

RoboCup

1

Generado por Doxygen 1.8.1.2

Sábado, 12 de Enero de 2013 02:21:22

Índice general

1	Índice de clases	1
1.1	Jerarquía de la clase	1
2	Índice de clases	3
2.1	Lista de clases	3
3	Documentación de las clases	5
3.1	Referencia de la Clase Datos	5
3.1.1	Descripción detallada	6
3.1.2	Documentación de las funciones miembro	6
3.1.2.1	getCiclo	6
3.1.2.2	getHearInst	6
3.1.2.3	getJugadorContrario	7
3.1.2.4	getJugadorPropio	7
3.1.2.5	getNumero	7
3.1.2.6	getSeelInst	7
3.1.2.7	getSenseInst	8
3.1.2.8	getSide	8
3.1.2.9	getultimaCadena	8
3.1.2.10	setCiclo	9
3.1.2.11	setDatosHear	9
3.1.2.12	setDatosHear	9
3.1.2.13	setDatosSee	9
3.1.2.14	setDatosSeePlayer	10
3.1.2.15	setDatosSense	11
3.1.2.16	setFalseHearEvent	11
3.1.2.17	setInit	12
3.1.2.18	setSide	12
3.1.2.19	setValor	12
3.2	Referencia de la Clase HearVar	13
3.2.1	Descripción detallada	13
3.2.2	Documentación de las funciones miembro	13

3.2.2.1	getinstruction	13
3.2.2.2	getside	13
3.2.2.3	setdata	13
3.3	Referencia de la Clase Jugador	14
3.3.1	Descripción detallada	16
3.3.2	Documentación del constructor y destructor	16
3.3.2.1	Jugador	16
3.3.3	Documentación de las funciones miembro	16
3.3.3.1	buscarIndicador	16
3.3.3.2	buscarPelota	17
3.3.3.3	colocar	17
3.3.3.4	comprobar_pelota	18
3.3.3.5	correr	18
3.3.3.6	getalpha	18
3.3.3.7	getnumero	19
3.3.3.8	getPosicion	19
3.3.3.9	girar	19
3.3.3.10	golpear	19
3.3.3.11	goTo	20
3.3.3.12	goTo	22
3.3.3.13	hearReferee	23
3.3.3.14	interpretar	24
3.3.3.15	mirar	25
3.3.3.16	mover	25
3.3.3.17	posicionPelota	25
3.3.3.18	seguirPelota	26
3.3.3.19	setalpha	26
3.3.3.20	setnumero	26
3.3.3.21	setPosicion	26
3.3.4	Documentación de los datos miembro	29
3.3.4.1	msg	29
3.4	Referencia de la Clase Parser	29
3.4.1	Descripción detallada	29
3.4.2	Documentación de las funciones miembro	29
3.4.2.1	leerparentesis	29
3.4.2.2	separardatos	30
3.5	Referencia de la Clase Portero	34
3.5.1	Descripción detallada	34
3.6	Referencia de la Clase Punto	35
3.6.1	Descripción detallada	35

3.7	Referencia de la Clase SeePlayer	35
3.7.1	Descripción detallada	35
3.7.2	Documentación de las funciones miembro	35
3.7.2.1	getDorsal	35
3.7.2.2	getEquipo	36
3.7.2.3	setDorsal	36
3.7.2.4	setEquipo	36
3.8	Referencia de la Clase SeeVar	37
3.8.1	Descripción detallada	37
3.8.2	Documentación de las funciones miembro	37
3.8.2.1	getBodydir	37
3.8.2.2	getDir	38
3.8.2.3	getDirchng	38
3.8.2.4	getDistchng	38
3.8.2.5	getHeaddir	39
3.8.2.6	setBodydir	39
3.8.2.7	setDir	39
3.8.2.8	setDirchng	39
3.8.2.9	setDist	40
3.8.2.10	setDistchng	40
3.8.2.11	setHeaddir	40
3.9	Referencia de la Clase SenseVar	40
3.9.1	Descripción detallada	41
3.9.2	Documentación de las funciones miembro	41
3.9.2.1	getQuality	41
3.9.2.2	getValue1	41
3.9.2.3	getValue2	41
3.9.2.4	getWidth	42
3.9.2.5	setQuality	42
3.9.2.6	setValue	42
3.9.2.7	setWidth	42
3.10	Referencia de la Clase Socket	43
3.10.1	Descripción detallada	43
3.10.2	Documentación de las funciones miembro	44
3.10.2.1	bufferChar2String	44
3.10.2.2	bufferString2Char	44
3.10.2.3	getSbuffer	44
3.10.2.4	print	44
3.10.2.5	setSbuffer	44
3.10.3	Documentación de los datos miembro	45

3.10.3.1	ccbuffer	45
3.10.3.2	n	45
3.10.3.3	sbuffer	45
3.11	Referencia de la Clase tcp	45
3.11.1	Descripción detallada	45
3.11.2	Documentación de los datos miembro	46
3.11.2.1	serv_addr	46
3.12	Referencia de la Clase TCPclient	46
3.12.1	Descripción detallada	46
3.12.2	Documentación de las funciones miembro	46
3.12.2.1	checkConnection	46
3.12.2.2	enviar	47
3.12.2.3	recibir	47
3.12.2.4	socket	47
3.13	Referencia de la Clase TCPserver	48
3.13.1	Descripción detallada	48
3.14	Referencia de la Clase Thread	48
3.14.1	Descripción detallada	49
3.14.2	Documentación de las funciones miembro	49
3.14.2.1	thread_function	49
3.15	Referencia de la Clase ThreadEnviar	50
3.15.1	Descripción detallada	50
3.16	Referencia de la Clase ThreadRecibir	50
3.16.1	Descripción detallada	51
3.17	Referencia de la Clase Udp	51
3.17.1	Descripción detallada	51
3.18	Referencia de la Clase UDPclient	51
3.18.1	Descripción detallada	52
3.19	Referencia de la Clase UDPserver	52
3.19.1	Descripción detallada	52

Capítulo 1

Índice de clases

1.1. Jerarquía de la clase

Esta lista de herencias esta ordenada aproximadamente por orden alfabético:

Datos	5
HearVar	13
Jugador	14
Portero	34
Parser	29
Punto	35
SeeVar	37
SeePlayer	35
SenseVar	40
Socket	43
tcp	45
TCPclient	46
TCPserver	48
Udp	51
UDPclient	51
UDPserver	52
Thread	48
ThreadEnviar	50
ThreadRecibir	50

Capítulo 2

Índice de clases

2.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Datos	5
HearVar	13
Jugador	14
Parser	29
Portero	34
Punto	35
SeePlayer	35
SeeVar	37
SenseVar	40
Socket	43
tcp	45
TCPclient	46
TCPserver	48
Thread	48
ThreadEnviar	50
ThreadRecibir	50
Udp	51
UDPclient	51
UDPserver	52

Capítulo 3

Documentación de las clases

3.1. Referencia de la Clase Datos

Métodos públicos

- void `setCiclo` (string _ciclo)
Datos::setCiclo.
- void `setValor` (string dato)
Datos::setValor Introducimos los valores de la instrucción.
- void `setDatosSee` (string instruccion, vector< string > valor)
Datos::setDatosSee Asignamos los valores de la instrucción See.
- void `setDatosSeePlayer` (string _equipo, string _dorsal, vector< string > valor)
Datos::setDatosSeePlayer Asignamos los valores de los jugadores que vemos.
- void `setDatosSense` (string instruccion, vector< string > valor)
Datos::setDatosSense Asignamos los valores de estado del propio jugador.
- void `setDatosHear` (string who, vector< string > data, string cadena)
Datos::setDatosHear Analizamos lo que oyen los jugadores.
- void `setDatosHear` (string who, string cadena)
Datos::setDatosHear.
- void `setInit` (vector< string > initData)
Datos::setInit Introducimos los valores iniciales dados por el servidor.
- int `getNumero` ()
Datos::getNumero.
- int `getCiclo` ()
Datos::getCiclo.
- void `setTrueHearEvent` ()
Datos::setTrueHearEvent Ponemos la variable hearEvent a true.
- void `setFalseHearEvent` ()
Datos::setFalseHearEvent Ponemos la variable hearEvent a false.
- void `setSide` (string _side)
Datos::setSide Asignamos el lado del campo al que pertenece el equipo de cada jugador.
- string `getSide` ()
Datos::getSide.
- string `getultimaCadena` ()
Datos::getultimaCadena.
- `SeePlayer` `getJugadorPropio` (int dorsal)
Datos::getJugadorPropio Acceso a la información de un jugador de nuestro equipo.

- [SeePlayer getJugadorContrario](#) (int dorsal)
Datos::getJugadorContrario Acceso a la información de un jugador del equipo contrario.
- [SeeVar getSeeInst](#) (string instruccion)
Datos::getSeeInst.
- [SenseVar getSenseInst](#) (string instruccion)
Datos::getSenseInst.
- [HearVar getHearInst](#) (string instruccion)
Datos::getHearInst.

Atributos públicos

- vector< string > [valores](#)
Vector donde se almacenan los valores de una instrucción.
- bool [hearEvent](#)
Nos dice si se ha producido algún Hear.

3.1.1. Descripción detallada

Definición en la línea 16 del archivo datos.h.

3.1.2. Documentación de las funciones miembro

3.1.2.1. int Datos::getCiclo ()

[Datos::getCiclo.](#)

Devuelve

Devolvemos el número del ciclo.

Definición en la línea 239 del archivo datos.cpp.

```
{
    return ciclo;
}
```

3.1.2.2. HearVar Datos::getHearInst (string instruccion)

[Datos::getHearInst.](#)

Parámetros

<i>instruccion</i>	Instrucción deseada.
--------------------	----------------------

Devuelve

Devuelve la información de una determinada instrucción de tipo Sense.

Definición en la línea 319 del archivo datos.cpp.

```
{
    return Hearinst[instruccion];
}
```

3.1.2.3. SeePlayer Datos::getJugadorContrario (int *dorsal*)

[Datos::getJugadorContrario](#) Acceso a la información de un jugador del equipo contrario.

Parámetros

<i>dorsal</i>	Número del jugador.
---------------	---------------------

Devuelve

Devuelve el jugador deseado.

Definición en la línea 289 del archivo datos.cpp.

```
{  
    return Seeplayer.at(2).at(dorsal);  
}
```

3.1.2.4. SeePlayer Datos::getJugadorPropio (int *dorsal*)

[Datos::getJugadorPropio](#) Acceso a la información de un jugador de nuestro equipo.

Parámetros

<i>dorsal</i>	Número del jugador.
---------------	---------------------

Devuelve

Devuelve el jugador deseado.

Definición en la línea 279 del archivo datos.cpp.

```
{  
    return Seeplayer.at(1).at(dorsal);  
}
```

3.1.2.5. int Datos::getNumero ()

[Datos::getNumero](#).

Devuelve

Devolvemos el número del jugador.

Definición en la línea 230 del archivo datos.cpp.

```
{  
    return numero;  
}
```

3.1.2.6. SeeVar Datos::getSeelInst (string *instruccion*)

[Datos::getSeelInst](#).

Parámetros

<i>instruccion</i>	Instrucción deseada.
--------------------	----------------------

Devuelve

Devuelve la información de una determinada instrucción de tipo See.

Definición en la línea 299 del archivo datos.cpp.

```
{  
    return Seeinst[instruccion];  
}
```

3.1.2.7. SenseVar Datos::getSenseInst (string *instruccion*)

[Datos::getSenseInst.](#)

Parámetros

<i>instruccion</i>	Instrucción deseada.
--------------------	----------------------

Devuelve

Devuelve la información de una determinada instrucción de tipo Sense.

