ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIONES

Titulación:

INGENIERO TÉCNICO EN INFORMATICA DE GESTIÓN

Nombre del proyecto:

"Aprendizaje y desarrollo de aplicaciones multimedia e interactivas en entorno Scratch/BYOB con Arduino y Kinect"

Autor: Jorge Wederago Jiménez

Tutor: Alfredo Pina Calafí

Pamplona, a 4 de Julio de 2013

Índice

Justificación del proyecto

Este proyecto, a mi parecer, parte del grave problema al que nos enfrentamos los ingenieros a la hora de comenzar una carrera que tenga formación en programación. Muchos de nosotros no tenemos muy claro el concepto de programación y, con ello, terminamos encontrando estas como las asignaturas más duras y de mayor complejidad de la carrera.

Gracias a la existencia de lenguajes como Scratch/BYOB encontramos una posible solución a este problema. El poder incluir estos lenguajes en la educación a niveles básicos, ESO y/o Bachiller en asignaturas tales como Tecnología facilitaría mucho la transición a lenguajes de mayor nivel para posibles futuros ingenieros.
Ideas similares se están comenzando a implementar gracias a iniciativas como la First Lego League la cual, a través de la asignatura de Tecnología, está comenzando a acercar la robótica y la programación a alumnos de todo el mundo.

Presentación

El objetivo principal de este proyecto es facilitar el aprendizaje y manejo de los entornos Scratch/BYOB. Se realizarán una serie de ejemplos de sencilla aplicación para acercar al usuario este tipo de programación y dar las nociones necesarias para llevar adelante pequeños proyectos educativos haciendo uso de estas tecnologías.

Una vez familiarizados con los entornos, se procederá a estudiar los procedimientos de comunicación exterior de los que dispone para así poder crear aplicaciones interactivas haciendo uso de tecnologías low cost y/o hardware libre.

Conocidas las características y fundamentos de la comunicación con Scratch se procederá al estudio de dos plataformas de software libre que permiten la interactuación con el entorno: Kinect2Scratch y Arduino4Scratch.

En Kinect2Scratch se introducirá el uso de este software y con ello, la realización de algunos ejemplos para entender de forma práctica como usarlo en nuestros proyectos.

Con Arduino4Scratch veremos cómo interactuar con Arduino y Scratch de modo que ambos sean receptores y emisores a la vez que desarrollamos ejemplos sencillos.

Con los conocimientos adquiridos a lo largo del aprendizaje de estos entornos intentaremos crear nuestras propias aplicaciones y ver qué utilidades le podemos dar.

Por último se hará un estudio con usuarios reales respecto a la manejabilidad y usabilidad de algunas aplicaciones desarrolladas mediante una serie de test especializados y se expondrán las competencias básicas se pueden trabajar mediante el uso de estos entornos.

Scratch

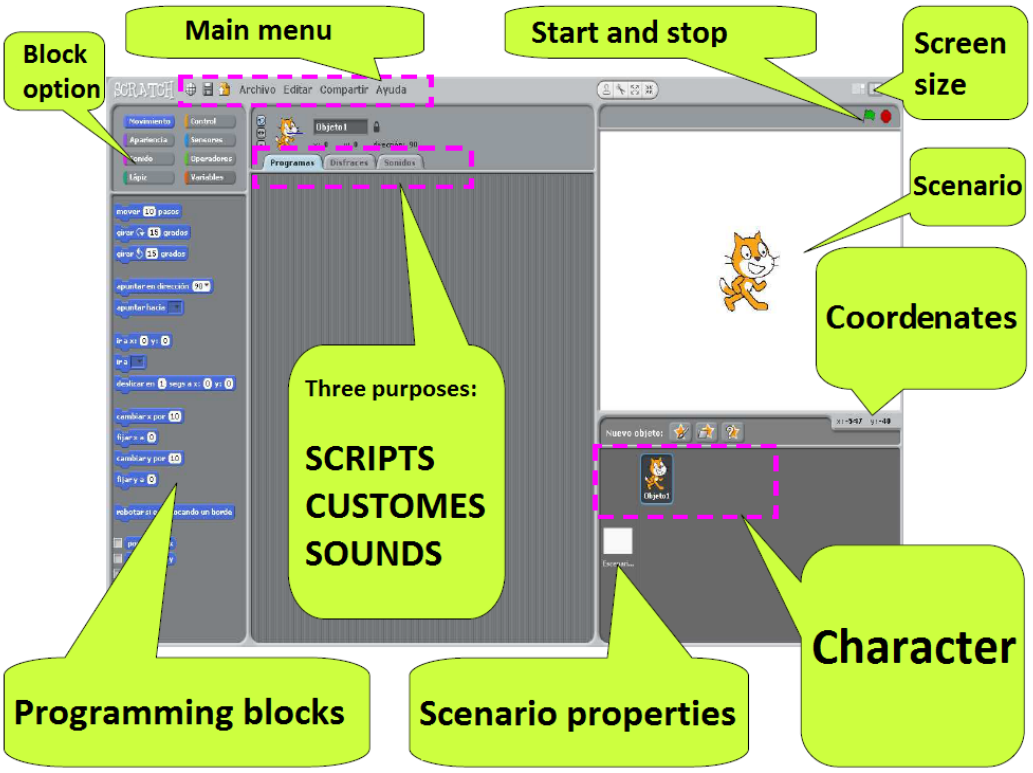
¿Qué es Scratch?

El entorno de aprendizaje Scratch permite un fácil acercamiento a la programación mediante el uso del interfaz utilizando "bloques" que representan las instrucciones básicas de la programación clásica. Se trata de un lenguaje que usa Scripts y Sprites. Su lema: *programa, crea y juega* nos da una clara idea de cuál es su primera intención.

Este entorno programado en Squeak (una implementación libre de Smalltalk-80) y a partir de su versión 2.0 en ActionScript, fue diseñado por un equipo de ingenieros, el *Lifelong Kindergarten Group*, del MIT con la intención de facilitar el acercamiento a los entornos de programación. El encargado del proyecto fue Mitchel Resnick y su primera aparición fue durante el verano de 2007.

El uso de estos bloques no solo facilita su entendimiento sino que además nos permite, de manera intuitiva, escribir de forma sintácticamente correcta. Esto se debe a que, los denominados bloques, poseen una forma física que los hace distinguibles unos de otros y del mismo modo indican el tipo de construcción que se nos permite hacer con ese tipo de órdenes.

Interfaz de Scratch



El interfaz de Scratch consta de los siguientes elementos:

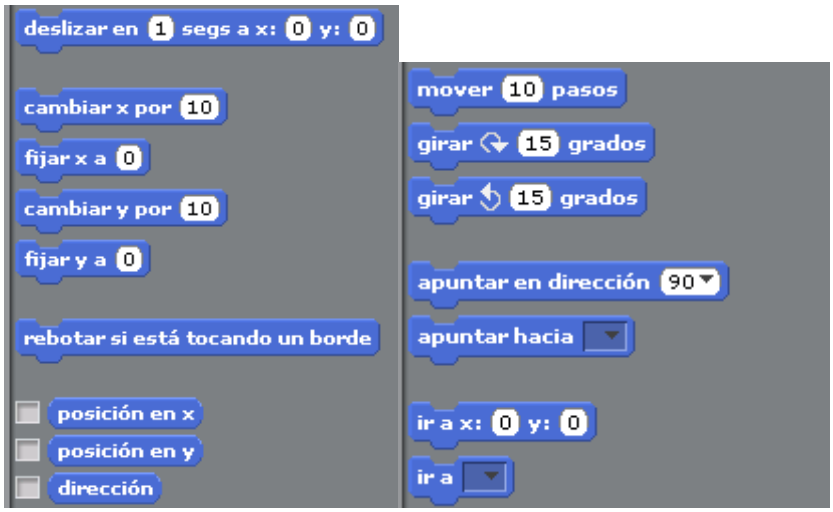
- Main Menu (menú principal): el menú principal donde encontramos (de izquierda a derecha): Cambio de Idioma, guardar, compartir proyecto, Archivo, Editar, Compartir y Ayuda.
- Block Option (opciones de bloques): en esta zona vemos los bloques divididos en las categorías que describiremos más adelante: Movimiento, Apariencia, Sonido, Lápiz, Control, Sensores, Operadores y Variables.
- Programming blocks (bloques de programación): aquí se nos presentan todos los bloques disponibles dentro de una categoría.
- Scripts (programas): zona donde prepararemos los programas de cada sprite. **IMPORTANTE: si hacemos doble click sobre cualquier bloque/conjunto de bloques en esta sección podemos ejecutar órdenes concretas o bloques concretos para así probar de manera más sencilla lo construido hasta el momento.**
- Customes (disfraces): distintas apariencias posibles para nuestro sprite.
- Sounds (sonidos): sonidos de los que dispone el sprite.
- Scenario properties (sprite del escenario): al seleccionarlo podremos tratarlo como a un sprite más.
- Characters (personajes/sprites): aquí están todos los personajes de nuestras historietas o juegos.
- Coordenates (coordenadas): coordenadas dentro del escenario en los ejes x e y.
- Scenario (escenario): Lugar donde podemos, de manera sencilla, situar las escenas en las que tendrán lugar nuestros videojuegos o historietas.

- Start & Stop:(bandera verde/roja) la bandera verde indica al sistema que queremos que comience a ejecutar la programación. El punto rojo detiene la ejecución en cualquier momento.
- Screen and Size: Con estos tres botones podemos modificar la estructura del interfaz, el situado a la izquierda da mayor peso a la zona de programación, el del centro lo equilibra y el de la derecha nos permite ver el escenario a pantalla completa.

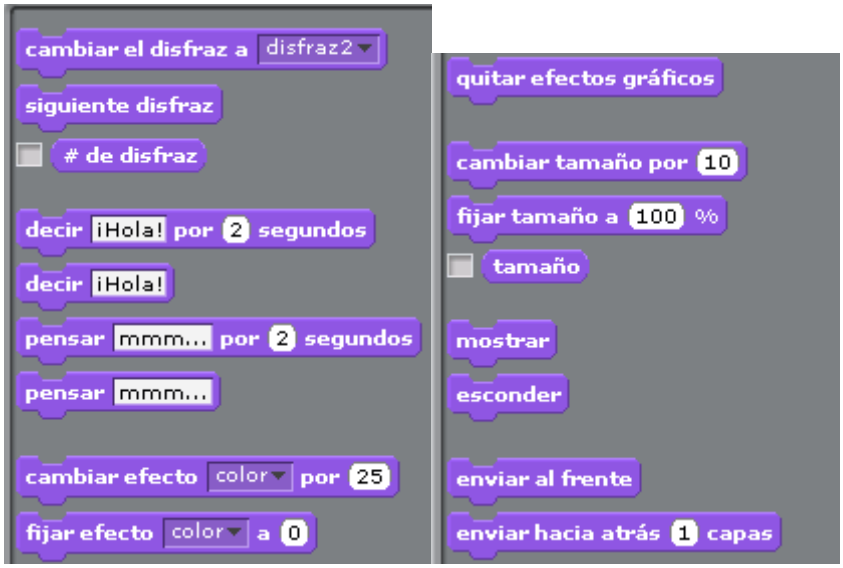
Bloques en Scratch

A continuación nombraremos e indicaremos la función de cada una de las distintas categorías que engloban los bloques anteriormente nombrados.

- Movimiento: En esta sección disponemos de todos las instrucciones que nos dan el control sobre el movimiento de los sprites.

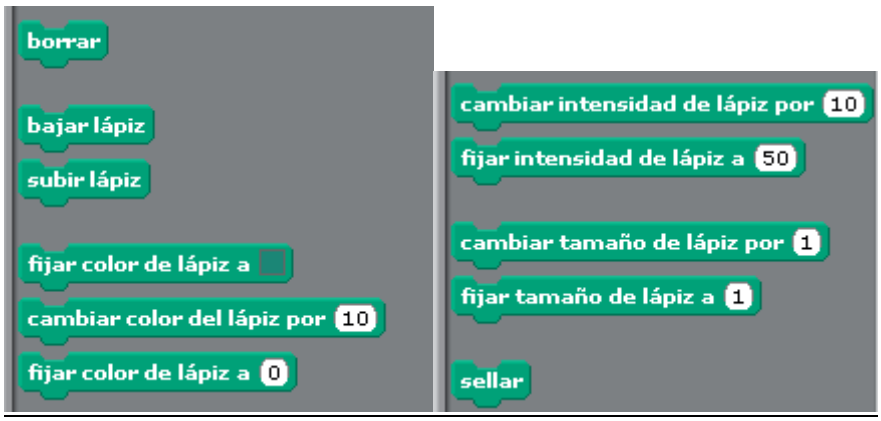


- Apariencia: Aquí podemos modificar el aspecto de los sprites. También contiene las órdenes que corresponden a la creación de diálogos.

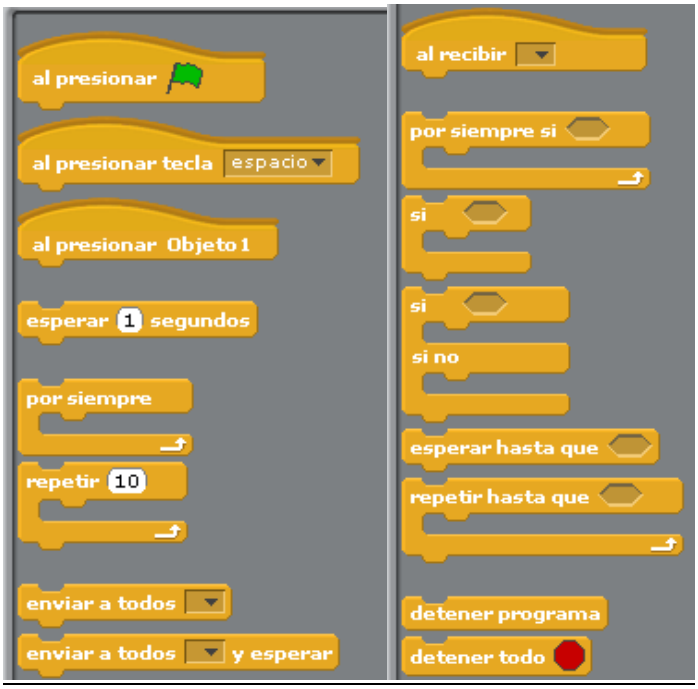


- Sonido: En esta zona encontraremos todo lo relacionado con el control de sonidos, en el aparecerán los sonidos propios de cada sprite.

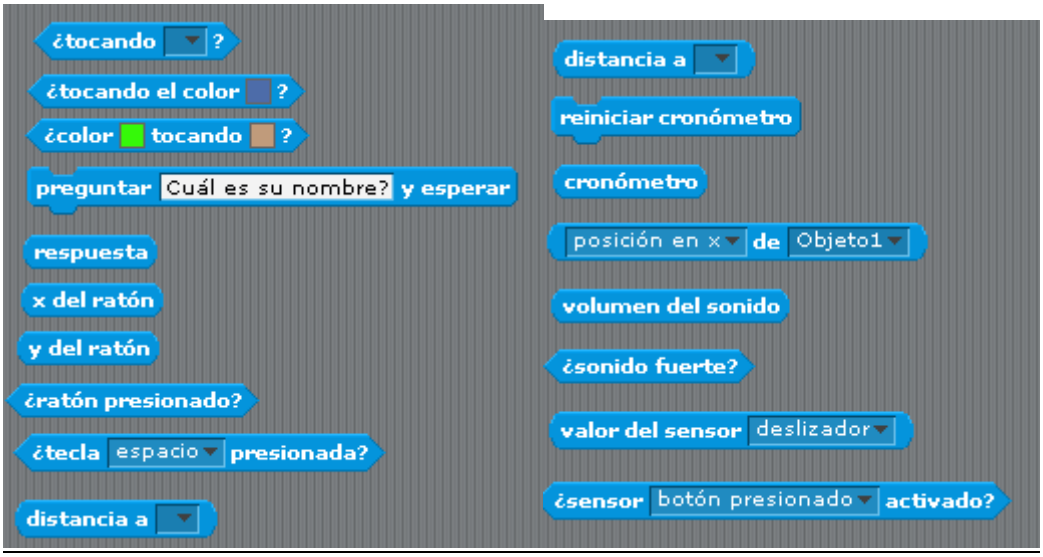
- Lápiz: Si queremos usar el escenario como un lienzo para aplicaciones de dibujo aquí encontraremos todo lo necesario



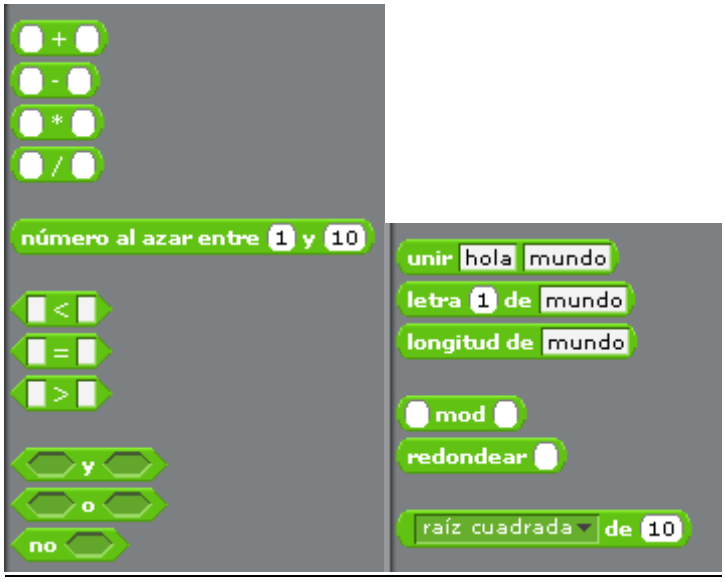
- General: Es el grupo más importante de órdenes, determinan la ejecución de nuestros programas.



- Sensores: A la hora de interactuar entre sprites o con hardware/software exterior, en este apartado tendremos lo necesario.



- Operadores: Agrupa todos los bloques relacionados con operaciones aritméticas y lógicas.



- Variables: En la sección de variables podemos crear variables globales o locales (de cada sprite) para los usos que necesitemos. Se explicarán detalladamente más adelante.

Primer contacto

Como ya se ha mencionado anteriormente, Scratch es un lenguaje de Scripts y Sprites. Antes de nada vamos a definir estos dos conceptos.

