISTA_[1]=='B' or VISTA_[1]=='P' or VISTA_[1]=='A' or VISTA_[1]=='M' or VISTA_[1]=='D'){  
181         accion = actTURN_R;  
182         cout << "Regla 1: Cuando ve que choca, entonces gira\n";  
183     }  
184     else{  
185         accion = actFORWARD;  
186         cout << "Regla 2: Sigo avanzando\n";  
187     }  
188  
189     // recuerdo la ultima accion realizada  
190     last_accion_ = accion;  
191  
192     return accion;  
193 }  
194 }
```

#### 4.4.3. Incluyendo un comportamiento simple II



Cuando implementamos el comportamiento anterior y lo observamos sobre el simulador, nos damos cuenta de que nuestro personaje entra en un ciclo rápidamente y que dicho comportamiento no nos llevará a descubrir el contenido del mapa

Vamos a cambiar este comportamiento para hacerlo un poco más sofisticado. Supongamos que voy avanzando y me encuentro con un obstáculo. Podría pensar que dicho obstáculo es un elemento aislado dentro del mapa que no me permite avanzar pero que, si lo rodeo por uno de sus lados, podré seguir avanzando manteniendo la dirección inicial.

Dicho comportamiento se podría describir con el siguiente conjunto de reglas:

1. *Si delante de mí hay un árbol, un precipicio, un muro o una puerta,*
  - 1.1. *Gira a la derecha.*
2. *Si no: Si acabo de girar a la derecha*
  - 2.1. *Avanza.*
3. *Si no: Si acabo de avanzar*
  - 3.1. *Gira a la izquierda*
4. *En otro caso, avanza*

Si se analizan las reglas anteriores, se puede ver que se intenta evitar un obstáculo por la derecha mediante la composición de las acciones “girar a la derecha”, “avanzar” y “girar a la izquierda”. En un agente reactivo, que en cada iteración sólo puede realizar una acción, cuando se desean encadenar acciones, es necesario definir variables de estado que me sitúen en qué parte del encadenamiento me encuentro. Para el comportamiento anterior, se necesitan 2 variables de estado, una que nos dice “ya he girado a la derecha y me toca avanzar” y otra que nos dice “he avanzado y ahora me toca girar a la izquierda”. A la primera de ellas la llamaremos ‘primer\_paso’ y, a la segunda, ‘segundo\_paso’. Estas variables son de tipo booleano y se deben definir (como todas las variables de estado) en el fichero ‘belkan.h’.

El procedimiento de declaración de variables de estado siempre es igual. Me sitúo en el fichero ‘.h’, me voy a la parte de declaración de variables de instancia de la clase (en nuestro caso, al final del fichero ‘belkan.h’) y las declaro. En nuestro ejemplo, quedarían como

```
.....
string role;
bool primer_paso, segundo_paso;
```

Una vez declaradas las variables, nos vamos al constructor de la clase (en nuestro caso, al principio del fichero ‘belkan.h’) y las inicializamos:

```
.....
for (int i=0;i<100;i++)
    for(int j=0;j<100;j++)
        mapa_solucion_[i][j]='?';
primer_paso = false;
```



Segundo\_paso = false;

Terminado el proceso de declaración e inicialización, ya podemos volver al método 'Think' de 'belkan.cpp' para definir el comportamiento anterior, que se muestra a continuación:

```
187 Agent::ActionType Agent::Think()
188 {
189     Agent::ActionType accion = actFORWARD; // Por defecto avanza
190
191     // tomar accion
192     if (VISTA[1]=='B' or VISTA[1]=='P' or VISTA[1]=='M' or VISTA[1]=='D'){
193         accion = actTURN_R;
194         cout << "Regla 1: Cuando ve que choca, entonces gira\n";
195         primer_paso=true;
196         segundo_paso=false;
197     }
198     else if (primer_paso){
199         accion = actFORWARD;
200         cout << "Regla 3: Estoy en el primer paso para rodear el obstaculo\n";
201         primer_paso=false;
202         segundo_paso=true;
203     }
204     else if (segundo_paso){
205         accion = actTURN_L;
206         cout << "Regla 4: Con este paso termino de rodear el obstaculo\n";
207         segundo_paso=false;
208     }
209     else{
210         accion = actFORWARD;
211         cout << "Regla 2: Sigo avanzando\n";
212     }
213
214     // recuerdo la ultima accion realizada
215     last_accion_ = accion;
216
217     return accion;
218 }
219
220
221 }
```

Como se puede observar, las reglas 3 y 4 se activan cuando se dispara la regla 1 y lo hacen dando valor a las variables de estado que acabamos de definir. Es muy importante mantener actualizadas estas variables, ya que ellas por sí solas pueden lanzar reglas y, si no están bien controladas, pueden provocar el descontrol de nuestro agente.