Definición en la línea 309 del archivo datos.cpp.

```
{  
    return Senseinst[instruccion];  
}
```

3.1.2.8. string Datos::getSide ()

[Datos::getSide.](#)

Devuelve

Devolvemos el lado del campo al que pertenece el equipo de cada jugador.

Definición en la línea 257 del archivo datos.cpp.

```
{  
    return side;  
}
```

3.1.2.9. string Datos::getultimaCadena ()

[Datos::getultimaCadena.](#)

Devuelve

Devuelve la última cadena interpretada por el parser.

En ocasiones es algo muy concreto que nos interese de la cadena. Se utiliza en la función hearReferee en la clase [Jugador](#).

Definición en la línea 269 del archivo datos.cpp.

```
{  
    return ultimaCadena;  
}
```

3.1.2.10. void Datos::setCiclo (string _ciclo)

[Datos::setCiclo.](#)

Parámetros

<i>_ciclo</i>	
---------------	--

Definición en la línea 176 del archivo datos.cpp.

```
{
    ciclo = atoi(_ciclo.c_str());
}
```

3.1.2.11. void Datos::setDatosHear (string who, vector< string > data, string cadena)

[Datos::setDatosHear](#) Analizamos lo que oyen los jugadores.

Parámetros

<i>who</i>	Emisor del mensaje.
<i>data</i>	
<i>cadena</i>	

Definición en la línea 150 del archivo datos.cpp.

```
{
    ultimaCadena.clear();
    ultimaCadena=cadena;
    Hearinst[who].setdata(data);
}
```

3.1.2.12. void Datos::setDatosHear (string who, string cadena)

[Datos::setDatosHear.](#)

Parámetros

<i>who</i>	
<i>cadena</i>	

Definición en la línea 162 del archivo datos.cpp.

```
{
    vector<string> data;
    data.push_back(cadena);
    data.push_back("");
    ultimaCadena.clear();
    ultimaCadena = cadena;
    Hearinst[who].setdata(data);
}
```

3.1.2.13. void Datos::setDatosSee (string instruccion, vector< string > valor)

[Datos::setDatosSee](#) Asignamos los valores de la instrucción See.

Parámetros

<i>instruccion</i>	Contiene el nombre del objeto que vemos.
<i>valor</i>	Vector que contiene los diferentes valores del objeto.

La funcion setDatosSee recibe un string con la informacion despues del parseo y distribuye los valores en la clase see. Siempre que reciba un valor recibe tambien la instruccion a la que pertenece para encontrarla en el map. < Variable auxiliar que facilita la conversión de string a double

Definición en la línea 25 del archivo datos.cpp.

```
{
    double j;

    string cadena; ///< Variable auxiliar que facilita la conversión de string
                  a double
    if (valor.size()>0){
        for (int i=0; i<valor.size();i++)
        {
            cadena=valor.at(i);
            j=atof(cadena.c_str());
            switch (i)
            {
                case 0:
                    Seeinst[instruccion].setDist(j);
                    break;
                case 1:
                    Seeinst[instruccion].setDir (j);
                    break;
                case 2:
                    Seeinst[instruccion].setDistchnng (j);
                    break;
                case 3:
                    Seeinst[instruccion].setDirchnng (j);
                    break;
                case 4:
                    Seeinst[instruccion].setBodydir (j);
                    break;
                case 5:
                    Seeinst[instruccion].setHeaddir (j);
                    break;
            }
        }
    }
}
```

3.1.2.14. void Datos::setDatosSeePlayer (string _equipo, string _dorsal, vector< string > valor)

[Datos::setDatosSeePlayer](#) Asignamos los valores de los jugadores que vemos.

Parámetros

<code>_equipo</code>	Nombre del equipo del jugador.
<code>_dorsal</code>	Número del jugador.
<code>valor</code>	Vector con los valores de posición y velocidad del jugador.

La funcion setDatosSeePlayer recibe un string con la informacion del equipo, otro con el dorsal del jugador y un vector con la información que nos otorga el see de cada jugador, y con estos datos crea y modifica el vector donde se almacenan todos los jugadores.

Definición en la línea 72 del archivo datos.cpp.

```
{
    double j;
    int equipo, dorsal;

    string cadena; //Variable auxiliar que facilita la conversión de string a
                  double

    if (_equipo.find("desconocido") != string::npos)
        equipo=0;
    else
        if (_equipo.find("Grupo_3") != string::npos)
            equipo=1;
        else
            equipo=2;
    if (_dorsal.find("desconocido") != string::npos)
        dorsal = 0;
    else
        dorsal=atoi(_dorsal.c_str());
}
```



```

Seeplayer.at(equipo).at(dorsal).setDorsal(dorsal);
Seeplayer.at(equipo).at(dorsal).setEquipo(equipo);

if (valor.size()>0)
{
    for (int i=0; i<valor.size();i++)
    {
        cadena.clear();
        cadena=valor.at(i);
        j=atoi(cadena.c_str());
        switch (i)
        {
            case 0:
                Seeplayer.at(equipo).at(dorsal).setDist(j);
                break;
            case 1:
                Seeplayer.at(equipo).at(dorsal).setDir (j);
                break;
            case 2:
                Seeplayer.at(equipo).at(dorsal).setDistchnng (j);
                break;
            case 3:
                Seeplayer.at(equipo).at(dorsal).setDirchnng (j);
                break;
        }
    }
}
}

```

3.1.2.15. void Datos::setDatosSense (string *instruccion*, vector< string > *valor*)

[Datos::setDatosSense](#) Asignamos los valores de estado del propio jugador.

Parámetros

<i>instruccion</i>	Contiene el nombre de la propiedad a evaluar.
<i>valor</i>	Contiene los valores de cada propiedad.

La función setDatosSense recibe un string con el nombre de la instrucción y un vector con los datos a guardar. La instrucción es el indicador del map donde se guarda la información para poder encontrarla después.

Definición en la línea 130 del archivo datos.cpp.

```

{
    if (instruccion.compare("view_mode")==0) //
    {
        Senseinst[instruccion].setQuality (valor[0]);

        Senseinst[instruccion].setWidth (valor[1]);
    }
    else
        Senseinst[instruccion].setValue(valor);
}

```

3.1.2.16. void Datos::setFalseHearEvent ()

[Datos::setFalseHearEvent](#) Ponemos la variable hearEvent a false.

Si leemos una cadena de tipo hear ponemos a 1 hearEvent

Definición en la línea 191 del archivo datos.cpp.

```

{
    hearEvent = false;
}

```

3.1.2.17. void Datos::setInit (vector< string > *initData*)

[Datos::setInit](#) Introducimos los valores iniciales dados por el servidor.

Parámetros

<i>initData</i>	Vector donde se almacenan los valores iniciales.
-----------------	--

En esta función almacenamos los valores iniciales que nos proporciona el servidor como son el dorsal y el lado del campo en el que jugamos. Si por lo que sea no recibimos ningún lado éste se guardará como "desconocido".

Definición en la línea 216 del archivo datos.cpp.

```
{
    side.clear();
    numero=atoi(initData.at(0).c_str());
    if (initData.at(1) != "l" && initData.at(1) != "r")
        side = "desconocido";
    else
        side = initData.at(1);
}
```

3.1.2.18. void Datos::setSide (string *_side*)

[Datos::setSide](#) Asignamos el lado del campo al que pertenece el equipo de cada jugador.

Si la cadena no es de tipo hear hearEvent se pone a 0

Parámetros

<i>_side</i>	Lado del campo.
--------------	-----------------

Definición en la línea 248 del archivo datos.cpp.

```
{
    side=_side;
}
```

3.1.2.19. void Datos::setValor (string *dato*)

[Datos::setValor](#) Introducimos los valores de la instrucción.

Parámetros

<i>dato</i>	String con el dato a introducir.
-------------	----------------------------------

Introducimos el dato correspondiente al vector valores donde se almacenan todos los datos de una determinada intrucción. Esta función se vital para el parser.

Definición en la línea 203 del archivo datos.cpp.

```
{
    valores.push_back (dato);
}
```

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/robocup/datos.h
- /home/carlos/info/RoboCup/src/libraries/robocup/datos.cpp

3.2. Referencia de la Clase HearVar

Métodos públicos

- void [setdata](#) (std::vector< std::string > data)
[HearVar::setdata](#) Introducimos los distintos datos de una instrucción tipo Hear.
- std::string [getinstruction](#) ()
[HearVar::getinstruction](#).
- std::string [getside](#) ()
[HearVar::getside](#).

3.2.1. Descripción detallada

Definición en la línea 6 del archivo hearVar.h.

3.2.2. Documentación de las funciones miembro

3.2.2.1. std::string HearVar::getinstruction ()

[HearVar::getinstruction](#).

Devuelve

Devuelve la instrucción de tipo Hear.

Definición en la línea 19 del archivo hearVar.cpp.

```
{
    return instruction;
}
```

3.2.2.2. std::string HearVar::getside ()

[HearVar::getside](#).

Devuelve

Devuelve el lado del campo.

Definición en la línea 28 del archivo hearVar.cpp.

```
{
    return side;
}
```

3.2.2.3. void HearVar::setdata (std::vector< std::string > data)

[HearVar::setdata](#) Introducimos los distintos datos de una instrucción tipo Hear.

Parámetros

<i>data</i>	Datos a introducir.
-------------	-------------------------------------

Definición en la línea 7 del archivo hearVar.cpp.

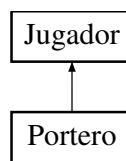
```
{
    instruction.clear();
    side.clear();
    instruction=data.at(0);
    side=data.at(1);
}
```

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/robocup/hearVar.h
- /home/carlos/info/RoboCup/src/libraries/robocup/hearVar.cpp

3.3. Referencia de la Clase Jugador

Diagrama de herencias de Jugador



Métodos públicos

- **Jugador** ()
Jugador::Jugador Constructor necesario para la realización de los tests.
- **Jugador** (int argc, char *argv[])
Jugador::Jugador Constructor de nuestro jugador.
- int **getnumero** ()
Jugador::getnumero.
- void **setnumero** (int Nnumero)
Jugador::setnumero Introducimos el dorsal al jugador.
- double **getalpha** ()
Jugador::getalpha.
- void **setalpha** (double num)
Jugador::setalpha Asignamos valor al ángulo del jugador.
- void **mover** (float pot)
Jugador::mover Función para mover a nuestro jugador.
- void **girar** (float ang)
Jugador::girar Función para girar el cuerpo de nuestro jugador.
- bool **golpear** (float pot, float dir)
Jugador::golpear Función para golpear la pelota.
- void **mirar** (float ang)
Jugador::mirar Función para girar la cabeza de nuestro jugador.
- void **interpretar** (string h)
Jugador::interpretar Interpretamos los datos recibidos desde el servidor.
- bool **buscarPelota** ()
Jugador::buscarPelota Función para buscar la pelota y poner al jugador mirando hacia él.
- void **seguirPelota** (bool balonzonaJugador)
Jugador::seguirPelota Función para perseguir la pelota si ésta está dentro de la zona de juego del jugador y golpearla cuando se de el caso.
- void **seguirPelota** ()

- Jugador::seguirPelota* Función para perseguir la pelota esté donde esté y golpearla cuando se de el caso.
- void **colocar** (double x, double y)
 - Jugador::colocar* Función para colocar nuestros jugadores por el campo al inicio del partido.
- void **comprobar_pelota** (string h)
 - Jugador::comprobar_pelota* Comprobamos si el jugador ve la pelota.
- void **correr** ()
 - Jugador::correr* Función para correr con la pelota dando pequeños golpes.
- void **buscarIndicador** (string indicador)
 - Jugador::buscarIndicador* Función para buscar algún indicador de la cadena See y mirar hacia él.
- void **esperarInicio** ()
 - Jugador::esperarInicio* Todos los jugadores menos uno esperan a que éste saque de centro del campo.
- void **setPosicion** (string h)
 - Jugador::setPosicion* Calculamos la posición absoluta de nuestro jugador en el campo en coordenadas cartesianas.
- **Punto** **getPosicion** ()
 - Jugador::getPosicion.*
- void **goTo** (**Punto** p)
 - Jugador::goTo* Función para mover nuestro jugador a cualquier punto del campo que queramos.
- void **hearReferee** ()
 - Jugador::hearReferee* Función que interpreta los mensajes del árbitro.
- void **goTo** (double x, double y)
 - Jugador::goTo* Función para mover nuestro jugador a cualquier punto del campo que queramos.
- void **posicionPorteria** ()
 - Jugador::posicionPorteria* Calculamos la dirección de la portería con respecto a nuestro jugador.
- bool **distanciaJugadoresPropios** (float distancia)
- bool **jugadorPropioCercaBalon** ()
- bool **distanciaJugadoresContrarios** (float distancia)
- **Punto** **posicionJugadorPropio** (int dorsal)
- **Punto** **posicionPelota** ()
 - Jugador::posicionPelota* Calculamos la posición absoluta de la pelota en el campo.

