PAC 2 Treball fi de carrera ETIG

Xavier Moreu Reñé

INDEX:

Descripció del projecte:

El meu projecte consistirà en una plana web en la que es podrà programar uns robots virtuals que es mouran per un entorn que es pot editar per l'usuari. Tot el entorn del robot o mapa, els robots i demés elements tindran persistència online en una base de dades.

Funcionalitats del projecte:

Per parlar de les funcionalitats dividiré el projecte en varies parts. Hi ha funcionalitats que estaran programades i les farà el Client o més ben dit el browser del client i d'altres el servidor.

- Registre, logging d'usuaris i modificació de dades:

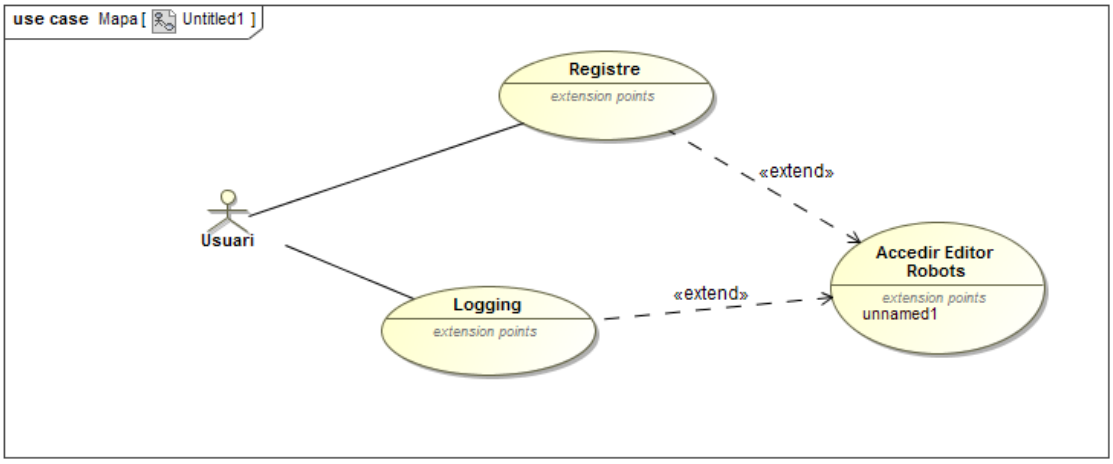
  Aquesta funcionalitat estarà programada en arxius JSP que per tant es compilaran en el servidor i mostraran el seu resultat al client.

  Consistirà en mostrar al usuari dues opcions, de logging o de registre.

  Si un usuari està registrat anirà a fer el logging amb el seu email i password que ha establert.

  Si un usuari no ho està, anirà a registrar-se i automàticament després podrà entrar al programa en si ja registrat.

  Aquestes funcionalitats requeriran per suposat, accés a la base de dades.

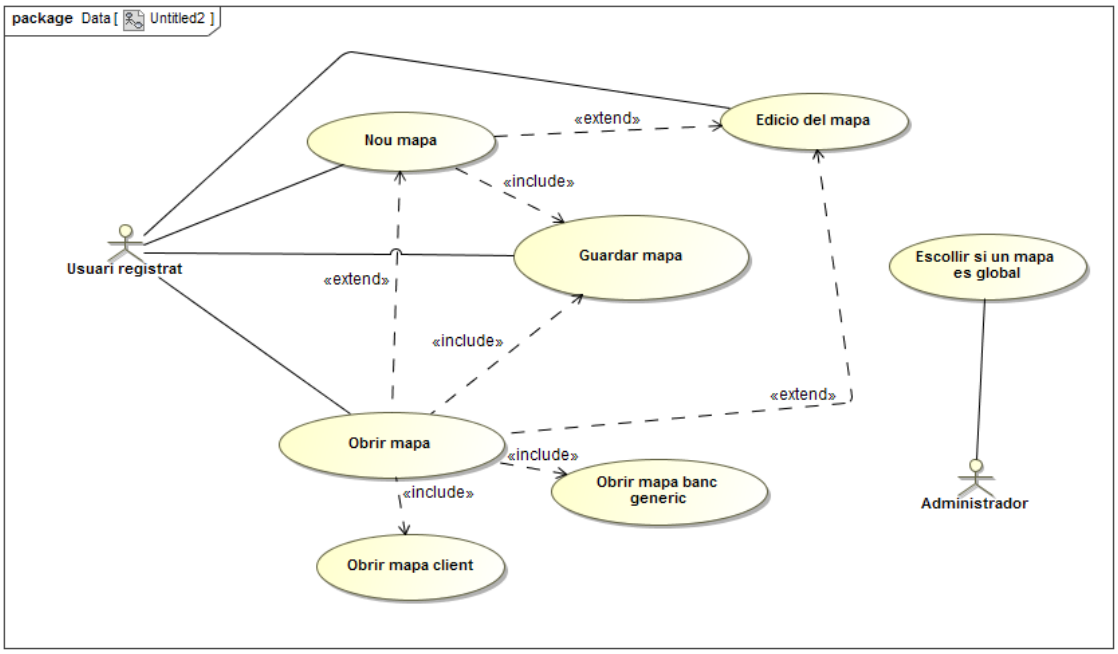


- Gestió de mapes: Crear un nou mapa, carregar un mapa del servidor, gravar un mapa o modificar un.

  Al entrar una vegada registrat, es crearà un nou mapa d'inici que podrà ser gravat al servidor o bé es podrà carregar un ja guardat i modificar-lo com es vulgui. A l'hora de carregar mapes també podrem carregar-ne un d'ús públic en un banc de mapes públics que només el administrador del programa triarà dels usuaris o farà ell mateix.

  Bàsicament un mapa tindrà un nom , una descripció i una sèrie d'objectes que el confeccionaran com son els murs, les balises i els robots. Encara que aquests últims encara no he determinat si es podran recuperar com a entitats apartades del mapa per poder-los carregar en un altra mapa i poder executar el seu codi en aquest.

  El diagrama de casos d'ús seria el següent:



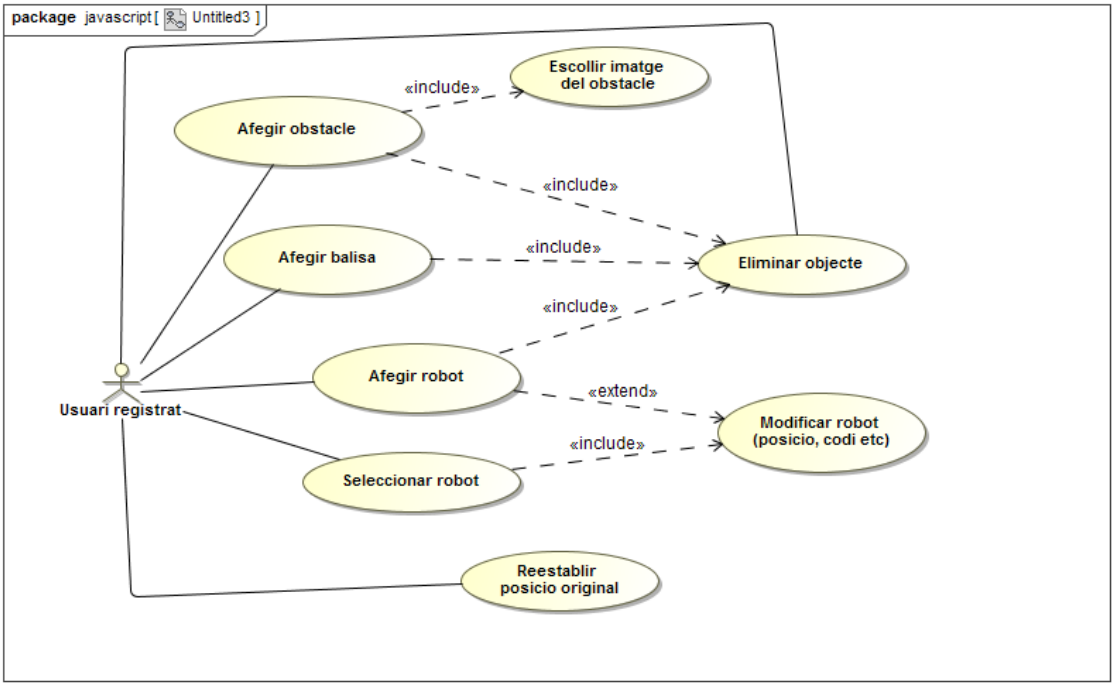
- Edició d'un mapa:

