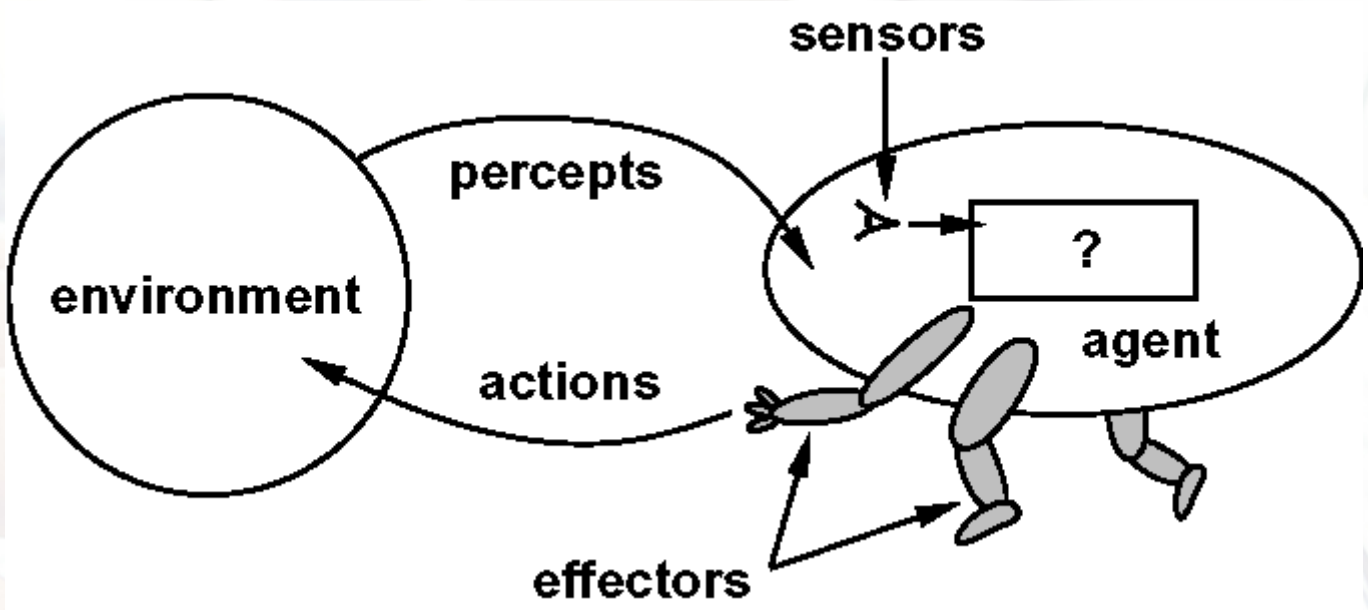
Prácticas de Laboratorio. SINT

Conocimiento del entorno (Búsquedas)