Si ejecutáis el código anterior os daréis cuenta que el personaje también entra en un ciclo. Pensad en cómo extender este comportamiento para que se mejorara la capacidad de exploración de nuestro personaje.

Los dos últimos ejemplos tratan de ilustrar una forma de empezar a construir nuestro agente, pero los comportamientos que aquí se muestran son demasiado básicos como para que funcionen bien en mapas



de la complejidad con la que vamos a trabajar. Por eso, en condiciones normales, estos dos comportamientos no sirven de solución a ninguna parte de la práctica.

Como idea general, un buen comportamiento de exploración del mapa iría más en la línea del cuento de Pulgarcito. Para quien no lo recuerde, Pulgarcito era un personaje de cuento que se adentraba en un bosque y, para recordar el camino de vuelta, dejaba miguitas de pan conforme avanzaba. Pensad que, en este caso, en vez de usar pan, se van dejando señales que indican de alguna manera cuándo fue la última vez que pasamos por ahí. Profundizad en esta idea y que no os pase como al célebre personaje del cuento, que no encontró el camino de vuelta, ya que los pájaros se comieron el pan de iba dejando.

## 5. Método de evaluación y entrega de prácticas

### 5.1. Método de evaluación

A partir del comportamiento entregado por el alumno en los ficheros 'belkan.cpp' y 'belkan.h' se procederá a hacer una simulación sobre un nuevo mapa de tamaño 100x100, donde cada agente compite sólo (es decir, no hay otros jugadores, sólo los personajes definidos en el juego y tenéis que considerar que estos últimos 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 que cada estudiante obtendrá será:

a)  $\text{Nota1} = (\% \text{ de coincidencia con el mapa oculto}) * 7$

b)  $\text{Nota2} = (\text{puntuación obtenida durante el juego}) / (\text{Max. P.O.G}) * 3$

donde **Max. P.O.G.** representa la máxima puntuación obtenida por un estudiante en ese grupo de prácticas.

La nota final se calculará de la siguiente manera:

$$\text{NotaFinal} = (\text{Nota1} + \text{Nota2}) * \text{Grado\_Satisfaccion\_Defensa}$$

donde Grado\_Satisfaccion\_Defensa es un valor en [0,1] indicando en qué medida el alumno ha sabido defender los comportamientos implementados en su agente en 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 'belkan.cpp' y 'belkan.h') 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	Fecha defensa
A3, B3, C3 y D2	10 de Mayo hasta las 23:00 horas	17 de Mayo
A1, B1	11 de Mayo hasta las 23:00 horas	18 de Mayo
C1	13 de Mayo hasta las 23:00 horas	20 de Mayo
A2,B2,C2,D1	16 de Mayo hasta las 23:00 horas	23 de Mayo

*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.*



## ANEXO I: Lista de Mensaje que proporciona el entorno

A continuación se lista la mayoría (no están todos, pero estos son los relevantes) de los mensajes que son capturados por el sensor 'MENSAJE\_' de nuestro agente. Se describe la siguiente lista para facilitar al estudiante la verificación de que una determinada acción se realizó de forma correcta.