- Cuando hablamos de un sprite, hacemos referencia a un mapa de bits dibujado en una pantalla de ordenador, es decir, nos referimos al objeto o personaje que vamos a hacer o hemos creado.

- Al hablar de un script nos referimos a la secuencia de órdenes por las que se rige el comportamiento de un sprite. Es decir, define el comportamiento de un sprite.

Para comenzar necesitamos un Sprite en el escenario. Para poner un sprite nuevo podemos hacer uso de las opciones de Nuevo Objeto que se encuentran debajo del escenario.

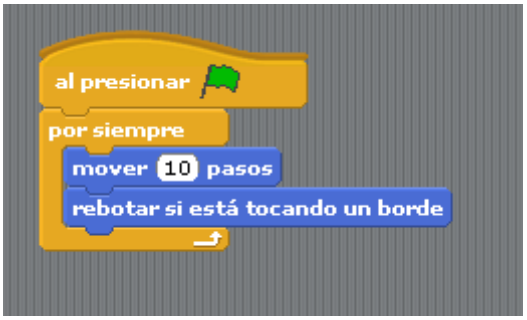
La primera opción nos permite dibujarlo usando el editor propio de Scratch, la segunda nos da la posibilidad de importar imágenes predeterminadas o personalizadas y la última opción añade uno de los predeterminados de manera aleatoria. Para este caso utilizaremos el que viene por defecto y lo situaremos en el lado izquierdo del escenario, para ello lo arrastraremos desde la posición en la que aparece.

A continuación clicaremos en la pestaña de programación y añadiremos el siguiente código.

El primer bloque, que pertenece al grupo general, indica al sprite que deseamos que empiece a ejecutar ese programa desde que pulsamos el botón bandera verde (recordemos que pulsar la bandera verde significa comenzar a ejecutar el programa).

El segundo bloque indica al sprite que queremos que se desplace 10 pasos, es decir, 10 unidades en el eje x.

Como podemos ver esta ejecución no aporta apenas sensación de movimiento así que vamos a probar con la siguiente.



Al código anterior le hemos añadido una nueva instrucción del grupo general que nos permite hacer las acciones que contiene de manera indefinida (o hasta que se

detenga la ejecución del programa). Además le hemos añadido el bloque "rebotar si está tocando un borde" con lo cual evitamos que nuestro sprite se salga de la pantalla. Para finalizar nos interesa introducir una opción nueva del sprite, los giros.

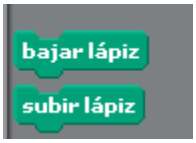
Las opciones de giros las encontramos en la parte superior del interfaz gráfico. Se trata de los tres botones de la parte izquierda. El de arriba, permite el giro del sprite en grados, el del centro lo limita sobre el eje vertical y el de más abajo no permite giros en el sprite, siempre estará orientado en la misma dirección. Esta opción es un detalle muy importante ya que en función de cual tengamos seleccionada limitará la capacidad de movimiento de nuestro sprite.

En el programa *ejemplos-apariencia* adjunto, se podrán encontrar una serie de scripts que nos permitirán, de manera sencilla, ver ejemplos de la amplia gama de opciones de que disponemos con tan solo el uso de los grupos de bloques: general, movimiento y apariencia.

Dibujando en el escenario

En este apartado veremos cómo convertir el escenario en un lienzo. Para ello haremos uso de los bloques en la sección de lápiz.

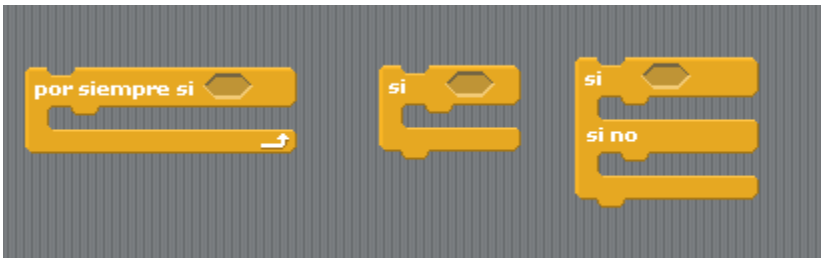
Los dos bloques principales de este grupo son los siguientes:



Bajar lápiz simula la acción de comenzar a escribir en el papel mientras que subir lápiz simula la acción de levantar la mano del papel. De esta manera tan sencilla simulamos la acción real de la escritura en papel.
El resto de opciones nos permiten personalizar el estilo de escritura, desde el color hasta la dureza del trazo.

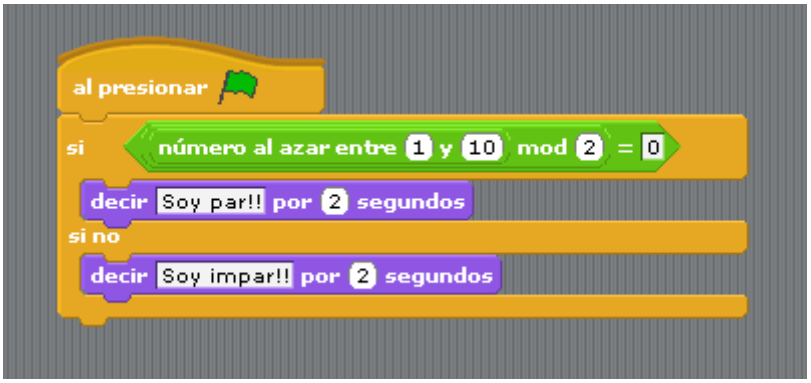
Se adjunta el programa *Ejemplo lienzo* con el que usamos el ratón como mano para dibujar. En este programa introduciremos la orden/bloque "si".

Este bloque permite sentencias y composiciones alternativas, es decir, comprueba si se cumple una condición y en función de si se cumple o no ejecutará el bloque de instrucciones que contenga. Esta instrucción la podemos encontrar de las siguientes formas:



La primera es una combinación del "por siempre" con el "si", con ella podremos ejecutar de forma indefinida lo que contenga el bloque siempre y cuando se cumpla la condición. La segunda es la que ya conocemos y la última nos permite ejecutar dos bloques distintos de instrucciones en función de si se da la condición o no.

Como observamos de forma clara, las condiciones tienen forma "romboide". De este modo sabemos de forma visual que tipo de instrucciones podemos usar como condiciones. Principalmente se trata de instrucciones que pertenecen al grupo de operadores y al grupo de sensores. Un ejemplo de uso de las instrucciones "si"



El siguiente script genera un aleatorio entre 1 y 10, lo divide de forma entera entre 2 y en función de si el resultado es 0 o no indica si es par o impar.

Por último encontraremos una instrucción de color azul claro que corresponde al grupo de sensores, lo explicaremos en el siguiente apartado.

Interacción en Scratch

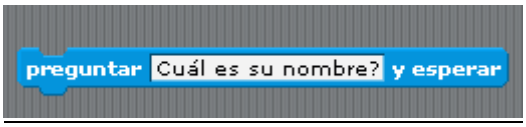
En este apartado vamos a conocer como simular interacciones en Scratch. Para ello usaremos principalmente los bloques pertenecientes al grupo sensores, pero también veremos que el grupo general contiene opciones interesantes.

Si vamos al bloque de sensores encontramos bloques tan intuitivos como los siguientes:

Con el primero podemos elegir con que sprite del programa está interactuando y con el segundo podemos hacer la misma comprobación usando colores. A continuación vamos a ver los distintos tipos de interacción:

- Entre sprites: Estos son los de la imagen anterior, nos permiten preguntar a los sprites su estado y si están interactuando con algo de su entorno. Como ya mencionamos en la sección anterior, estos bloques se suelen usar en alternativas o condicionales.

- Por teclado: Esto lo hacemos mediante el siguiente bloque:



  Esta instrucción nos permite mediante un cuadro de diálogo obtener un parámetro a través del teclado. En este caso el nombre del usuario. Este dato será guardado en el bloque respuesta que encontramos en sensores. A partir de ahora nos referiremos a ese tipo de bloques como variables, las cuales, se introducen en el siguiente apartado.
- Con el ratón: Podemos comprobar en todo momento la situación del ratón, la distancia entre el ratón y el sprite en el que se está ejecutando el código y comprobar si el botón izquierdo está siendo pulsado.

En el programa *sensores y variables* que se adjunta encontramos una serie de ejemplos sencillos que muestran el funcionamiento de algunos de los bloques de sensores. También se introducen conceptos de variables las cuales explicaremos a continuación

¿Qué son las variables?

Las variables, como en todo lenguaje de programación, son estructuras que pueden ir modificando su contenido a lo largo de la ejecución según se precise. En

Scratch tenemos una serie de variables ya definidas por el propio sistema, algunos ejemplos son:

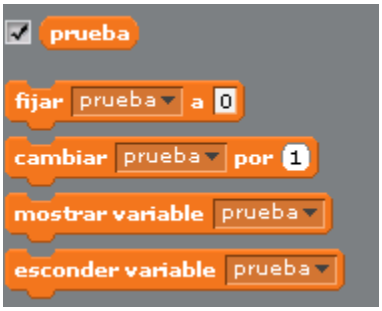
Todas las variables definidas por el sistema contienen datos a tiempo real, las 2 primeras, por ejemplo, contienen en todo momento las coordenadas del ratón en el programa.

Este tipo de estructura es extremadamente útil cuando tratamos la interacción en Scratch, no solo nos permiten conocer el estado de lo que representan, sino que además nos permiten llevar cuenta de vidas, puntuación, tiempo, opciones, etc. dentro de nuestro programa.

Para trabajar con variables vamos a su grupo en el interfaz y encontraremos las opciones:

Con el primer botón crearemos una variable única con un nombre a nuestra elección y con una lista definiremos una estructura que es capaz de contener todas las variables que queramos para así poder tratarlas en grupo en vez de una en una.

Al crear una variable aparecerán las siguientes opciones:



El tick en el lado izquierdo del nombre de nuestra variable indica que, en el escenario, se está mostrando su contenido. Podemos controlar su aparición quitándole el tick o bien usando las instrucciones mostrar/esconder variable. Por otro lado, podemos fijar y cambiar el contenido de la variable con dos bloques que nos quedan, son muy útiles para resetear el valor de la variable y/o cambiar su valor según necesitemos.

En el ejemplo *sensores y variables* encontraremos un pequeño contador que nos permitirá entender mejor el uso de estas estructuras.

Con las listas se trabaja de forma similar añadiendo las funciones de añadir, eliminar y sustituir objetos/variables de la lista.

Con esta exposición termina la introducción al lenguaje Scratch.

BYOB

¿Qué es BYOB?

BYOB (build your own blocks) es una versión extendida de Scratch. Incluye todas las funciones de Scratch además de permitirnos la creación de nuestros propios bloques, es decir, podemos agrupar un conjunto de instrucciones que vayamos a usar de manera periódica en un único bloque para futuros usos e incluso dejarla como parte de nuestra librería.

Actualmente se encuentra en la versión 4.0 y ha cambiado su nombre por Snap!, esta nueva versión nos permite trabajar en línea con el programa a través de nuestro navegador sin necesidad de instalar ningún tipo de Software.

Dada la complejidad de los nuevos añadidos al lenguaje me limitare a comentar algunos de los que considero más importantes ya que requieren un conocimiento más profundo de los lenguajes de programación necesario para entender el funcionamiento de Scratch.

Añadidos de BYOB/Snap!

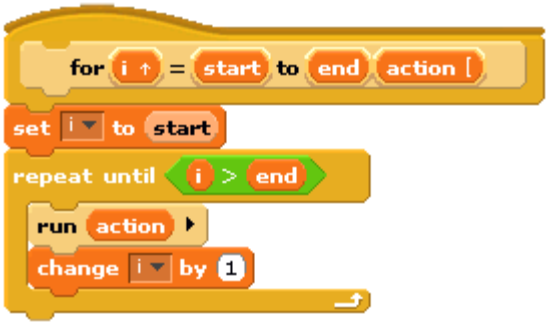
Algunos de los extras que encontramos en BYOB/Snap! respecto a Scratch es la aparición de nuevos bloques que nos permiten la creación de funciones. Estos bloques son los siguientes:

Gracias a estos bloques tenemos acceso a todas las opciones que nos brinda el uso de funciones en un programa software.

Los bloques “run” y “call” permiten la ejecución y creación de funciones que, o bien podemos definir dentro del mismo bloque, o bien tenemos definidas aparte gracias a la creación de bloques propios.

Este ejemplo representa el bloque “for” que encontramos en Scratch.

Y su implementación en BYOB sería:



El bloque “launch” nos permite la ejecución de scripts independientes al del propio sprite. Gracias a esto podemos simular el funcionamiento de hilos de ejecución, es decir, poner en funcionamiento programas de manera paralela en vez de forma secuencial. Esto permite ejecuciones mucho más eficientes.

Las flechitas que encontramos en los bloques descritos anteriormente nos permiten pasar parámetros a las llamadas de la función, por ejemplo en el caso del for le pasaríamos el valor de la i.

Este es solo uno de los ejemplos de la capacidad que tiene BYOB/Snap! frente a Scratch.

En definitiva, gracias a BYOB/Snap! tenemos acceso a un lenguaje de programación mucho más avanzado y similar al que nos brindan lenguajes tales como, Java o C++.

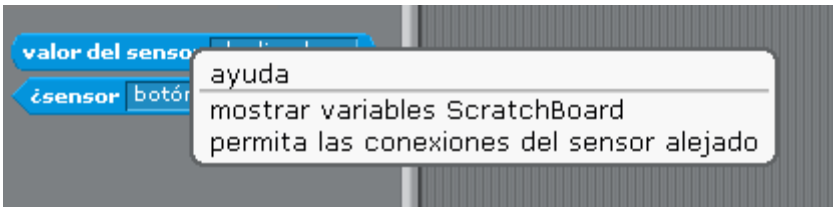
Interacción con el exterior

Una vez familiarizados con el mundo de Scratch procederemos al estudio de una de sus características más atractivas, su capacidad de recibir datos de elementos externos y trabajar con ellos como si de sensores propios de Scratch se trataran.

¿Cómo se comunica?

El entorno Scratch dispone de un sistema de comunicaciones con aplicaciones externas mediante el uso del protocolo TCP/IP. Esta comunicación se realiza a través del puerto 42001.

Para permitir que Scratch reciba/envíe señales debemos abrir dicho puerto. Para ello basta con ir al apartado sensores, situarnos en cualquiera de los dos últimos bloques y un click con el botón derecho.



Al seleccionar "permita las conexiones del sensor alejado" el programa nos devolverá una notificación que nos indica que el puerto está preparado.

Una vez realizada esta acción podemos comunicarnos con Scratch mediante el siguiente protocolo.

sensor-update nombre_de_la_variable valor_numero
broadcast nombre_de_la_senal

Mandando mensajes con la estructura anteriormente descrita somos capaces de crear nuevos sensores a los que podemos ir dando valores así como actualizar los valores que contienen. Por desgracia, debido a como recibe los mensajes Scratch (obliga a que los dos primeros bytes del comando o mensaje indiquen el tamaño del mensaje),

no podemos probarlo mediante telnet por lo que necesitaremos de la creación de pequeños programas que nos permita comunicarnos con él.

Mediante el uso de telnet podemos probar como Scratch manda señales hacia el exterior. Para ello usaremos el envío de señales propio de Scratch.

Pero antes deberemos habernos conectado por telnet al puerto, tras estar conectados mandamos la señal y obtenemos lo siguiente.

```
laptop@laptop-PC:~$ telnet 172.18.65.25 42001
Trying 172.18.65.25...
Connected to 172.18.65.25.
Escape character is '^]'.
broadcast "probando!!"
```

De este modo vemos como, usando las señales propias de Scratch, también nos podemos comunicar con elementos exteriores.

Por otro lado, las variables creadas en Scratch también son comunicadas al exterior cada vez que son actualizas para que todos los dispositivos que estén conectados tengan los valores al día.

Dicho esto podemos concluir que los datos que son transmitidos desde el exterior a Scratch son, o bien señales o bien valores que generaran sensores nuevos o los actualizarán. Y los que son retransmitidos desde Scratch al exterior serán señales o actualizaciones de las variables creadas en Scratch.

Ejemplos de comunicación: Python

Ya hemos visto como Scratch puede comunicarse mediante el uso de broadcast y cambios en el valor de sus variables globales con aplicaciones capaces de establecer conexiones remotas. Ahora vamos a centrarnos en mandar señales desde el exterior a Scratch, para ello nos serviremos de los protocolos descritos anteriormente.

sensor-update nombre_de_la_variable valor_numero
broadcast nombre_de_la_senal

Para este primer intento de comunicación con Scratch vamos a hacer uso de un pequeño programa en Python.

Toda comunicación con Scratch se hace mediante el uso de sockets, por lo tanto la primera parte de nuestro código consistirá en la inclusión de las librerías necesarias para ello.

```
from array import array
import socket
import time

HOST = 'localshot'
PORT = 42001
```

También aprovechamos para definir el host y el puerto al que el socket debe conectarse, como ya se ha mencionado el puerto que usa Scratch es el 42001.

