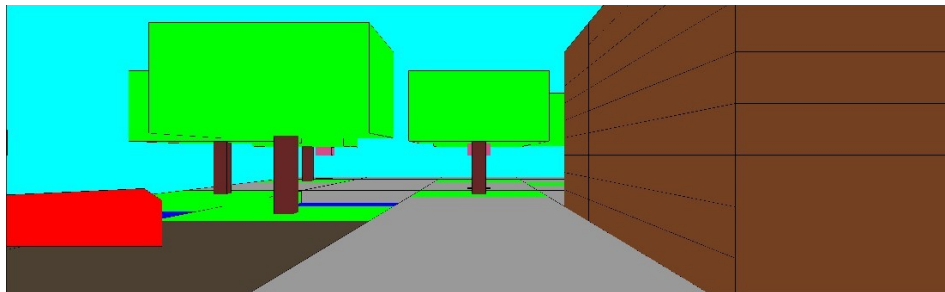


INTELIGENCIA ARTIFICIAL

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

Práctica 2



Agentes Reactivos/Deliberativos: los extraños mundos de BelKan

**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA
ARTIFICIAL
UNIVERSIDAD DE GRANADA
Curso 2017-2018**



1. Introducción

La segunda práctica de la asignatura *Inteligencia Artificial* consiste en el diseño e implementación de un agente deliberativo, capaz de percibir el ambiente y actuar considerando una representación de las consecuencias de sus acciones y siguiendo un proceso de búsqueda. 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.

1.1. Motivación

En esta práctica se diseñará e implementará un agente reactivo y deliberativo 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 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=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*.

En esta práctica, centraremos nuestros esfuerzos en implementar el comportamiento de un “personaje virtual” asumiendo un comportamiento reactivo y deliberativo. 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,


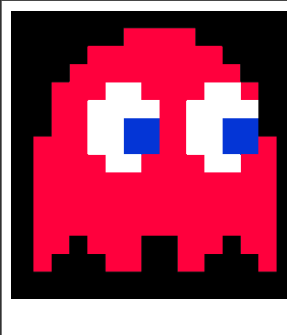
Departamento de Ciencias de la Computación e Inteligencia Artificial

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 sí 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, aunque aparentemente los 4 fantasmas presentan el mismo comportamiento, en realidad no es así, cada uno de ellos tiene su propia personalidad.

	<p>El fantasma amarillo se hace llamar 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.</p> <p>Es por ello que cuando fue denominado lento, no se refería a la velocidad con la que va por el laberinto, sino a lo lento y estúpido que podría ser tomando decisiones.</p> <p>Si nos fijamos en las direcciones que toma en cada cruce del laberinto, veremos que Clyde en realidad no persigue a Pacman. Quizás no sienta una especial aversión hacia él y no sea tan vengativo como los demás fantasmas, o, tal vez, sea tan estúpido que se pierde dando vueltas por los pasillos.</p>
	<p>El fantasma azul se hace llamar Inky. 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>Pinky es el fantasma rosa, también llamado Speedy (en inglés, rápido). Este fantasma es bastante metódico y, aunque tampoco es denominado rápido por su velocidad, es muy veloz tomando buenas decisiones.</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 Blinky y es conocido como Shadow (en inglés, Sombra). Es el más agresivo de los cuatro fantasmas y su nombre viene dado porque casi siempre es la sombra de Pacman, persiguiéndolo incansablemente.</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 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).