  Un mapa es el entorn gràfic i virtual pel qual un robot es mourà. Serà un espai gràfic dividit en quadrants per una quadrícula (en principi amb resolució de 32 píxels per quadrant) i a on a cada quadrant hi hauran una sèrie de objectes. Per tant el mapa es composa de una sèrie de objectes entre els quals es troben:

  - Obstacles com son murs o arbres que impediran el trànsit del robot per el mapa.
  - Balises: Les balises podran ser agafades i transportades per un robot i tornades a deixar.
  - Robots: Es podran incloure una sèrie de robots dins d'un mapa. Cada robot es desplaçarà per la quadricula del mapa desenvolupant accions de detecció de obstacles o agafant i deixant balises etc. A més hi haurà la opció de fer moure el robot manualment.
  - Diferents textures de terra: aquesta opció no és segur que la inclogui
  - Altres: Deixo un marge per incloure altres tipus de objectes. Per exemple estudio la possibilitat de incloure balises emissores de IR perquè el robot les usi per conèixer la seva posició

Lo important aquí és saber que es podran eliminar tots els objectes però només en el cas dels robots es podrà modificar la seva posició i les seves propietats al ser aquest el objecte més complexa del mapa.

Per últim es podrà restablir la posició original de les balises i dels robots



- Edició de un robot:

  D'un robot es podrà editar el seu nom per identificar-lo i pot ser que el seu aspecte.

  També la seva posició original en el mapa en un moment donat.

  Però el més important és la edició i execució del seu codi que el farà moure per la pantalla.

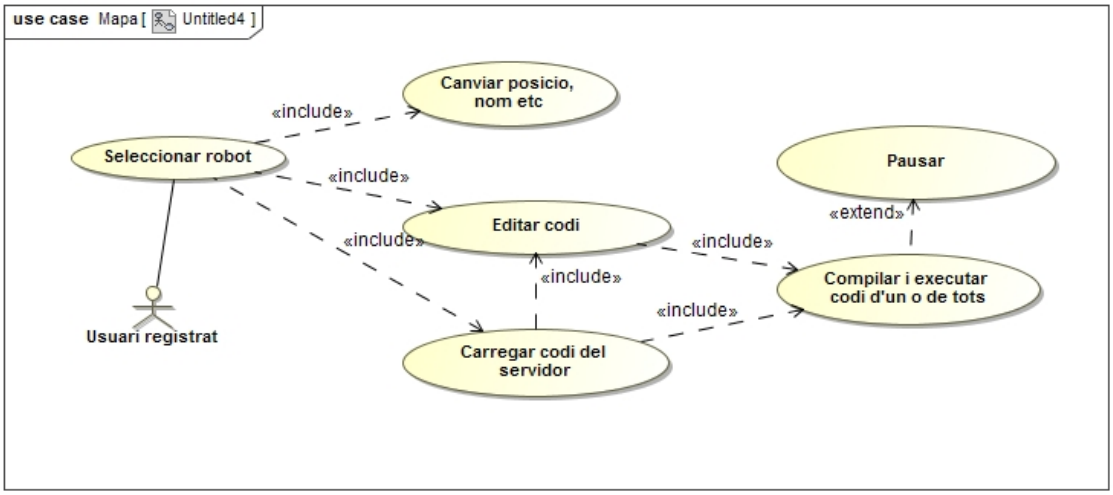
  Un usuari podrà carregar del servidor un codi implementat per ell. Quan salvi les dades del mapa es guardarà també el codi que es podrà carregar independentment del robot al qual pertany.

  Podrà executar el codi del robot que s'està editant de tal manera que el robot es mogui i actuï segons les instruccions d'aquest. Podrà fer que el codi es pausi en un punt per veure el valor dels registres o de la memòria RAM del robot que seran també visualitzats.

  Es podrà escollir si es vol executar el codi de un robot, o tots els robots del mapa.

També estudio la possibilitat de entrar en mode debugger de manera que polsant una tecla vagi instrucció per instrucció.

La funció de compilació del codi escrit la farà el servidor enviant-li el codi i el servidor retornarà el codi compilat i serà el browser del client el que interpreti i executi el codi compilat.



Tecnologies i la seva descripció:

Javascript:

El llenguatge javascript és un llenguatge que interpreta la majoria dels navegadors d'enguany i és el llenguatge amb el que treballaré més per fer aquest projecte.

Javascript no és un llenguatge purament orientat al objecte tot i que es poden definir objectes i treballar amb ells i inclús classes.

La definició de classes es fa d'una manera una mica especial. Una classe es defineix com una funció qualsevol. Per crear les propietats de la classe indicaré amb el terme this. I el nom del paràmetre.

Function MevaClasse(parametre1,parametre2){

this.parametre1=parametre1;

this.parametre2=parametre2;

}

De manera que si jo invoco

var meva=new MevaClasse("x",3);

Crearé un objecte, meva. Si poso meva.parametre1 faig referencia al paràmetre 1 que el puc modificar o consultar.

La creació de mètodes de una classe és força peculiar, es fa usant la propietat prototype que tenen els objectes i igualant-la a una funció que volem fer.

Per exemple seria:

MevaClasse.prototype.operacioAFer=function(parametre){

}

També podem crear relacions de herència definint una funció novament i especificant que s'ha de cridar la funció constructora pare:

```
function Mur(posX,posY,nom,url){
    ObjecteMapa.call(this,posX,posY,nom);
    this.tipus=1;
//    this.img=new Image;
//    this.img.src=url;
    this.url=url;

};
Mur.prototype=Object.create(ObjecteMapa.prototype);
Mur.prototype.constructor=ObjecteMapa;
```

En la imatge veiem com es fa. De la classe pare s'invoca la funció constructora amb call i després es descriuen els paràmetres propis de la subclasse.

A més per definir que tindrà els mètodes de la classe pare es posa la propietat prototype de la classe filla igualada a la del pare i s'afegiran després els mètodes propis del fill.

La herència va bé perquè podem fer polimorfisme de manera que amb el operador typeof podem esbrinar de quina subclasse és un objecte que l'hem passat com a tipus classe pare.