Atributos públicos

- clock_t **te**
- clock_t **tr**
- bool **veoPelota**
 - Booleano que dice si el jugador está viendo la pelota o no.
- bool **cercaLinea**
 - Booleano que dice si el jugador está cerca de alguna línea del borde del campo.
- **Punto** **posicion**
 - Posición de nuestro jugador.
- **Punto** **pos_inicial**
 - Posición inicial donde se coloca nuestro jugador.
- **Punto** **pos_play**
 - Posición donde se coloca nuestro jugador durante el partido.
- **UDPClient** **cliente**
 - Variable de tipo socket UDP.
- **Thread****Enviar** **enviar**
 - Variable *Thread* para enviar información.
- **Thread****Recibir** **recibir**
 - Variable *Thread* para recibir información.
- **Parser** **parser**

Variable de tipo parser.

- [Datos datos](#)

Aquí almacenamos nuestros datos.

- `int` [refereeMode](#)

Modo de juego dictado por el árbitro.

- `bool` [thread_flag](#)

Flag necesario para sincronizar los flags.

- `pthread_mutex_t` [_mutex](#)

Mutex necesario para sincronizar los flags.

- `Jugador *` [jugador](#)

Necesario para el thread.

- `stringstream` [msg](#)

3.3.1. Descripción detallada

Definición en la línea 22 del archivo jugador.h.

3.3.2. Documentación del constructor y destructor

3.3.2.1. `Jugador::Jugador (int argc, char * argv[])`

[Jugador::Jugador](#) Constructor de nuestro jugador.

Parámetros

<i>argc</i>	Número de parametros que tiene la cadena que pasamos por valor.
<i>argv</i>	Array donde se almacenan los parámetros. Los parametros son la dirección del servidor y el puerto.

Definición en la línea 46 del archivo jugador.cpp.

```
{
    cicloGolpeo = 0;
    te = clock();
    tr = clock();
    thread_flag=false;
    veoPelota=false;
    pthread_mutex_init (&this->_mutex, (pthread_mutexattr_t*) NULL);
    cliente.checkConnection(argc);
    cliente.socket(argv);
    recibir.start(this);
    sleep(0.5);
    cliente.enviar("(init Grupo_3)");
    thread_flag = false;
    sleep(2);
    thread_flag = false;
    numero = datos.getNumero();
    cout << "Jugador con el dorsal " << numero << " juega en el lado " << datos
        .getSide() << endl;
}
```

3.3.3. Documentación de las funciones miembro

3.3.3.1. `void Jugador::buscarIndicador (string indicador)`

[Jugador::buscarIndicador](#) Función para buscar algún indicador de la cadena See y mirar hacia él.

Parámetros

<i>indicador</i>	Nombre del indicador que queremos buscar.
------------------	---

Definición en la línea 340 del archivo jugador.cpp.

```
{
    while (true)
    {
        if(cliente.getSbuffer().find("see")!=string::npos &&
            cliente.getSbuffer().find(indicador)!=string::npos)
        {
            while (5 < datos.getSeeInst(indicador).getDir(
            ) < -5)
            {
                girar(datos.getSeeInst(indicador).getDir
                ());
                usleep(15000);
            }
            break;
        }
        else
        {
            girar(20);
            usleep(15000);
        }
    }
}
```

3.3.3.2. bool Jugador::buscarPelota ()

Jugador::buscarPelota Función para buscar la pelota y poner al jugador mirando hacia él.

Devuelve

La funcion buscar pelota devuelve FALSE si no esta viéndola y TRUE si la ve

Definición en la línea 255 del archivo jugador.cpp.

```
{
    if (veoPelota)
    {
        if (-5 < datos.getSeeInst("ball").getDir() &&
            datos.getSeeInst("ball").getDir() < 5)
            return true;
        else
        {
            //girar (datos.getSeeInst("ball").getDir());
            if (datos.getSeeInst("ball").getDir()<0)
            {
                girar (-5);
                usleep(15000);
            }
            else
            {
                girar (5);
                usleep(15000);
            }
        }
        return true;
    }
    else
    {
        girar(20);
        usleep(15000);
        return false;
    }
}
```

3.3.3.3. void Jugador::colocar (double x, double y)

Jugador::colocar Función para colocar nuestros jugadores por el campo al inicio del partido.

Parámetros

x	Coordenada X de la posición del jugador.
y	Coordenada Y de la posición del jugador.

Definición en la línea 239 del archivo jugador.cpp.

```
{
    msg.str("");
    if(x<(-52.5) || x>(52.5))
        cout<<"Imposible colocar el jugador, coordenada X incorrecta"<<endl;
    if(y<-34 || y>34)
        cout<<"Imposible colocar el jugador, coordenada Y incorrecta"<<endl;
    msg<<"move "<<x<<" "<<y<<" ";
    pos_inicial.setX(x);
    pos_inicial.setY(y);
}
```

3.3.3.4. void Jugador::comprobar_pelota (string h)

[Jugador::comprobar_pelota](#) Comprobamos si el jugador ve la pelota.

Parámetros

<i>h</i>	Cadena recibida desde el servidor.
----------	------------------------------------

Definición en la línea 106 del archivo jugador.cpp.

```
{
    if (h.find("see")!=string::npos && h.find("ball")!=string::npos)
        veoPelota=true;
    else
        if (h.find("see")!=string::npos && h.find("ball")==string::npos)
            veoPelota=false;
}
```

3.3.3.5. void Jugador::correr ()

[Jugador::correr](#) Función para correr con la pelota dando pequeños golpes.

En esta primera versión no se utiliza.

Definición en la línea 324 del archivo jugador.cpp.

```
{
    if (buscarPelota())
    {
        if(golpear(20,dirPorteria))
            usleep(15000);
        else
            mover(70);
        usleep(15000);
    }
}
```

3.3.3.6. double Jugador::getalpha ()

[Jugador::getalpha](#).

Devuelve

Devolvemos el ángulo del jugador.

Definición en la línea 98 del archivo jugador.cpp.

```
{
    return alpha;
}
```


3.3.3.7. int Jugador::getnumero ()[Jugador::getnumero.](#)**Devuelve**

Devolvemos el número del jugador.

Ver también

Definición en la línea 80 del archivo jugador.cpp.

```
{  
    return numero;  
}
```

3.3.3.8. Punto Jugador::getPosicion ()[Jugador::getPosicion.](#)**Devuelve**

Devuelve la posición del jugador.

Definición en la línea 602 del archivo jugador.cpp.

```
{  
    return posicion;  
}
```

3.3.3.9. void Jugador::girar (float ang)[Jugador::girar](#) Función para girar el cuerpo de nuestro jugador.**Parámetros**

<i>ang</i>	Ángulo que va a girar el jugador.
------------	-----------------------------------

Según el manual de la RoboCup el parámetro del comando "turn" no es un ángulo como tal sino un momento que se calcula en función de parametros dados por el servidor.

Definición en la línea 214 del archivo jugador.cpp.

```
{  
    double momento;  
    momento=ang*(1.0+5*(datos.getSenseInst("speed").getValue1  
        ( )));  
    msg.str("");  
    momento = checkAngulo(momento);  
    msg<<"turn "<<momento<<" ";  
}
```

3.3.3.10. bool Jugador::golpear (float pot, float dir)[Jugador::golpear](#) Función para golpear la pelota.

Parámetros

<i>pot</i>	Potencia con la que golpear la pelota.
<i>dir</i>	Dirección en la que queremos mandar la pelota.

Devuelve

Si las condiciones para golpear la pelota han sido favorables la función devuelve TRUE.

Definición en la línea 175 del archivo jugador.cpp.

```
{
    msg.str("");
    float potencia = checkPotencia(pot);
    float direccion = checkAngulo(dir);
    if (cicloGolpeo == datos.getCiclo()) //Evitamos golpear 2
        veces en el mismo ciclo
        return false;
    else
        if (veoPelota && 0 < datos.getSeeInst("ball").
            getDist() && datos.getSeeInst("ball").getDist() < 0.7)
        {
            cout<<"Golpeo con una potencia: "«potencia<<endl;
            msg << "(kick " « potencia « " " « direccion « " ";
            usleep(15000);
            cicloGolpeo = datos.getCiclo();
            return true;
        }
        else
            return false;
}
```

3.3.3.11. void Jugador::goTo (Punto p)

[Jugador::goTo](#) Función para mover nuestro jugador a cualquier punto del campo que queramos.

Parámetros

<i>p</i>	Variable de tipo PUNTO donde asignamos la posición de destino.
----------	--

Esta función nos permite llevar a nuestro jugador a cualquier punto del campo, en ella tenemos en cuenta en todo momento la posición de la pelota y si nos la encontramos de camino dejamos de ir a nuestro destino y salimos de la función para ir hacia la pelota. Posición relativa a nosotros

Distancia relativa a nosotros

Vamos a la posición deseada mientras no tengamos la pelota cerca

Posición relativa a nosotros

Distancia relativa a nosotros

Angulo relativo a nosotros en grados

Definición en la línea 615 del archivo jugador.cpp.

```
{
    double dist,angle,rel_x,rel_y,alpha2,rel_angle, ciclo;
    ciclo = datos.getCiclo();
    /** Posición relativa a nosotros*/
    rel_x= p.getX() -posicion.getX();
    rel_y= p.getY() -posicion.getY();

    /** Distancia relativa a nosotros*/
    dist=sqrt((rel_x*rel_x)+(rel_y*rel_y));
    /** Vamos a la posición deseada mientras no tengamos la pelota cerca */

    angle = acos(rel_x/dist)*180/3.141592;

    if (rel_y>0)
        angle = 360 - angle;
```

```

rel_angle=angle-alpha2;

if (rel_angle<0)
    rel_angle=360+rel_angle;

girar(rel_angle);

while(dist>2 && datos.getCiclo()-ciclo < 100)
{
    if ((veoPelota==true && datos.getSeeInst("ball"
).getDist()<10) || datos.hearEvent==true)
    {
        break;
    }
    else
    {
        alpha2=alpha;

        /** Posición relativa a nosotros*/
        rel_x= p.getX() - posicion.getX();
        rel_y= p.getY() - posicion.getY();

        /** Distancia relativa a nosotros*/
        dist=sqrt((rel_x*rel_x)+(rel_y*rel_y));

        /** Angulo relativo a nosotros en grados*/

        angle = acos(rel_x/dist)*180/3.141592;
        if (rel_y>0)
            angle = 360 - angle;

        rel_angle=angle-alpha2;
        if (rel_angle<0)
            rel_angle=360+rel_angle;

        // cout<<"x= "«posicion.getX()«" , y= "«posicion.getY()«"
        ,alpha= "«alpha«endl;
        // cout<<"angle= "«angle«" , dist= "«dist «" , rel_x=
        "«rel_x«" , rel_y= "«rel_y«endl;
        // cout<<"angulo relativo = "« rel_angle «endl;

        if (cercaLinea || posicion.getX()< -60 ||
posicion.getX()>60 || posicion.getY()<-50 || posicion.getY()>50
        )
        {
            //girar(45);
            //buscarPelota();
            //usleep(15000);
            cercaLinea = false;
            break;
        }

        if (rel_angle< 355 && rel_angle > 5)
        {
            if (rel_angle <=180)
            {
                girar(-5);
                usleep(15000);
                // mover(80);
                // usleep(15000);

            }
            if (rel_angle >180)
            {
                girar(5);
                usleep(15000);
                //mover(80);
                //usleep(15000);

            }
        }
        else
        {
            mover(80);
            //cout<<"me nuevo"«endl;
            usleep(15000);

        }

    }
}

cout<<"He llegado"«endl;
mover(0);
usleep(15000);
buscarPelota();
}

```

3.3.3.12. void Jugador::goTo (double x, double y)

Jugador::goTo Función para mover nuestro jugador a cualquier punto del campo que queramos.