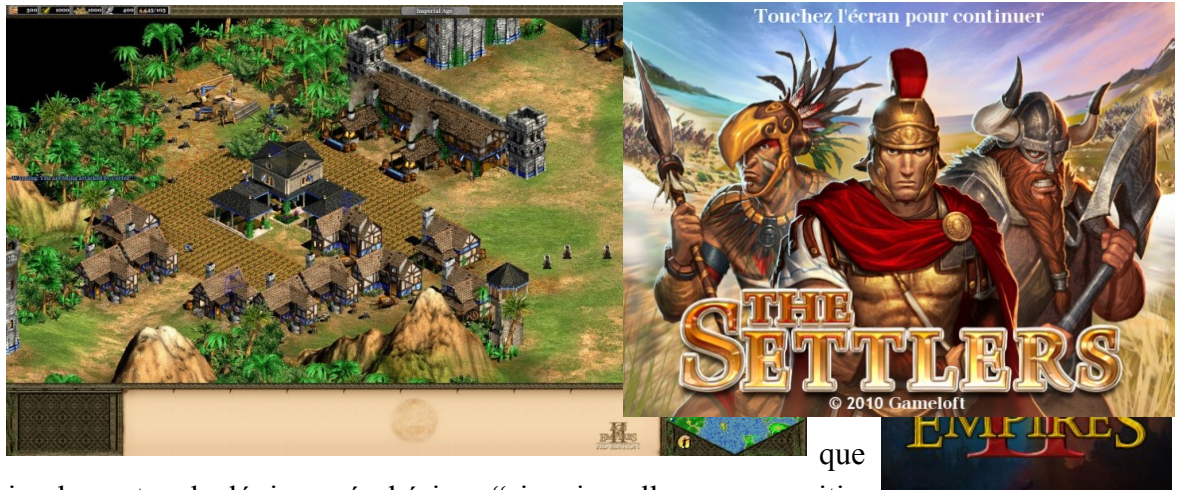
También estamos acostumbrados, y por eso uno puede llegar a pensar, que los agentes reactivos estén vinculados con el comportamiento de los personajes secundarios de los juegos. Pero no es así, especialmente en los juegos de estrategia en tiempo real. Un juego que es algo antiguo, pero es claro ejemplo 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

Departamento de Ciencias de la Computación e Inteligencia Artificial

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 qué 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/deliberativos.

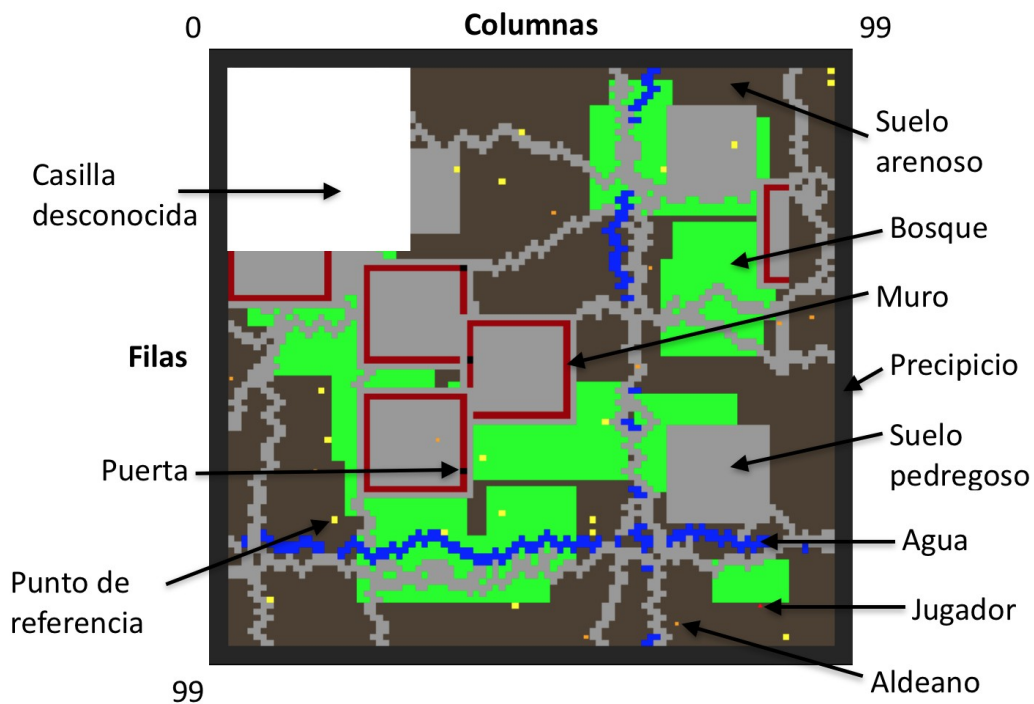
2.1. El escenario de juego

Este juego se desarrolla sobre un mapa cuadrado bidimensional discreto que contiene como máximo 100 filas y 100 columnas. El mapa representa los accidentes geográficos de la superficie de 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 donde la primera componente representa la fila y la segunda representa la columna dentro de nuestro mapa. Como ejemplo usaremos un mapa de tamaño 100x100 de caracteres. Fijaremos sobre este mapa las siguientes consideraciones:

- La casilla superior izquierda del mapa es la casilla [0][0].
- La casilla inferior derecha del mapa es la casilla [99][99].

