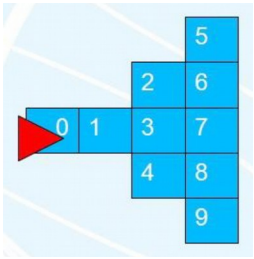
1. Introducción.

El movimiento de mi agente se podría decir que está dividido en varias partes; una parte destinada a la obtención de dos puntos kilométricos, una parte destinada a la recolección de objetos valiosos, otra parte destinada a la entrega de dichos objetos y una última parte destinada a moverse por el mapa.

2. Obtención de puntos kilométricos.

Para esta tarea, me he basado principalmente en el sensor VISTA_ de nuestro agente. Este sensor es un vector que representa la visión de la siguiente manera:



Una vez ha entrado un punto kilométrico en la vista del agente, es cuando comienza nuestra tarea de ir hacia el. Esta tarea la podemos dividir en 3 subtareas:
- Cuando el punto kilométrico esta en la casilla 1, 3 o 7.
- Cuando el punto kilométrico esta en la casilla 2, 6 o 5.
- Cuando el punto kilométrico esta en la casilla 4, 8 o 9.

Los movimientos a realizar están guiados por variables, pero también están influenciados por el sensor VISTA_ del agente, ya que, aunque haya que realizar una serie de movimientos para llegar al PK, si hay obstáculos estos movimientos no se pueden realizar.

3. Obtención de objetos.

Al igual que hemos hecho en el caso de los Pks haremos lo mismo en el caso de los objetos. En cuanto uno de ellos entre en nuestro campo de visión, iremos a por el. Pero con una condición, ese objeto debe ser uno de los que dan puntuación en el juego, es decir, el lingote, el Óscar de la academia, la manzana o los algoritmos.
Dado que la mochila tiene una capacidad limitada, para esta tarea junto con la siguiente, llevaremos un contador indicando el número de de objetos que tenemos.

4. Entrega de objetos.

Una vez más, al igual que hemos hecho en los dos casos anteriores, esta tarea comienza cuando tenemos un personaje en nuestro campo de visión y, además, tenemos algún objeto que entregar. También, esta tarea está guiadas por variables, pero también por lo que vemos en el sensor VISTA_. Al igual que sucede en la obtención de objetos, no todos los personajes nos interesan, ya que no todos nos dan puntuación. Por ello, sólo iremos a por un personaje si este es una princesa, Leonado, un princeso o el profe de IA.

5. Moverse por el mapa.

Para moverse por el mapa, en un principio pensé en implementar una versión de pulgarcio (versión que finalmente he implementado) la cuál consistía en llevar un contador en cada casilla que indicase el número de veces que he pasado por ella, y en cada decisión de hacia donde moverme, lo que hacía era elegir la casilla que menor peso (número de veces) tenía. Esta idea funcionaba, pero tenía algunos problemas de implementación, ya que en determinadas situaciones, el agente se quedaba "pillado" en una zona y no podría moverse. Por ello, cambié de estrategia.

Finalmente he optado por implementar un estrategia basada en la idea de pulgarcito, es decir, llevar un contador para cada casilla que indique en que iteración he pasado por dicha casilla. Para ello he usado una matriz como la que sigue:

```
int pulgarcito[200][200];
```

Para poder movernos, en cada momento, realizo tres pasos:
- Percepción del entorno.
- Decisión del movimiento.
- Actualización de información.

5.1. Percepción del entorno.

El mapa entorno es siempre el mismo independientemente de la orientación. Pero la percepción del mismo no. Veamos esto con más detalle. En mi caso, vamos a decidir que movimiento hacer entre girar a la derecha, girar a la izquierda o seguir hacia adelante. Para ello, debemos obtener el valor de las casillas situadas a la derecha, a la izquierda y al frente de nuestro agente para, en una etapa posterior, poder decidir que acción realizar.

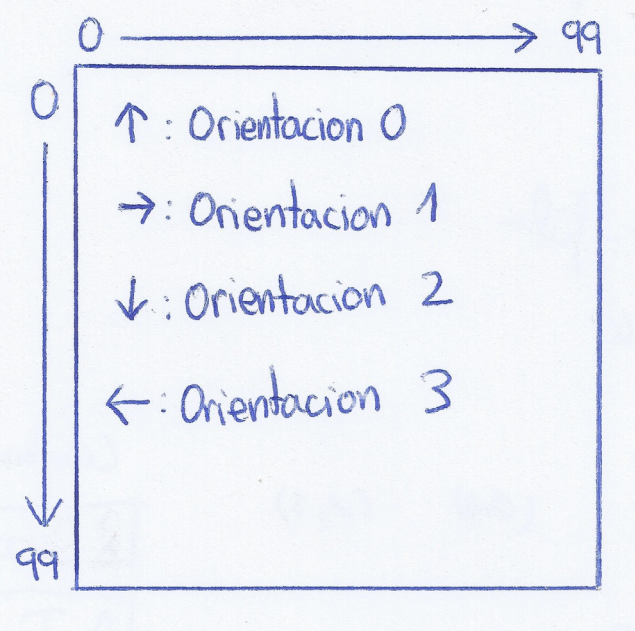
El calculo de las casillas sigue la misma metodología independientemente de la orientación, el pseudocódigo sería el que sigue:

```
if((mapa_entorno_[izquierda] es bosque, agua, precipicio, muro o puerta) ||
(mapa_objetos_[izquierda] es algun personaje)
   izquierda = 30000;
else if (mapa_entorno_[izquierda] es ? && mapa_objetos_[izquierda] es ?)
   izquierda = 0;
else
   izquierda = pulgarcito de la izquierda del agente;

if((mapa_entorno_[derecha] es bosque, agua, precipicio, muro o puerta) ||
(mapa_objetos_[derecha] es algun personaje)
   derecha = 30000;
else if (mapa_entorno_[derecha] es ? && mapa_objetos_[derecha] es ?)
   derecha = 0;
else
   derecha = pulgarcito de la derecha del agente;

if((mapa_entorno_[frente] es bosque, agua, precipicio, muro o puerta) ||
(mapa_objetos_[frente] es algun personaje)
   frente = 30000;
else if (mapa_entorno_[frente] es ? && mapa_objetos_[frente] es ?)
   frente = 0;
else
   frente = pulgarcito del frente del agente;
```