Universidad de Huelva

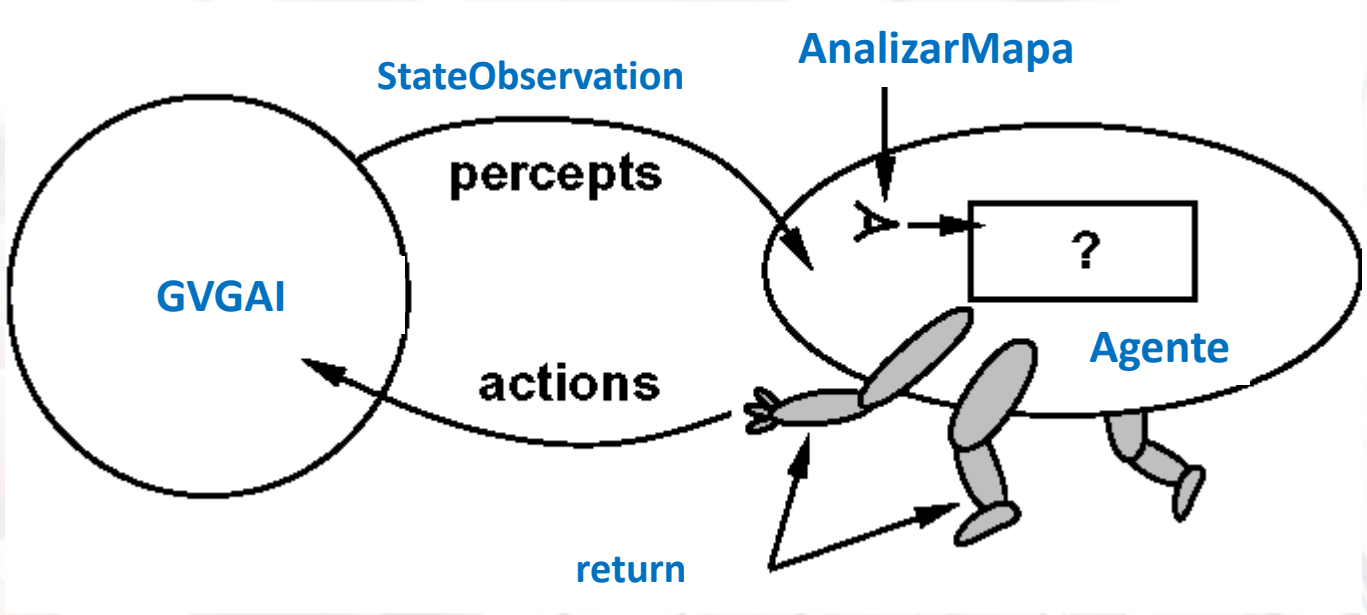
TEORIA

- Según hemos visto en clase de teoría…



EN NUESTRO CASO

- Para nosotros



EN NUESTRO CASO

- Para nosotros

PERCEPCIÓN

```
public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {
    // Proceso de decisión y envío de la acción correspondiente
    RAZONAMIENTO
    return ACTIONS.ACTION_NIL;
}
```

ACCIÓN

StateObservation

Una primera tarea para la implementación de cualquier problema de agentes y de cualquier otra cosa es: el estudio del entorno.

La CLASE que nos trae la información del entorno es: **StateObservation**

Ofrece MUCHA información sobre el estado del juego, mapas, objetos, etc…

Centrémonos por ahora en el MAPA.

StateObservation: MAPA

StateObservation.getObservationGrid()

Devuelve una cuadrícula (array bidimensional) con todas las observaciones en el nivel, accesible por las coordenadas x, y de la cuadrícula.

Cada celda de la cuadrícula tiene un ancho y alto de píxeles: **getBlockSize**().

Cada celda está referenciada por 2 coordenadas contiene una lista con todas las observaciones en esa posición.