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



Los elementos permanentes en el terreno son los siguientes:

- *Árboles o Bosque*, 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. Nuestro personaje no sabe nadar... con lo cual si cae en una de estas casillas morirá.

- *Precipicios*, que se codifica con el carácter ‘P’ y tiene asociado el color negro. Nuestro personaje no sabe volar... con lo cual si cae en una de estas casillas morirá.
- *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. Esto no quiere decir que no hay precipicios en el resto del mapa.

Sobre esta superficie pueden existir elementos que tienen la capacidad de moverse por sí mismos. Estos elementos son:

- *Jugador*, se codifica con el carácter ‘j’ y se muestra como un triángulo rojo. Éste es nuestro personaje, sólo habrá un jugador a la vez.
- *Aldeano*, se codifican con el carácter ‘a’ y se muestra como un cuadrado naranja. Son habitantes anónimos del mundo que se desplazan a través del mapa sin un cometido específico, simplemente intentan molestarnos en nuestros movimientos. Son sólo molestos, no son peligrosos.

2.2. Nuestro Personaje

Obviamente nuestro personaje es el protagonista de la historia y debe llevar a cabo su **objetivo: ir de un punto específico en el mapa (origen) a otro punto específico en el mapa (destino)**. 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

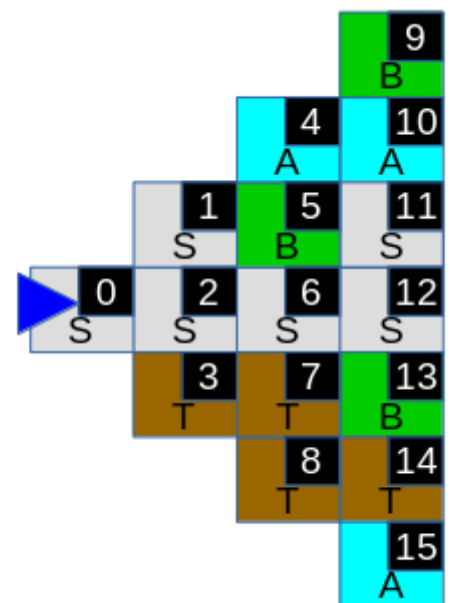
que es raro. Yo sospecho, sólo sospecho, que viene de alguna versión no comercializada del “*Trivial*”. A propósito, ¿recuerdas a qué categoría se asociaba el quesito rojo en este juego? Es oír Trivial y te salen las preguntas de forma inconsciente...

2.2.1. Sensores del 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 del 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.



De igual manera se estructura el vector **superficie** pero, en este caso, indicando qué objetos móviles se encuentran en cada una de esas casillas.

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

- **Sensor de choque (colision)**: Es una variable booleana que se podrá en verdadero en caso que la última acción del jugador haya ocasionado un choque con un árbol, muro o puerta.
- **Sensor de vida (reset)**: Es una variable booleana que se podrá en verdadero en caso que la última acción del jugador le haya llevado a su muerte... por ejemplo, se ha precipitado al vacío en un precipicio o se ha ahogado en el agua. No os pongáis tristes, ya que morir en el juego implica que se vuelve a reaparecer en el mismo mundo, con una nueva posición pero siempre mirando al norte.
- **Sensores de mensaje (mensajeF, mensajeC)**: En algunas ocasiones se le enviarán al jugador mensajes a través de estos dos sensores de tipo entero (ej: mensajeF = 10, mensajeC = 55), como ser su posición en el mapa (más detalles en la siguiente sección).
- **Sensor de destino (destinoF, destinoC)**: En todo momento, la posición del destino se puede averiguar a través de estos dos sensores de tipo entero. ¡Atención! Las coordenadas de destino se pueden cambiar por la interfaz del usuario (pinchando en algún lugar del mapa o cambiando a mano los valores de Fila y Columna en Ir a..., ver captura de pantalla en la sección 4.2.1). Esto no sucede en la versión sin interfaz.