```
"Alguien te golpeo con una piedra. Quedas reiniciado. "  
"Caiste al agua. "  
"Menudo golpe contra el arbol. "  
"Este golpe ha sido fatal, quedas reiniciado. "  
"Por que te suicidas tirandote por el precipicio!!!. "  
"Intentando atravesar un muro, Copperfield?. "  
"Intentando atravesar una puerta cerrada, Eres un fantasma?. "  
"Has chocado con un obstaculo movil. "  
"Asi me gusta, que no te precipites!!!. "  
"Tienes que tener las manos libres para recoger un objeto. "  
"Recogido bikini. "  
"Recogida piedra. "  
"Recogidas llaves. "  
"Recogida rosa. "  
"Recogido lingote de oro. "  
"Recogido Oscar, no sabes la ilusion que me hace y lo quiero dedicar... "  
"Recogidas zapatillas. "  
"Recogida pala. "  
"Recogida manzana. "  
"Recogido algoritmo, no se como, pero lo he recogido. "  
"No tienes nada en las manos para soltar. "  
"Aqui no se puede soltar. Busca otro sitio. "  
"Soltado bikini."  
"Soltada piedra."  
"Soltada llaves."  
"Soltada rosa."  
"Soltado lingote de oro."  
"Soltado Oscar."  
"Soltada zapatillas."  
"Soltada pala."  
"Soltada manzana."  
"No tienes nada para guardar. "  
"No lo puedes guardar, la mochila esta llena. "  
"Guardado objeto en la mochila. "  
"Tienes las manos ocupadas!!!. "  
"Tienes la mochila vacia!!!. "  
"Usando bikini."  
"Usando piedra."  
"Usando llaves."  
"Usando rosa."  
"Usando lingote de oro."  
"Usando Oscar."  
"Usando zapatillas."  
"Usando pala."  
"Usando manzana."
```



ugr

Universidad de Granada

Departamento de Ciencias de la Computación  
e Inteligencia Artificial



"No tienes en la mano nada que dar. "  
"No hay nadie a quien darle eso. "  
"No le interesa lo que le quieres dar. "  
"Objeto Entregado. "  
"Muy bien, y te parece bonito lanzar aire; Muy original. "  
"Eres salvaje, eso no se puede lanzar. "  
"Se oye ruido de pajaros, le diste a un arbol. "  
"No destroces el mobiliario urbano!! ". "  
"No le tires piedras a mi puerta, gamberros!! ". "  
"Esto es intolerable; Quedas reiniciado. "  
"Le diste al oso!!! Buena Punteria; Consigues 100 puntos. "  
"Le diste a otro jugador!!! dice que ya te pillara; Consigues 50 puntos. "  
"No le diste a nada. "  
"Has encontrado un bikini, y parece de tu talla. "  
"Has encontrado una piedra, que suerte la mia!! ". "  
"Has encontrado una llave, que puerta abrira?. "  
"Has encontrado una rosa, ayyyyyyyyyyyyyy!! pincha. "  
"Has encontrado un lingote de oro, bien bien BIEN!!! ". "  
"Has encontrado un Oscar, no sabia que se pudiera encontrar esto!!! ". "  
"Has encontrado unas zapatillas, con sus cordones y todo!!! ". "  
"Has encontrado una pala, si, una pala de cavar; Para cavar estoy yo!! ". "  
"Has encontrado una manzana, y por poco no te dio en la cabeza Einstein. "  
"Has encontrado un algoritmo, UN ALGORITMO!!!, COMO UN ALGORITMO?. "  
"Hola Aldeano. "  
"Hola Princesa. "  
"Hombre!! Leonardo Di Caprio. "  
"Eres el principe valiente? "  
"Jod...!! Una bruja. "  
"Querido Profe!! ". "  
"Un oso!! pies para que os quiero!!! "  
"Menudo regalo de mie...!! Anda alejate de mi vista"  
"PK fila: XX columna: XX. "  
"Te han dado XX puntos"