A continuación preparamos el código encargado de abrir el socket y establecer la conexión.

```
socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket.connect((HOST, PORT))
```

Para un uso más cómodo del programa, vamos a permitir al usuario introducir por pantalla el código que desea mandar indicándole la escritura correcta.

```
raw_input()
resp=raw_input("Uso: [sensor nombre_sensor valor | broadcast nombre_señal]: ")
datos=resp.split(' ')
```

Con la función split nos preparamos para ver qué clase de comando ha elegido y la tratamos según la elección.

```
opcion=datos[0]

if opcion == "sensor" :
        sensor=datos[1]
        valor=datos[2]
        comando= "sensor-update " + sensor + " " + valor
        comandoAScratch(comando)
if opcion == "broadcast" :
        sensor=datos[1]
        comando="broadcast \"" + sensor +"\""
        print comando
        comandoAScratch(comando)
```

Por último definimos la función comandoAScratch que será la encargada de crear y mandar la trama completa que se debe mandar a Scratch.

```
def comandoAScratch(cmd):
    n = len(cmd)
    a = array('c')
    a.append(chr((n >> 24) & 0xFF))
    a.append(chr((n >> 16) & 0xFF))
    a.append(chr((n >>  8) & 0xFF))
    a.append(chr(n & 0xFF))
    socket.send(a.tostring() + cmd)
```

Para terminar cerramos la conexión y con ello acaba la ejecución del programa.

```
socket.close()
```

Se adjunta el programa *comunicacionScratch.py.*

Si ejecutamos el programa comunicacionScratch.py

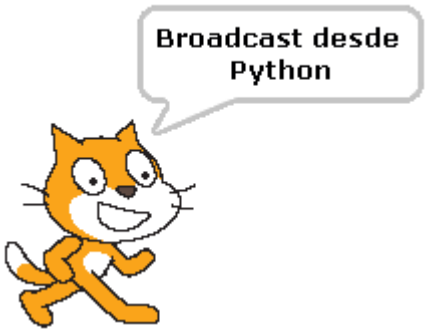
```
python sendScratchCommand.py
```

y escribimos

```
Conectando...
conectado
Uso: [sensor nombre_sensor valor | broadcast nombre_senal]: broadcast python
broadcast "python"
closing socket...
done
```

Ahora, en Scratch preparamos un receptor de señal como el siguiente

Al mandar el comando desde Python obtenemos



A continuación probaremos los sensores:

```
Uso: [sensor nombre_sensor valor | broadcast nombre_senal]: sensor python 5
```

En el aparatado de sensores de Scratch encontraremos lo siguiente

Y si pulsamos para ver su valor veremos que es

Ejemplos de comunicación: ActionScript3

Este es un ejemplo análogo al anterior haciendo uso de ActionScript3.

```
var nuevo:TextField= new TextField();
nuevo.type=TextFieldType.INPUT;
addChild(nuevo);
nuevo.x=230;
nuevo.y=200;
nuevo.width=200;
nuevo.height=30;
nuevo.border=true;
nuevo.addEventListener(KeyboardEvent.KEY_UP,checkForReturn);
```

Creamos un campo de texto para que sea más fácil la introducción del comando en el programa. Además creamos un eventListener que será el encargado de detectar que se pulsa la tecla Enter para el envío del mensaje. El listener tendrá el siguiente código:

```
function checkForReturn(event:KeyboardEvent){
    if(event.charCode==13){
        acceptInput();
    }
}
```

El código 13 es el que indica que se trata de un Enter. Tras comprobar que si se trata de un enter llamamos a la función acceptInput que será la encargada de comprobar el contenido del texto y derivarlo en la función que le corresponda. Para ello haremos uso de expresiones regulares, con ellas comprobaremos si hace referencia a un sensor o a un broadcast.

```
function acceptInput(){
    var entrada:String="";
    entrada=nuevo.text;
    var broadcast_pattern:RegExp=/broadcast\s.*/;
    var sensor_pattern:RegExp=/sensor\s.*\s\d/;
    var resultado:Array;
    resultado=broadcast_pattern.exec(entrada);
    if (resultado!=null){
        broadcast(resultado.toString());
    }
    resultado=sensor_pattern.exec(entrada);
    if (resultado!=null){

        var result:Array;
        result=entrada.split(" ");
        update(result[1],result[2]);
    }
}
```

Tras la comprobación del tipo de mensaje que es procederemos a mandar los valores a las funciones encargadas de construir el mensaje de forma correcta.

```
function update(variable:String,value:*):void {
  if(!sock.connected) return
  value=String(value);
  sendScratchMessage('sensor-update "'+variable+'" "'+value+'"');
}

function broadcast(broadcast:String):void {
    if(!sock.connected) return
    var index:int
    index=broadcast.indexOf(" ") +1
    broadcast=broadcast.substring(index)
    sendScratchMessage('broadcast "'+ broadcast+'"');
}
```

Como se puede observar el mensaje que construyen es similar al que generábamos en Python.

Por último llamamos a la función encargada de mandar el mensaje a Scratch, pero antes de eso necesitamos definir el socket que utilizaremos para mandarlo.

```
var sock:Socket = new Socket();
sock.connect('localhost',42001)
```

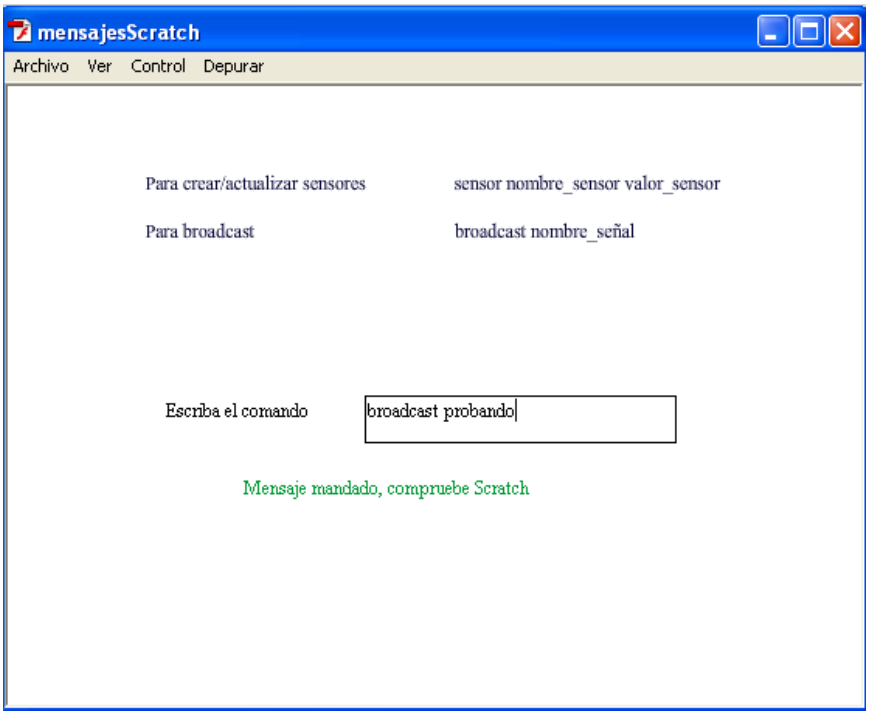
Como ya tenemos definido el socket podemos proceder a llamar a la función que mandará el mensaje por él.

```
function sendScratchMessage(s:String):void{
  var bytes:ByteArray=new ByteArray();
  bytes[0]=0;
  bytes[1]=0;
  bytes[2]=0;
  bytes[3]=s.length;
  for (var i:Number=0; i<4; i++) {
    sock.writeByte(bytes[i]);
  }
  sock.writeMultiByte(s,"us-ascii");
  sock.flush();
}
```

Este programa podemos encontrarlo adjunto con funcionalidades extra (escucha lo que manda Scratch) con el nombre *mensajeScratch.fla.*

Para poder probarlo vamos a crear un sencillo programa que consta de 3 scripts. Por un lado prepararemos una parte capaz de mandar mensajes y otra de recibirlos.

Una vez tenemos preparados los dos bloques nos aseguramos de que tenemos abierto el puerto para conexiones remotas y ejecutamos el programa Flash.



Escribimos lo que aparece en la imagen y al mandarlo observamos que en Scratch sucede lo siguiente.

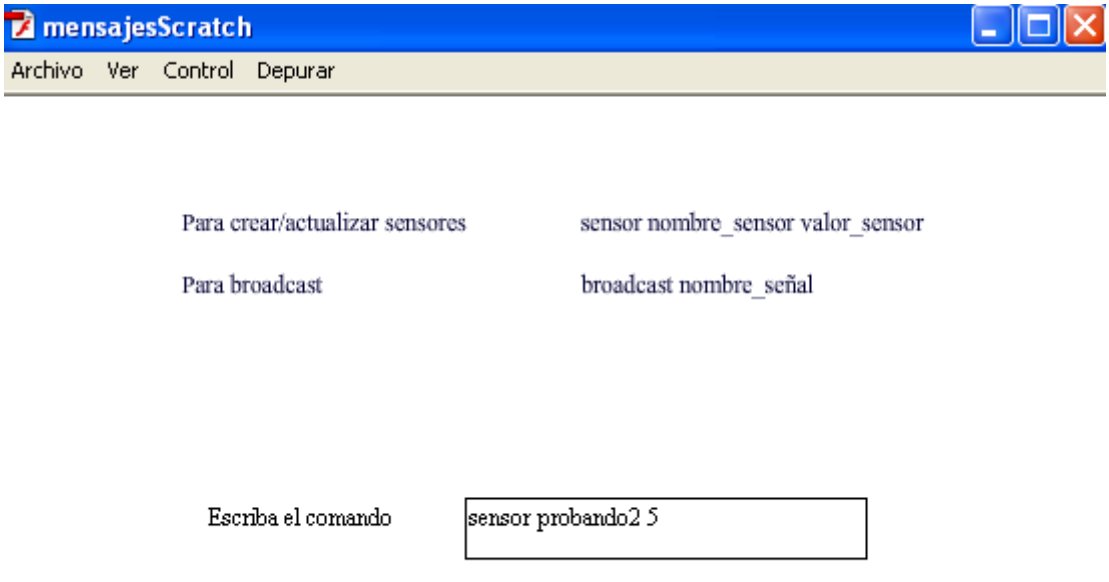
Como podemos ver el mensaje ha sido mandado correctamente.

A continuación ejecutaremos el bloque encargado de mandar la señal desde Scratch. En flash veremos aparecer lo siguiente en la salida estandar.

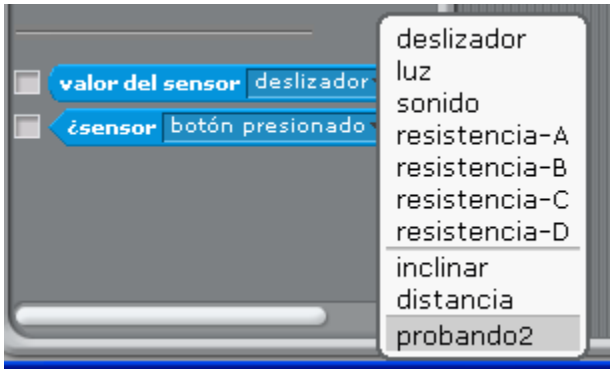
```
broadcast "probando desde Scratch"
Scratch ha hecho un broadcast:  probando desde Scratch
```

De este modo vemos que Flash también es capaz de leer las señales desde Scratch.

Ahora vamos a estudiar la comunicación que se realiza mediante variables en ambos sentidos. Para empezar vamos a mandar un mensaje de tipo sensor desde Flash.



Entonces, al acceder a la parte de sensores de Scratch, nos encontraremos con:

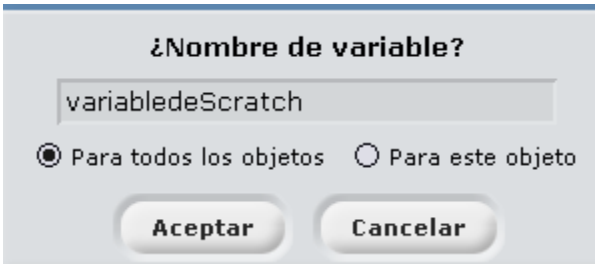


Vemos que el sensor ha sido recibido, ahora comprobaremos su valor seleccionándolo y activando el tick.

Y en el escenario nos saldrá el valor del sensor, en este caso 5.

Usando este método podemos actualizar las veces que queramos el valor del sensor para así poder simular la toma de datos externos.

Veamos el sistema contrario, actualizar valores desde Scratch. Para ello se hace uso de variables así que crearemos una nueva.



Una vez creada la variable vemos que en la salida estándar de flash aparece lo siguiente

```
sensor-update "variabledeScratch" 0
```

A partir de este momento cada vez que actualicemos el valor de esta variable se mandará un mensaje similar a flash. Por ejemplo si usamos esto

En Flash obtendremos

```
sensor-update "variabledeScratch" 1
```

Conclusión de comunicaciones

Como hemos podido ver con estos dos ejemplos, Scratch permite comunicaciones externas de manera muy sencilla. Cualquier lenguaje con capacidad de conexión mediante el uso de Sockets puede servirnos para crear un programa fácil capaz de permitirnos conexiones con él así como de simular el funcionamiento de sensores para una mayor interactividad. En las próximas secciones vamos a estudiar la interacción con Scratch a través de los dispositivos externos Kinect y Arduino.

Kinect

¿Qué es Kinect?

Kinect fue anunciada por primera vez durante el E3 (Electronic Entertainment Expo) de 2009, originalmente conocido como Project Natal, fue definida como un controlador de juego libre creado por Alex Kipman. Desarrollado por Microsoft como periférico para su videoconsola Xbox 360 y lanzado en 2010. Un año después se lanzó una versión compatible con PC para los sistemas operativos Windows 7 (y posteriormente Windows 8) además de un Software Development Kit (SDK) libre para brindar a la comunidad de desarrolladores e investigadores académicos con la oportunidad de trabajar con toda la potencia del dispositivo. El SDK permite la creación de aplicaciones en C++, C# y Visual Basic a través de Microsoft Visual Studio.

Algunas de las características de este SDK son:

- Acceso al flujo de datos no procesados de profundidad, cámara de color y micrófonos.
- Capacidad de seguimiento esquelético de hasta dos personas para la creación de aplicaciones de gesticulación.
- Procesamiento de audio: supresión de ruido, cancelación del eco y, a través del API de Windows, reconocimiento de voz.
- Fácil instalación y una extensa documentación técnica.

Características técnicas

El sensor Kinect es una barra horizontal de unos 23 cm situada sobre una base con eje de articulación de rótula.

Cuenta con una cámara RGB, un sensor de profundidad, un micrófono de múltiples matrices y un procesador encargado de ejecutar el software que trae implementado. Este software es el encargado de capturar el movimiento del esqueleto en 3D, reconocimiento facial y de voz. El sistema de micrófonos múltiples permite la participación en los chats de Xbox Live (plataforma online de videojuegos de Microsoft) sin el uso de auriculares gracias a la focalización y supresión del ruido que es capaz de procesar.

El sensor de profundidad está formado por un haz de infrarrojos combinado con un sensor monocromo que permite a Kinect ver cualquier sala en 3D bajo cualquier tipo de luz ambiental.

La cámara RGB usa un flujo de 8 bits capaz de retransmitir a una resolución VGA de 640x480 haciendo uso del *filtro de Bayer*. También es capaz de trabajar a 1240x1024 con tasas de fotogramas más bajas así cómo usar otros formatos de colores tales como el espacio de color *YUV*.

El sensor monocromático es capaz de trabajar con resoluciones de 640x480 con una profundidad de 11 bits. Con ello se obtienen hasta 2048 niveles de sensibilidad. Es capaz de proporcionar una visión en directo de la vista de la cámara monocroma resoluciones de 640x480 y 1240x1024.

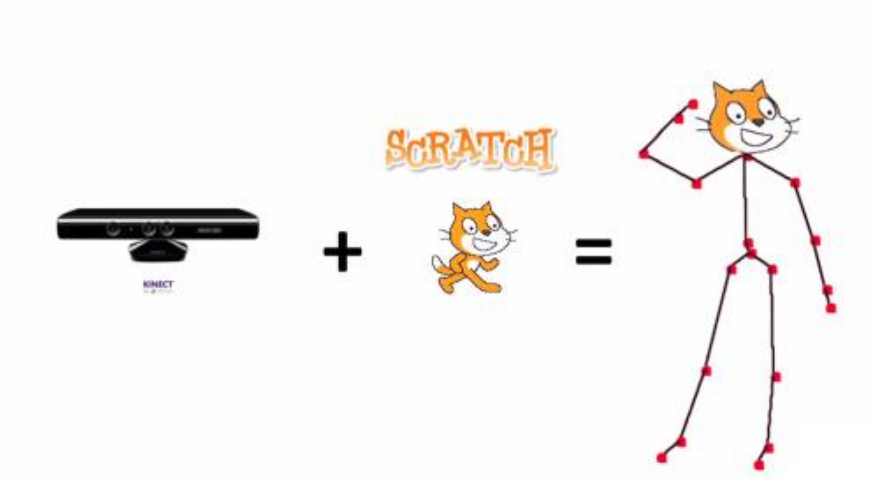
Se precisa una distancia de entre 1.2m hasta 3.5m para que el uso del software de Kinect sea eficiente, por lo tanto el espacio necesario aproximadamente para hacer uso de todo su potencial es de unos $6m^2$.

Debido a la motorización de la rótula del dispositivo Kinect fue necesaria la adición de una fuente de energía externa ya que el puerto USB propio de la consola Xbox 360 no era capaz de proporcionarla. En la consola Xbox 360 S este problema fue solucionado mediante la implementación de un puerto auxiliar capaz de alimentarla.

Kinect2Scratch (K2S)

Ahora que ya conocemos el dispositivo Kinect, el contenido de su SDK y las capacidades del mismo vamos a estudiar el funcionamiento de un software llamado Kinect2Scratch.

Podemos definir Kinect2Scratch como:



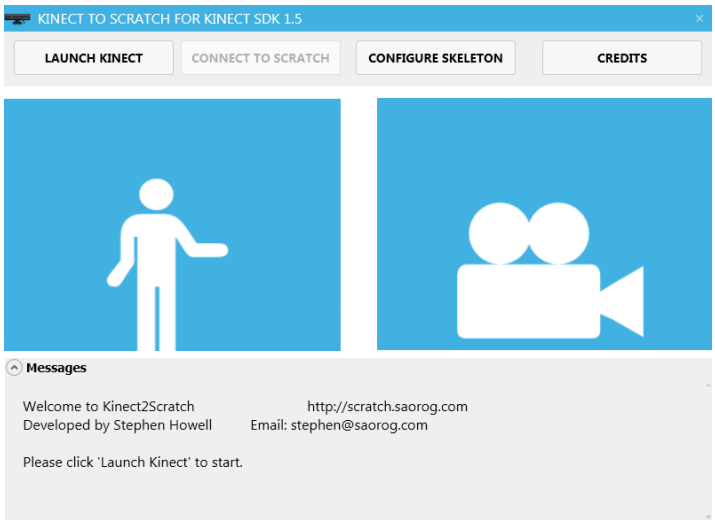
Este software ha sido desarrollado por Stephen Howell y está disponible en http://scratch.saorog.com. Actualmente se encuentra en su versión 2.5.

Basado en la mismas ideas comentadas en el capítulo de *Interacción con el exterior* y haciendo uso del SDK de Kinect, Stephen Howell ha desarrollado un software capaz de transmitir una gran parte de los datos propios de Kinect a Scratch dando a los usuarios, de manera gratuita, una nueva manera de trabajar e interactuar con Scratch.