2.2.2. Datos del personaje compartidos con el entorno

Dentro de la definición del agente hay una matriz llamada **mapaResultado** en donde se puede interactuar con el mapa. Todo cambio en esta matriz se verá reflejado automáticamente en la interfaz gráfica.

2.2.3. Acciones que puede realizar el personaje

Nuestro personaje puede realizar varias acciones distintas durante el juego:

- *actFORWARD*: le permite avanzar a la siguiente casilla del mapa siguiendo su orientación actual. Para que la operación se finalice con éxito es necesario que la casilla de destino sea transitable para nuestro personaje.
- *actTURN_L*: le permite mantenerse en la misma casilla y girar a la izquierda 90° teniendo en cuenta su orientación.
- *actTURN_R*: le permite mantenerse en la misma casilla y girar a la derecha 90° teniendo en cuenta su orientación.
- *actIDLE*: pues como su nombre indica, no hace nada.

3. Objetivo de la práctica

El objetivo de esta práctica es dotar de un comportamiento inteligente a nuestro personaje usando un agente reactivo/deliberativo para definir las habilidades que le permitan dentro del juego según el nivel seleccionado.

Para que sea más fácil resolver esta práctica, se han diseñado tres niveles con dificultad incremental.

Nivel 1: Agente deliberativo en un isla desierta

Nuestro personaje se encuentra en una isla desierta, es decir, solo existe él y nadie más (no hay aldeanos). Su misión es ir a un punto destino del mapa. En este nivel nuestro personaje tiene un mapa completo de la isla y conoce todos sus recovecos. Nuestro personaje no puede atravesar bosques, ni agua, ni muros, ni puertas, ni precipicios. Por tanto, deberá esquivar estos tipos de terreno ya que sino chocará (en el caso de los muros, puertas y árboles) o morirá (en caso de agua o precipicio).

Nuestro personaje aparecerá de forma aleatoria sobre un mundo de BelKan concreto (la isla desierta), el cual no cambiará durante el juego, conociendo su posición (a través de los sensores **mensajeF** y **mensajeC** que se activa cuando el jugador debe decidir su primera acción) y orientación sobre el mapa (siempre mira al norte al iniciar). Su objetivo es crear y llevar a cabo un plan de movimientos en el mapa (agente

deliberativo) para llegar con seguridad desde el origen al destino en una cantidad de acciones finita (máx. 500) o 300 segundos totales (lo que suceda antes). ¡No debe morir injustificadamente ni chocarse con nada! Para ello es obligatorio implementar alguno de los algoritmos vistos en clase.

Nivel 2: Agente reactivo/deliberativo en un poblado

Al igual que en el nivel anterior, conocemos la isla de “pe” a “pa”. En este nivel la cosa se complica un poco: ya no estamos solos... En esta isla hay aldeanos que se pasean como si estuvieran en su casa sin un destino fijo. Un aldeano puede moverse continuamente, puede quedarse quieto, etc. Es decir, para ir de un punto a otro del mapa puede que nuestro plan inicial no funcione correctamente ya que nos crucemos en el camino con un aldeano que nos entorpezca... En ese caso deberemos integrar comportamientos reactivo y deliberativo, de manera que el agente debe encontrar un plan de navegación (comportamiento deliberativo) y, durante su ejecución, el agente debe considerar que otros objetos móviles (como aldeanos) pueden hacer que falle la ejecución del plan.

Los aldeanos pueden detectarse gracias a un sensor que tiene nuestro personaje, es decir, no podemos saber donde están los aldeanos en el mapa, pero si podremos verlos con nuestro sensor de superficie, si estamos cerca de ellos. Para optar por este nivel es necesario tener hecho el nivel anterior. Para este nivel también se cuenta con un número de 500 acciones como máximo o 300 segundos totales (lo que suceda antes). También es importante recordar que el jugador no debe morir injustificadamente ni chocarse con nada ni nadie.

Nivel 3: Agente reactivo/deliberativo haciendo turismo aventura

En este nivel la cosa se complica, ya que no nos han dado un mapa de la isla y ¡no tenemos idea de donde estamos! Eso si, nos dicen que miramos al norte, algo es algo. Deberemos ir descubriendo el mapa poco a poco. Pero para empezar debemos saber en que fila y columna estamos. Para ello debemos recorrer el mapa hasta encontrar una casilla PK (punto de referencia) que nos indicara la posición real en fila y columna y desde ese momento podremos ir construyendo nuestro propio mapa a medida que nos dirigimos al destino. Para optar por este nivel es necesario tener hecho el nivel anterior.