StateObservation: MAPA

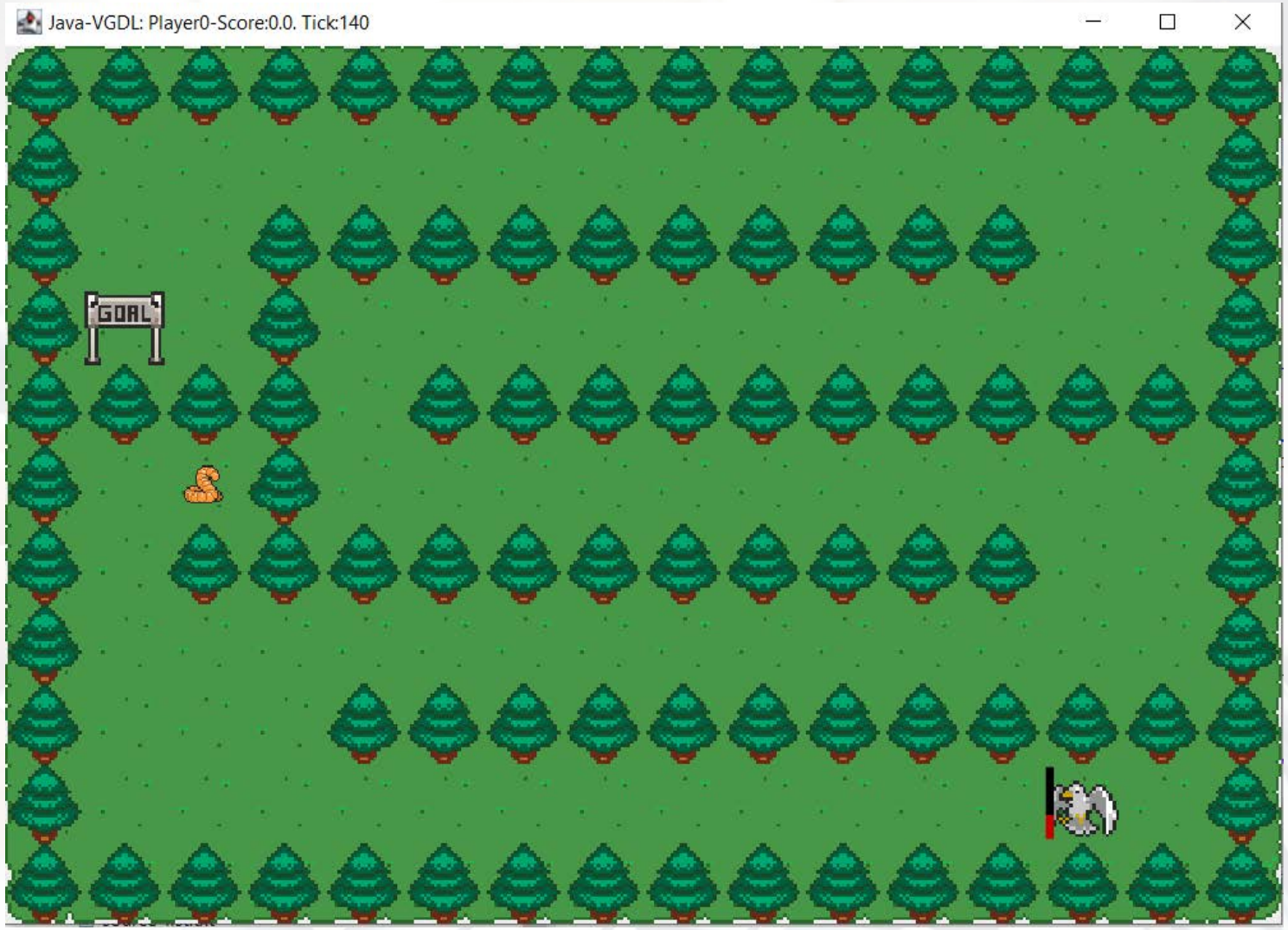
```
stateObs.getObservationGrid()
```

Tenga en cuenta que la misma observación puede ocupar más de una celda de cuadrícula.

¿QUÉ SIGNIFICA ESTO?

Devuelve: la cuadrícula de observaciones.

StateObservation: MAPA



StateObservation: MAPA

EJERCICIO 1

Seleccionar el juego 50

Implementar un jugador que en el “act” represente el mapa del juego en modo texto: PONIENDO UN “*” en la casilla que tiene ALGÚN CONTENIDO y muestre número de filas, número de columnas, tamaño del bloque y dimensión del mapa. Solo se muestra una vez.

```
Numero de filas: 11
Numero de columnas: 16
Diemension block: 50
Dimesión del mapa: 800 x 550

****************
*              *
*  **********  *
** *           *
**** ***********
* **           *
* ***********  *
*              *
*   ***********
*             * *
****************
```

StateObservation: OBJETOS

Los objetos del mapa son instancias de la clase core.game.Observation

Contienen todas sus propiedades, de las cuales, algunas son:

category e **itype**: son para definir el tipo de objeto.

obsID: el identificador ÚNICO del objeto.

position: posición del objeto… OJO para saber la celda hay que dividir por ***getBlockSize***(). ¿qué significa esto?

StateObservation: OBJETOS

EJERCICIO 2

Seleccionar el juego 50

Identificar las categorías y tipos del mapa 50 y pintarlos con símbolos diferentes.

Mostrar al final las posiciones de la meta, de la serpiente y del jugador.