Canvas:

Per la part gràfica del projecte faré servir l'objecte Canvas de HTML5. Canvas és un objecte del codi html com pot ser INPUT o TEXTAREA que es defineix amb un tag amb el mateix nom.

Es pot definir així:

<canvas id="lienzo" width="1024" height="500"></canvas>

Es molt important definir l'amplada i altura del objecte que serà l'area on dibuixarem.

Per manegar l'objecte, dibuixar-hi, canviar-li propietats etc farem servir javascript que és un llenguatge que es compila i interpreta en el browser del client.

Per fer un dibuix en el canvas hem de referenciar el objecte que manega la gestió de dibuix i que té definits tots els mètodes de dibuix. En altres llenguatges com el Java fem servir el objecte "Graphics" que el conté el JPanel o altres tipus de classes. En el cas de canvas l'objecte equivalent és "Context". De manera que podem dibuixar algo en el canvas obtenint el objecte context que conté i invocant els seus mètodes.

Un exemple:

El primer que hem de fer és referenciar per DOM el objecte canvas definit en el codi HTML

```
var canvas = document.getElementById("micanvas");
```

Despres s'ha de referenciar l'objecte Context que conté

```
var ctx = canvas.getContext("2d");
```

i tot seguit ja podem dibuixar un rectangle en ell amb el mètode fillRect

ctx.fillRect(10,10,200,200); que ens dibuixa un rectangle en la posició 10, 10 en píxels començant per la part superior esquerra i amb una amplada de 200 per 200 pixels.

Li podem afegir a més el estil ctx.fillStyle="#f00";

Però el primer que em vaig plantejar és si podria dibuixar una imatge carregada en una carpeta del nostre servidor o de qualsevol perquè jo volia crear sprites i representar-los en el canvas.

Hi ha una classe anomenada Image que ho fa.

En un objecte Image hi ha un paràmetre src o source en el qual es defineix

De manera que amb les següents instruccions:

var img=new Image();

img.src=”robot.png”;

Ja tindríem els bits de la imatge en l’objecte Image llestos per representar en el canvas

L’objecte context té un mètode molt important anomenat drawImage que el que fa és dibuixar un objecte img en el canvas al qual pertany o també dibuixar un nou canvas dins un altra.

Per exemple:

context.drawImage.drawImage(img,0,0,32,32);

Dibuixa el robot.png dins del canvas en la posició 0,0 i amb l’amplada 32 per 32 adaptant la imatge a aquesta amplada.

A més, drawImage té els següents paràmetres en la funció:

context.drawImage(*img,sx,sy,swidth,sheight,x,y,width,height*);

On img és l’objecte Image o un altra objecte canvas

sx,sy,swidth i sheight perquè sèntengui son els valors que defineixen la area de retall de la imatge img que es vol pegar al canvas.

X,y, w, h son els valors que defineixen posició i dimensions on es vol pegar el retall dins del canvas.

Per tant si tinc una sèrie de sprites en una imatge puc anar retallant subimatges d’aquest a imatge i crear nous canvas amb cadascun de les subimatges, cosa que em facilita molt la feina.

Per tant quasi bé tinc tot el que he de saber de canvas per crear el mapa de entorn per on es mouran els robots.

Una cosa que em preocupava és una vegada sabuda la manera de dibuixar les imatges, la pregunta era com les podria moure. Necessitava alguna manera de cridar una funció determinada cada x temps. La funció és setInterval del objecte Window. Si es crida enviant-li un valor en milisegons que representa el temps que ha de passar fins que la funció es cridi donarà l’efecte d’animació desitjat.

setInterval(actualitzar,1000) per exemple, crida la funció actualitzar cada segon que transcorre.

El problema que trobava és que una imatge o sprite, quan es movia per pantalla perquè canviava el seu x,y, no s’esborrava la imatge en la posició anterior. Això es soluciona dibuixant un fons i després la imatge. El fons pot dibuixat per el mètode clearRect de context.

Un altra problema era que quan les imatges es movien per el canvas, feien pampallugues. Aquest efecte no desitjable es degut a que es va redibuixant el objecte o objectes sobre el canvas. La solució és la antiga tècnica del doble buffer. Consisteix en dibuixar totes les imatges en un context de un nou canvas i després dibuixar en el canvas destí el canvas nou.

Perquè s'entengui:

```
var canvas2 = document.createElement('canvas');

var context2 = canvas2.getContext('2d');
```

i en la funció que es va cridant cada interval de temps:

```
function actualitzar(){

        context2.clearRect(0,0,lienzo.width,lienzo.height);

        context2.drawImage(img,0,0,32,32);
        contextLienzo.clearRect(0,0,lienzo.width,lienzo.height);

        contextLienzo.drawImage(canvas2,0,0);

}
```

Com veiem es va dibuixant el canvas2 en el canvas original.

Un altra problema que he vist és què passa si tinc molts objectes a dibuixar com serà el cas ja que un mapa estarà composat per elements gràfics per dibuixar els murs etc. Aquests elements seran estàtics menys els robots i les balises. Quan es redibuixa la escena en cada iteració, si faig un bucle for per exemple, farà que vagi molt lent tot ja que tinc que redibuixar un per un els elements. Però com que son estàtics la majoria el que vaig pensar de fer és dibuixar en un principi els elements del mapa en un context de un canvas de fons i després dibuixar aquest canvas que tindria la mida del canvas visible en cada iteració de setInterval.

Al final el que faré és un dibuix per capes.

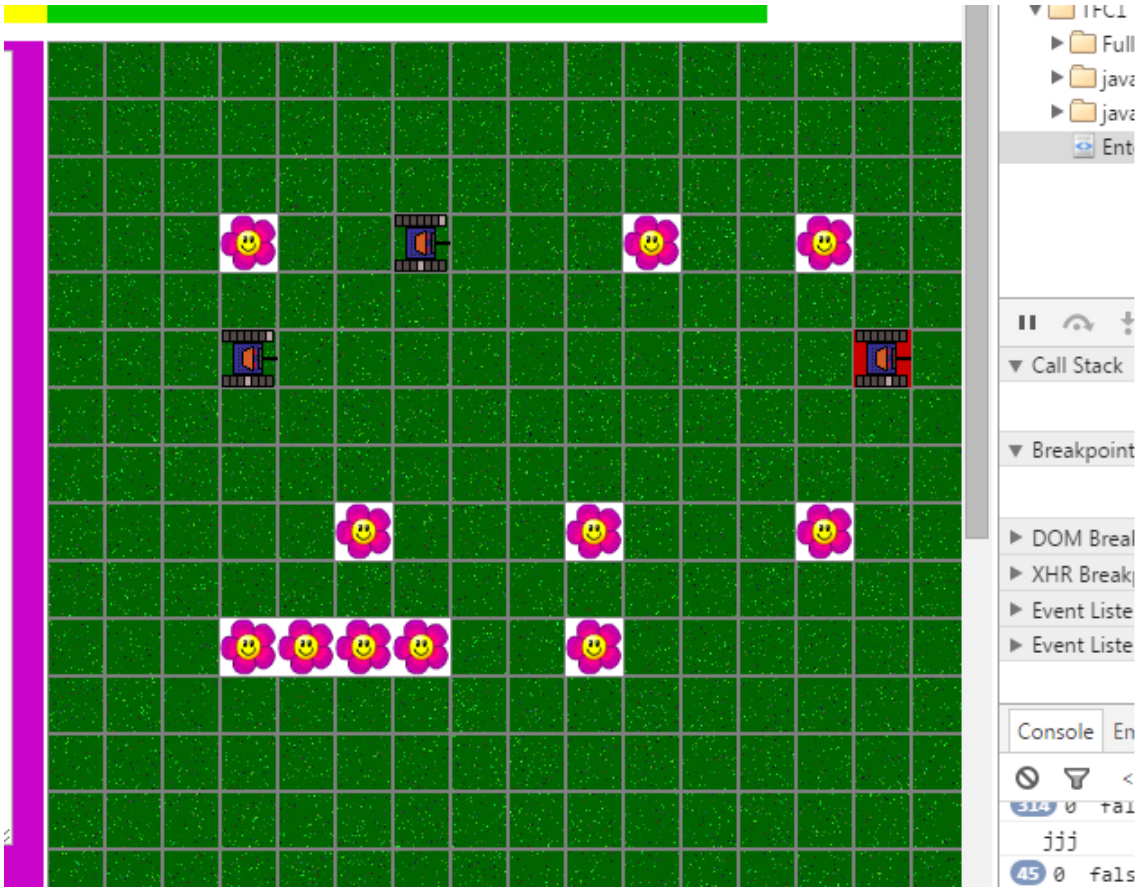
La capa de fons o canvas de fons contindrà una graella per determinar més be el posicionament dels objectes. A més he dissenyat una funció que dibuixa píxels amb color aleatori per crear un cert efecte noise. Si el fons és verd i els píxels van en diferents tonalitats verdoses es crea un efecte de gespa.