En este nivel cada vez que lleguemos al destino, éste cambiará de sitio. Es decir, deberemos tratar de conseguir llegar a la mayor cantidad de destinos posible utilizando como máximo 2000 acciones o 300 segundos totales (lo que suceda antes). También es importante recordar que el jugador no debe morir injustificadamente ni chocarse con nada ni nadie.

4. El Software

Para la realización de la práctica se proporciona al alumno una implementación tanto del entorno simulado del mundo en donde se mueve nuestro personaje como de la estructura básica del agente reactivo/deliberativo.

4.1. Instalación

Se 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 PRADO. 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 existe un fichero 'install.sh' que instala los paquetes necesario y compila el código. Luego se ejecuta 'make' para compilar cada vez que se edite el código.

Hay un elemento adicional 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). Es necesario tener instalada la librería 'freeglut3'.

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 Programing 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.”

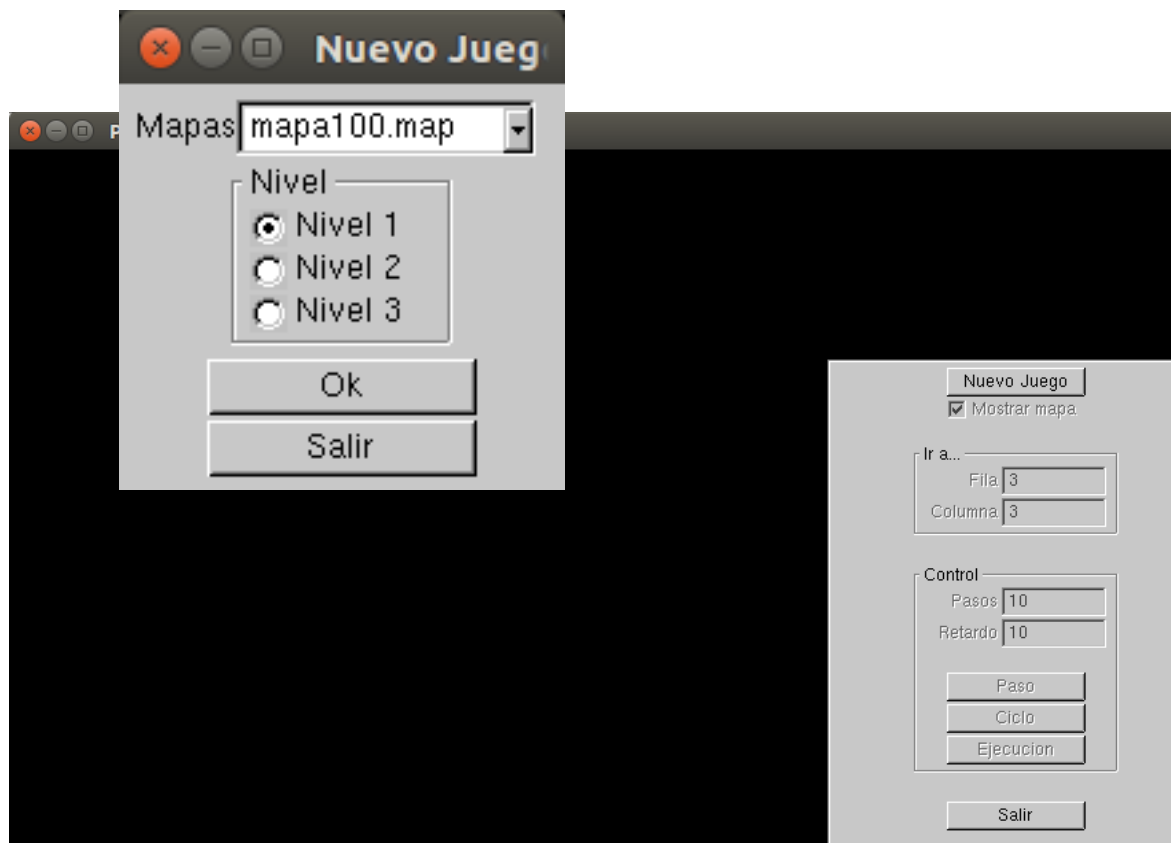
Y de aquí proviene la razón de que no os funcione bien del todo. Si vuestra tarjeta gráfica está diseñada específicamente para Direct3D (que es la que usan la mayoría de los juegos de ordenador), trabajará con OpenGL en modo simulación y, a veces, esa simulación no es tan buena. Un síntoma de esto suelen 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.