SOLO SE DEBE PINTAR UNA SOLA VEZ TODO

```
 * WARNING: Time limitations based on WALL TIME on Windows *
Numero de filas: 11
Numero de columnas: 16
Diemension block: 50
Dimesión del mapa: 800 x 550

################
#              #
#  ##########  #
#G #           #
#### ###########
# S#           #
# ###########  #
#              #
#   ############
#            X #
################

Posición elemento meta: 50.0 : 150.0

Posición elemento serpiente: 100.0 : 250.0

Posición avatar: 650.0 : 450.0

Posición elemento 11,16: 750.0 : 500.0\n\n
Result (1->win; 0->lose): Player0:0, Player0-Score:0.0, timesteps:285
```

StateObservation: MAPA MEJOR

La percepción (StateObservation) tiene más métodos más concretos y útiles.

Se pueden utilizar para no tener que recorrer el doble bucle del grid.

Los objetos los colocaremos en las casillas en función de la discretización de la posición.

Podemos inicializar parte del mapa en el constructor del agente.

Actualizaremos en el “act” las partes movibles.

StateObservation: MAPA MEJOR

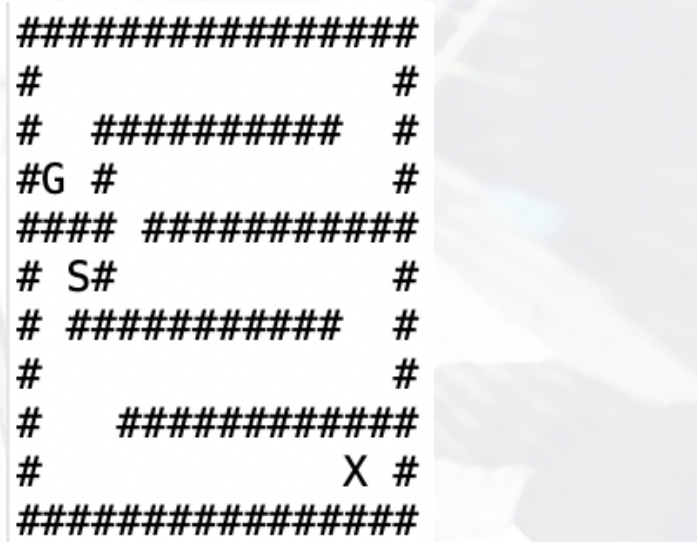
EJERCICIO 3

Seleccionar el juego 50

Crear un agente que inicialice las partes fijas en el constructor y las partes móviles en el ”act” y que se actualice en cada ciclo.

Luego pintar el mapa.

- getMovablePositions();
- getPortalsPositions();
- getAvatarPosition() (…. /stateObs.getBlockSize());)



EJERCICIO 4

Seleccionar el juego 50

Hacer que el Jugador se mueva aleatoriamente y comprobar la visualización.

Seleccionar el juego 46

Hacer que el Jugador se mueva aleatoriamente y comprobar la visualización.

