# Inteligencia Artifical: Práctica 1

**Daniel Alconchel Vázquez** **Grupo 1**

La idea que he utilizado es avanzar en la dirección donde existan más casillas **'?'** en mapaResultado.

**Aclaración:**

Debido a que he cambiado de idea varias veces (la verdad, se me ha antragantado bastante la práctica) el código está bastante desordenado, e, incluso, hay variables inutilizadas.

## 1. Antes de decidir movimiento

En cada llamada al método **think**, procesa el movimiento anterior, para asi actualizar las variables de fila y columna, la orientación, o, incluso, si ha ocurrido un **reset**, restaurar el valor inicial de algunas variables.

***Línea 8 - Línea107***

## 2. Procesado del mapa

Si está bien situado, es decir, está en nivel 0 o ha pasado por una casilla de posicionamiento, entonces, en cada iteración, añade a MapaResultado lo que encuentra en su rango de visión (Las 16 componentes del vector).

En caso de no estar ubicado, utilizo una matriz auxiliar interna, inicializada al doble de tamaño que el mapa original y en la que nuestro personaje aparece internamente representado en las coordenadas (MapaResultado.size(), MapaResultado.size()), mirando al norte (esto permite evitar cores).

Una vez se ubica, realizo una rotación y una traslación para solapar todo lo que ha ido anotando en este mapa interno en nuestro mapa resultado.

***Línea 123 - Línea 137***

## 3. Casillas de Interés

Consideramos casillas de interés las de posicionamiento (si no estamos ubicados), las de zapato o bikini (si no disponemos de ellos) y las de recarga.

Nuestro personaje irá avanzando por el mapa, con un movimiento que explicaremos más adelante. Detendrá dichas elecciones en el momento que ve algo interesante en su rango de visión. Para ello usamos la función:

```
void ComprobarVision(Sensores sensores);
```

Esta función marca que casilla interesante estamos viendo y la posición en la que se encuentra. Para planear como llegar hasta dicha casilla, usaremos la función:

```
void BuscarInteres(int posicion);
```

La cual, añade una seria de acciones a un vector de acciones, para ser realizadas en las siguientes iteraciones. Es importante destacar que en caso de ver dos o más casillas de interés a la vez, prioriza siempre posicionamiento (si no está bien posicionado), bikini y zapatillas (si no las tiene), y ya, por último, recarga.

## 4. Movimiento Principal

Este es el algoritmo que más problemas me ha traido, por lo que está 0 optimizado de tantos errores cometidos, e, incluso, tiene variables un tanto inservibles.

Pese a que existe una función de movimiento para cuando está ubicado y otra para cuando no, la idea es la misma.

```
Action MovimientoNoUbicado(Sensores sensores);
Action MovimientoUbicado(Sensores sensores);
```

Eligen avanzar en la dirección menos explorada. Para ello, realizan un barrido hacia el norte, sur, este y oeste de la porsición del personaje, con un ancho de 7 (que es el máximo rango de visión que tiene nuestro "robot").

Como defecto del cambio de código, de ir solventando los errores que surgian, etc... En caso de que se use MovimientoUbicado, el personaje va apuntando en una matríz numérica cuántas veces ha pasado por cada zona, como si un mapa de calor se tratase. Esto era porque, originalmente, intenté realizar un cálculo de probabilidades, en la que la toma de decisiones fuera directamente proporcional al número de 0's que hay en una dirección (casillas no

visitadas) e inversamente proporcional al número de veces que ha pasado por una serie de casillas (casillas con un valor numérico distinto de 0, que indica cuántas veces se ha pasado por una zona).

Como no conseguí realizarlo correctamente (pese a la gran cantidad de horas intentando arreglarlo e, incluso, con alguna sugerencia de algún comapañero), tuve que descartar la idea, simplificarla y de ahí que el código resultante no sea tan eficiente y estructurado como uno desearía.

Continuando con la explicación, como hemos dicho, avanza en la dirección donde se encuentren mayor cantidad de **'?'**. En el momento que se choca (consideraremos chocarse como encontrarse con un muro o precipicio, un árbol y no tener zapatillas o agua y no tener bikini), entonces, revalua que dirección tomas y se dirige hacia la misma. En dicho cálculo intervienen los métodos:

```
void Dividircargas();
int ElegirRegion();
void ResetearRegion();
void Girar(int region);
void CalcularTendencia();
void Dividircargas_Interna();
```

Ahora, tenemos que distinguir dos casos:

- Si se choca con un árbol (y no tiene zapatillas) o agua (y no tiene bikini), puede ocurrir que decida avanzar en la misma dirección. En ese caso, escoge la segunda región más interesante.
- Si se choca con un muro o precipicio y considera que es óptimo seguir abanzando en la misma dirección, entonces, se pone a seguirlo, buscando una apertura:
    - ***Línea 600-640***
    - ***Línea 561-571***

Inicialmente, elige girar a un lado. A continuación, avanzará. Si se choca con algo que no es un muro (o precipicio), para de seguir el muro (o precipicio). En otro caso, vuelve a girar en la misma dirección que ha elegido inicialmente.

Por último, de este método comentar que también considera "chocarse" como que todas las casillas que vería si avanza una posición ya están exploradas.

# 5. Aldeanos y Lobos

En caso de toparse en frente suya con un aldeano o lobo, simplemente resetea las variables necesarias y gira hacia uno de los lados, de forma aleatoria.

## 6. Completado

Hay una función que completa las casillas no vistas, teniendo en cuenta las que les rodea. Esta función hay casos en los que es bastante "extremista", ya que, por ejemplo, mi algoritmo hace que en vértigo saque muy poco porcentaje o incluso nulo, pero al completar por defectos con precipicios, pues acaba pasando el mapa con un 40% o más.

```
unsigned char suponer(int fil, int col);
void InferirCasilla();
```

Se puede desactivar esta funcionalidad comentando las líneas ***210-213***

## 7. Comentarios

El código tiene bastantes desperfectos en determinadas situaciones. Por ejemplo, si por alguna razón se mete en una situación en la que todas las casillas de delante de su campo de visión (en todas las direcciones) están exploradas, se mete en un bucle en el que cambia de dirección constantemente.

Otro desperfecto es que no he optimizado las recargas y, por último, puede ocurrir que al seguir un muro o precipicio (como elige una dirección en la que girar todo el rato) se meta también en algún bucle, porque tiene un obstáculo pegado al mismo y gira dos veces, volviendo a la esquina original y repitiendo el proceso indefinidamente.