Parámetros

x	Coordenada X de la posición de destino.
y	Coordenada Y de la posición de destino.

Esta función nos permite llevar a nuestro jugador a cualquier punto del campo, en ella tenemos en cuenta en todo momento la posición de la pelota y si nos la encontramos de camino dejamos de ir a nuestro destino y salimos de la función para ir hacia la pelota. Posición relativa a nosotros

Distancia relativa a nosotros

Vamos a la posición deseada mientras no tengamos la pelota cerca

Posición relativa a nosotros

Distancia relativa a nosotros

Angulo relativo a nosotros en grados

Definición en la línea 725 del archivo jugador.cpp.

```
{
    double dist,angle,rel_x,rel_y,alpha2,rel_angle, ciclo;
    ciclo = datos.getCiclo();
    /** Posición relativa a nosotros*/
    rel_x= x -posicion.getX();
    rel_y= y -posicion.getY();

    /** Distancia relativa a nosotros*/
    dist=sqrt((rel_x*rel_x)+(rel_y*rel_y));
    /** Vamos a la posición deseada mientras no tengamos la pelota cerca */

    angle = acos(rel_x/dist)*180/3.141592;

    if (rel_y>0)
        angle = 360 - angle;

    rel_angle=angle-alpha2;

    if (rel_angle<0)
        rel_angle=360+rel_angle;

    girar(rel_angle);

    while(dist>2 && datos.getCiclo()-ciclo < 100)
    {
        if ((veoPelota==true && datos.getSeeInst("ball"
        ).getDist(<10) || datos.hearEvent==true)
        {
            break;
        }
        else
        {
            alpha2=alpha;

            /** Posición relativa a nosotros*/
            rel_x= x - posicion.getX();
            rel_y= y - posicion.getY();

            /** Distancia relativa a nosotros*/
            dist=sqrt((rel_x*rel_x)+(rel_y*rel_y));

            /** Angulo relativo a nosotros en grados*/

            angle = acos(rel_x/dist)*180/3.141592;
            if (rel_y>0)
                angle = 360 - angle;

            rel_angle=angle-alpha2;
            if (rel_angle<0)
                rel_angle=360+rel_angle;

            // cout<<"x= "«posicion.getX()«" , y= "«posicion.getY()«"
            ,alpha= "«alpha«endl;
            // cout<<"angle= "«angle«" , dist= "«dist«" , rel_x=
            "«rel_x«" , rel_y= "«rel_y«endl;
```

```

        // cout<<"angulo relativo = "<< rel_angle <<endl;

        if (cercaLinea)
        {
            //girar(45);
            //usleep(15000);
            cercaLinea = false;
            break;
        }

        if (rel_angle< 355 && rel_angle > 5)
        {
            if (rel_angle <=180)
            {
                girar(-5);
                usleep(15000);
                //mover(80);
                //usleep(15000);

            }
            if (rel_angle >180)
            {
                girar(5);
                usleep(15000);
                //mover(80);
                //usleep(15000);

            }
        }
        else
        {
            mover(80);
            //cout<<"me muevo"<<endl;
            usleep(15000);
        }
    }
}
mover(0);
usleep(15000);
buscarPelota();
}

```

3.3.3.13. void Jugador::hearReferee ()

Jugador::hearReferee Función que interpreta los mensajes del árbitro.

Con esta función devolvemos un numero entero según sea la accion indicada por el arbitro. Este numero lo utilizaremos despues para colocar a los jugadores según sea necesario.

Definición en la línea 829 del archivo jugador.cpp.

```

{
    bool mySide;
    string instruction;
    if ((datos.hearEvent == true) && (datos.getultimaCadena()
        .find("referee")!=string::npos ||
        datos.getultimaCadena()
        .find("goal") != string::npos ||
        datos.getultimaCadena()
        .find("play_on") != string::npos))
    {
        //cout<<datos.getultimaCadena()<<endl;
        instruction = datos.getHearInst("referee").
        getinstruction();
        if (datos.getSide() == datos.getHearInst("
        referee").getside())
            mySide = true;
        else
            mySide = false;

        if(instruction == "play_on")
            refereeMode = 0;

        if (instruction == "kick_off")
        {
            if (mySide)
                refereeMode = 1; //1 si saca de centro nuestro
            equipo

```

```

        else
            refereeMode = 2; //2 si saca de centro el equipo
    contrario
    }

    if (instruction == "kick_in")
    {
        if (mySide)
            refereeMode = 3; //3 si saca de banda nuestro equipo
        else
            refereeMode = 4; //4 si saca de banda el equipo
    contrario
    }

    if (instruction == "free_kick")
    {
        if(mySide)
            refereeMode = 5; //5 si saca la falta mi equipo
        else
            refereeMode = 6; //6 si saca la falta el quipo
    contrario
    }

    if (instruction == "corner_kick")
    {
        if(mySide)
            refereeMode = 7; //7 si saca de corner mi equipo
        else
            refereeMode = 8; //8 si saca el corner el equio
    contrario
    }

    if (instruction == "goal_kick")
    {
        if (mySide)
            refereeMode = 9; //9 Si saca de porteria mi equipo
        else
            refereeMode = 10; //10 si saca de porteria el equipo
    contrario
    }

    if (instruction == "before_kick_off")
        refereeMode = 11; //11 si estamos en before_kick_off

    if (instruction == "offside")
    {
        if (mySide)
            refereeMode = 12; //12 si el fuera de juego es del
otro equipo
        else
            refereeMode = 13; //13 si el fuera de juego es de mi
equipo
    }

    if(instruction == "half_time")
    {
        if (datos.getSide() == "1")
            datos.setSide("r");
        else
            datos.setSide("l");
        refereeMode = 14; //14 despues de cambiar el lado de
juego en el descanso
    }
    }
}

```

3.3.3.14. void Jugador::interpretar (string h)

[Jugador::interpretar](#) Interpretamos los datos recibidos desde el servidor.

Parámetros

<i>h</i>	Cadena que vamos a interpretar enviada por el servidor.
----------	---

Definición en la línea 160 del archivo jugador.cpp.

```

{
    parser.separardatos(h,datos);

```

```

    comprobar_pelota(h);
    setPosicion(h);
    posicionPorteria();
    hearReferee();
}

```

3.3.3.15. void Jugador::mirar (float *ang*)

[Jugador::mirar](#) Función para girar la cabeza de nuestro jugador.

Parámetros

<i>ang</i>	Ángulo que giramos la cabeza del jugador.
------------	---

Definición en la línea 227 del archivo jugador.cpp.

```

{
    float angulo = checkAngulo(ang);
    msg.str("");
    msg<<"(turn_neck "«angulo»" );
}

```

3.3.3.16. void Jugador::mover (float *pot*)

[Jugador::mover](#) Función para mover a nuestro jugador.

Parámetros

<i>pot</i>	Potencia con la que aceleramos al jugador.
------------	--

Definición en la línea 199 del archivo jugador.cpp.

```

{
    float potencia = checkPotencia(pot);
    msg.str(""); //BORRAMOS EL CONTENIDO PORQUE SI NO SE AÑADE AL FINAL
    msg<<"(dash "«potencia»" );
}

```

3.3.3.17. Punto Jugador::posicionPelota ()

[Jugador::posicionPelota](#) Calculamos la posición absoluta de la pelota en el campo.

Devuelve

Devolvemos la posición de la pelota.

Definición en la línea 143 del archivo jugador.cpp.

```

{
    double x,y,angle;
    Punto pos;
    angle = alpha - datos.getSeeInst("ball").getDir() ;
    x = posicion.getX() + (datos.getSeeInst("ball").
        getDist()*cos(angle));
    y = posicion.getY() + (datos.getSeeInst("ball").
        getDist()*sin(angle));
    pos.setX(x);
    pos.setY(y);
    return pos;
}

```

3.3.3.18. void Jugador::seguirPelota (bool zonaBalon)

[Jugador::seguirPelota](#) Función para perseguir la pelota si ésta está dentro de la zona de juego del jugador y golpearla cuando se de el caso.

Parámetros

<i>zonaBalon</i>	Booleano que dicta si la pelota está dentro de la zona o no.
------------------	--

Definición en la línea 291 del archivo jugador.cpp.

```
{
    if (buscarPelota() == true && zonaBalon == true &&
        jugadorPropioCercaBalon() == false)
    {
        double velocidad = datos.getSeeInst("ball").getDist()*20
        ;
        mover(velocidad);
        usleep(15000);
    }

    if (golpear(14*sqrt(distPorteria),dirPorteria))
        usleep(15000);
}
```

3.3.3.19. void Jugador::setalpha (double num)

[Jugador::setalpha](#) Asignamos valor al ángulo del jugador.

Parámetros

<i>num</i>	Valor a asignar.
------------	------------------

Definición en la línea 89 del archivo jugador.cpp.

```
{
    alpha=num;
}
```

3.3.3.20. void Jugador::setnumero (int Nnumero)

[Jugador::setnumero](#) Introducimos el dorsal al jugador.

Parámetros

<i>Nnumero</i>	Número de dorsal.
----------------	-------------------

Definición en la línea 71 del archivo jugador.cpp.

```
{
    numero = Nnumero;
}
```

3.3.3.21. void Jugador::setPosicion (string h)

[Jugador::setPosicion](#) Calculamos la posición absoluta de nuestro jugador en el campo en coordenadas cartesianas.