4.2. Funcionamiento del Programa

Existen dos ficheros ejecutables: Belkan y BelkanSG. El primero corresponde al simulador con interfaz gráfica, mientras que el segundo es un simulador en modo *batch* sin interfaz.

4.2.1. Interfaz gráfica

Para ejecutar el simulador con interfaz hay que escribir “./Belkan”. Se tiene la posibilidad de cambiar la semilla del generador de números seguido de un número. Por



ejemplo, “./Belkan 1”.

Al arrancar el programa, nos mostrará una ventana como la que se muestra en la figura anterior, que es la ventana principal. Para iniciar el programa se debe elegir el botón “Nuevo Juego” que abriría la siguiente ventana:

En esta nueva ventana se puede elegir el mapa con el cual trabajar (debe estar dentro de la carpeta “mapas”) y el nivel deseado. Seleccionaremos el “Nivel 1” como ejemplo con el mapa “mapa30.map”, y presionaremos el botón de “Ok”.



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

Los botones ‘Paso’, ‘Ciclo’ y ‘Ejecución’ son las tres variantes que permite el simulador para avanzar en la simulación. El primero de ellos, ‘Paso’, invoca al agente que se haya definido y devuelve la siguiente acción. El botón, ‘Ciclo’ realiza varios pasos con un retardo especificado (en décimas de segundo). Por último el botón ‘Ejecución’ realiza una ejecución completa de la simulación.

4.2.2. Sistema *batch*

Para ejecutar el simulador con sin interfaz gráfica hay que escribir “./BelkanSG” seguido de una serie de parámetros:

- fichero de mapa
- semilla para inicializar el generador de números aleatorios
- nivel (1, 2 o 3)
- fila origen
- columna origen
- pares de (fila, columna) destino

Por ejemplo podemos ejecutar “./BelkanSG mapas/mapa100.map 1 3 4 4 3 3 5 2” lo cual utilizará el mapa indicado con una semilla 1 en el nivel 3 e intentará enviar el jugador desde la posición (4,4) hasta la posición (3,3) y luego la (5,2). En caso de quedarse sin destinos, los elegirá al azar.

Al finalizar la ejecución nos el tiempo consumido, la cantidad de fallos que hemos cometido al descubrir el mapa, el número de colisiones que hemos sufrido, la cantidad de muertes por caídas (al agua o a algún precipicio) y la cantidad destinos alcanzados.

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.hpp’ y ‘jugador.cpp’. Estos archivos contendrán los comportamientos implementados para nuestro agente deliberativo/reactivo. Además, dentro de estos dos archivos, no se puede eliminar nada de lo que hay originalmente, únicamente se puede añadir. Para aclarar esto mejor, pasamos a ver con un poco más de detalle cada uno de estos archivos.

Empecemos con el archivo ‘jugador.hpp’, sin nos vamos a la parte inferior del código vemos la declaración de los datos privados. Tiene declaradas varias variables de estado:

Departamento de Ciencias de la Computación e Inteligencia Artificial

- `fil, col`: posición en fila y columna donde está posicionado el jugador. Se inicializa en 99 tanto `fil` como `col`.
- `brujula`: orientación del jugador (0 -> norte, 1 -> este, 2 -> sur, 3 -> oeste). Se inicializa en 0.
- `destino`: datos sobre la fila, columna y orientación del objetivo. Se inicializa en -1 todos los campos de su estructura.
- `plan`: lista de acciones que determinan un plan de movimientos. Se inicializa en una lista vacía.

El estudiante puede agregar tantas variables de estado como crea convenientes.

Como ya vimos, existe una variable `mapaResultado` que se incluye a partir del fichero `'comportamiento.hpp'` en donde se encuentra el mapa. En los dos primeros niveles esta variable se utilizaría básicamente como si fuera de solo lectura, mientras que en el nivel 3 la matriz viene inicializada con '?' y se debe ir completando a medida que se descubre el mapa (al llenarla se irá automáticamente dibujando en la interfaz gráfica).