En el caso de que no podamos movernos a esa casilla (primer if), le asignamos un valor alto para que no sea elegido a la hora de decidir que movimiento realizar. Sin embargo, si no hemos visitado esa casilla, le asignamos 0, un valor muy bajo para facilitar el hecho de que sea visitada. En cualquier otro caso, simplemente obtenemos el valor de pulgarcito para dicha casilla.

El problema es como determinar que casilla debo mirar, ya que la izquierda, la derecha y el frente del agente varía según su orientación.

Para ver esta parte más claramente nos vamos a apoyar en la siguiente imagen:



De una forma muy abstracta tenemos que:

```
if(orientacion == 0){
   izquierda = m[y_][x_-1]
   derecha = m[y_][x_+1]
   frente = m[y_-1][x_]
}
else if(orientacion == 1){
   izquierda = m[y_-1][x_]
   derecha = m[y_+1][x_]
   frente = m[y_][x_+1]
}
else if(orientacion == 2){
   izquierda = m[y_][x_+1]
   derecha = m[y_][x_-1]
   frente = m[y_+1][x_]
}
else if(orientacion == 3){
   izquierda = m[y_+1][x_]
   derecha = m[y_-1][x_]
   frente = m[y_][x_-1]
}
```

donde m representa la matriz correspondiente, teniendo en cuenta el pseudocódigo asociado al calculo de izquierda, derecha y frente descrito con anterioridad.

5.2. Decisión del movimiento.

Una vez hemos calculado el valor que tiene cada casilla a la cual nos podemos mover. Dado que en las variables izquierda, derecha y frente tenemos asignado el peso de cada casilla, lo más lógico es moverse a la casilla cuyo peso se menor. En base a esta idea, realizaremos la toma de decisiones.

5.3. Actualización de información.

Una vez hemos realizado el movimiento, debemos actualizar el valor de pulgarcito para la nueva casilla. Pero, ¿y si no hay nueva casilla? Es decir, solo vamos a actualizar el valor de pulgarcito si el movimiento elegido es ir al frente. En caso de que sea un giro, como estamos en la misma casilla, mantendremos el valor que tenía pulgarcito para esa casilla.

6. Solución.

Una vez realizadas las tareas anteriores, sólo nos queda guardar lo que nuestro agente ha descubierto en el mapa solución. Para ello nos orientaremos con los puntos kilométricos que hemos obtenido en el punto 2 de esta memoria. A la vez que guardabamos los puntos kilométricos reales, guardábamos las coordenadas de nuestro agente en dichos puntos. A continuación hay que determinar cual es la orientación real de nuestro mapa, para ello calcularemos 4 valores y en base a ellos determinaremos la orientación correcta:

```
f1 = PK1_fila - PK2_fila;
f2 = myPK1_fila - myPK2_fila;
c1 = PK1_columna - PK2_columna;
c2 = myPK1_columna – myPK2_columna;

if(f1 == f2)
   Estamos orientados.
else if (f1 == -f2)
   Estamos al reves.
else if (f1 == c2)
   Girar 90º a la derecha.
else if (f1 == -c2)
   Girar 90º a la izquierda.
```

Una vez determinada la orientación, sólo hay que girar la matriz mapa_entorno_ y guardarla en mapa_solucion_ de forma correcta, es decir, teniendo en cuenta los puntos kilométricos reales como los “puntos kilométricos” de mi agente.
Por ejemplo, si PK1_fila vale 60 y PK1_columna vale 10 y myPK1_fila vale 100 y myPK1_columna vale 16, en la matriz solución, en la componente [60][10] va a haber el contenido de mapa_entorno_[100][16].
Antes de realizar la tarea de traslación hay que tener en cuenta que, al rotar la matriz, también hay que “rotar” los “puntos kilométricos” de mi agente.

```
if(f1 == f2){
   No hay que hacer ningun cambio.
}
else if (f1 == -f2){
   myPK1_fila = 199 - myPK1_fila;
   myPK1_columna = 199 - myPK1_columna;
}
```

```
else if (f1 == c2){
   aux = myPK1_fila;
   myPK1_fila = myPK1_columna;
   myPK1_columna = aux;
   myPK1_columna = 199 - myPK1_columna;
}
else if (f1 == -c2){
   aux = myPK1_fila;
   myPK1_fila = myPK1_columna;
   myPK1_columna = aux;
   myPK1_fila = 199 - myPK1_fila;
}
```

Finalmente, una vez se ha trasladado el map_entorno_ a la solución, mediante un algoritmo basado en dos bucles for, recorro la matriz solución intentando rellenar los huecos que mi agente no ha descubierto pero que sabemos que son así, por ejemplo, rellenar los edificios, los precipicios y, suponiendo que dentro de agua hay agua y que dentro de bosque hay bosque, las zonas de agua y bosque.