Utilizando Kinect2Scratch

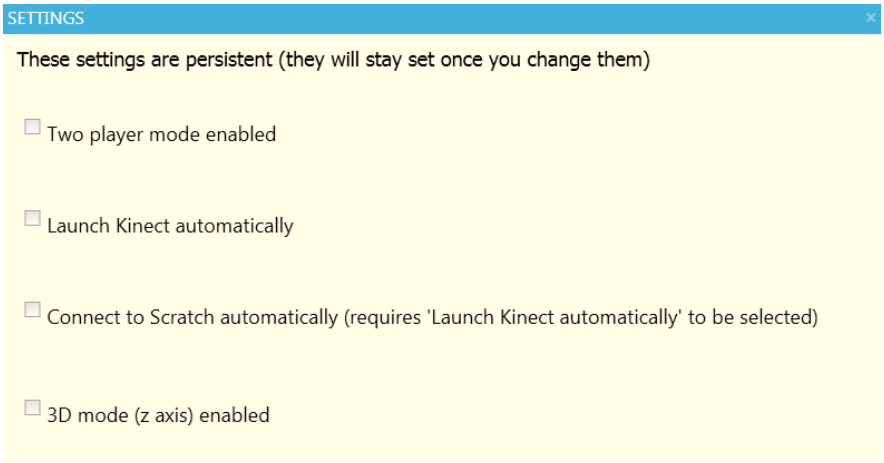
Antes de nada, debemos asegurarnos de que disponemos de todo lo necesario para hacer que el software de Kinect funciona, es decir, necesitamos un sistema operativo Windows 7 u 8, y la versión más reciente del SDK de Kinect, el cual podemos encontrar en la web de Microsoft. Una vez instalado el SDK de Kinect y Kinect2Scratch procederemos a lanzar el software.

Al lanzar el software se nos mostrará un interfaz similar al siguiente:



Este interfaz, muy intuitivo, nos da un acceso rápido a todas las opciones de las que dispone el software.

Comencemos por configure skeleton:

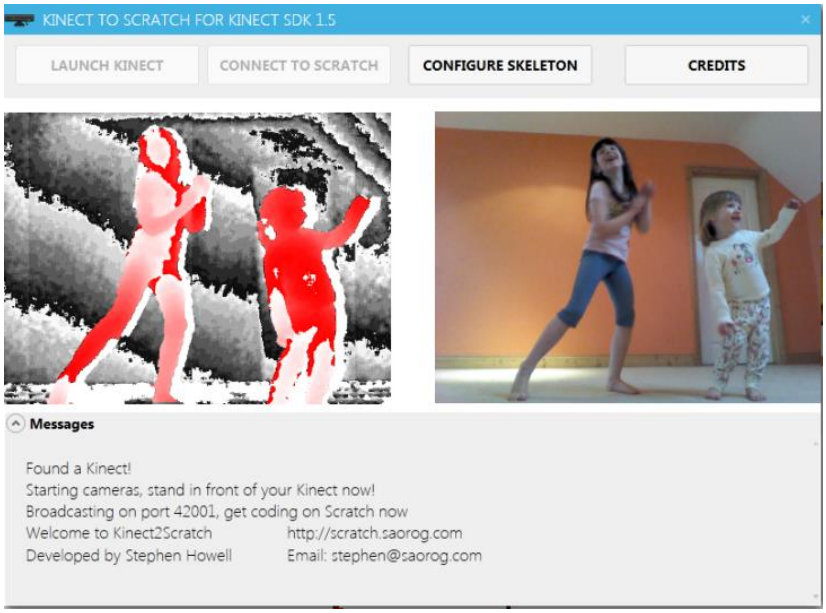


En esta ventana tenemos acceso a los parámetros de configuración del software. Las opciones son:

1. Two player mode enabled: Activando esta opción permitimos que Kinect mande a Scratch datos de dos jugadores o esqueletos.
2. Launch Kinect automatically: Permite al software lanzar la Kinect al inicio de la aplicación sin necesidad de hacerlo manualmente agilizando el proceso de puesta en marcha.
3. Connect to Scratch automatically: Opción que necesita de la anterior por defecto, posibilita que el software comience a mandar los datos desde Kinect a Scratch nada más lanzarla.
4. 3d mode enabled: Permite el envío de datos sobre el eje z, profundidad.

Ahora que ya conocemos las opciones básicas de Kinect2Scratch vamos a ver como comenzar el proceso de comunicación

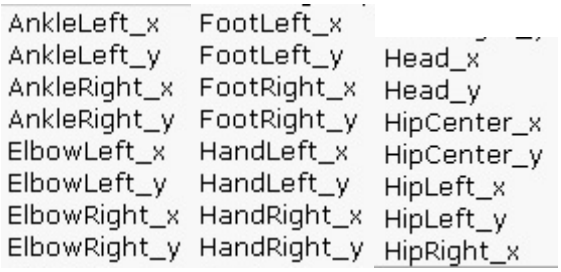
Para ello pulsaremos el botón *Launch Kinect* y, si tenemos conectada una Kinect al sistema operativo el interfaz debería ser similar al siguiente.



En la parte izquierda tenemos las imágenes del sensor monocromo y a la derecha las de la cámara. De las imágenes que pertenecen a la cámara monocroma veremos que aparecen figuras resaltadas en rojo, esto significa que Kinect lo reconoce como un esqueleto y ya es capaz de trabajar con los datos del mismo.

Una vez vemos que funciona de manera correcta podemos pasar a pulsar el botón *Connect To Scratch.* Este botón pone en funcionamiento el protocolo de comunicación entre Kinect y Scratch, ya podemos empezar a trabajar en Scratch.

Recordemos que para que el paso de datos entre ambos dispositivos sea correcto deberemos haber activado la opción de dispositivos externos en el grupo de sensores. Una vez activado el acceso a sensores remotos el bloque "valor del sensor ---" tendrá acceso a todos los datos transmitidos desde Kinect.



Estos son algunos ejemplos de lo que encontraremos dentro de ese bloque. Como se puede observar se trata de las coordenadas x e y de cada una de las partes de nuestro cuerpo. En caso de que tuviésemos activada la opción de 3D, vista en el apartado anterior, también tendríamos acceso a la coordenada Z de cada parte de nuestro cuerpo. Del mismo modo, si estuviese activada la opción de dos jugadores, tendríamos acceso a las coordenadas de nuestro segundo jugador. Para diferenciarlos tendríamos por ejemplo AnkleLeft_x y AnkleLeft_x2, de este modo haría referencia a uno de los dos esqueletos sobre los que obtendríamos información a través del dispositivo Kinect.

Ahora que ya sabemos dónde encontrar los datos que nos manda Kinect ha llegado el momento de crear nuestra primera aplicación haciendo uso de algún parámetro externo.

Primer contacto con Kinect2Sctrach

Para este primer ejemplo vamos a utilizar como base el programa con el que aprendimos a usar el lápiz.

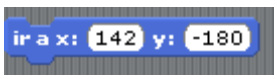
Una vez tenemos el ejemplo abierto lo que vamos a hacer es sustituir toda interacción a través de teclado o ratón por interacción con Kinect.

Para ello comenzaremos cambiando el esconder del programa por un mostrar ya que nos interesa tener un puntero.

Ahora que ya tenemos el puntero visible podemos cambiarlo a una forma que nos convenga, por defecto será el gato de Scratch.

Una vez tenemos un puntero lo siguiente es controlarlo con, por ejemplo, nuestra mano derecha. Para ello sustituiremos la instrucción

por un bloque que nos simule el movimiento del puntero en función de los parámetros pasados por Kinect sobre nuestra mano derecha. ¿Cómo podemos hacerlo? Con la orden



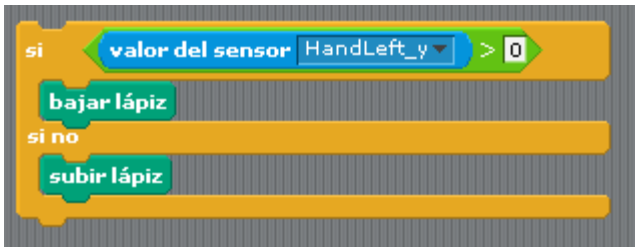
del mismo modo que el bloque anterior permitía una actualización constante de los valores x e y del ratón gracias a que estaba contenido en un "bloque por siempre". De modo que, si tomamos los valores del sensor que corresponden a la mano derecha

y los ponemos en las casillas que corresponden a x e y, simularemos el funcionamiento del primer bloque pero con las coordenadas de nuestra mano derecha.

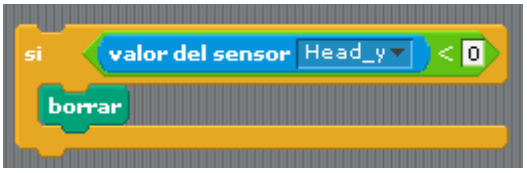
Ya tenemos nuestro puntero se dirige en función de nuestra mano derecha. A continuación vamos a definir cuándo debe escribir en el papel.

Vamos a definir que, en caso de que la mano izquierda este sobre el eje x de Scratch (media altura de nuestro cuerpo) escriba y en caso de estar por debajo no escriba.

Una vez más vamos a hacer uso de la instrucción condicional, en el programa original comprobábamos si el botón del ratón estaba pulsado esta vez, nos interesa saber si el eje y está por encima de 0.



¡Listo!, ahora solo falta decidir cuándo vamos a borrar el lienzo. He optado por un gesto que no hagamos de manera automática para evitar borrados indeseados, inclinar la cabeza. Como ya hemos visto, la posición 0 del eje y para Scratch está más o menos a media altura de nuestro cuerpo por lo que podemos usar aquí también este parámetro para definir hasta donde debe ser la inclinación. Así que cambiamos la función de borrado por barra espaciadora que teníamos y añadimos al bucle por siempre el siguiente código



Con esto ya tenemos terminado nuestro programa. Al ejecutarlo y probarlo un poco vemos como falla de vez en cuando el seguimiento de la mano derecha, esto se debe a que a pesar de que el ratio de refresco de los datos es, a priori suficiente, para una aplicación que pretende trabajar con esta "precisión" se queda corto. Con esto en mente ya podemos comenzar a trabajar y crear nuestros propios programas haciendo uso de Kinect2Scratch.

Se adjunta el programa *Ejemplo lienzo Kinect*

Conclusión

Con este ejemplo hemos visto de una manera sencilla como obtener los datos de Kinect y hacernos una pequeña idea de la capacidad de interactividad que nos brinda el uso de este software y de su hardware.

La capacidad de crear aplicaciones interactivas mediante el uso de este software permite el acercamiento de estas tecnologías a todos los públicos ya que, como todos sabemos, las cosas son siempre más llamativas si se prestan a la interactuación (un claro

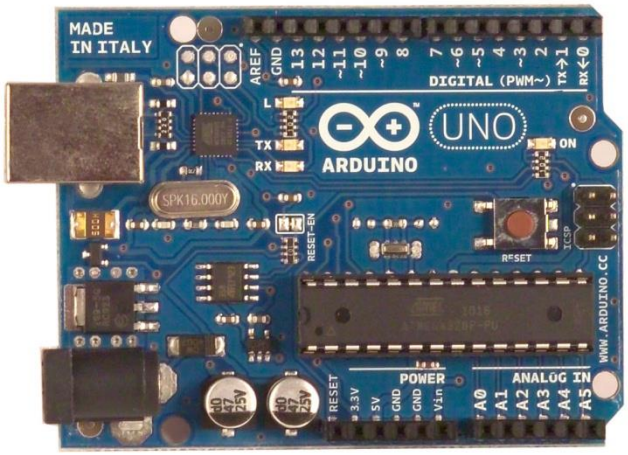
ejemplo es el éxito y la rápida adaptación de la población al uso de dispositivos Android/IOS).

A pesar de ello, uno de los grandes problemas que encontraremos será la usabilidad de estas tecnologías. Es decir, a pesar de ser interactivo ¿es de fácil uso? O ¿es práctico su uso? Más adelante haremos plantearemos algunos test especializados para comprobar este factor y obtener datos con usuarios reales.

Arduino

¿Qué es Arduino?

Arduino es una plataforma de hardware libre basada en una placa con un microcontrolador y un entorno de desarrollo principalmente utilizado para crear prototipos flexibles y de fácil uso. Fue creado para ser una plataforma de fácil acceso para todo aquel interesado en crear entornos y objetos interactivos.



La imagen superior muestra la placa *Arduino Uno*, esta es la que se utilizó para este proyecto.

El hardware Arduino consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Esto lo dota de una gran sencillez de utilización y de un bajo coste.

El software de desarrollo para este dispositivo es un entorno que implementa el lenguaje de programación Processing/Wiring.

Arduino puede usarse para desarrollar objetos autónomos y ser conectado con software en un ordenador (Flash por ejemplo). Además se puede programar en otros entornos como Java, ActionScript, C++, etc.

Especificaciones

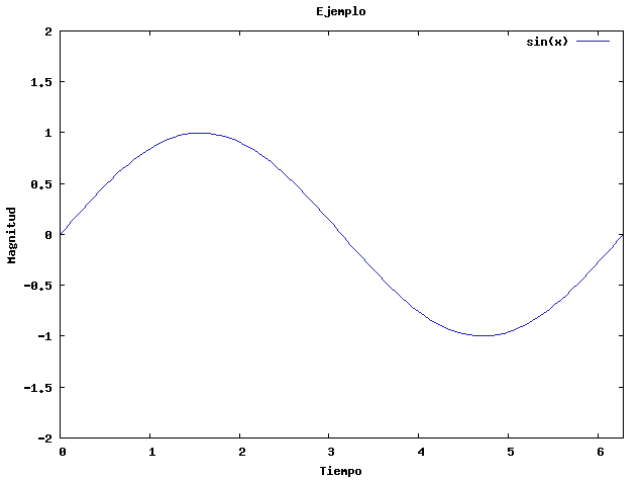
Dado que se ha trabajado sobre la placa Arduino Uno nombraremos las especificaciones de esta. Pero en cualquier caso la mayoría tienen unas características similares.

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

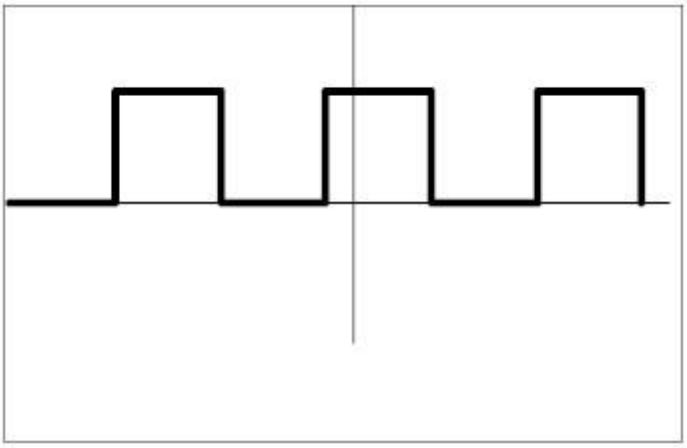
Lo más destacable de estas características, y lo que vamos a usar principalmente son las entradas y salidas (INPUTS/OUTPUTS) de señales digitales y analógicas.

Una entrada o input es un módulo que permite el envío de señales al sistema, en este caso el microcontrolador. Una salida (u output) por el contrario es el módulo encargado de mandar ese tipo de señales.

Además tenemos los dos tipos de señales: digital y analógica. Una señal analógica es aquella que viene definida por una función matemática continua. Es decir, dentro del rango de una señal puede tomar todos los valores del espectro.



Mientras que la señal digital es aquella que representa valores discretos y suelen representarse mediante el uso de dos niveles, High y Low (alto y bajo o 1 y 0).



Instalación del entorno Arduino

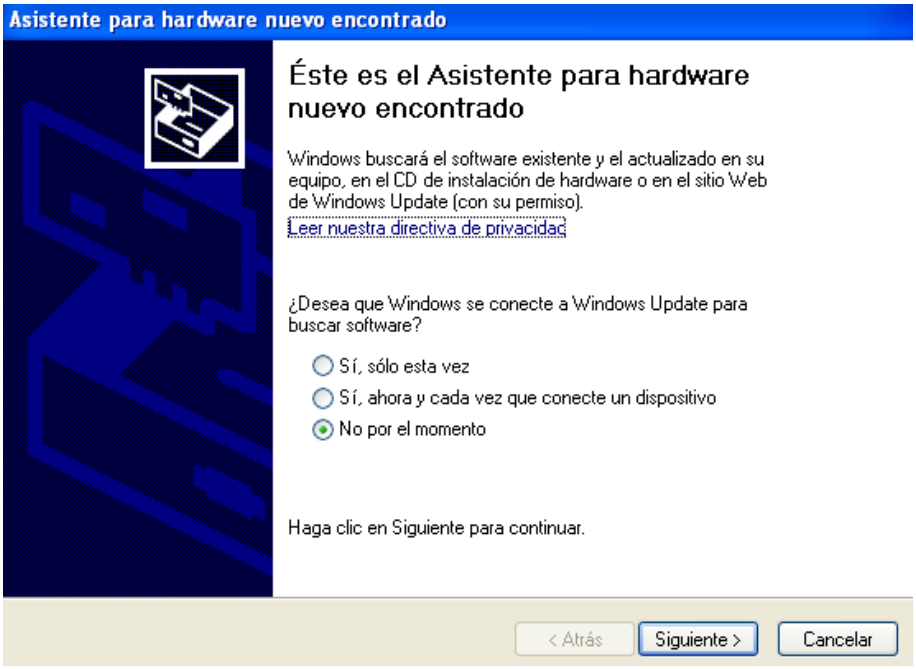
Para poder trabajar con Arduino debemos tener instalado un intérprete de Java y hacernos con la última versión del IDE de Arduino, que es de distribución gratuita.

A continuación se describe el proceso de instalación de los drivers de la placa.