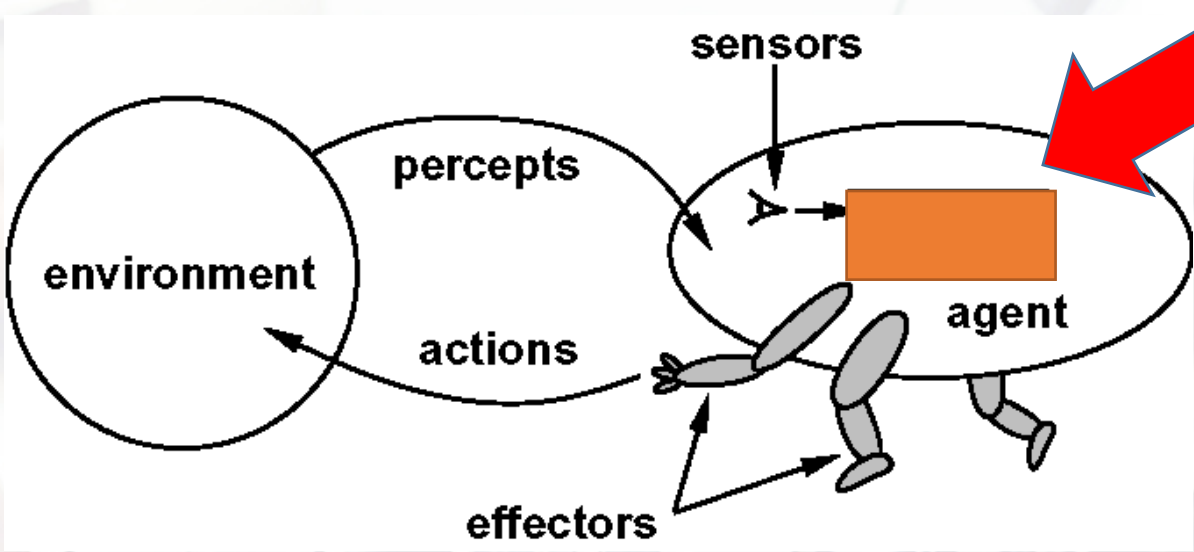
Reorganización del código

Para establecer mejor la filosofía de agentes…

NECESITAMOS REORGANIZAR EL CÓDIGO

Vamos a dotar al agente de…

CEREBRO

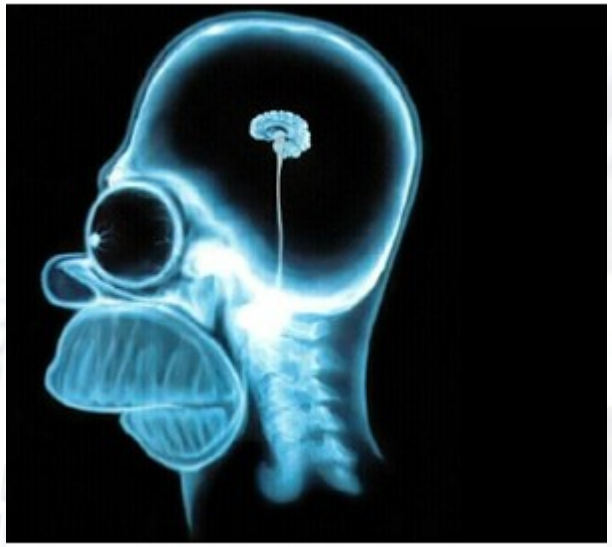


CEREBRO

En el cerebro vamos a concentrar todas las tareas inteligentes.

Dentro del cerebro tendremos:

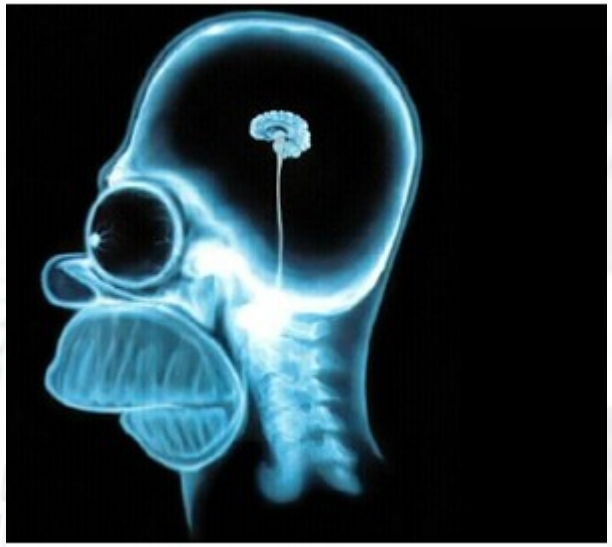
- Nuestra representación del mundo
- Nuestro motor de razonamiento



CEREBRO

Vamos a implementar distintos tipos de Cerebros:

- Por lo que DEBERÁ SER UN INTERFAZ
- Las clases serán implementaciones de éste
- El resto permanecerá MUY similar.