Després una segona capa que serà el canvas de mapa on dibuixaré tots els objectes del mapa estàtics com els murs.

I finalment una capa de elements mòbils on dibuixaré tots els elements mòbils com ara els robots i les balises.

En cada iteració per tant es dibuixaran les tres capes una sobre l'altra amb el ben entès que les dues capes primeres no canvien.

El resultat és el capturat en la pantalla seguent:



En la imatge veiem els robots i els murs en forma de flor(encara els tinc que dissenyar).

Un altra efecte que vull crear és el scrolling de la escena. Aquest efecte encara no l'he estudiat perquè no tinc clar de fer-ho.

Aquest scrolling representa problemes a superar. Per exemple, puc fer que el canvas de fons i el canvas de mapa es moguin (varii el seu x, y dins del canvas) però llavors els he de definir més grans i no sé si donarà això problemes de memòria.

Un altra solució és fer un scrolling per sectors. El scrolling per sector es basa en dividir el mapa en sectors i quan un objecte mòbil com un robot està a prop de sortir de la pantalla visible es carrega un altra sector del mapa on el robot quedi per la mitat per exemple. Així s'estalvia memòria perquè només es redibuixa la part visible.

JQUERY i AJAX:

Jquery és un framework de javascript que ens facilita molt l'accés a elements del html i a la invocació de events d'aquests. Com per exemple la detecció del click del mouse en el canvas.

Però el més important és que és una eina que facilita molt més la comunicació AJAX amb el servidor, cosa que hauré de fer molt per guardar dades, carregar-les o simplement per si vull que el servidor executi operacions i retorni un resultat (com ara la compilació del codi virtual que la faré així).

Per detectar que el botó del mouse s'ha polsat sobre el canvas escriurem la instrucció:

```
$('canvas#lienzo').mousedown(function(e){


});
```

On entre parèntesis li pesem com a paràmetre la funció que s'ha d'invocar. Podríem haver posat $('canvas#lienzo').mousedown(polsaMouse) ;

```
function polsaMouse(e){

}
```

Com veiem és molt fàcil fer referència a un component del html posant el seu id. En aquest el canvas amb id =lienzo.

Per fer peticions a servidor usant la tecnologia Ajax amb jquery és tremendament fàcil amb la funció ajax que incorpora.

La estructura general seria aquesta:

```
$.ajax({
        type: "POST", //Mètode de enviament al servidor
          data: <Les dades que enviem>,
        url: <El arxiu jsp que processarà les dades>,
        success: function(data) {
           var res = data;
                <El que es farà si la resposta del servidor té èxit. "data" conté les dades que retorna el servidor. Pot ser un xml, un json, text etc>
           } ,
          error: function(e) {
                <El que es farà en cas de fracàs de l'enviament>
        }
```

});

En url podríem posar també un arxiu php depenent de la plataforma amb la que treballem. En el cas que ens ocupa és un arxiu servlet jsp.

El tema de les dades és important. No és el mateix enviar una cadena simple de texte que una sèrie de dades com un array o un objecte.

Per fer això es pot escollir bàsicament dos tecnologies, JSON o XML.

Tecnologia de encapsulació de dades JSON:

Com he dit abans, quan tenim dades que han de viatjar del client al servidor per AJAX, i aquestes son nombroses necessitem organitzar-les i que siguin de fàcil accés per una de les parts (servidor i client).

XML és una tecnologia que organitza les dades en forma de etiquetes que tenen paràmetres i que contenen altres etiquetes o subetiquetes etc de forma que té estructura d'arbre i permet una localització ràpida de una dada per tenir aquesta estructura de dades. HTML És un tipus de XML. En android també s'usa i en molts frameworks també. La tecnologia XML és molt eficient ja que per exemple, en una plana web, algunes subetiquetes seran imatges que carreguen lent i en canvi el browser llegirà el html global i deixarà la càrrega d'imatges pel final. Això fa que ja es carregui el text i l'estructura bàsica de la pàgina. Android també funciona així perquè un dispositiu mòbil ha de tenir un sistema de presentació més òptim.

Dit això que toca fora de tema, el sistema que he triat és JSON ja que l'intercanvi es farà entre dos llenguatges que són java. Javascript i java que incorporen funcions de ràpida lectura i escriptura de format JSON. Si per exemple es preveies la creació d' un programa d'escriptori amb C++ o altra llenguatge interessaria més usar XML perquè és més universal .

JSON *JavaScript Object Notation* és una nomenclatura que estructura les dades entre claudàtors o bràquets i les separa en comes. És una estructura menys universal i es basa en la programació orientada al objecte. Es va crear per javascript que incorpora funcions que accedeixen fàcilment al json i creen objectes automàticament de forma molt més sencilla que fer-ho amb XML.

Un exemple seria

Json={"Persones":[{"nom":"Joan","edat":36},{"nom":"Pep","edat":18}],"Robots":[{"model":"k-456","autonomia":4}]}

De aquí amb la funció de jquery parseJSON obtindríem ja la estructura en objectes.

Per accedir al nom de la segona persona n'hi hauria prou en posar json.Persones[1].nom que escriuria Pep.

Com veiem és molt ràpid.

En la banda del servidor, en el servlet JSP tenim una biblioteca que fa el mateix però amb algunes diferències. En Java, s'han de definir les classes per crear nous objectes a diferència de javascript en el que es treballa amb objectes però no cal definir-los .

La biblioteca s'anomena Gson i és molt fàcil llegir un json i representar-lo en objecte sempre i quan li donem la classe i subclasses que ha de usar per copiar dades.