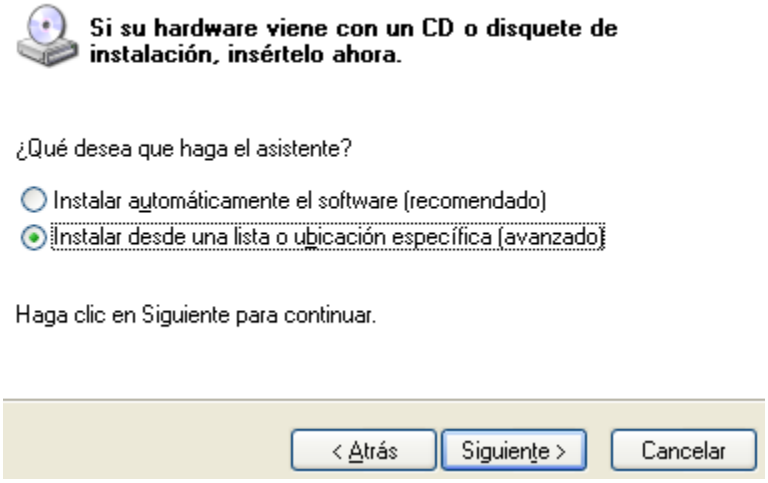
1. Conectamos la placa a un puerto USB y con ello debería encenderse un LED verde en el que estará escrito ON o PWR.
2. Una vez está bien conectado dependiendo del sistema operativo procederemos de la siguiente manera
    a. Windows 7: El driver debería instalarse de manera automática, en caso de no ser así seguir los mismos pasos que para la instalación en Windows XP.

    b. Windows XP: Al conectar la placa aparecerá el asistente para instalación de hardware de Windows.

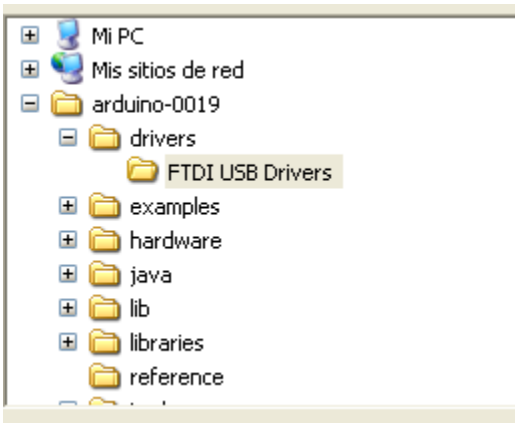
        i. Elegiremos la tercera opción: “No por el momento” y pulsaremos siguiente.



ii. En la siguiente pantalla elegiremos la segunda opción: "Instalar desde una lista o ubicación específica (avanzado)".

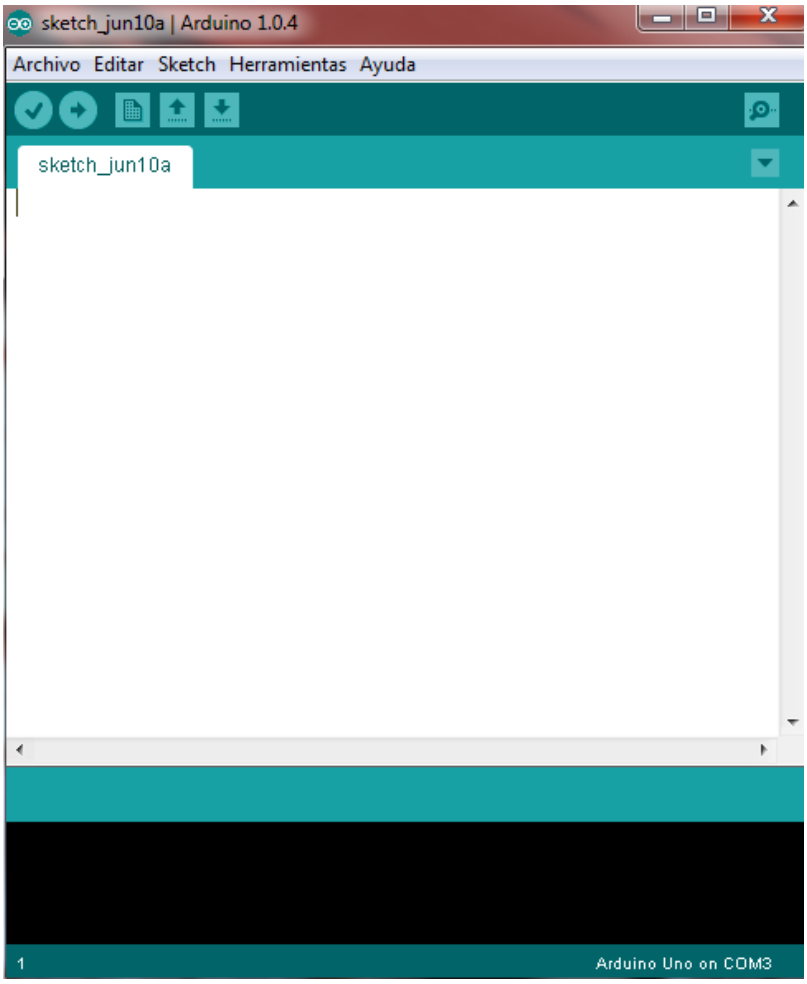


iii. A continuación nos aparecerá una pantalla donde debemos elegir la ruta del archivo/carpeta que contiene el driver que deseamos instalar. Seleccionaremos la opción "Incluir esta ubicación en la búsqueda" y buscaremos la carpeta drivers/FTI USB Drivers



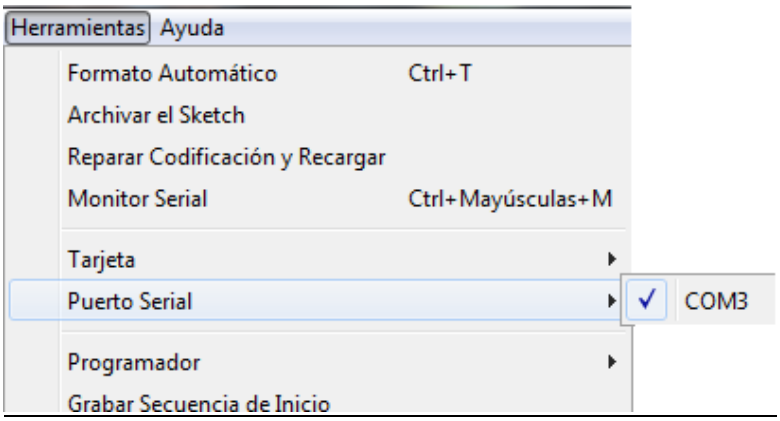
iv. Le daremos a siguiente y se instalarán los componentes necesarios para su correcto funcionamiento.

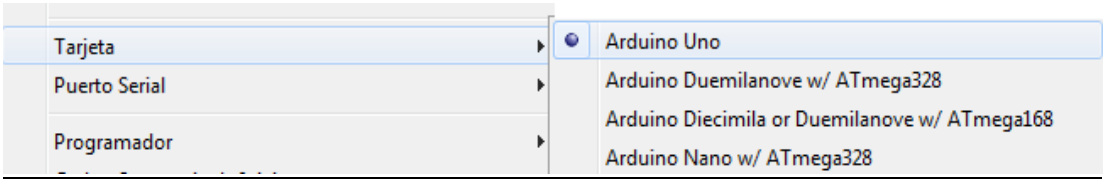
Una vez instalado correctamente procederemos a lanzar la aplicación y aparecerá el siguiente interfaz:



Este interfaz podemos dividirlo en 4 zonas:

1. Parte superior: en ella encontramos las opciones de File, Edit, Sketch, Tools y Help. Todas ellas contienen las opciones típicas de estos apartados.
    a. Archivo: nos permite trabajar con todo lo relacionado con ficheros. **También encontraremos una opción que da acceso directo a ejemplos de programación.**
    b. Editar: herramientas de edición de texto.
    c. Sketch: es el nombre utilizado para los programas que generamos. Aquí tendremos las opciones de comprobar si es correcto y compilación.
    d. Herramientas: herramientas del programa, nos permite elegir el puerto COM que queremos usar así como definir el tipo de placa que estamos usando.





    e. Ayuda: colecciones de ayuda.
2. Barra de accesos directos: se sitúa debajo de la barra de herramientas. Las opciones que nos ofrece de izquierda a derecha son: Verificar, Cargar en placa, nuevo, abrir fichero, guardar fichero y, a la derecha del todo, monitorear el puerto COM.
3. Parte central: aquí escribiremos nuestros programas o Sketch.
4. Parte inferior: la ventana de color negro es la ventana de salida del código en ejecución. Nos mostrará errores de compilación o tiempos de carga en caso de que todo haya ido bien.

Introducción al lenguaje Processing

Arduino dispone de una gran variedad de ejemplos para que el usuario. Vamos a centrarnos en dos puntos, Outputs e Inputs ya que son los más importantes.

Outputs

Comencemos por uno de los más sencillos. Seleccionamos Archivo → Ejemplos → Basics → Blink

Este será el código que contiene el ejemplo:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Este ejemplo se compone de dos funciones, estas funciones siempre forman parte del código.

1. Función setup()

   Es la encargada de indicar a la placa qué formato debe tomar cada uno de los pines de placa. En este caso indicamos al pin 13 que debe ser un output.

2. Función loop()

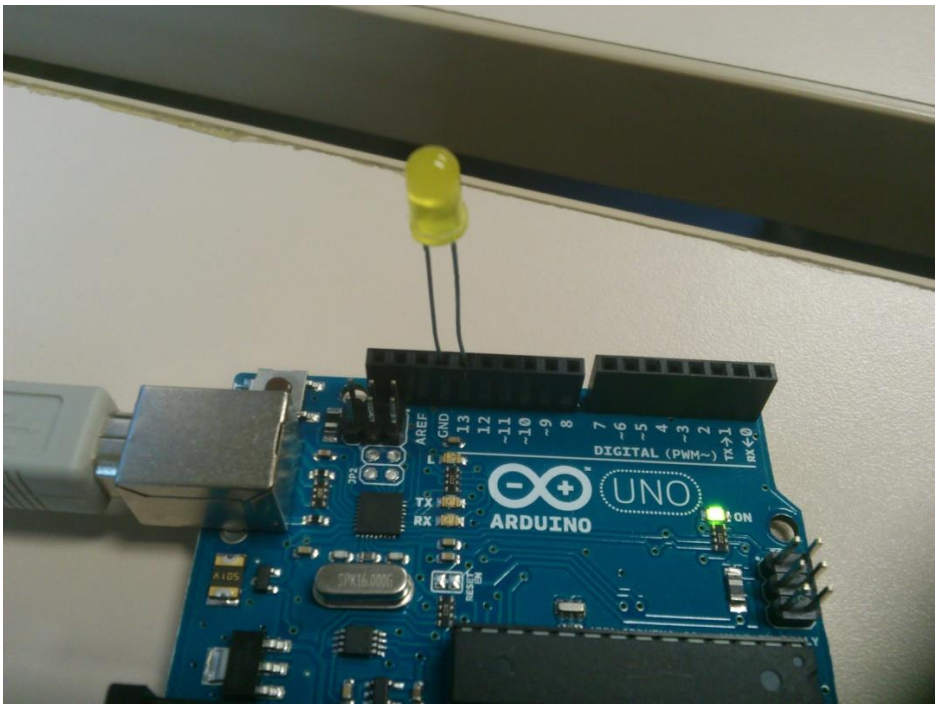
   Se trata del bucle principal del programa, es el encargado de ejecutar el código. En este caso consiste en poner el LED en ON, esperar 1 segundo y ponerlo en OFF y esperar otro segundo. Esto se repite de manera indefinida hasta que el usuario lo aborta de manera manual pulsando el botón reset de la placa (botón físico que encontramos en l aplaca).

Por lo tanto tenemos que la función digitalWrite () mediante el uso de dos parámetros configura una salida.

Probamos el ejemplo dándole al botón de acceso rápido para cargar y … ¿no nos hemos olvidado algo? Así es, para que la ejecución del programa sea visible y comprobemos su correcto funcionamiento, ¿no deberíamos tener un LED conectado?

Como hemos indicado en el código que vamos a usar el PIN 13, tenemos que conectar un LED a este PIN. Recordemos que los LED tienen 2 patillas, una más corta que la otra. La corta debe conectarse a una toma de tierra o GRD, mientras que la otra la podemos conectar al PIN que la activa

Ya que se trata de nuestro primer ejemplo no haremos uso de la placa prototipo, vamos a conectar la patilla larga en 13 y la corta en el GND que se sitúa a su lado de la siguiente manera.



En caso de que ya tengamos el programa cargado, el LED comenzará a iluminarse. En caso contrario le daremos a cargar el programa en la placa y observaremos el cambio

Ya hemos visto, de manera muy sencilla, cómo funcionan las salidas u outputs de Arduino).

A continuación vamos a ver un ejemplo de entradas.

Inputs

Tras comprobar el funcionamiento de las salidas por parte de Arduino, ahora solo nos queda ver como tomar datos de elementos externos. Para ello haremos uso del emisor más básico de todos, un pulsador.

Este pulsador es digital y, al igual que todos los demás, solo tiene dos posibles señales: pulsado y no pulsado, es decir 1 y 0 respectivamente. Una vez que tenemos claro este concepto pasamos a abrir el ejemplo que encontramos en Archivo → Ejemplos → Digital → Button

El código del ejemplo es el siguiente:

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

En este ejemplo se introducen las constantes y las variables. Las constantes son valores permanentes a lo largo de todo el programa, es decir no cambian su valor como lo harían las variables.

Como constantes tendremos los pines de los que haremos uso, el 2 para el pulsador y el 13 para el LED.
Como variable usaremos una llamada “buttonState” que indicará el estado del pulsador, es decir, tomará el valor que leamos desde él.

Ahora vamos a observar cómo están definidas las funciones setup () y loop ()

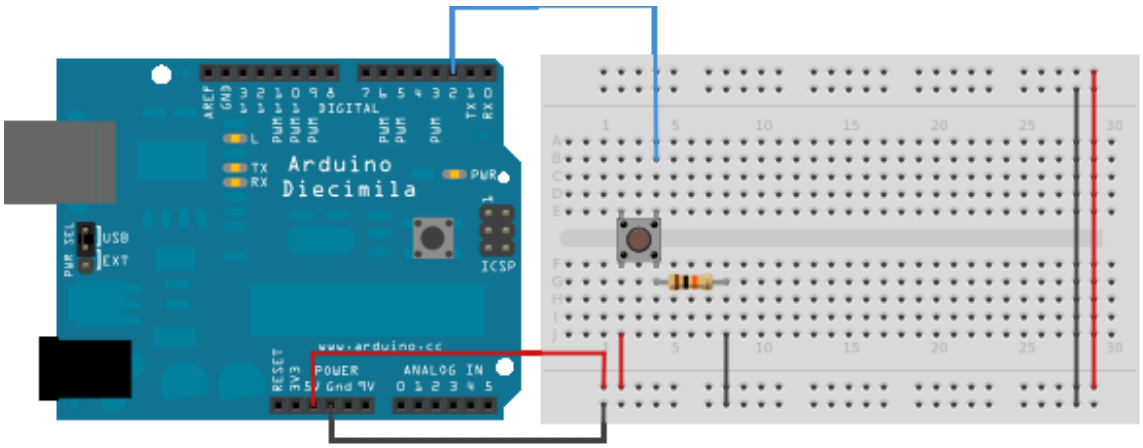
1. Función setup()

De forma análoga a lo que hicimos en ella durante el ejemplo del output declararemos el modo de funcionamiento de cada uno de los PIN que vamos a utilizar. En este caso usaremos de nuevo el PIN 13 como una salida y el PIN 2 como una entrada. Además definimos el 13 como un PIN de LED y el 2 como un botón.

2. Función loop()

   El bucle principal de este programa se encargará de leer en todo momento el estado del pulsador y guardarlo en la variable "buttonState" y en función de su valor, HIGH (1) o LOW (0) encenderá el LED del PIN 13 como lo hacíamos durante el ejemplo de salidas.

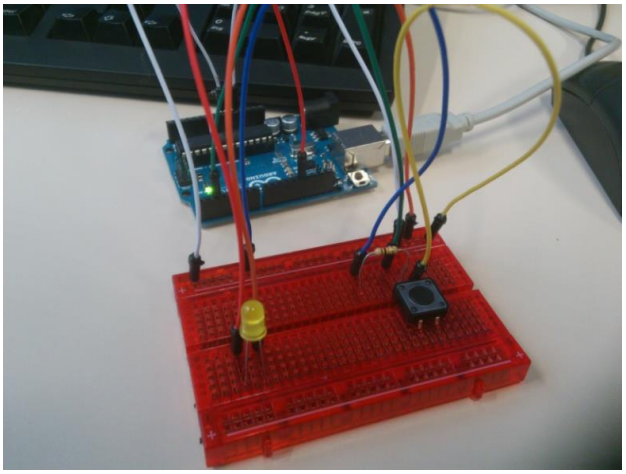
Una vez comprendido el código debemos pasar a crear el circuito que nos permitirá comprobar su funcionamiento. En este caso haremos uso de una placa prototipo, una resistencia de 10KΩ, un pulsado y un LED. Se adjunta la siguiente imagen que muestra el esquema de montaje para un pulsador:



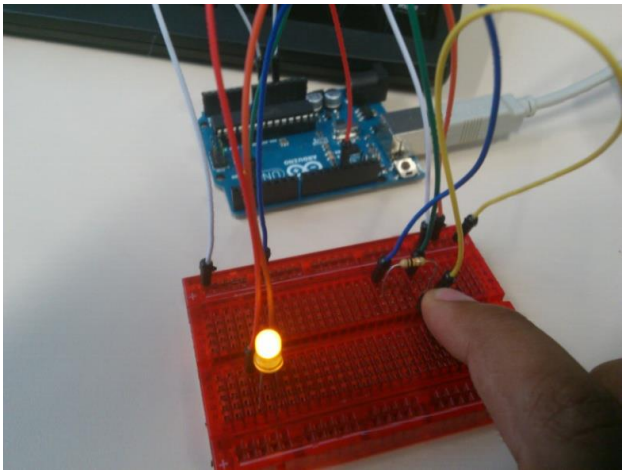
El pulsador necesita hacer uso de la corriente de 5V, por lo que debemos alimentarla con ella por una de sus patillas. En otra conectaremos la resistencia de 10KΩ junto con el cable que lo conecta el PIN 2. No olvidemos conectar la segunda patilla de la resistencia con la toma a tierra (GND).

Por último podemos conectar el LED de manera directa a tierra y al PIN 13 o hacer uso de la placa prototipo.