Parámetros

<i>h</i>	Cadena recibida desde el servidor de dónde sacamos la información.
----------	--

Definición en la línea 446 del archivo jugador.cpp.

```
{
    double linedir, linedist, flagdir, flagdist=200, flag_x, flag_y, abs_x, abs_y,
    alpha2;
    if(h.find("see")!=string::npos)
    {
        if (h.find("(line r)")!=string::npos)
        {
            linedist = datos.getSeeInst("line r").getDist();
            linedir = 90 + datos.getSeeInst("line r").getDir
        ()
            ;

            if (datos.getSeeInst("line r").getDir()>0)

                linedir = 270 + datos.getSeeInst("line r").
getDir();
        }

        if (h.find("(line l)")!=string::npos)
        {
            linedist = datos.getSeeInst("line l").getDist();
            linedir = 90 + datos.getSeeInst("line l").getDir
        ()
            ;

            if (datos.getSeeInst("line l").getDir()<0)

                linedir = 270 + datos.getSeeInst("line l").
getDir();
        }

        if (h.find("(line t)")!=string::npos)
        {
            linedist = datos.getSeeInst("line t").getDist();
            linedir = datos.getSeeInst("line t").getDir();

            if (datos.getSeeInst("line t").getDir()<0)

                linedir = 180 + datos.getSeeInst("line t").
getDir();
        }

        if (h.find("(line b)")!=string::npos)
        {
            linedist = datos.getSeeInst("line b").getDist();
            linedir = 180 + datos.getSeeInst("line b").getDir
        ()
            ;

            if (datos.getSeeInst("line b").getDir()<0)

                linedir = 360 + datos.getSeeInst("line b").
getDir();
        }

        if (h.find("flag")!=string::npos)
        {
            if (h.find("(flag l b)")!=string::npos)
            {
                if(flagdist>datos.getSeeInst("flag l b").getDist
        ())
                {
                    flagdir= this->datos.getSeeInst("flag l b").
getDir();
                    flagdist=this->datos.getSeeInst("flag l b").
getDist();
                    flag_x=-52.5;
                    flag_y=34;
                }
            }
            if (h.find("(flag l t)")!=string::npos)
            {
                if(flagdist>datos.getSeeInst("flag l t").getDist
        ())
                {
                    flagdir= this->datos.getSeeInst("flag l t").
getDir();
                    flagdist=this->datos.getSeeInst("flag l t").
getDist();
                    flag_x=-52.5;
                    flag_y=-34;
                }
            }
            if (h.find("(flag c t)")!=string::npos)
            {
                if(flagdist>datos.getSeeInst("flag c t").getDist
        ())
```

```

        {
            flagdir= this->datos.getSeeInst("flag c t").
getDir();
            flagdist=this->datos.getSeeInst("flag c t").
getDist();
            flag_x=0;
            flag_y=-34;
        }
        if (h.find("(flag r t)")!=string::npos)
        {
            if(flagdist>datos.getSeeInst("flag r t").getDist
())
            {
                flagdir= this->datos.getSeeInst("flag r t").
getDir();
                flagdist=this->datos.getSeeInst("flag r t").
getDist();
                flag_x=52.5;
                flag_y=-34;
            }
        }
        if (h.find("(flag r b)")!=string::npos)
        {
            if(flagdist>datos.getSeeInst("flag r b").getDist
())
            {
                flagdir= this->datos.getSeeInst("flag r b").
getDir();
                flagdist=this->datos.getSeeInst("flag r b").
getDist();
                flag_x=52.5;
                flag_y=34;
            }
        }
        if (h.find("(flag c b)")!=string::npos)
        {
            if(flagdist>datos.getSeeInst("flag c b").getDist
())
            {
                flagdir= this->datos.getSeeInst("flag c b").
getDir();
                flagdist= this->datos.getSeeInst("flag c b")
.getDist();
                flag_x=0;
                flag_y=34;
            }
        }
        if (h.find("(goal l)")!=string::npos)
        {
            if(flagdist>datos.getSeeInst("goal l").getDist()
)
            {
                flagdir= this->datos.getSeeInst("goal l").
getDir();
                flagdist=this->datos.getSeeInst("goal l").
getDist();
                flag_x=-52.5;
                flag_y=0;
            }
        }
        if (h.find("(goal r)")!=string::npos)
        {
            if(flagdist>datos.getSeeInst("goal r").getDist()
)
            {
                flagdir= this->datos.getSeeInst("goal r").
getDir();
                flagdist=this->datos.getSeeInst("goal r").
getDist();
                flag_x=52.5;
                flag_y=0;
            }
        }
        if (h.find("(flag c)")!=string::npos)
        {
            if(flagdist>datos.getSeeInst("flag c").getDist()
)
            {
                flagdir= this->datos.getSeeInst("flag c").
getDir();
                flagdist=this->datos.getSeeInst("flag c").
getDist();
                flag_x=0;
                flag_y=0;
            }
        }
    }
}

```

```

    }
    alpha = linedir;
    alpha2 = (linedir - flagdir)*3.141592/180;

    abs_x=flag_x-(flagdist*(cos(alpha2)));
    abs_y=flag_y+(flagdist*(sin(alpha2)));

    posicion.setX(abs_x);
    posicion.setY(abs_y);
    if (linedist<2)
        cercaLinea = true;
}
}

```

3.3.4. Documentación de los datos miembro

3.3.4.1. stringstream Jugador::msg

Almacenamos el mensaje que vayamos a enviar desde el thread, es de tipo stringstream, el cual nos permite introducir comodamente los valores a enviar, para utilizar esta variable como string basta con llamar a "msg.str()"

Definición en la línea 58 del archivo jugador.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/robocup/jugador.h
- /home/carlos/info/RoboCup/src/libraries/robocup/jugador.cpp

3.4. Referencia de la Clase Parser

Métodos públicos

- void leerparentesis (string cad)
 - void leerespacios (string cad)
 - void separardatos (string cad, Datos &valor)
- Parser::separardatos.*

Atributos públicos

- vector< int > inicio
- vector< int > fin1
- vector< int > fin2
- vector< int > espacios
- Datos valor

3.4.1. Descripción detallada

Definición en la línea 12 del archivo parser.h.

3.4.2. Documentación de las funciones miembro

3.4.2.1. void Parser::leerparentesis (string cad)

Nos evitamos leer el primer parentesis en las cadenas mas largas

Borramos el (de inicio anterior, solo nos interesa el ultimo

Definición en la línea 5 del archivo parser.cpp.

```

{
    int i=0;
    if (cad.compare(1,3,"see")==0 || cad.compare(1,5,"sense")==0)
        i=1; /** Nos evitamos leer el primer parentesis en las cadenas mas
            largas */
    for(i;i<cad.size();i++)
    {
        if(cad.at(i)=='(')
        {
            inicio.push_back(i);
            for(int j=i+1;j<cad.size();j++)
            {
                if(cad.at(j)=='(')
                {
                    inicio.pop_back(); /** Borramos el ( de inicio anterior,
solo nos interesa el ultimo */
                    inicio.push_back(j);
                    while(cad.at(j)!='(')
                        j++;
                    fin1.push_back(j);
                    j++; /** Necesrio para no leer 2 veces la misma posicion*/
                }
                if(cad.at(j)==')')
                {
                    fin2.push_back(j);
                    j++; /** Necesrio para no leer 2 veces la misma posicion*/
                    if (j>=cad.size())
                        i=cad.size()-1; /** Comprobaremos despues la ultima
posicion otra vez*/
                    else
                        i=j;
                    break; /** Salimos del segundo bucle for */
                }
            }
        }
        if(cad.at(i)==')')
            fin2.push_back(i);
    }
}

```

3.4.2.2. void Parser::separardatos (string cad, Datos & valor)

[Parser::separardatos.](#)

Parámetros

<i>cad</i>	
<i>valor</i>	

< Variables para manejar la posicion de inicio de lectura

< Variables para manejar la longitud de lectura

< Borramos los vectores para evitar

< posibles superposiciones de datos

Leemos la cadena see si es superior a 15 caracteres

Hago tantas lecturas como parentesis tenga

Almaceamos en cadena los terminos que indican de que instruccion se trata

Copiamos en cadena la instruccion que se comparara completa

Almaceamos en comando los terminos que indican de que instruccion se trata

Almaceamos en comando el ultimo valor de la secuencia

Almaceamos en comando los valores intermedios

Almacenamos las cadenas de tipo sense body

Hago tantas lecturas como parentesis tenga

Almaceamos en cadena los terminos que indican de que instruccion se trata

Buscamos los espacios y mandamos el comando a la clase de datos

Almaceamos en comando los terminos que indican de que instruccion se trata

Almaceamos en comando el ultimo valor de la secuencia

Almaceamos en comando los valores intermedios

Almacenamos las cadenas de tipo hear

Almaceamos en 'cada' quien realiza el hear

Almaceamos en cadb los terminos que indican de que instruccion se trata

Almacenamos en cadb el lado de la instruccion

Definición en la línea 57 del archivo parser.cpp.

```
{
    string buffer,instruccion,dato; //Variable necesaria para poder buscar
    espacios, ya que tenemos un char y le pasamos un string.
    int i=0;
    int a,c; ///< Variables para manejar la posicion de inicio de lectura
    int b,d; ///< Variables para manejar la longitud de lectura
    int t;
    inicio.clear();
    fin1.clear();          ///< Borramos los vectores para evitar
    fin2.clear();          ///< posibles superposiciones de datos
    espacios.clear();

    leerparentesis(cad);
    leerespacios(cad);
    // Guardamos el ciclo de envio
    if (cad.find("init") != string::npos);
    else
    {
        if(espacios.size()<2)
            a = fin2.at(0)-espacios.at(0)-1;
        else
            a=espacios.at(1)-espacios.at(0)-1;
        b=espacios.at(0)+1;
        char ciclo[a];
        bzero(ciclo,a+1);
        cad.copy(ciclo,a,b);
        valor.setCiclo(ciclo);
    }
    if (cad.find("see")!=string::npos && cad.size()>15 ) /** Leemos la cadena
    see si es superior a 15 caracteres*/
    {
        if(cad.size()<10) cout<<"Cadena see corta"<<endl; //Si solo llega una
        instruccion de tipo (see xxxx) la deshechamos
        else
        {
            for(inicio.at(i);i<inicio.size();i++)/** Hago tantas lecturas como
            parentesis tenga */
            {
                a=fin1.at(i)-inicio.at(i)-1;
                b=inicio.at(i)+1;
                c=fin2.at(i)-fin1.at(i)-2;
                d=fin1.at(i)+2;
                char inst[a]; /** Almaceamos en cadena los terminos que indican
                de que instruccion se trata*/
                char data[c];
                bzero(inst,a+1);
                bzero(data,c+1);
                cad.copy(inst,a,b); /** Copiamos en cadena la instruccion que
                se comparara completa*/
                instruccion=inst;/**LA INSTRUCCION SE PASARA CADA VEZ QUE
                ENVIEMOS UN DATO
                cad.copy(data,c,d);
                buffer.clear();
                buffer=data;
                espacios.clear();
                leerespacios(buffer);
                valor.valores.clear();
                for(int j=0;j<=espacios.size();j++)
                {
                    if (j==0)
                    {
                        t=espacios.at(j);
                        char dat[t]; /** Almaceamos en comando los terminos que
                        indican de que instruccion se trata*/
                        bzero(dat,t+1);
                        buffer.copy(dat,t,0);
                        dato.clear();
                    }
                }
            }
        }
    }
}
```

```

        dato=dat;
        valor.setValor(dato); //Introducimos el valor
    en el vector
    }

    if (j==espacios.size())
    {
        t=buffer.size()-espacios.at(j-1);
        char dat[t]; /** Almacenamos en comado el ultimo valor
de la secuencia*/
        bzero(dat,t+1);
        buffer.copy(dat,t,espacios.at(j-1)+1);
        dato.clear();
        dato=dat;
        valor.setValor(dato); //Introducimos el valor en
el vector
    }

    if(j!=0 && j!=espacios.size())
    {
        t=espacios.at(j)-espacios.at(j-1)-1;
        char dat[t]; /** Almacenamos en comando los valores
intermedios*/
        bzero(dat,t+1);
        buffer.copy(dat,t,espacios.at(j-1)+1);
        dato.clear();
        dato=dat;
        valor.setValor(dato); //Introducimos el valor
en el vector
    }
}
if (instruccion.find("player") != string::npos) /* Si vemos un
jugador hay que separar la instruccion y en algunas
instruccion no es completa ya que no reconoce el
dorsal*/
{
    string equipo, dorsal;
    leerespacios(instruccion);
    if (espacios.size() == 2)
    {
        t = espacios.at(1)-espacios.at(0)-1;
        char dat[t];
        bzero(dat,t+1);
        instruccion.copy(dat,t,espacios.at(0)+1);
        equipo.clear();
        equipo=dat;
        t = instruccion.size() - espacios.at(1);
        bzero(dat,t+1);
        instruccion.copy(dat,t,espacios.at(1)+1);
        dorsal.clear();
        dorsal=dat;
    }
    else
    {
        equipo = "desconocido";
        dorsal = "desconocido";
    }
    valor.setDatosSeePlayer(equipo,dorsal,
valor.valores);
}
else
    valor.setDatosSee (instruccion, valor.valores
); //Introducimos en cada instrucción sus valores correspondientes
}
}
valor.setFalseHearEvent();
}

/** Almacenamos las cadenas de tipo sense body */
if(cad.find("sense_body")!=string::npos)
{
    for(inicio.at(i);i<inicio.size();i++)/** Hago tantas lecturas como
parentesis tenga*/
    {
        if(fin1.size()==0) //Para instrucciones globales
            a=fin2.at(i)-inicio.at(i)-1;
        else
            a=fin1.at(i)-inicio.at(i)-1; //Para sense_body
        b=inicio.at(i)+1;
        char cadena[a]; /** Almacenamos en cadena los terminos que indican
de que instruccion se trata*/
        bzero(cadena,a+1);
        cad.copy(cadena,a,b);
        buffer.clear();
    }
}