Per exemple,

gson.fromJson(json, Persona.class);

On s'indica la classe que prèviament ha de estar definida i el que farà la funció és copiar les dades del JSON i igualar-les als paràmetres de la classe.

Per transferir via Client (javascript) --→Servidor(JSP) usarem la funció JSON.stringify(<Objecte a convertir>) de la llibreria de javascript json. Al costat del servidor es farà servir la funció gson.fromJson(<json rebut>,< Classe a usar>);

Per transferir de Servidor(JSP) →Client(javascript) Usaré la funció gson.toJson(<Objecte a transferir>). I de la banda del client (javascript) faré servir la funció jQuery.parseJSON(respostaServidor); que em transformarà les dades textuals en objectes javascript.

Hibernate per la permanència de dades:

Hibernate és un framework que facilita la permanència de dades en una base de dades. Funciona a base de arxius XML que contenen els paràmetres per connectar amb la BD i on informen de com es relacionen les classes de negoci amb les taules etc. Permet fer consultes, insercions etc de manera més ràpida i que les classes getters i setters puguin contenir els seus valors ràpidament. També permet obtenir a partir de taules de la BD els getters i setters corresponents de manera automatitzada sense clicar ni una línia de codi.

En el meu projecte vaig dubtar de usar-ho perquè l'esforç de aprendre'l a fer servir no compensava el de crear jo mateix les consultes SQL etc. A més com que el programa principal treballa en javascript, els getters i setters serien prescindibles. Tot i així com que es tracta de un projecte de carrera he decidit fer-lo servir per aprendre un framework nou. He tingut problemes a la hora de construir els JSON amb la classe Gson i els getters i setters obtinguts de la BD i tindré que fer solucions una mica rudimentàries.

A continuació explicaré com funciona Hibernate:

Manegar Hibernate amb Netbeans és molt sencill:

Primer de tot es clica el hibernate configuration Wizard que és un assistent per la configuració de l'arxiu hibernate.cfg.xml que tindrà en format xml totes les dades per connectar amb la meva BD

L'aspecte que acaba tenint és

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>

    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>

    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/bdejemplo</property>

    <property name="hibernate.connection.username">root</property>

    <property name="hibernate.connection.password">1234</property>

    <mapping resource="SetersGeters/Persona.hbm.xml"/>

    <mapping resource="SetersGeters/Usuari.hbm.xml"/>

    <mapping resource="SetersGeters/Mapa.hbm.xml"/>

    <mapping resource="SetersGeters/Objectemapa.hbm.xml"/>

  </session-factory>

</hibernate-configuration>
```

Com veiem està el driver usat, la url de la base de dades, el nom d'usuari, el password i tota una sèrie de tags anomenats mapping resource que especifica les classes unsades per guardar-les a la BD.

Per crear els setters i getter primer hem de crear un arxiu hibernate.reveng.xml amb l'assistent de netbeans "Reverse engineering Wizard" que el que fa és fer crear l'arxiu llegint l'estructura de dades de la nostra BD.

Una vegada fet això s'executa l'assistent "Hibernate mapping files and POJO's from data base" que el que fa és que a partir dels dos xml creats, crea els getters i setters corresponents junt amb els arxius NOMCLASSE.hbm.xml. Aquests arxius xml son els més importants ja que especifiquen quina relació tenen els noms dels paràmetres dels getters i setters respecte amb el nom del camp en que es guardaran o consultaran a la BD. També especifiquen si tenen subobjectes etc que ajudarà en les consultes.

Un exemple de Hibernate mapping és el seguent:

```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
```

```
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<!-- Generated 05/10/2015 11:10:39 by Hibernate Tools 4.3.1 -->

<hibernate-mapping>

   <class name="SetersGeters.Mapa" table="mapa" catalog="bdejemplo" optimistic-
lock="version">

      <id name="idMapa" type="java.lang.Long">

         <column name="idMapa" />

         <generator class="identity" />

      </id>

      <many-to-one name="usuari" class="SetersGeters.Usuari" fetch="select">

         <column name="idUsuari" unique="true" />

      </many-to-one>

      <property name="nomMapa" type="string">

         <column name="nomMapa" length="50" />

      </property>

      <set name="objectemapas" table="objectemapa" inverse="true" lazy="true"
fetch="select">

         <key>

            <column name="idMapa" not-null="true" unique="true" />

         </key>

         <one-to-many class="SetersGeters.Objectemapa" />

      </set>

   </class>

</hibernate-mapping>
```

Com veiem relaciona cada paràmetre de una classe amb la columna de la bd que el guardarà.

Per definir el id que identifica a cada registre es fa amb el tag id. Dins d'aquest el tag generator class es refereix a com es generarà aquest id. En el cas aquest com que és un valor auto incremental el proporciona la base de dades s'ha de posar identity.

Una relació ‘ many to one’ o de varis a molts especifica que pot tenir varis objectes d’aquesta classe el objecte especificat en la relació. En aquest cas és la propietat usuari que refereix a la taula Usuaris i que ens diu que varis mapes pertanyen a un usuari.

Per definir una relació ‘one to many’ el que es farà és definir en la classe que pertany al one diguéssim, un paràmetre List o Set. En el XML el tag és Set si és set. S’ha d’especificar el key de la base de dades de la part ‘many’ o sigui de cada element del Set que es relacionarà amb l’id del mapa (one) que els conté. Especifica la clau forànea. Com a paràmetre del tag Set podem especificar el comportament al guardar un objecte Mapa en vers a guardar o no els subobjectes de la relació ‘one to many’. Si posem cascade=’all’ especificarem que al guardar un Mapa, Hibernate guardi tots els objectes o robots. També que s'esborrin en el cas de fer un remove.

A més en el arxiu hibernate.cfg.xml s’especificarà a on estan aquests arxius hibernate-mapping.

Amb totes aquestes dades , la biblioteca d’hibernate ja està preparada per fer consultes, gravar dades etc.

Es crea una classe que gestiona la connexió i sessió amb la BD i creem una classe que faci les operacions que vulguem.

Per exemple, si volem obtenir el llistat de mapes de un usuari determinat ho farem així:

```
public List getListaMapas(long idUsuari){

    Session session = HibernateUtil.getSessionFactory().openSession();

    session.beginTransaction();

    List result = session.createQuery("from Mapa where idUsuari="+idUsuari).list();

    session.getTransaction().commit();

    return result;

  }
```

Com veiem creem una sessió amb la classe HibernateUtil, li diem que iniciem una transacció amb la BD i fem el query. Ens retornaria en aquest cas un llista de objectes de la classe Mapa demanats.

Per guardar un mapa faríem el següent:

```
public void guardaMapa2(Mapa mapa)  {

      Session session = HibernateUtil.getSessionFactory().openSession();
```

```
        session.beginTransaction();
        session.saveOrUpdate(mapa);
        session.getTransaction().commit();
        session.close();