El resultado obtenido es algo similar a lo siguiente:



Al pulsar el botón se encenderá la luz.



Conclusión

Como hemos podido ver, gracias a estos dos ejemplos, el lenguaje Processing, del que hace uso Arduino, es capaz de gestionar entradas y salidas de una manera muy sencilla. Con estos conocimientos básicos ya podemos empezar a crear nuestros propios programas para Arduino e ir ampliando nuestro repertorio con nuevos dispositivos de entrada y salida.

Scratch4Arduino (S4A)

Ahora que tenemos unas nociones básicas del funcionamiento de Arduino vamos a estudiar un software desarrollado por un grupo formado por Marina Conde, Víctor Casado, Joan Güell, José García y Jordi Delgado con ayuda del "Citilab Smalltalk Programming Group".

Este software nos permite la interacción en ambos sentidos entre Arduino y Scratch con un ratio de actualización de unos 75 ms.

Cabe destacar que este software tiene programado por defecto el uso de los PIN. Dispone de 6 entradas analógicas (las que trae cada placa por defecto), 2 entradas digitales (correspondientes a los PIN 2 y 3), 3 entradas analógicas (PIN digital 5,6 y 9), 3 salidas digitales (PIN 10,11 y 13) y 4 salidas especiales para conectar servomotores (PIN digital 4, 7, 8 y 12).

Este software es libre y podemos obtenerlo en la web del grupo Citilab http://seaside.citilab.eu/scratch/arduino

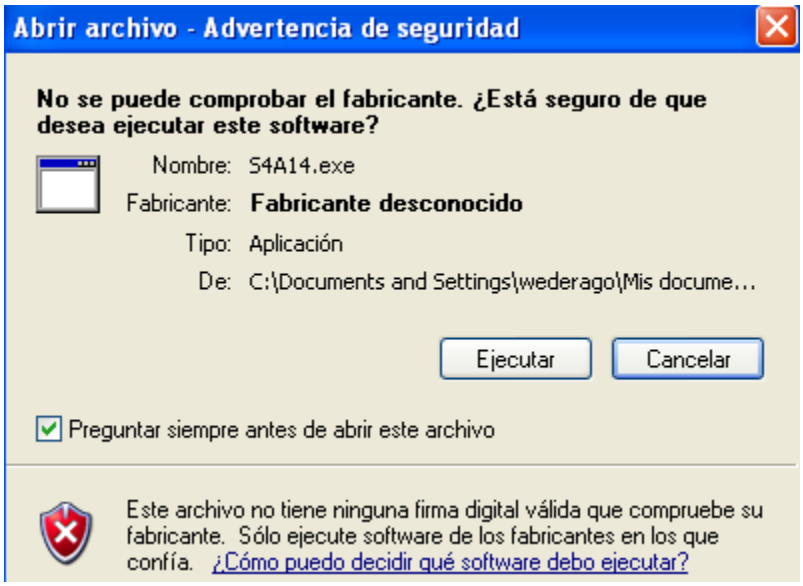
Utilizando Scratch4Arduino

Para hacer uso de este software necesitamos tener instalado el entorno de Arduino. Este paso ya ha sido explicado en el apartado anterior por lo que procederemos a la instalación de S4A.

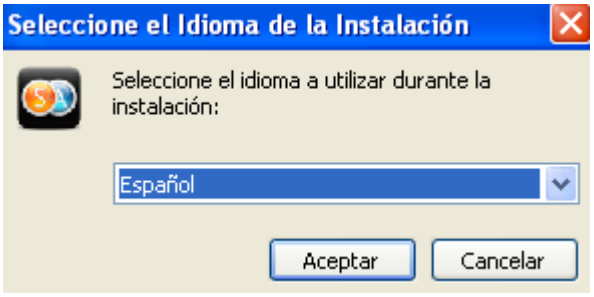
En la web del proyecto, en el apartado de descargas (downloads) encontraremos instaladores para Windows, Mac y Linux.

En nuestro caso procedemos a descargarnos el de Windows y lo ejecutamos.

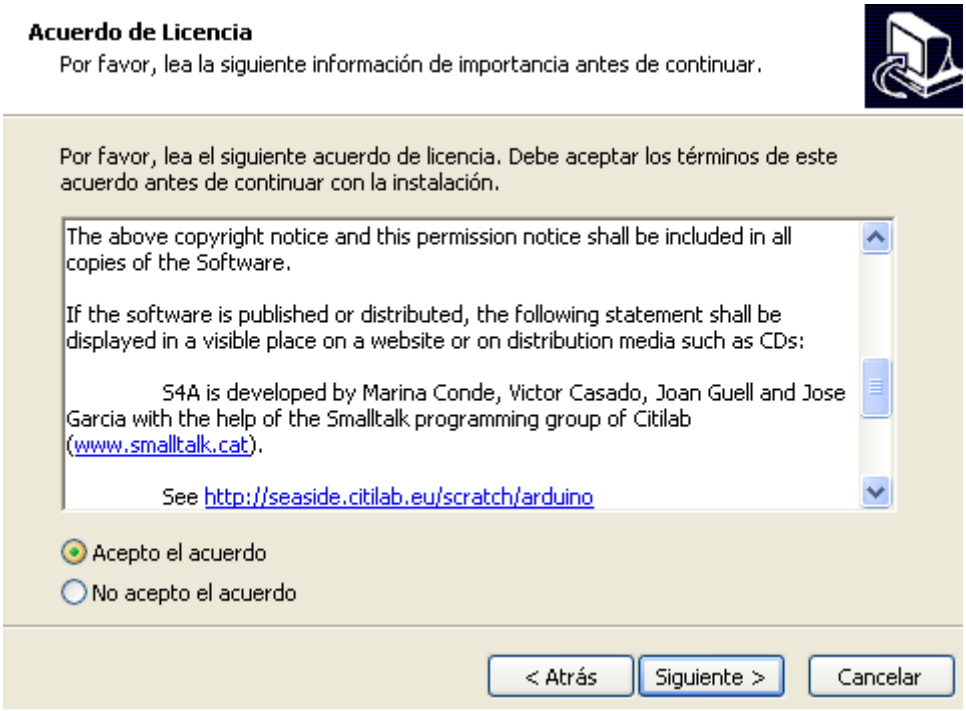
En caso de aparecer lo siguiente le daremos a ejecutar



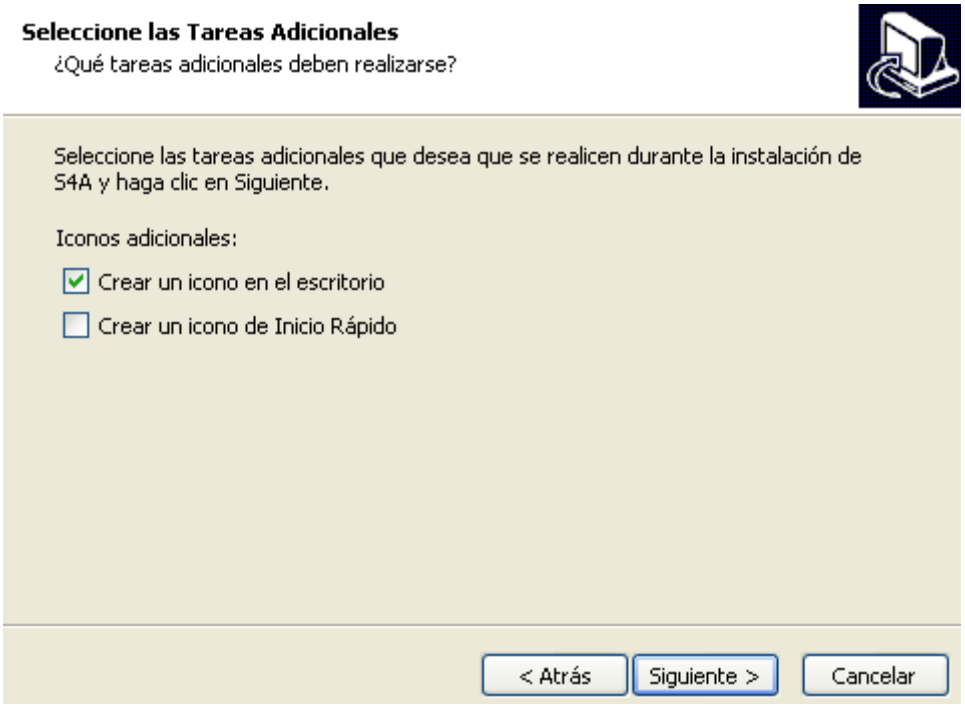
Seleccionamos idioma y procedemos a su instalación



En las siguientes pantallas elegimos siguiente y aceptamos los términos de uso.



Seguimos presionando siguiente y elegimos si queremos crear algún tipo de acceso directo.



Ya tenemos instalado el software S4A. Ahora necesitamos que Arduino sea capaz de mandar datos a Scratch al mismo tiempo que los recibe.

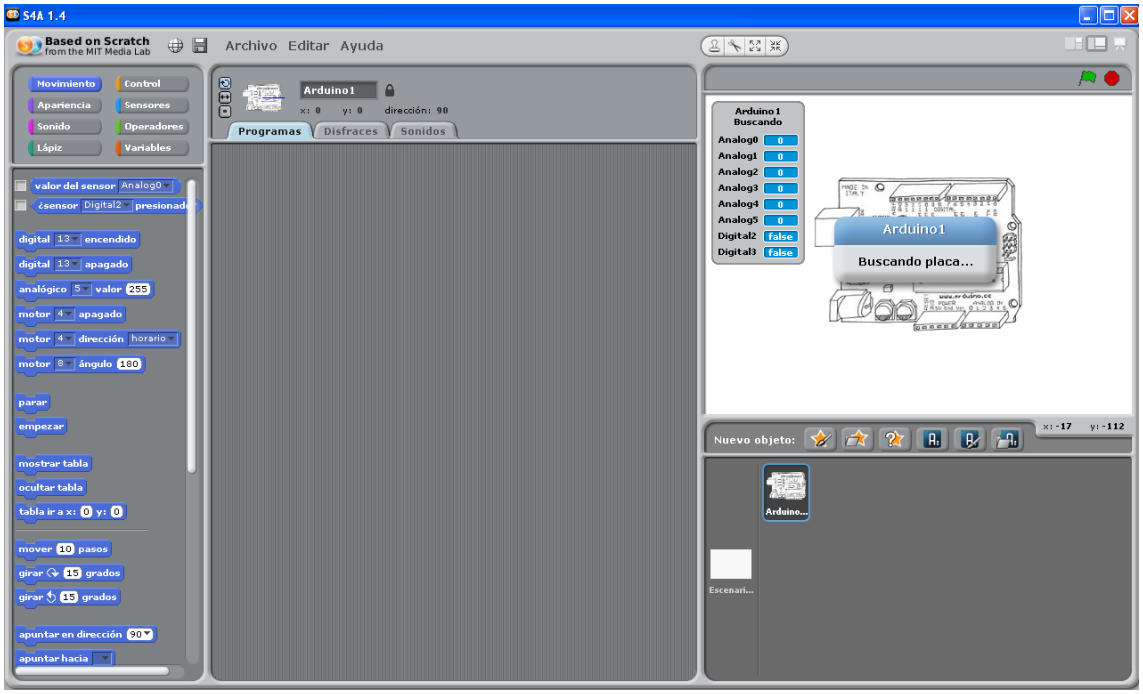
Para ello de la misma web de Citilab nos descargamos el programa de Arduino que se encarga de ello asignando a los PIN la funcionalidad anteriormente descrita. Descargaremos el firmware del paso 2 que se indica en la web.
http://seaside.citilab.eu/scratch?_s=_ZXxID7E_t-r92kJ&_k=04llCkg5bVRVUPf6

Una vez descargado tan solo tenemos que cargar el archivo en la placa y ya podremos establecer la comunicación entre el hardware y el software.

Como ya tenemos subido el Sketch a la placa podemos cerrar el entorno de Arduino y comenzar a trabajar con S4A.

Al lanzar S4A existe la posibilidad de que nos solicite un archivo *.image*. Este fichero lo encontraremos en la misma ruta en la que instalamos S4A con el nombre S4A.image. Lo seleccionamos y ya estará listo S4A.

El interfaz de S4A es muy similar al de Scratch, pero añadiendo bloques que permiten enviar y recibir datos desde Arduino.



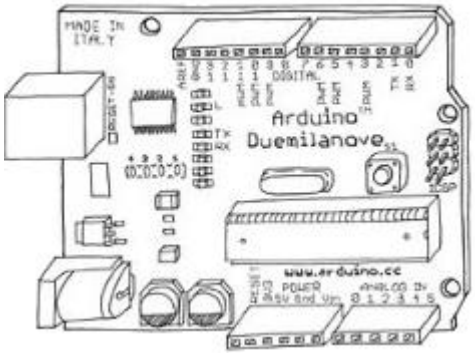
Durante los primeros segundos de ejecución de S4A se buscará una placa compatible para trabajar. Una vez localizada desaparecerá el mensaje y ya podremos trabajar.

Si echamos un vistazo a las categorías de bloques encontraremos todos los nuevos bloques en Movimiento. Existen varios tipos de bloques que nos permiten el control sobre todos y cada uno de los PIN descritos anteriormente. Son los siguientes:

Los bloques, “valor del sensor X” y “¿sensor X presionado?”, son similares a los que encontramos en Sensores y nos permiten comprobar el estado de los PIN digitales y analógicos. Además de bloques exclusivos para el uso de PIN digitales y analógicos. Por último encontraremos los encargados de los motores.

Los bloques de “ocultar tabla”, “mostrar tabla” y “tabla ir a: x, y” hacen referencia al a tabla que posee cada uno de los “Objetos Arduino” que tengamos en nuestro programa de Scratch.

Un “Objeto Arduino” es un sprite que tiene acceso pleno a la comunicación entre Scratch y Arduino, es decir, posee todos los bloques que permiten la conexión. Siempre que empezamos un proyecto nuevo tendremos un objeto Arduino por defecto con el siguiente disfraz.



En caso de necesitar más sprites que tengan acceso a estos datos, podemos hacer uso de los nuevos iconos que han aparecido a la derecha de los botones que usábamos para crear nuevos sprites en Scratch.

Los nuevos botones que tenemos nos permiten, de izquierda a derecha:

1. Crear un nuevo objeto Arduino: Crear un nuevo objeto con el mismo disfraz que nuestro primer objeto Arduino.

2. Dibujar objeto Arduino: Nos da acceso a las herramientas de dibujo de Scratch para que creemos nuestro objeto Arduino al igual que haríamos con cualquier otro.
3. Escoger desde fichero: Con esta opción tomamos uno de los sprites predefinidos o importados que tengamos y le da acceso a los datos y bloques exclusivos de Arduino.

Ya estamos familiarizados con las nuevas funciones de las que disponemos ahora, vamos a crear nuestra primera aplicación.

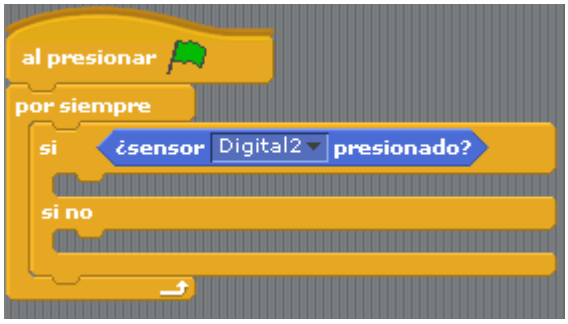
Primer contacto con Scratch4Arduino

En la introducción a Arduino vimos como encender un LED haciendo uso de un botón. Ahora vamos a hacer lo mismo a través de S4A. Para ello usaremos el mismo esquema de montaje que usamos en Arduino.

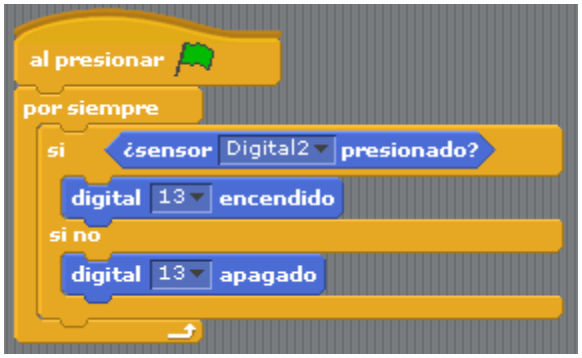
Una vez tenemos el circuito montado abrimos S4A y comenzamos creando un bloque que compruebe en todo momento un valor, esto lo hacemos con un bloque "por siempre" y un "si". Para este ejemplo usaremos el "si" que nos permite ejecutar dos bloques en función de si se da la condición o no. Y no nos olvidemos del bloque "al presionar empezar"

Con esta estructura estaremos comprobando en todo momento "algo" y en función de si se cumple o no ejecutaremos una acción u otra.

Ahora tenemos que ver que es ese "algo" que queremos comprobar. Al tratarse de un botón sabemos que es un digital input así que si buscamos en los bloques propios de un objeto Arduino encontraremos "¿sensor digitalX pulsado?". Este es el bloque que necesitamos, ya que hemos conectado el pulsador al PIN 2 elegimos ese.



Ahora nos falta saber qué queremos hacer cuando esté pulsado. Como queremos hacer lo mismo que en el ejemplo de introducción a Arduino, al pulsarlo se encenderá el LED 13 y si no lo apagará.



Y con esto ya tenemos uno de nuestros primeros programas de Arduino convertido a S4A.