```

```

        buffer=cadena;
        /** Buscamos los espacios y mandamos el comando a la clase de datos
    */
    espacios.clear();
    leerespacios(buffer);
    valor.valores.clear();
    for(int j=0;j<=espacios.size();j++)
    {
        if (j==0)
        {
            t=espacios.at(j);
            char inst[t]; /** Almacenamos en comando los terminos que
indican de que instruccion se trata*/
            bzero(inst,t+1);
            buffer.copy(inst,t,0);
            instruccion.clear();
            instruccion=inst;
        }

        if (j==espacios.size())
        {
            t=buffer.size()-espacios.at(j-1);
            char dat[t]; /** Almacenamos en comando el ultimo valor de la
secuencia*/
            bzero(dat,t+1);
            buffer.copy(dat,t,espacios.at(j-1)+1);
            dato.clear();
            dato=dat;
            valor.setValor(dato); //Introducimos el valor en el
vector
        }

        if(j!=0 && j!=espacios.size())
        {
            t=espacios.at(j)-espacios.at(j-1)-1;
            char dat[t]; /** Almacenamos en comando los valores
intermedios*/
            bzero(dat,t+1);
            buffer.copy(dat,t,espacios.at(j-1)+1);
            dato.clear();
            dato=dat;
            valor.setValor(dato); //Introducimos el valor en el
vector
        }
    }

    valor.setDatosSense (instruccion,valor.valores)
; //Introducimos en cada instruccion sus valores correspondientes
}
valor.setFalseHearEvent();
}

/** Almacenamos las cadenas de tipo hear */
if(cad.find("hear") !=string::npos)
{
    if(cad.find("play_on") != string::npos)
        valor.setDatosHear("referee","play_on");
    else
        if(cad.find("goal_l") != string::npos || cad.find("goal_r") !=
string::npos)
            valor.setDatosHear("referee","goal");
        else
        {
            valor.valores.clear();
            a=espacios.at(2)-espacios.at(1)-1;
            b=espacios.at(1)+1;
            string who,inst,side;
            char cada[a]; /** Almacenamos en 'cada' quien realiza el hear*/
            bzero(cada,a+1);
            cad.copy(cada,a,b);
            who.clear();
            who=cada;
            a=cad.size()-espacios.at(2)-4; //NO LEEMOS EL LADO
            b=espacios.at(2)+1;
            char cadb[a]; /** Almacenamos en cadb los terminos que indican
de que instruccion se trata*/
            bzero(cadb,a+1);
            cad.copy(cadb,a,b);
            inst.clear();
            inst=cadb;
            valor.setValor(inst);
            b=cad.size()-2;
            char cadc[1]; /** Almacenamos en cadb el lado de la instruccion
*/
            bzero(cadc,a+1);
            cad.copy(cadc,1,b);
            side.clear();

```

```

        side=cado;
        valor.setValor(side);
        valor.setDatosHear(who, valor.valores, cad);
    }

    valor.setTrueHearEvent();
}
if(cad.find("init")!=string::npos)
{
    valor.valores.clear();
    a=espacios.at(2)-espacios.at(1)-1;
    b=espacios.at(1)+1;
    string numero, side;
    char cadnumero[a];
    bzero(cadnumero, a+1);
    cad.copy(cadnumero, a, b);
    numero.clear();
    numero=cadnumero;
    valor.setValor(numero);
    b=espacios.at(0)+1;
    char cadside[a];
    bzero(cadside, a+1);
    cad.copy(cadside, 1, b);
    side.clear();
    side=cadside;
    valor.setValor(side);
    valor.setInit(valor.valores);
    valor.setFalseHearEvent();
}
}

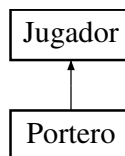
```

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/parser/parser.h
- /home/carlos/info/RoboCup/src/libraries/parser/parser.cpp

3.5. Referencia de la Clase Portero

Diagrama de herencias de Portero



Métodos públicos

- **Portero** (int argc, char *argv[])
- void **coger** ()
Portero::coger Función para coger la pelota.

Otros miembros heredados

3.5.1. Descripción detallada

Definición en la línea 6 del archivo portero.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/robocup/portero.h
- /home/carlos/info/RoboCup/src/libraries/robocup/portero.cpp

3.6. Referencia de la Clase Punto

Métodos públicos

- **Punto** (double a, double b)
- void **printPunto** ()
- void **setX** (double _x)
- void **setY** (double _y)
- double **getX** ()
- double **getY** ()

3.6.1. Descripción detallada

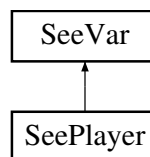
Definición en la línea 6 del archivo punto.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/punto/punto.h
- /home/carlos/info/RoboCup/src/libraries/punto/punto.cpp

3.7. Referencia de la Clase SeePlayer

Diagrama de herencias de SeePlayer



Métodos públicos

- int **getDorsal** ()
SeePlayer::getDorsal.
- int **getEquipo** ()
SeePlayer::getEquipo.
- void **setDorsal** (int _dorsal)
SeePlayer::setDorsal Guardamos el número del jugador.
- void **setEquipo** (int _equipo)
SeePlayer::setEquipo Guardamos el equipo.

3.7.1. Descripción detallada

Definición en la línea 8 del archivo seePlayer.h.

3.7.2. Documentación de las funciones miembro

3.7.2.1. int SeePlayer::getDorsal ()

SeePlayer::getDorsal.

Devuelve

Devuelve el dorsal del jugador.

Definición en la línea 7 del archivo seePlayer.cpp.

```
{  
    return dorsal;  
}
```

3.7.2.2. int SeePlayer::getEquipo ()

[SeePlayer::getEquipo](#).

Devuelve

Devuelve el equipo.

Definición en la línea 17 del archivo seePlayer.cpp.

```
{  
    return equipo;  
}
```

3.7.2.3. void SeePlayer::setDorsal (int *_dorsal*)

[SeePlayer::setDorsal](#) Guardamos el número del jugador.

Parámetros

<i>_dorsal</i>	Dorsal del jugador.
----------------	---------------------

Definición en la línea 26 del archivo seePlayer.cpp.

```
{  
    dorsal=_dorsal;  
}
```

3.7.2.4. void SeePlayer::setEquipo (int *_equipo*)

[SeePlayer::setEquipo](#) Guardamos el equipo.

Parámetros

<i>_equipo</i>	
----------------	--

Definición en la línea 35 del archivo seePlayer.cpp.

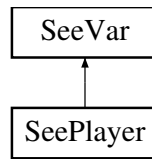
```
{  
    equipo=_equipo;  
}
```

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/robocup/seePlayer.h
- /home/carlos/info/RoboCup/src/libraries/robocup/seePlayer.cpp

3.8. Referencia de la Clase SeeVar

Diagrama de herencias de SeeVar



Métodos públicos

- void [setDist](#) (double a)
[SeeVar::setDist](#) Guardamos dist.
- double **getDist** ()
- void [setDir](#) (double a)
[SeeVar::setDir](#) Guardamos dir.
- double [getDir](#) ()
[SeeVar::getDir](#).
- void [setDistchng](#) (double a)
[SeeVar::setDistchng](#) Guardamos distchng.
- double [getDistchng](#) ()
[SeeVar::getDistchng](#).
- void [setDirchng](#) (double a)
[SeeVar::setDirchng](#) Guardamos dirchng.
- double [getDirchng](#) ()
[SeeVar::getDirchng](#).
- void [setBodydir](#) (double a)
[SeeVar::setBodydir](#) Guardamos bodydir.
- double [getBodydir](#) ()
[SeeVar::getBodydir](#).
- void [setHeaddir](#) (double a)
[SeeVar::setHeaddir](#) Guardamos headdir.
- double [getHeaddir](#) ()
[SeeVar::getHeaddir](#).

3.8.1. Descripción detallada

Definición en la línea 5 del archivo seeVar.h.

3.8.2. Documentación de las funciones miembro

3.8.2.1. double SeeVar::getBodydir ()

[SeeVar::getBodydir](#).

Devuelve

Devuelve bodydir.

Definición en la línea 8 del archivo seeVar.cpp.

```
{  
    return bodydir;  
}
```

3.8.2.2. double SeeVar::getDir ()

[SeeVar::getDir.](#)

Devuelve

Devuelve dir.

Definición en la línea 17 del archivo seeVar.cpp.

```
{  
    return dir;  
}
```

3.8.2.3. double SeeVar::getDirchng ()

[SeeVar::getDirchng.](#)

Devuelve

Devuelve dirchng.

Definición en la línea 26 del archivo seeVar.cpp.

```
{  
    return dirchng;  
}
```

3.8.2.4. double SeeVar::getDistchng ()

[SeeVar::getDistchng.](#)

Devuelve

Devuelve distchng

Definición en la línea 40 del archivo seeVar.cpp.

```
{  
    return distchng;  
}
```

3.8.2.5. double SeeVar::getHeaddir ()

[SeeVar::getHeaddir](#).

Devuelve

Devuelve headdir.

Definición en la línea 49 del archivo seeVar.cpp.

```
{  
    return headdir;  
}
```

3.8.2.6. void SeeVar::setBodydir (double a)

[SeeVar::setBodydir](#) Guardamos bodydir.

Parámetros

<i>a</i>	Dato a guardar
----------	----------------

Definición en la línea 58 del archivo seeVar.cpp.

```
{  
    bodydir=a;  
}
```

3.8.2.7. void SeeVar::setDir (double a)

[SeeVar::setDir](#) Guardamos dir.

Parámetros

<i>a</i>	Dato a guardar
----------	----------------

Definición en la línea 68 del archivo seeVar.cpp.

```
{  
    dir=a;  
}
```

3.8.2.8. void SeeVar::setDirchng (double a)

[SeeVar::setDirchng](#) Guardamos dirchng.