    }
```

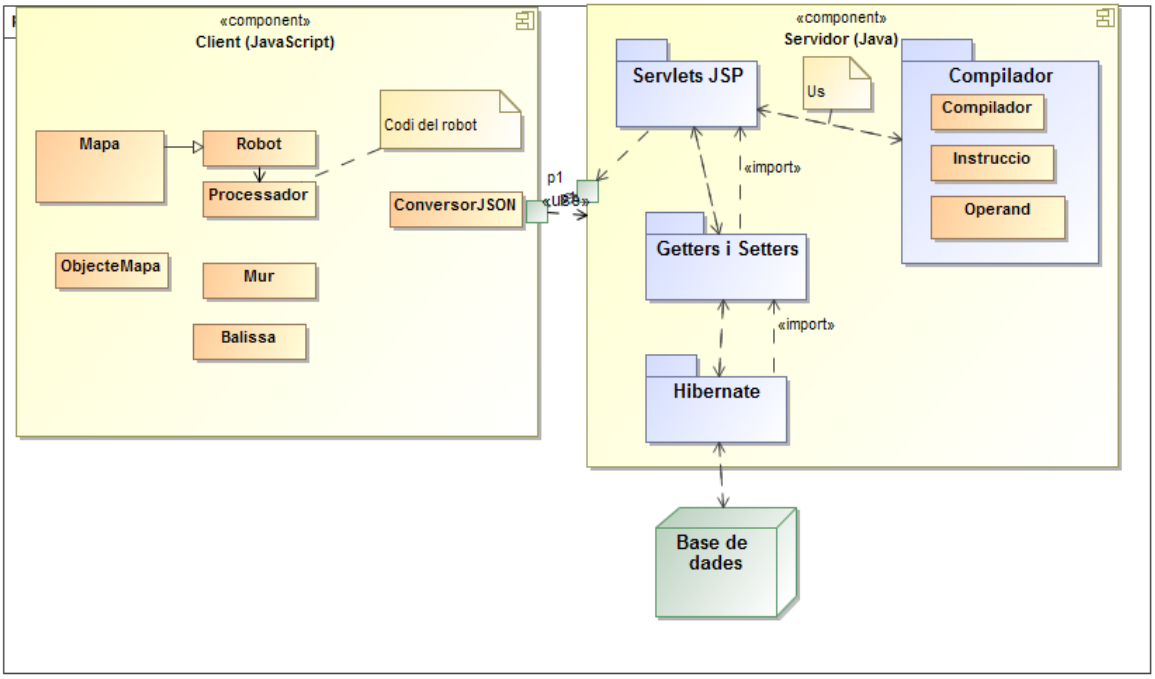
Com veiem, cridant el mètode save o saveOrUpdate i passant l'objecte Mapa que ja conté les dades a guardar en els seus paràmetres, es guardaran les dades a la BD.

Com he dit abans, en una aplicació en la que el servidor fa les operacions de tota la aplicació ens simplifica enormement la feina perquè no tindrem que crear els setters getters i després li podrem afegir els mètodes que volguem. A més no tindrem en molts casos que fer cap consulta SQL cosa que per a mi no representa molta dificultat. Al final acabes necessitant fer consultes HQL (Consultes de hibernate) que en definitiva el SQL una mica simplificat.

Arquitectura del projecte:

Amb tot això, passaré a descriure gràficament i amb paraules les parts del projecte i com es gestionarà tot i en quin llenguatge. També especificaré els diagrames UML de les classes i les taules de la BD.

A continuació un esquema global de la interacció entre els diferents components:



Com podem veure hi ha tres components que s'han de comunicar. El Client, el servidor i la base de dades.

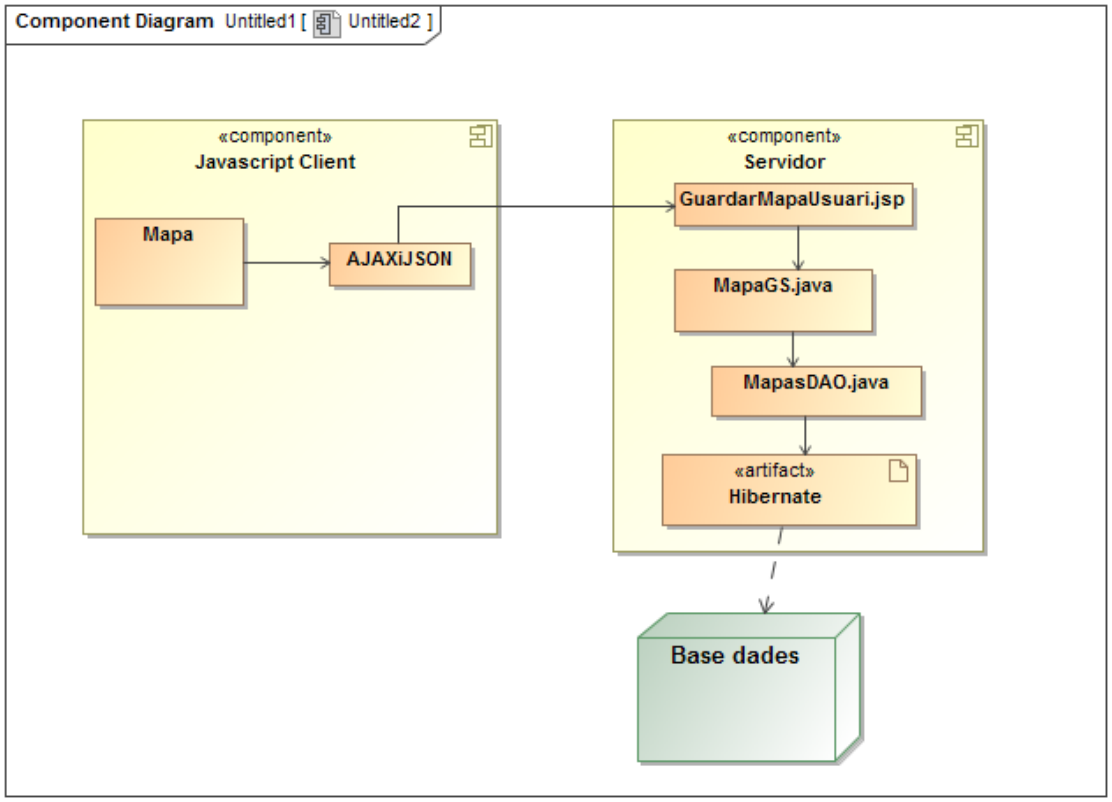
El client representa el browser o navegador amb el que s'accedirà a la web creada. El llenguatge emprat és HTML i JavaScript que la majoria de navegadors interpreten.

En javascript especificaré una sèrie de classes necessàries perquè el software treballi. Una de les classes s'encarregarà de la conversió dels objectes a JSON i viceversa i de la comunicació amb la tecnologia AJAX amb el servidor. Per fer aquesta conversió necessitaré crear uns objectes de suport amb els mateixos paràmetres en quan a nom que la base de dades on es guardaran.

En la banda del servidor tindrem una sèrie de arxius JSP servlets que rebran informació via JSON del client i enviaran d'aquesta manera també la informació.

Aquests arxius faran servir una sèrie de classes com son la Gson, els Setters i Getters i classes per guardar a i consultar la BD a traves del paquet Hibernate i gràcies als arxius XML que requereix. També farà servir el paquet Compilador que contindrà les classes pròpies d'aquesta funció complexa una vegada hagi rebut un codi String del client. El servlet rebra per JSON el codi i usarà les classes del paquet per compilar-lo i retornar-lo igualment via JSON.

El procés de guardar un mapa per exemple seguirà el següent esquema:



Com podem veure el objecte Mapa en javascript serà convertit en una cadena de text JSON per la classe AJAXiJSON i enviat al servidor.

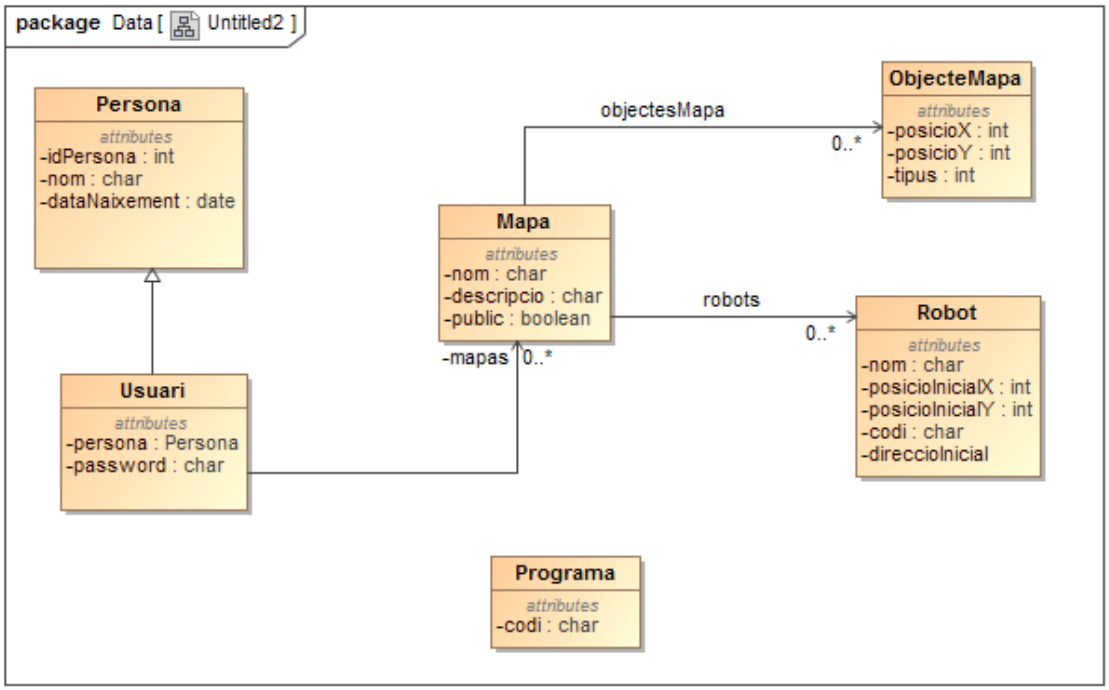
El servidor tindrà un servlet JSP anomenat GuardarMapaUsuari .jsp que crearà a partir del json amb ajut de la llibreria Gson un objecte Mapa amb tots els subobjectes que conté. Aquest objecte serà de la classe Mapa que hauré creat amb l'assistent de hibernate.

Després amb la classe MapasDAO que serà un gestor que interactua amb hibernate, guardaré el objecte mapa en la BD.

Per recuperar un mapa es seguirà el procés contrari.

A continuació faré un esquema del diagrama de classes en la banda del servidor:

Tindrem els getters i setters obtinguts de la BD:



Com podem veure és un diagrama molt sencill. He volgut simplificar molt la estructura dels objectes i fer una base de dades senzilla. Les taules de la base de dades bàsicament tindran els mateixos paràmetres i les relacions seran una referència a nivell de un identificatiu numèric que identificarà la taula a la qual pertany la relació.

Observem en aquest diagrama que un usuari registrat tindrà una sèrie de mapes creats per ell que poden ser públics o privats. Aquest paràmetre com he dit en els cassos d'ús serà modificat per el administrador només.

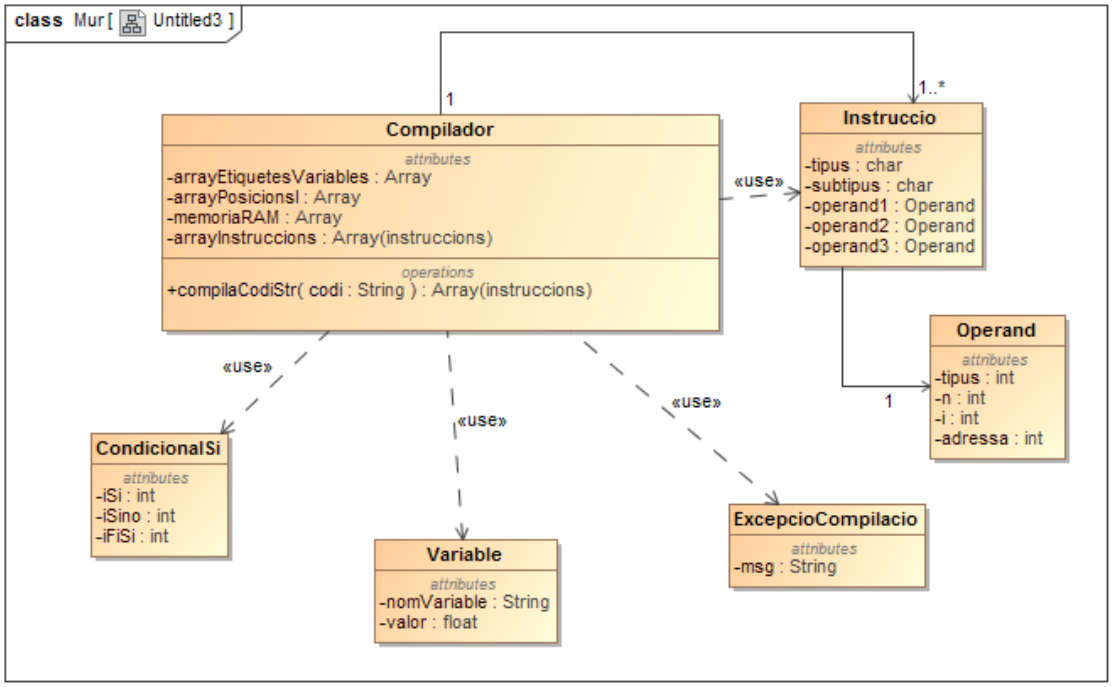
Cada mapa tindrà una sèrie de objectes que representen fragments de mur o bé balises o qualsevol objecte que se m'acudeixi. Els diferenciarà el atribut tipus.