Conclusión

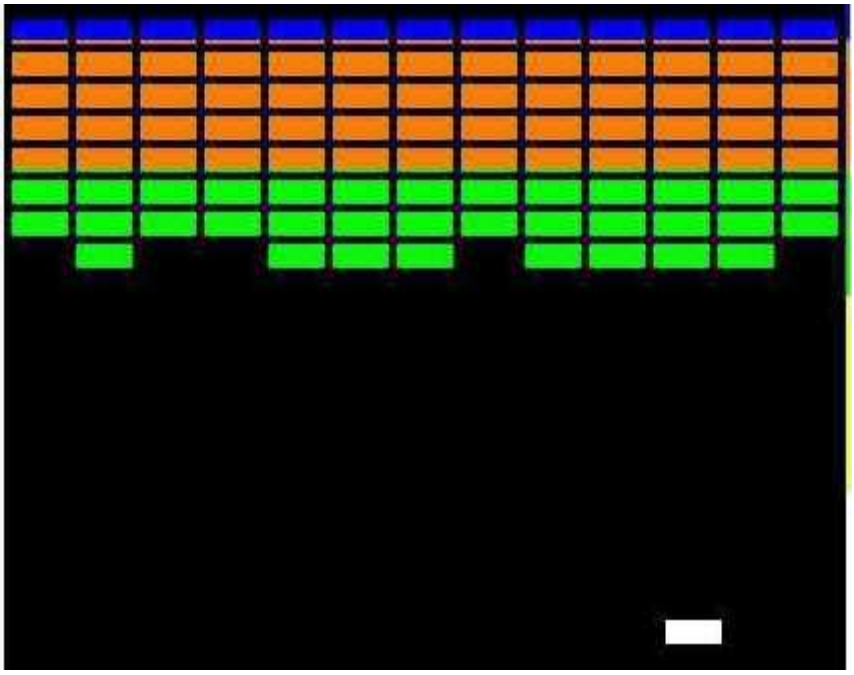
Con el ejemplo anterior hemos visto como, de manera sencilla, podemos hacer en S4A lo que podríamos hacer usando Processing sin muchos conocimientos del entorno de Arduino. Haciendo uso de la programación intuitiva y gráfica de Scratch tenemos acceso a todo el mundo de posibilidades que nos brinda Arduino, eso sí, con la limitación en el número de entradas y salidas de las que dispone la placa y están pre-configuradas en S4A.

Proyecto Educativo: Kinect y Arduino

La idea

Con los conocimientos que hemos adquirido hasta ahora, podemos embarcarnos en el desarrollo de una aplicación propia. Para ello he elegido como temática el juego Breakout desarrollado por Atari lanzado en 1976.

El juego, basado en el Pong de Atari, consiste en una pelota que se mueve por la pantalla rebotando y una "raqueta" en la parte inferior que debe evitar que salga la pelota por la parte inferior de la pantalla. Además tenemos una serie de bloques repartidos por la pantalla que debemos destruir utilizando la pelota. La pantalla acaba cuando hemos acabado con todos los bloques.



¿Qué puede aportar Kinect/Arduino?

Ahora que ya sabemos que vamos a hacer pensemos en como añadir funcionalidades a través del hardware y el software que hemos visto hasta ahora. Dado que el jugador solo controla el paddle o raqueta inferior hagamos que este sea controlado usando ese software/hardware.

De este modo tenemos dos maneras de trabajar, manejar el paddle haciendo uso de Kinect (tendremos que ver que queremos que mueva el paddle) y con Arduino.

¿Qué vamos a usar?

Ya sabemos dónde vamos a usar la interacción externa podemos centrarnos en la funcionalidad del juego.

Antes de nada necesitamos, el paddle, los bloques, una bola y algún fondo.

Para la bola usaremos el editor de sprites y crearemos una bola roja.

Es el momento de dibujar un paddle. (en mi caso busqué uno en internet)

Para los bloques dibujamos algo sencillo. (En mi caso he dibujado un bloque con algo de textura en CS4).

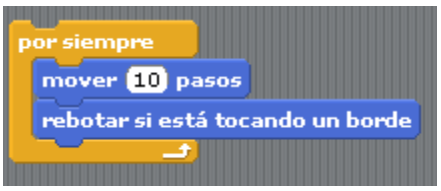
Una vez importados todos los sprites vamos a pensar en las funcionalidades básicas de cada uno de ellos.

Los Scripts

La bola

Comenzaremos con el elemento más importante del juego, la bola. Como hemos descrito antes su funcionalidad es la de ir rebotando por todo lo que toca. Ahora tenemos que pensar como simular este funcionamiento.

En una de las primeras secciones de este documento hablamos de los botones que permiten elegir las funciones de giro. Para esta bola nos interesa hacer uso de la primera de todas, la que le permite rotar los 360º de este modo podemos simular su movimiento dándole direcciones o rotándola en grados y haciendo que gire.



Añadiendo este bloque al script de la bola le dejamos girar y rebotar de una manera que simula bastante bien el funcionamiento que buscamos. Si probamos este bloque vemos que rebota en los cuatro lados del escenario y, para nuestro juego, necesitamos que si llega a la zona inferior el juego acabe.

La manera más sencilla de conseguir esto es creando en el escenario una línea en la parte inferior de un color y usar el bloque del apartado de sensores que nos permite comprobar si existe interacción entre el sprite y un color determinado.

Para ello seleccionamos el fondo para nuestro primer nivel y le añadimos la línea de color.

Ahora vamos a preparar el bloque de interacción con la línea de color para la bola.

Como podemos ver en el script, hemos añadido al bloque de movimiento de la bola el código necesario para detectar el fin de juego. La variable “juego_On” indica el estado de juego y tiene 4 posibles valores.

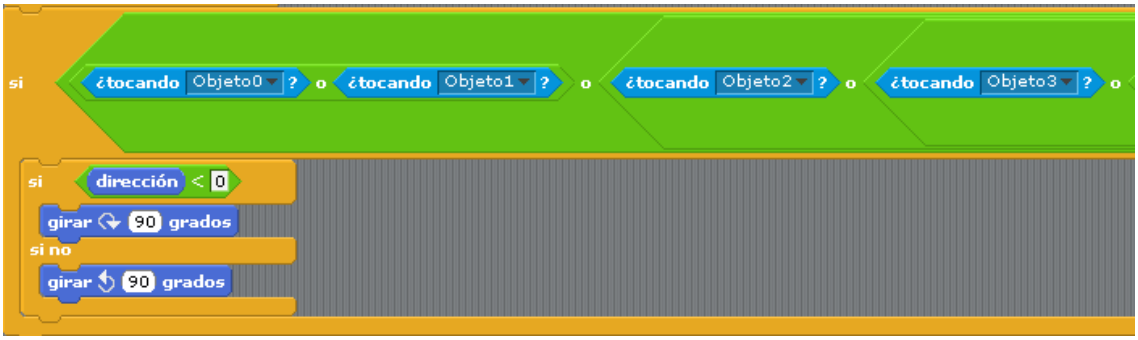
0.-el juego está preparado para comenzar a la espera de la señal.
1.-el juego está en curso.
2.-el juego ha terminado
3.-el juego está en la pantalla de instrucciones

Después de cambiar el estado de “juego_On” mandamos una señal a todos con “fin_juego”. Esta señal es la encargada de comunicar a todo nuestro programa que debe pasar a modo fin_juego. Como queremos que el juego acabe, le decimos al sprite que se haga invisible y por último que detenga su ejecución. El bloque “detener programa” finaliza la ejecución de todo el código de un sprite.

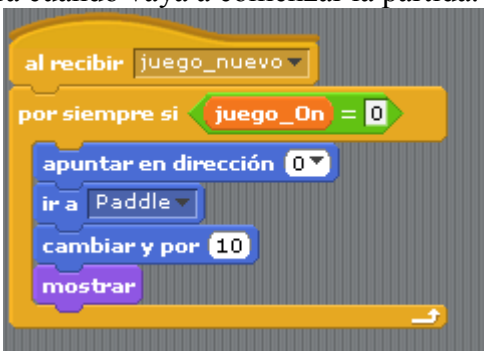
Para acabar con el script de la bola tan solo nos falta comprobar impactos con los bloques y/o paddle. Para ello añadiremos dos bloques de código al principal.

Con el primero vamos a comprobar el impacto con el paddle. En función de con qué dirección venga le daremos una nueva orientación. Diferenciamos dos casos para darle un poco más de realismo.

Para acabar con el movimiento del sprite añadimos el bloque encargado de comprobar los impactos con los bloques. Para ello haremos uso de bloques lógicos “o” encadenados para todos y cada uno de los bloques.



Ya tenemos definido todo el movimiento de la bola así como sus colisiones. Ahora vamos a definir una señal llamada "juego_nuevo" que se encargue de poner la pelota sobre el paddle para cuando vaya a comenzar la partida.

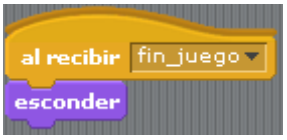


Para finalizar vamos a definir qué se debe hacer al recibir las señales de victoria y fin de juego

Los bloques

Ya tenemos preparada la bola, ahora veamos como programar los bloques que debemos destruir. Dado que se comportan de la misma manera merece la pena programar solo uno y luego clonarlo añadiendo las variaciones necesarias. Ya que los bloques son estáticos tan solo debemos limitarnos a indicarles que deben hacer al detectar un impacto.

Empecemos definiendo su comportamiento ante las señales victoria y fin de juego. Deben limitarse a volverse invisibles.



Del mismo modo al comienzo de la aplicación queremos que no sean visibles durante la pantalla de presentación.

Puesto que hemos decidido trabajar usando señales en los bloques vamos a definir las señales según el nombre de los niveles. El código es similar para todos los niveles y bloques que queramos poner. En él le indicaremos la posición en la que debe estar y, que al ser impactado, debe sumar uno a la puntuación y desaparecer.

El bloque de cambiar efecto color, lo ponemos para dar a cada bloque un tono de color aleatorio.

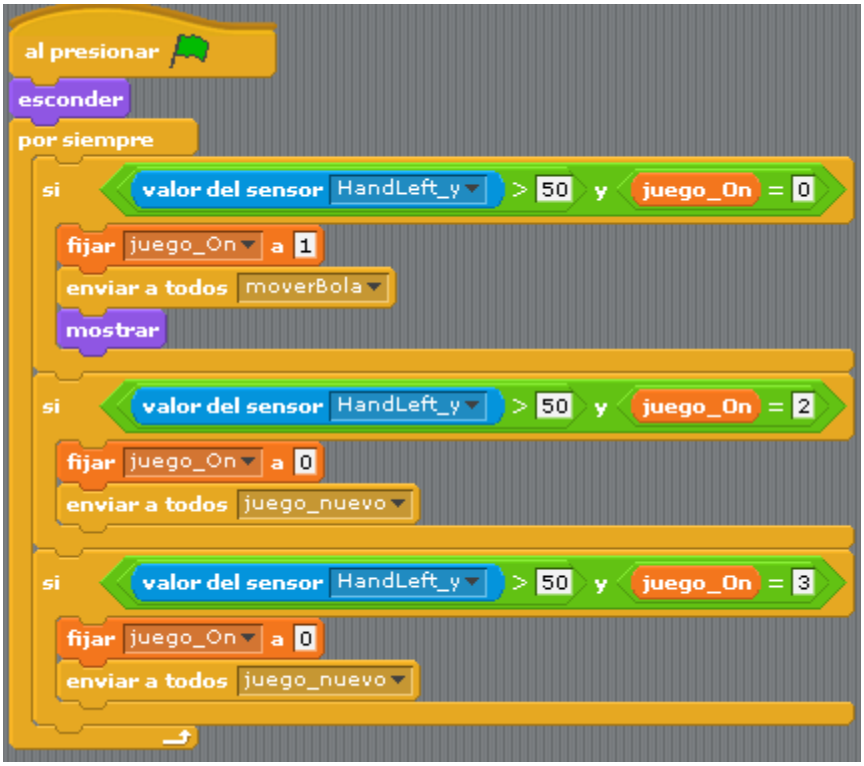
Con esta programación podemos crear todos los niveles de dificultad que consideremos.

El paddle

El paddle es el lugar donde vamos a proporcionar interactividad al juego haciendo uso de S4A y/o K2S. Comenzaremos con el código que corresponde a K2S.

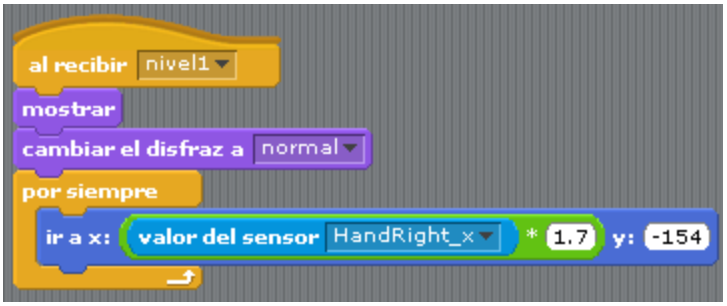
Para este juego he decidido que el paddle se controle con la mano derecha, mientras que para mandar señales de inicio de partida usaré la mano izquierda.

Para la mano izquierda comprobaremos su altura y en función de si está levantada o no comenzará el juego.



Como podemos observar, se limita a comprobar el estado de juego y mandar la señal correspondiente. La señal MoverBola es la encargado de indicar a la bola que debe comenzar a moverse y juego_nuevo indica que hay que preparar todo para un comenzar de nuevo. Estas señales serán lanzadas si la posición de la mano izquierda en el eje y supera la coordenada 50.

El código de la mano derecha corresponde al movimiento y lo haremos de la siguiente manera.



El movimiento del paddle se activa desde que comienza el nivel 1 hasta que termina el juego. Como podemos ver el paddle se sitúa y = -154 y el valor en el eje x varía en función del de la mano derecha del jugador. El coeficiente 1.7 lo usamos para hacer más manejable las distancias del juego ya que son relativas respecto a la cámara.

Respecto a las señales de victoria y fin de juego se limitan a lo mismo que en el resto de casos, a esconder el paddle.

El escenario

Finalmente tenemos que definir el script del escenario que es el encargado de mandar la mayoría de las señales que utiliza el juego.

El código propio del escenario consiste en resetear todas las variables que puedan estar cambiadas antes del inicio de un nuevo juego así de cómo mandar las señales de cambio de nivel y con ello cambiar su disfraz.

Personalizando nuestro proyecto

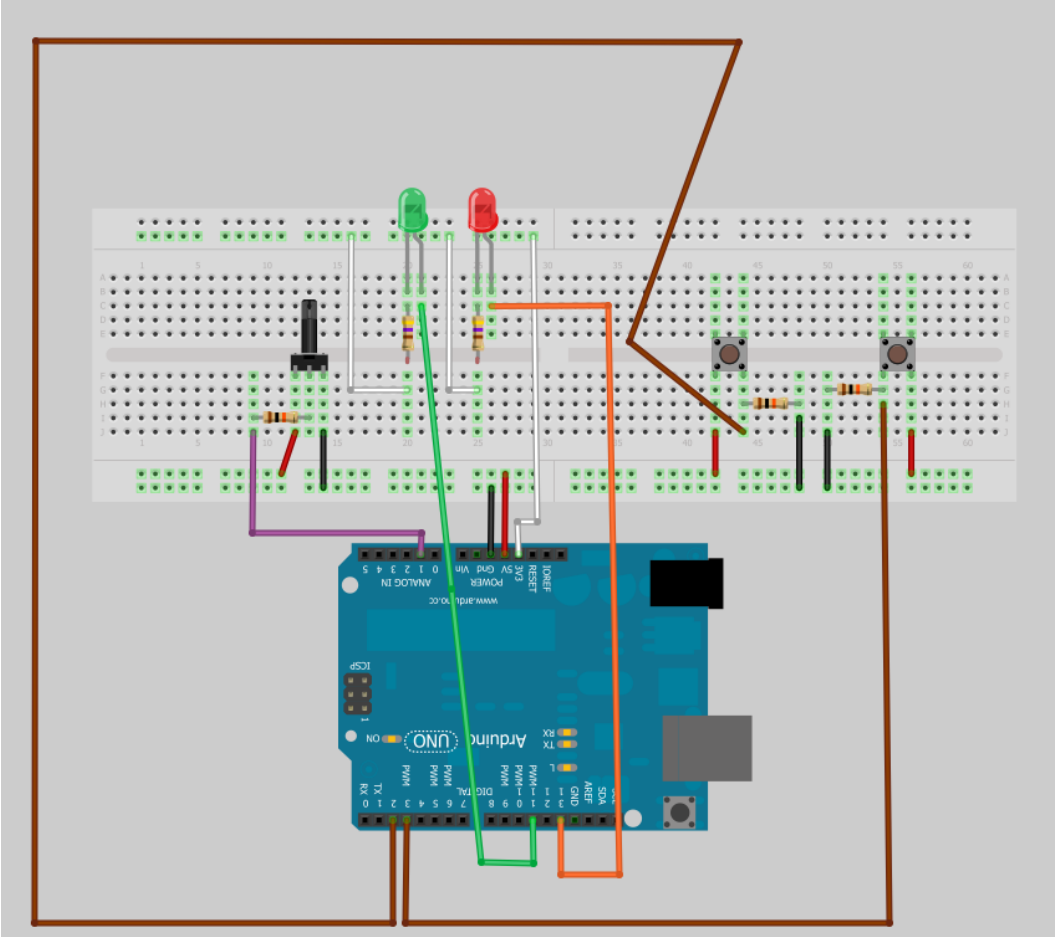
Ya tenemos el funcionamiento básico del juego en marcha, ahora falta darle nuestro "toque" personal para hacerlo verdaderamente nuestro. El juego que se adjunta contiene 2 disfraces extra para el paddle, que le permiten disparar y hacerse más largo para llegar más fácilmente a la bola.

Se adjunta el programa *ArkanoidK2S.*

Alternativa S4A

En la explicación anterior hemos optado por usar Kinect como control. Ahora vamos a ver como tendríamos que haberlo planteado de haber elegido Arduino.

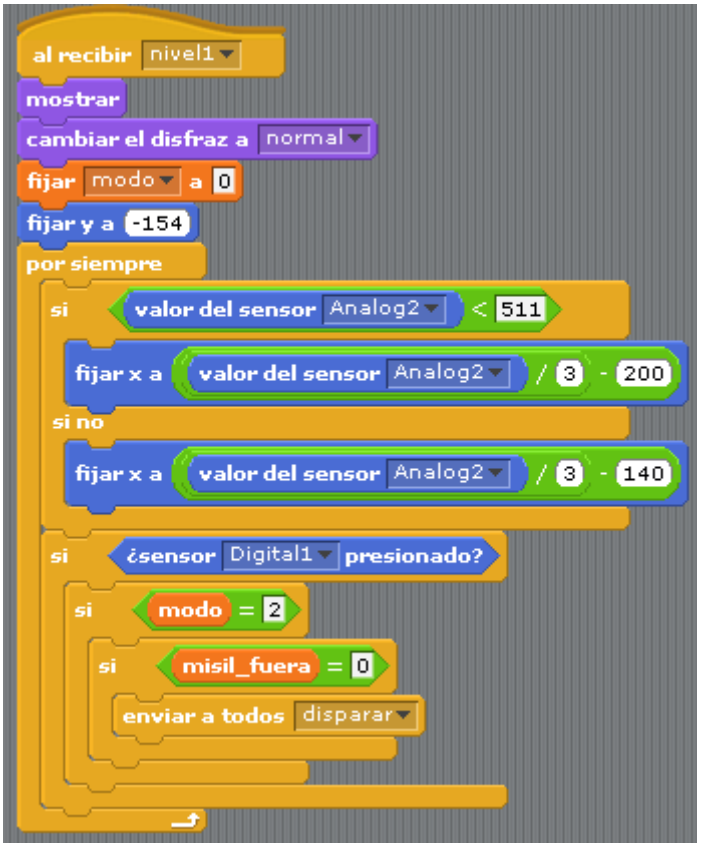
Para empezar comenzaremos con la creación del esquema del circuito que utilizaremos.