Parámetros

<i>a</i>	Dato a guardar
----------	----------------

Definición en la línea 77 del archivo seeVar.cpp.

```
{  
    dirchng=a;  
}
```

3.8.2.9. void SeeVar::setDist (double a)

[SeeVar::setDist](#) Guardamos dist.

Parámetros

<i>a</i>	Dato a guardar
----------	----------------

Definición en la línea 86 del archivo seeVar.cpp.

```
{
    dist=a;
}
```

3.8.2.10. void SeeVar::setDistchnng (double a)

[SeeVar::setDistchnng](#) Guardamos distchnng.

Parámetros

<i>a</i>	Dato a guardar
----------	----------------

Definición en la línea 95 del archivo seeVar.cpp.

```
{
    distchnng=a;
}
```

3.8.2.11. void SeeVar::setHeaddir (double a)

[SeeVar::setHeaddir](#) Guardamos headdir.

Parámetros

<i>a</i>	Dato a guardar
----------	----------------

Definición en la línea 104 del archivo seeVar.cpp.

```
{
    headdir=a;
}
```

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/robocup/seeVar.h
- /home/carlos/info/RoboCup/src/libraries/robocup/seeVar.cpp

3.9. Referencia de la Clase SenseVar**Métodos públicos**

- void [setQuality](#) (string a)
[SenseVar::setQuality](#) Guardamos la variable quality.
- std::string [getQuality](#) ()
[SenseVar::getQuality](#).

- void `setWidth` (string a)
`SenseVar::setWidth` Guardamos la variable width.
- std::string `getWidth` ()
`SenseVar::getWidth`.
- void `setValue` (std::vector< string > a)
`SenseVar::setValue` Guardamos las variables value1 y value2.
- double `getValue1` ()
`SenseVar::getValue1`.
- double `getValue2` ()
`SenseVar::getValue2`.

3.9.1. Descripción detallada

Definición en la línea 8 del archivo senseVar.h.

3.9.2. Documentación de las funciones miembro

3.9.2.1. string SenseVar::getQuality ()

`SenseVar::getQuality`.

Devuelve

Devuelve quality.

Definición en la línea 8 del archivo senseVar.cpp.

```
{  
    return quality;  
}
```

3.9.2.2. double SenseVar::getValue1 ()

`SenseVar::getValue1`.

Devuelve

devuelve value1

Definición en la línea 26 del archivo senseVar.cpp.

```
{  
    return value1;  
}
```

3.9.2.3. double SenseVar::getValue2 ()

`SenseVar::getValue2`.

Devuelve

Devuelve value2

Definición en la línea 35 del archivo senseVar.cpp.

```
{  
    return value2;  
}
```

3.9.2.4. `string SenseVar::getWidth ()`

[SenseVar::getWidth](#).

Devuelve

Devuelve width.

Definición en la línea 17 del archivo `senseVar.cpp`.

```
{
    return width;
}
```

3.9.2.5. `void SenseVar::setQuality (string a)`

[SenseVar::setQuality](#) Guardamos la variable quality.

Parámetros

<code>a</code>	Dato a guardar
----------------	----------------

Definición en la línea 44 del archivo `senseVar.cpp`.

```
{
    quality.clear();
    quality=a;
}
```

3.9.2.6. `void SenseVar::setValue (std::vector< string > a)`

[SenseVar::setValue](#) Guardamos las variables value1 y value2.

Parámetros

<code>a</code>	Vector de dato a guardar
----------------	--------------------------

Definición en la línea 64 del archivo `senseVar.cpp`.

```
{
    if (a.size()==1)
    {
        stringstream ss(a[0]);
        ss>>value1;
        ss.str("");
    }
    else
    {
        stringstream ss;
        ss<< a[0]<<" "<<a[1];
        ss>>value1<>value2;
        ss.str("");
    }
}
```

3.9.2.7. `void SenseVar::setWidth (string a)`

[SenseVar::setWidth](#) Guardamos la variable width.

Parámetros

a	Dato a guardar
---	----------------

Definición en la línea 54 del archivo senseVar.cpp.

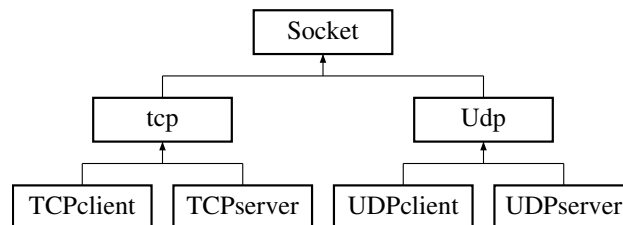
```
{
    width.clear();
    width=a;
}
```

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/robocup/senseVar.h
- /home/carlos/info/RoboCup/src/libraries/robocup/senseVar.cpp

3.10. Referencia de la Clase Socket

Diagrama de herencias de Socket



Métodos públicos

- void [bufferChar2String](#) (char *cbuf)
- void [bufferString2Char](#) (string sbuf)
- void [print](#) ()
- virtual void **checkConnection** (int argc)=0
- virtual void **socket** (char *argv[]=0)
- virtual void **enviar** (string buffer)=0
- virtual void **recibir** ()=0
- string [getSbuffer](#) ()
- string [setSbuffer](#) (string sSbuffer)

Atributos protegidos

- int **sockfd**
- int [n](#)
- string [sbuffer](#)
- char * **cbuffer**
- char [ccbuffer](#) [1024]

3.10.1. Descripción detallada

Definición en la línea 17 del archivo socket.h.

3.10.2. Documentación de las funciones miembro

3.10.2.1. void Socket::bufferChar2String (char * *cbuf*)

Conversión de tipo <char> a tipo <string> < Borramos lo que hubiera guardado en sbuffer

< Copiamos lo que haya en cbuf a sbuffer

Definición en la línea 12 del archivo socket.cpp.

```
{
    sbuffer.clear(); /**< Borramos lo que hubiera guardado en sbuffer */
    sbuffer = cbuf; /**< Copiamos lo que haya en cbuf a sbuffer */
}
```

3.10.2.2. void Socket::bufferString2Char (string *sbuf*)

Conversión de tipo <string> a tipo <char>

Definición en la línea 20 del archivo socket.cpp.

```
{
    int i=sbuf.length()+1;
    cbuffer = new char[i];
    strcpy(cbuffer,sbuf.c_str());
}
```

3.10.2.3. string Socket::getSbuffer ()

Función que devuelve el valor del atributo sbuffer

Definición en la línea 37 del archivo socket.cpp.

```
{
    return sbuffer;
}
```

3.10.2.4. void Socket::print ()

Imprimimos por pantalla lo que esté almacenado en

Parámetros

<i>ccbuffer</i>	
-----------------	--

Definición en la línea 30 del archivo socket.cpp.

```
{
    cout<<sbuffer<<endl;
}
```

3.10.2.5. string Socket::setSbuffer (string *sSbuffer*)

Funciones para modificar y recuperar el atributo sbuffer

Definición en la línea 42 del archivo socket.cpp.

```
{
    sbuffer = sSbuffer;
}
```

3.10.3. Documentación de los datos miembro

3.10.3.1. `char Socket::ccbuffer[1024]` `[protected]`

En cbuffer almacenamos los char que enviamos y en ccbuffer los que recibimos

Definición en la línea 23 del archivo socket.h.

3.10.3.2. `int Socket::n` `[protected]`

Parámetros

<code>sockfd</code>	es el file descriptor de nuestro socket y
<code>n</code>	es donde almacenamos los <code><return></code> de

Ver también

enviar y
recibir

Definición en la línea 20 del archivo socket.h.

3.10.3.3. `string Socket::sbuffer` `[protected]`

Aquí almacenamos lo que recibimos en formato `<string>`

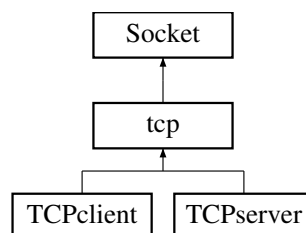
Definición en la línea 22 del archivo socket.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- `/home/carlos/info/RoboCup/src/libraries/sockets/socket.h`
- `/home/carlos/info/RoboCup/src/libraries/sockets/socket.cpp`

3.11. Referencia de la Clase tcp

Diagrama de herencias de tcp



Atributos protegidos

- `struct sockaddr_in` `serv_addr`

Otros miembros heredados

3.11.1. Descripción detallada

Definición en la línea 6 del archivo TCP.h.

3.11.2. Documentación de los datos miembro

3.11.2.1. `struct sockaddr_in tcp::serv_addr` `[protected]`

Estructura donde se almacenan los datos del servidor

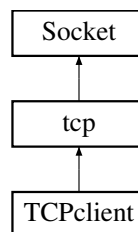
Definición en la línea 9 del archivo TCP.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- `/home/carlos/info/RoboCup/src/libraries/sockets/TCP.h`
- `/home/carlos/info/RoboCup/src/libraries/sockets/TCP.cpp`

3.12. Referencia de la Clase TCPclient

Diagrama de herencias de TCPclient



Métodos públicos

- void `socket` (char *argv[])
- void `checkConnection` (int argc)
- void `enviar` (string buffer)
- void `recibir` ()

Otros miembros heredados

3.12.1. Descripción detallada

Definición en la línea 6 del archivo tcpclient.h.

3.12.2. Documentación de las funciones miembro

3.12.2.1. `void TCPclient::checkConnection (int argc)` `[virtual]`

Comprobamos la conexión.

Parámetros

<code>argc</code>	Contiene el número de argumentos del programa.
-------------------	--

Si el número de argumentos es menos del deseado entonces se muestra un error por pantalla y se sale del programa

Implementa `Socket`.

Definición en la línea 47 del archivo tcpclient.cpp.

```

{
    if (argc < 3)
    {
        cout<<"ERROR, falta la ip del servidor o el puerto"<<endl;
        exit(0);
    }
}

```

3.12.2.2. void TCPClient::enviar (string buffer) [virtual]

Función Enviar

Parámetros

<i>buffer</i>	Mensaje a enviar
---------------	------------------

Implementa [Socket](#).

Definición en la línea 62 del archivo tcpclient.cpp.

```

{
    bufferString2Char(buffer);
    n = write(sockfd,cbuffer,strlen(cbuffer));
    if (n < 0) error("ERROR writing to socket");
}

```

3.12.2.3. void TCPClient::recibir () [virtual]

Función Recibir

Implementa [Socket](#).

Definición en la línea 75 del archivo tcpclient.cpp.

```

{
    bzero(ccbuffer,1024);
    n = read(sockfd,ccbuffer,255);
    if (n < 0)
        error("ERROR reading from socket");
    bufferChar2String(ccbuffer);
}

```

3.12.2.4. void TCPClient::socket (char * argv[]) [virtual]

Creación del socket

Parámetros

<i>argv[]</i>	Aquí se pasa por parametro la dirección IP del servidor y el número de puerto.
---------------	--

Creamos el socket y vamos comprobando que no hay errores durante el proceso.

Implementa [Socket](#).