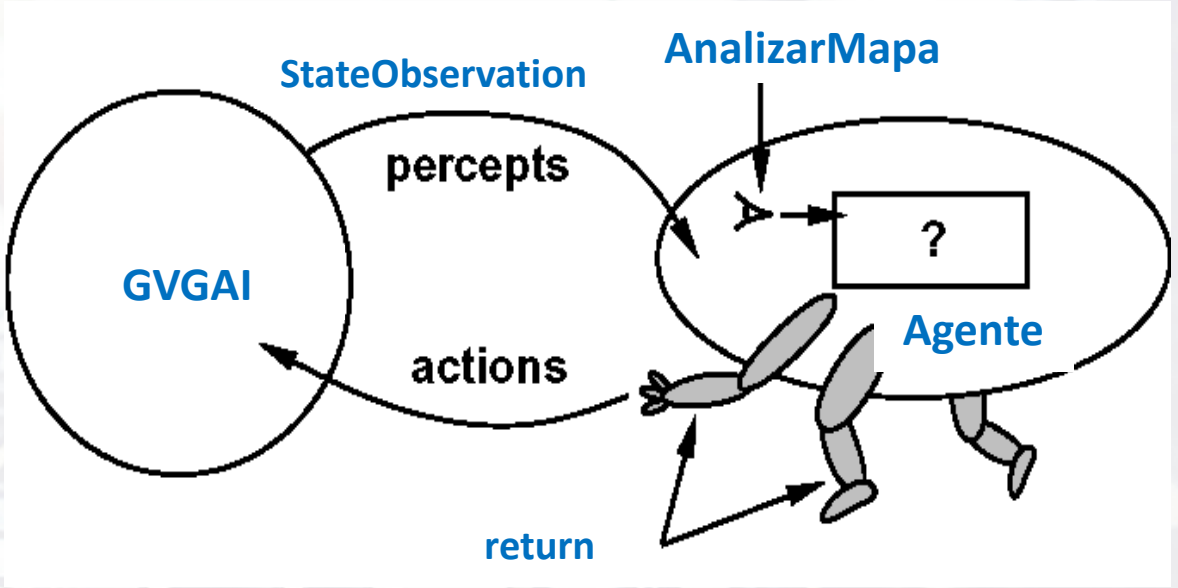


Reorganización del código

CEREBRO

Toda clase que implemente el cerebro deberá tener:

- **AnalizarMapa**
  - que es la forma de analizar el entorno
- **Pensar**
  - Sistema de razonamiento
- **Return**
  - del tipo ACTIONS



EJERCICIO 5

Crear el interfaz Cerebro con los métodos dichos. Crear una clase que implemente Cerebro y que tenga todos los métodos del ejercicio 4. Crear un agente donde en el constructor SOLO tenga la construcción del cerebro (con las observaciones de parámetros), En el “act” SOLO 2 Métodos:

- Actualizar el cerebro y
- Llamar a Pensar del Cerebro y les devuelva la acción.

Reorganización del código

EJERCICIO 5

```
Cerebro cerebro;

/**
 * initialize all variables for the agent
 * @param stateObs Observation of the current state.
 * @param elapsedTimer Timer when the action returned is due.
 */
public Agent01_05(StateObservation stateObs, ElapsedCpuTimer elapsedTimer){
    cerebro = new Aleatoria(stateObs);
}

/**
 * return ACTION_NIL on every call to simulate doNothing player
 * @param stateObs Observation of the current state.
 * @param elapsedTimer Timer when the action returned is due.
 * @return  ACTION_NIL all the time
 */
@Override
public ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {
    cerebro.analizarMundo(stateObs);
    return cerebro.pensar();
    // Proceso de decisión y envío de la acción correspondiente

}
```

StateObservation: PRACTICAR

EJERCICIO 6

Mostrar toda la información posible vista de forma ordenada del **juego 89**

- Dimensión del mapa, número de elementos fijos, número de elementos móviles, tipos de elementos, tamaño del bloque, etc…
- Posición del jugador en el formato: posicion[fila,columna]
- Posición de los adversarios: posicion[fila,columna]
- Finalmente el mapa