També tindrà una sèrie de Robots cadascun amb unes propietats bàsiques que li definiran la seva posició inicial dins del mapa i el seu codi per executar-se i moure's en l'entorn.

Per un altra banda tindrem la classe Programa que servirà per guardar i recuperar en la BD els codis independentment de que pertanyin o no a un robot.

Com he dit anteriorment la compilació del codi de cada robot es generarà a la banda del servidor.

Per fer això tindrem una sèrie de classes per realitzar aquesta complexa operació:



La classe Compilador és molt complexa, Contindrà una sèrie de mètodes per realitzar la traducció de un codi escrit en una cadena de caràcters, a una sèrie de instruccions que el processador que estarà programat en el client amb javascript, podrà interpretar i executar.

La classe Instrucció farà entendre com funciona el processador virtual.

Una instrucció serà de un tipus determinat com podrà ser de tipus “operació” o de tipus “salt” i tindrà un subtipus com pot ser “suma” o en el cas de tenir tipus “salt” podrà ser “sempre” o “mesgran”, “igual” etc

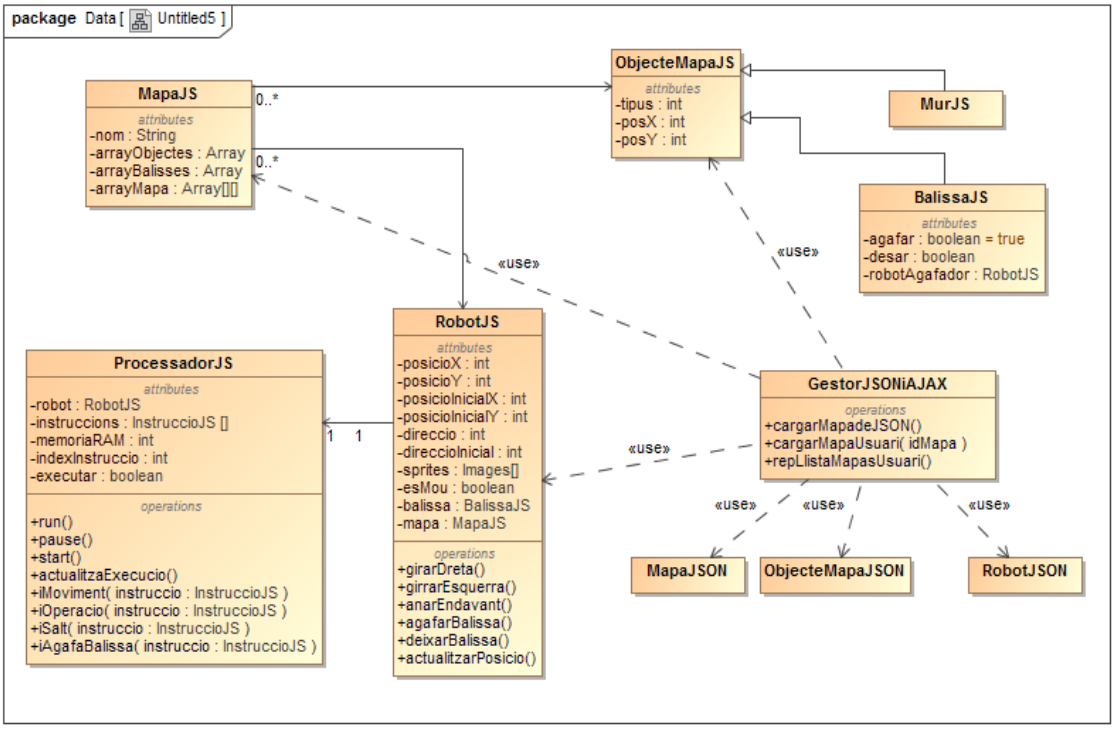
Per altra banda tindrà tres operadors que especificaran posicions de memòria ram virtual o bé nombres.

El operand1 normalment es referirà a una posició de memòria destí. Els operands dos i tres seran o bé posicions de memòria o nombres amb els quals s’operarà. Per això la classe Operand s’especificarà per un tipus que definirà si s’ha de tractar com una posició de memòria o un nombre. I tindrà una sèrie de paràmetres com n,i,adressa que son valors que especifiquen ubicacions de memòria o nombres. “n” és un nombre amb coma flotant que serà el nombre que s’ha de llegir per la instrucció si el tipus és “0”. “adreça” i “i” donaran una posició de la memòria ram sumant ambdós valors.

He vist que aquesta estructura de dades de la instrucció com a tal serà la més eficaç. Sempre intentant que els valors a guardar i consultar siguin enters o string petits perquè s’executin el més ràpid possible pel processador real.

Les classes CondicionalSi,Variable i ExcepcioCompilacio son classes que necessitarà el compilador per realitzar la seva tasca.

A continuació el diagrama de classes en la banda del client (javascript):



Com veiem és quelcom complexa i he deixat de dibuixar algunes classes com podrien ser la classe que controla els sprites dels robots o la que controla la animació i dibuixat.

La estructura Mapa, ObjectesMapa, Robots etc és molt semblant al diagrama de classes dels setters i getters que teníem al servidor però aquí tindran multitud de operacions que realitzaran les accions per tal de que els robots es moguin, interactuïn amb l'entorn etc etc.

La classe RobotJS és la més complexa juntament amb la seva súbdita classe ProcessadorJS. La classe RobotJS té totes les propietats i mètodes necessàries per moure's per la pantalla i interactuar amb el seu entorn (detectant col·lisions, agafant balises etc) Però la classe que realment el fa moure és la classe ProcessadorJS que el que fa és interpretar les instruccions de un array i canviant els valors de la memòria ram virtual expressada com un array de valors (long). El mètode +actualitzaExecucio() farà això i cridarà mètodes com iMoviment o iOperació segons el tipus de instrucció.

Finalment, la classe GestorJSONiAJAX s'encarregarà de enviar per la tecnologia AJAX les dades en format JSON al servidor i viceversa amb ajut de unes classes còpia de les principals (MapaJSON, ObjecteMapaJSON etc) que faran que la informació estigui ben estructurada en la sentència JSON.

Una mica de MySQL:

Per crear la base de dades i les taules, camps, claus foranies etc, he usat el Workbench de MySQL que simplifica i accelera molt el treball. A continuació mostro totes les sentències SQL per crear les principals taules de la BD.