Definición en la línea 18 del archivo tcpclient.cpp.

```

{
    sockfd = ::socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
    }
}

```

```

        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
          (char *)&serv_addr.sin_addr.s_addr,
          server->h_length);
    serv_addr.sin_port = htons(atoi(argv[2]));
    if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR connecting");
}

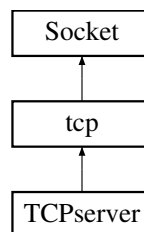
```

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/sockets/tcpclient.h
- /home/carlos/info/RoboCup/src/libraries/sockets/tcpclient.cpp

3.13. Referencia de la Clase TCPserver

Diagrama de herencias de TCPserver



Métodos públicos

- void **socket** (char *argv[])
- void **checkConnection** (int argc)
- void **conectar** ()
- void **enviar** (string buffer)
- void **recibir** ()

Otros miembros heredados

3.13.1. Descripción detallada

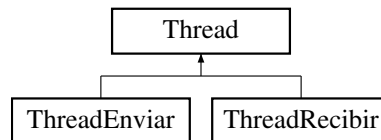
Definición en la línea 6 del archivo tcpserver.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/sockets/tcpserver.h
- /home/carlos/info/RoboCup/src/libraries/sockets/tcpserver.cpp

3.14. Referencia de la Clase Thread

Diagrama de herencias de Thread



Clases

- struct **Thread_Obj**

Métodos públicos

- void **start** (void *param=NULL)
- bool **isRunning** ()

Métodos protegidos

- **Thread** ()
default constructor Protected to avoid instantiation
- void * **join** ()
- void * **end** ()

Métodos protegidos estáticos

- static void * **thread_function** (void *param)
thread_function

Atributos protegidos

- bool **_terminate**
The Thread must terminate.

3.14.1. Descripción detallada

Definición en la línea 9 del archivo Thread.h.

3.14.2. Documentación de las funciones miembro

3.14.2.1. void * Thread::thread_function (void * *param*) [static],[protected]

thread_function

Parámetros

<i>param</i>	void *
--------------	--------

Devuelve

Definición en la línea 36 del archivo Thread.cpp.

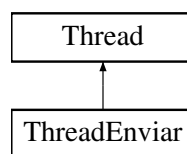
```
{  
    Thread_Obj* t_obj = (Thread_Obj*)param;  
    return t_obj->thr->run(t_obj->param);  
}
```

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/threads/Thread.h
- /home/carlos/info/RoboCup/src/libraries/threads/Thread.cpp

3.15. Referencia de la Clase ThreadEnviar

Diagrama de herencias de ThreadEnviar



Métodos públicos

- int **end** ()

Otros miembros heredados

3.15.1. Descripción detallada

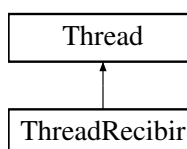
Definición en la línea 7 del archivo ThreadEnviar.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/threads/ThreadEnviar.h
- /home/carlos/info/RoboCup/src/libraries/threads/ThreadEnviar.cpp

3.16. Referencia de la Clase ThreadRecibir

Diagrama de herencias de ThreadRecibir



Métodos públicos

- int **end** ()

Otros miembros heredados

3.16.1. Descripción detallada

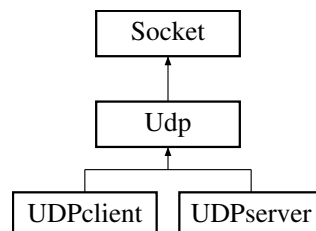
Definición en la línea 6 del archivo ThreadRecibir.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/threads/ThreadRecibir.h
- /home/carlos/info/RoboCup/src/libraries/threads/ThreadRecibir.cpp

3.17. Referencia de la Clase Udp

Diagrama de herencias de Udp



Atributos protegidos

- int **n**
- unsigned int **length**
- socklen_t **fromlen**
- struct sockaddr_in **server from**
- struct hostent * **hp**
- char **buf** [1024]

Otros miembros heredados

3.17.1. Descripción detallada

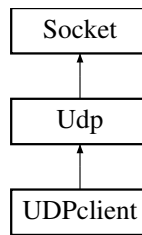
Definición en la línea 6 del archivo udp.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/sockets/udp.h
- /home/carlos/info/RoboCup/src/libraries/sockets/udp.cpp

3.18. Referencia de la Clase UDPclient

Diagrama de herencias de UDPclient



Métodos públicos

- void **socket** (char *argv[])
- void **checkConnection** (int argc)
- void **enviar** (string buffer)
- void **recibir** ()

Otros miembros heredados

3.18.1. Descripción detallada

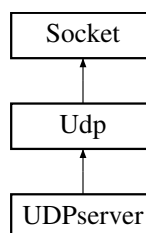
Definición en la línea 6 del archivo udpclient.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/sockets/udpclient.h
- /home/carlos/info/RoboCup/src/libraries/sockets/udpclient.cpp

3.19. Referencia de la Clase UDPserver

Diagrama de herencias de UDPserver



Métodos públicos

- void **socket** (char *argv[])
- void **checkConnection** (int argc)
- void **enviar** (string buffer)
- void **recibir** ()

Otros miembros heredados

3.19.1. Descripción detallada

Definición en la línea 6 del archivo udpserver.h.

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- /home/carlos/info/RoboCup/src/libraries/sockets/udpserver.h
- /home/carlos/info/RoboCup/src/libraries/sockets/udpserver.cpp

Índice alfabético

bufferChar2String
 Socket, [44](#)
bufferString2Char
 Socket, [44](#)
buscarIndicador
 Jugador, [16](#)
buscarPelota
 Jugador, [17](#)

ccbuffer
 Socket, [45](#)
checkConnection
 TCPClient, [46](#)
colocar
 Jugador, [17](#)
comprobar_pelota
 Jugador, [18](#)
correr
 Jugador, [18](#)

Datos, [5](#)
 getCiclo, [6](#)
 getHearInst, [6](#)
 getJugadorContrario, [6](#)
 getJugadorPropio, [7](#)
 getNumero, [7](#)
 getSeelInst, [7](#)
 getSenseInst, [8](#)
 getSide, [8](#)
 getultimaCadena, [8](#)
 setCiclo, [8](#)
 setDatosHear, [9](#)
 setDatosSee, [9](#)
 setDatosSeePlayer, [10](#)
 setDatosSense, [11](#)
 setFalseHearEvent, [11](#)
 setInit, [11](#)
 setSide, [12](#)
 setValor, [12](#)

enviar
 TCPClient, [47](#)

getBodydir
 SeeVar, [37](#)
getCiclo
 Datos, [6](#)
getDir
 SeeVar, [38](#)
getDirchng
 SeeVar, [38](#)
getDistchng
 SeeVar, [38](#)
getDorsal
 SeePlayer, [35](#)
getEquipo
 SeePlayer, [36](#)
getHeaddir
 SeeVar, [38](#)
getHearInst
 Datos, [6](#)
getJugadorContrario
 Datos, [6](#)
getJugadorPropio
 Datos, [7](#)
getNumero
 Datos, [7](#)
getPosicion
 Jugador, [19](#)
getQuality
 SenseVar, [41](#)
getSbuffer
 Socket, [44](#)
getSeelInst
 Datos, [7](#)
getSenseInst
 Datos, [8](#)
getSide
 Datos, [8](#)
getValue1
 SenseVar, [41](#)
getValue2
 SenseVar, [41](#)
getWidth
 SenseVar, [41](#)
getalpha
 Jugador, [18](#)
getinstruction
 HearVar, [13](#)
getnumero
 Jugador, [18](#)
getside
 HearVar, [13](#)
getultimaCadena
 Datos, [8](#)
girar
 Jugador, [19](#)
goTo
 Jugador, [20](#), [22](#)

- golpear
 - Jugador, [19](#)
- hearReferee
 - Jugador, [23](#)
- HearVar, [13](#)
 - getInstruction, [13](#)
 - getSide, [13](#)
 - setData, [13](#)
- interpretar
 - Jugador, [24](#)
- Jugador, [14](#)
 - buscarIndicador, [16](#)
 - buscarPelota, [17](#)
 - colocar, [17](#)
 - comprobar_pelota, [18](#)
 - correr, [18](#)
 - getPosicion, [19](#)
 - getAlpha, [18](#)
 - getNumero, [18](#)
 - girar, [19](#)
 - goTo, [20](#), [22](#)
 - golpear, [19](#)
 - hearReferee, [23](#)
 - interpretar, [24](#)
 - Jugador, [16](#)
 - mirar, [25](#)
 - mover, [25](#)
 - msg, [29](#)
 - posicionPelota, [25](#)
 - seguirPelota, [25](#)
 - setPosicion, [26](#)
 - setAlpha, [26](#)
 - setNumero, [26](#)
- leerparentesis
 - Parser, [29](#)
- mirar
 - Jugador, [25](#)
- mover
 - Jugador, [25](#)
- msg
 - Jugador, [29](#)
- n
 - Socket, [45](#)
- Parser, [29](#)
 - leerparentesis, [29](#)
 - separardatos, [30](#)
- Portero, [34](#)
- posicionPelota
 - Jugador, [25](#)
- print
 - Socket, [44](#)
- Punto, [35](#)
- recibir
 - TCPClient, [47](#)
- sbuffer
 - Socket, [45](#)
- SeePlayer, [35](#)
 - getDorsal, [35](#)
 - getEquipo, [36](#)
 - setDorsal, [36](#)
 - setEquipo, [36](#)
- SeeVar, [37](#)
 - getBodyDir, [37](#)
 - getDir, [38](#)
 - getDirChng, [38](#)
 - getDistChng, [38](#)
 - getHeaddir, [38](#)
 - setBodyDir, [39](#)
 - setDir, [39](#)
 - setDirChng, [39](#)
 - setDist, [39](#)
 - setDistChng, [40](#)
 - setHeaddir, [40](#)
- seguirPelota
 - Jugador, [25](#)
- SenseVar, [40](#)
 - getQuality, [41](#)
 - getValue1, [41](#)
 - getValue2, [41](#)
 - getWidth, [41](#)
 - setQuality, [42](#)
 - setValue, [42](#)
 - setWidth, [42](#)
- separardatos
 - Parser, [30](#)
- serv_addr
 - tcp, [46](#)
- setBodyDir
 - SeeVar, [39](#)
- setCiclo
 - Datos, [8](#)
- setDatosHear
 - Datos, [9](#)
- setDatosSee
 - Datos, [9](#)
- setDatosSeePlayer
 - Datos, [10](#)
- setDatosSense
 - Datos, [11](#)
- setDir
 - SeeVar, [39](#)
- setDirChng
 - SeeVar, [39](#)
- setDist
 - SeeVar, [39](#)
- setDistChng
 - SeeVar, [40](#)
- setDorsal
 - SeePlayer, [36](#)
- setEquipo

- SeePlayer, [36](#)
- setFalseHearEvent
 - Datos, [11](#)
- setHeaddir
 - SeeVar, [40](#)
- setInit
 - Datos, [11](#)
- setPosicion
 - Jugador, [26](#)
- setQuality
 - SenseVar, [42](#)
- setSbuffer
 - Socket, [44](#)
- setSide
 - Datos, [12](#)
- setValor
 - Datos, [12](#)
- setValue
 - SenseVar, [42](#)
- setWidth
 - SenseVar, [42](#)
- setalpha
 - Jugador, [26](#)
- setdata
 - HearVar, [13](#)
- setnumero
 - Jugador, [26](#)
- Socket, [43](#)
 - bufferChar2String, [44](#)
 - bufferString2Char, [44](#)
 - ccbuffer, [45](#)
 - getSbuffer, [44](#)
 - n, [45](#)
 - print, [44](#)
 - sbuffer, [45](#)
 - setSbuffer, [44](#)
- socket
 - TCPclient, [47](#)
- TCPclient, [46](#)
 - checkConnection, [46](#)
 - enviar, [47](#)
 - recibir, [47](#)
 - socket, [47](#)
- TCPserver, [48](#)
- tcp, [45](#)
 - serv_addr, [46](#)
- Thread, [48](#)
 - thread_function, [49](#)
- thread_function
 - Thread, [49](#)
- ThreadEnviar, [50](#)
- ThreadRecibir, [50](#)
- UDPclient, [51](#)
- UDPserver, [52](#)
- Udp, [51](#)