En el archivo `'jugador.cpp'` se describe el comportamiento del agente. En este archivo podemos añadir tantas funciones como necesitemos, pero las únicas funciones que se pueden modificar, y sólo para añadir código, son `'pathFinding'` y `'think'`. El método `'think'` se dedica a definir las reglas que regirán su comportamiento. Este método implementa el comportamiento del agente y, como se puede observar, devuelve un valor de tipo `'Action'`. Este es un tipo de dato enumerado declarado en `'comportamiento.hpp'` que puede tomar los valores `{actFORWARD, actTURN_L, actTURN_R, actIDLE}` como ya hemos comentado anteriormente.

5. Método de evaluación y entrega de prácticas

5.1. Método de evaluación

A partir del comportamiento entregado por el alumno en los ficheros antes mencionados y del nivel elegido por el mismo, se realizarán varias simulaciones con diferentes mapas (de diferente tamaño, pero con un máximo de 100x100) y con diferentes destinos, tanto alcanzables como no alcanzables (por ejemplo, una posición ocupada por un árbol). El simulador no debe "colgarse" en ningún momento y debe, por supuesto, compilar sin problemas.



Se valorará la cantidad de destinos alcanzados en cada simulación y restará puntos cualquier colisión o muerte innecesaria.

La nota final se calculará de la siguiente manera:

$$\text{Nota final} = \text{Nota Practica} * \text{Defensa}$$

donde “Defensa” es un valor en [0,1] indicando en que medida el alumno ha sabido defender los comportamientos implementados en su agente en el proceso de defensa.

- Si se opta por el Nivel 1 entonces la nota máxima alcanzable será de 3.
- Si se opta por el Nivel 2, la nota máxima alcanzable será de 4.
- Si se opta por el Nivel 3 la nota máxima alcanzable será de 6 (si se logra al menos llegar a un destino) y hasta 10 dependiendo de la cantidad de destinos alcanzados.

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 dos ficheros deberán entregarse mediante la plataforma web de la asignatura, en un fichero ZIP que no contenga carpetas ni subcarpetas. El archivo ZIP deberá contener sólo el código fuente de estos dos ficheros con la solución del alumno así como un fichero de documentación en formato PDF que describa el comportamiento implementado con un máximo de 5 páginas.

No se evaluarán aquellas prácticas que contengan ficheros ejecutables o virus

5.3. 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
TODOS	22 de Abril hasta las 23:00 horas	Semana del 30 de Abril



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.

5.4. Observaciones Finales

Esta **práctica es INDIVIDUAL**. El profesorado para asegurar la originalidad de cada una de las entregas, someterá a estas a un procedimiento de detección de copias. En el caso de detectar prácticas copiadas, los involucrados (tanto el que se copió como el que se ha dejado copiar) tendrán suspensa la asignatura. Por esta razón, recomendamos que en ningún caso se intercambie código entre los alumnos. No servirá como justificación del parecido entre dos prácticas el argumento “*es que la hemos hecho juntos y por eso son tan parecidas*”, o “*como estudiamos juntos, pues...*”, ya que como se ha dicho antes, **las prácticas son INDIVIDUALES**.

Como se ha comentado previamente, el objetivo de la defensa de prácticas es evaluar la capacidad del alumno para enfrentarse a este problema. Por consiguiente, se asume que todo el código que aparece en su práctica ha sido introducido por él por alguna razón y que dicho alumno domina perfectamente el código que entrega. Así, si durante cualquier momento del proceso de defensa el alumno no justifica adecuadamente algo de lo que aparece en su código, la práctica se considerará copiada y tendrá suspensa la asignatura. Por esta razón, aconsejamos que el alumno no incluya nada en su código que no sea capaz de explicar qué misión cumple dentro de su práctica y que revise el código con anterioridad a la defensa de prácticas.

Por último, las prácticas presentadas en tiempo y forma, pero no defendidas por el alumno, se considerarán como no entregadas y el alumno obtendrá la calificación de 0. El supuesto anterior se aplica a aquellas prácticas no involucradas en un proceso de copia. En este último caso, el alumno tendrá suspensa la asignatura.