```
CREATE TABLE `mapa` (
  `idMapa` bigint(20) NOT NULL AUTO_INCREMENT,
  `nomMapa` varchar(50) DEFAULT 'nouMapa',
  `idUsuari` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`idMapa`),
  KEY `pertanyUsuari_idx` (`idUsuari`),
  CONSTRAINT `pertanyUsuari` FOREIGN KEY (`idUsuari`) REFERENCES `usuari` (`idUsuari`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=31 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

CREATE TABLE `objectemapa` (
  `tipus` int(11) NOT NULL,
  `posicioX` int(11) NOT NULL,
  `posicioY` int(11) NOT NULL,
  `idMapa` bigint(20) NOT NULL,
  `idObjecte` bigint(20) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`idObjecte`),
  KEY `pertanyMapa_idx` (`idMapa`),
  CONSTRAINT `pertanyMapa` FOREIGN KEY (`idMapa`) REFERENCES `mapa` (`idMapa`) ON DELETE CASCADE ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=204 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;
```

```
CREATE TABLE `robot` (
  `idRobot` bigint(20) NOT NULL AUTO_INCREMENT,
  `nom` varchar(45) DEFAULT NULL,
  `posicioInicialX` int(11) DEFAULT NULL,
  `posicioInicialY` int(11) DEFAULT NULL,
  `direccioInicial` int(11) DEFAULT NULL,
  `idMapa` bigint(20) NOT NULL,
  `codi` longtext,
  PRIMARY KEY (`idRobot`),
  KEY `pertanyMapa_idx` (`idMapa`),
  CONSTRAINT `pertanyMapa2` FOREIGN KEY (`idMapa`) REFERENCES `mapa` (`idMapa`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=22 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

CREATE TABLE `usuari` (
  `idUsuari` bigint(20) NOT NULL AUTO_INCREMENT,
  `password` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `nom` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`idUsuari`),
  UNIQUE KEY `idUsuari_UNIQUE` (`idUsuari`),
  UNIQUE KEY `email_UNIQUE` (`email`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;
```