En este circuito hemos puesto dos pulsadores, dos LED y un potenciómetro. Las funciones de los pulsadores van a ser, de izquierda a derecha, disparar y comenzar una nueva partida. El potenciómetro nos permitirá desplazar el paddle en la dirección que lo rotemos y los LED indicarán el estado de la partida, rojo para derrota y verde para victoria.

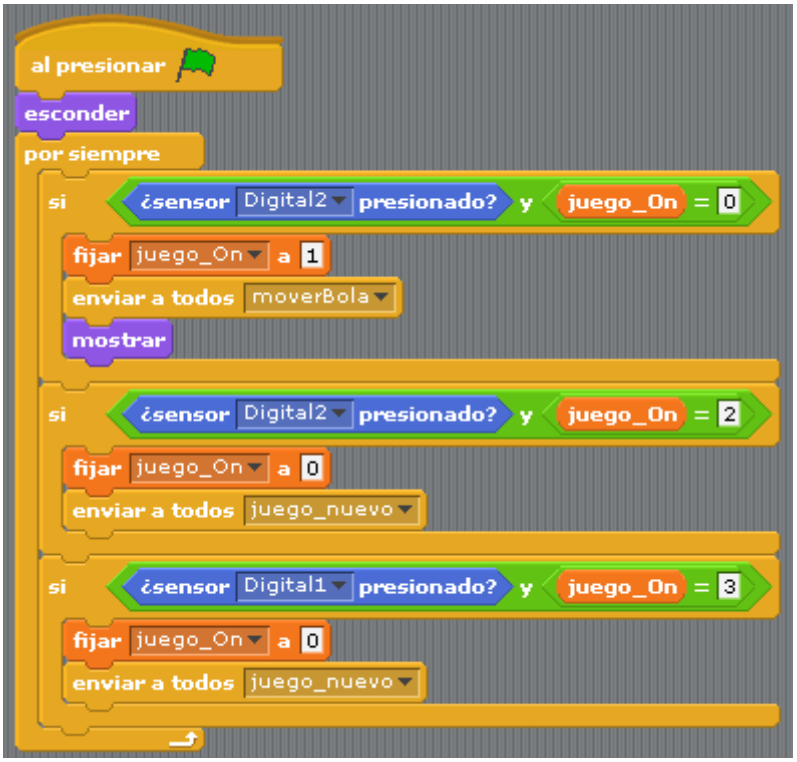
Ahora que tenemos montado nuestro controlador de juego vamos a darle las funcionalidades que hemos mencionado anteriormente.

Comenzaremos con el funcionamiento del paddle, para ello modificaremos el script de movimiento que teníamos para Kinect y lo cambiaremos por el siguiente. Recordemos que para poder hacer esto debemos estar trabajando en S4A.



En el código anterior hemos sustituido el movimiento por el valor del sensor analógico. Sabemos que el valor va desde 0 hasta 1023 por lo que tomamos 511 como punto central. Con el punto central definido, nos centramos en dar valores al paddle en el eje x. Esto lo conseguimos gracias a las expresiones matemáticas que tenemos dentro de cada uno de los bloques "fijar x". Por último, el proyecto adjunto tiene la funcionalidad de disparar así que comprobamos en qué modo está el paddle y si está en modo disparo le permitimos disparar.

Ahora solo nos queda activar el juego usando los botones en vez de con la posición de la mano izquierda. Para ello sustituiremos el bloque anterior por este.



Gracias a estos dos cambios ya tenemos las funcionalidades en Arduino en vez de Kinect. Cabe destacar que S4A permite la ejecución junto con K2S de modo que podemos usar los dos dispositivos a la vez.

Se adjunta el programa *ArkanoidS4A.*

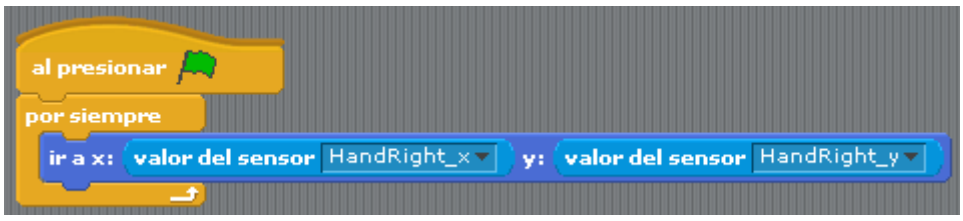
Proyecto Educativo Musical: Kinect

La idea

En este segundo proyecto se pretende trabajar, desde el punto de vista educativo, la discriminación auditiva junto con la motricidad y la lateralidad. Este será el interfaz de la aplicación:

¿Qué vamos a usar?

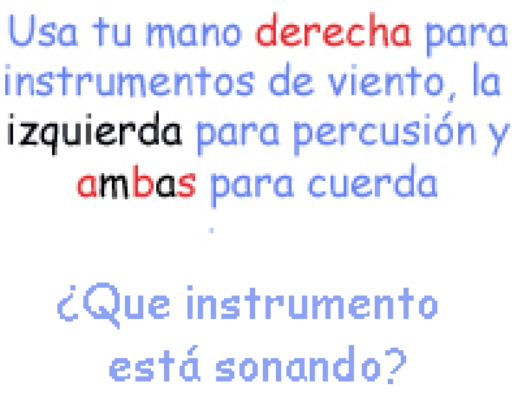
Ya que vamos a trabajar capacidades motrices haremos uso de K2S. Para empezar, vamos a definir cómo vamos a representar nuestras manos en el juego. He optado por el uso de dos puntos creados con el propio editor de Scratch, uno negro y uno rojo. El punto negro representará la mano izquierda y el rojo la derecha. Esto lo conseguiremos con el siguiente código.



Gracias a este código tenemos a tiempo real las coordenadas de nuestra mano derecha y, por lo tanto, sensación de interactuación completa. El código de la mano izquierda es idéntico, pero cambiando el nombre del sensor al de la mano izquierda.

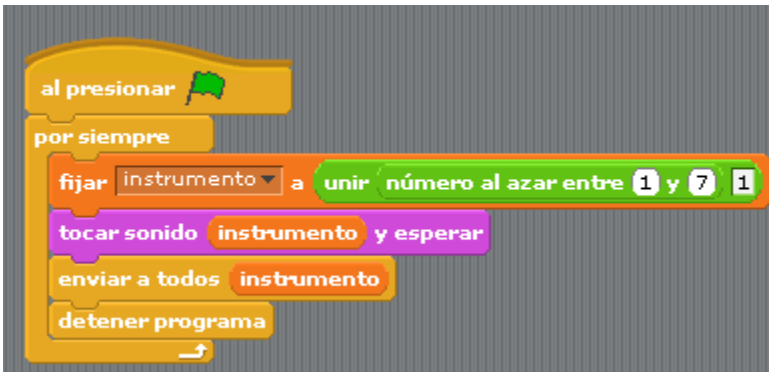
El escenario

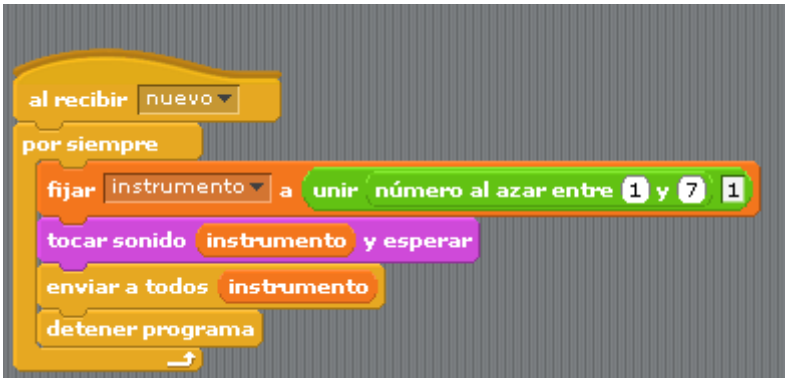
Para esta aplicación he optado por un diseño sencillo. Un fondo blanco sobre el que escribiremos las instrucciones del juego.



Como podemos observar las instrucciones que hacen referencia a cada una de las manos tienen el color correspondiente a la mano que representan para así dar una ayuda gráfica al usuario.

El escenario será el encargado de hacer funcionar toda nuestra aplicación. Comenzaremos escribiendo los siguientes script.





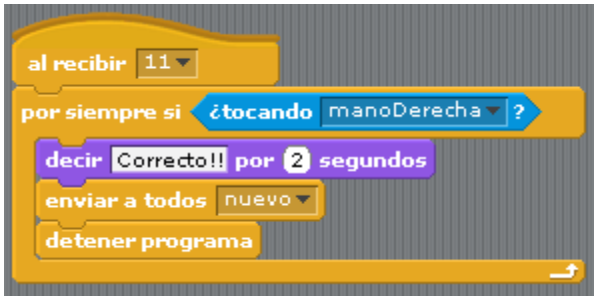
Como podemos ver ambos bloques son idénticos. El primero es el encargado de reproducir un instrumento al empezar la aplicación y el segundo hace lo mismo, pero solo cuando recibe la señal de instrumento nuevo que será enviada por los instrumentos cuando se responda correctamente al instrumento actual.

Para la elección de un instrumento primero tendremos que haber importado los sonidos al escenario. Esto podemos hacerlo bien arrastrando los mp3 al sprite que representa el escenario o bien con el comando importar que encontraremos en el apartado sonidos del mismo. En mi caso los he nombrado con un número del 1 al 7 y lo concateno con otro 1 al final. He procedido de esta manera porque Scratch no me permite nombrarlos con tan solo un número.

Haciendo uso de una variable que guardará el valor del instrumento que sonará aleatoriamente reproducimos el sonido y mandamos a todos los sprites el número generado aleatoriamente.

Los instrumentos

La última instrucción del escenario consistía en enviar a los instrumentos el valor de la variable, de modo que cada instrumento pudiese comprobar si es el que debe estar a la espera de la interacción con las manos. Un código similar al que describiré a continuación será el que encontraremos en cada uno de los instrumentos.



El primer bloque detecta si es este instrumento el que debe comenzar a comprobar si es interactuado. Con el siguiente bloque decimos si la respuesta era

correcta y mandamos la señal encargada de que se dé una nueva reproducción de sonido.

Este código solo varía en la señal de recepción entre cada uno de los instrumentos y la comprobación de que mano/s está interactuando.

Se adjunta el fichero *instrumentos*.

Pruebas con usuarios

Ahora que ya sabemos crear aplicaciones debemos centrarnos en cómo de útil puede resultar el uso y aplicación de las mismas. Para ello vamos a pedir a un grupo de personas que interactúen con nuestras aplicaciones. Una vez hayan probado las aplicaciones les pediremos que rellenen unos test especializados en la manejabilidad y usabilidad de software para hacernos una idea de cómo de útiles pueden resultar estas aplicaciones. Vamos a plantear los test sobre las aplicaciones desarrolladas en el apartado anterior con Kinect.

Estos test se han llevado a cabo sobre un conjunto de 6 personas, 4 de ellos con nociones de informática avanzada de edades comprendidas entre 22 y 25 años, un sujeto de 8 años y un adulto con conocimientos básicos de informática.

Test de esfuerzo

El test utilizado para obtener los resultados que describiremos a continuación podemos encontrarlo en el Anexo A.

Para la realización de este test se preguntó a los sujetos como ordenarían según importancia los seis factores del estudio y se obtuvo la siguiente clasificación.

Effort → Frustration → Performance → Physical Demand → Mental Demand → Temporal Demand

Los razonamientos, en general, fueron los siguientes: El esfuerzo para hacer acabar con la tarea es considerado lo más importante, seguido por la frustración. La frustración puede provocar el rechazo a esta tecnología por lo que es algo a tener muy en cuenta. En tercer lugar tenemos performance, es decir, la sensación que provoca al usuario el acabar (o no) la tarea asignada. Por último tenemos en orden esfuerzo físico, mental y temporal. Estos tres apartados son los considerados de menor importancia ya que gracias al uso de estas tecnologías estos esfuerzos se minimizan.

Ahora que ya tenemos la escala de pesos procedemos a realizar un test virtual. Este test es idéntico al que se nos presenta en formato físico en el Anexo A.

A continuación un ejemplo de la versión digital realizada por uno de los encuestados.

	Rating	Tally	Weight
Mental Demand	25	0	0
Physical Demand	35	1	0.06666666666666667
Temporal Demand	40	2	0.13333333333333333
Performance	50	5	0.3333333333333333
Effort	35	4	0.26666666666666666
Frustration	25	3	0.2

Overall = 38.666666666666664

Tras la elaboración de este test a todos los individuos, se han obtenido unas medias de 30.5445 puntos sobre 100 para la aplicación musical y de 32.3778 para el videojuego. Con estos datos podemos determinar que las aplicaciones que hemos desarrollado no suponen un gran esfuerzo al usuario. Esto las convierte una herramienta de trabajo válida y de posible interés para proyectos educativos.

Test de usabilidad

La usabilidad hace referencia a la facilidad con la que un sujeto puede adaptarse al uso de una tecnología.

Algunas de las definiciones que encontramos para este término que no pertenece a la RAE si no que ha sido tomado del inglés *usability* son:

- Jakob Nielsen definió la usabilidad como *"el atributo de calidad que mide lo fáciles que son de usar las interfaces Web."*
- También tenemos una definición formal definida por la ISO. *"La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso."*

Para la realización de este test hemos hecho uso del test de Brook que encontraremos en el Anexo B, en este mismo anexo encontraremos el método de evaluación que utiliza.

En este test nos es posible desglosar los resultados en cada uno de los campos a evaluar.

- *I think I would like to use this system frequently:* una media de 3.16 puntos en este apartado nos demuestra que los usuarios están dispuestos a usarlo con frecuencia.
- *I found the system unnecessarily complex:* en este apartado se obtuvo una media de 3.5 puntos, esto nos indica que el sistema es de uso sencillo e intuitivo
- *I though the system was easy to use:* un 3.5 nos demuestra que, desde el punto de vista de los usuarios que han realizado el test, el programa es de uso muy sencillo.

- *I think that I would need the support of a technical person to be able to use this system:* un 3.834 de media indica que para hacer uso de esta aplicación no se prevé la necesidad de asistencia técnica.
- *I found the various functions in this system were well integrated:* aquí se obtiene una media de 3.667. Los usuarios consideran que las funciones que se proponen están bien integradas.
- *I thought there was too much inconsistency in this system:* un 3.5 de media da a entender que no se encuentra apenas inconsistencia en el sistema.
- *I would imagine that most people would learn to use this system very quickly:* el 3.667 que se obtiene como media demuestra que los usuarios encuentran el funcionamiento muy sencillo.
- *I found the system very cumbersome to use:* un 3.3 indica que los ususarios no encontraban el sistema muy incómodo
- *I felt very confident using the system:* 3.667 es la media que se obtiene en este campo. Esta nos indica que el usuario se siente cómo haciendo uso del sistema.
- *I needed to learn a lot of things before I could get going with this system:* el 3.834 de media en este apartado demuestran que los ususarios piensan que no se necesitan de muchos conocimientos técnicos para poner el sistema en marcha.

De la realización de este test obtenemos las puntuaciones de 87.5 puntos y 88.3 sobre 100 puntos para las aplicaciones de música y videojuego respectivamente. Este resultado nos indica que esta clase de herramientas interactivas son de muy fácil uso y apenas requieren una formación o explicación antes de comenzar a utilizarlas lo que las hace ideales para los ámbitos educativos.

Conclusiones

Gracias a los test de usabilidad y esfuerzo podemos decir con una gran certeza que esta clase de aplicaciones podrían suponer un gran aporte a la educación y tratamiento de problemas tales como la dislexia. Estas cualidades lo dotan como una manera sencilla, divertida y eficaz de trabajar con personas con problemas como el déficit de atención y dificultades de aprendizaje.

Competencias educativas

Según la normativa europea que desarrolla el Real Decreto 1631/2006 se han establecido ocho competencias básicas en la educación secundaria obligatoria (ESO), que son:

1. Competencia en comunicación lingüística.
2. Competencia matemática.
3. Competencia en el conocimiento y la interacción con el mundo físico.
4. Tratamiento de la información y competencia digital.
5. Competencia social y ciudadana.
6. Competencia cultural y artística.
7. Competencia para aprender a aprender.
8. Autonomía e iniciativa personal.

Mediante el uso de estas tecnologías podemos plantearnos dos puntos de vista. Por un lado tendríamos el de desarrollador y conocedor de estas herramientas y por el otro el del usuario.

Competencias como usuario

Estas vienen intrínsecas al tipo de aplicaciones que utilicemos. Dentro del abanico de posibilidades que se pueden manejar con un conocimiento básico podríamos hablar de trabajar las siguientes competencias:

- Competencia lingüística: una serie de aplicaciones que ayuden al entendimiento y práctica de las estructuras gramaticales formarían una buena base para esta competencia.
- Matemática: aplicaciones que enseñen los conceptos par e impar, que ayuden a relacionar el nombre con cada uno de los números en una operación o indicar por teclado cuál es el resultado de una operación que se plantea de manera aleatoria son solo algunos ejemplos de aplicaciones que ayudarían a desarrollar esta competencia.
- Competencia cultural y artística: juegos interactivos que permitan relacionar obras de arte con datos sobre ellas y aplicaciones de tipo lienzo serían perfectas para el desarrollo de esta competencia.
- Autonomía e iniciativa: aplicaciones que planteen retos o decisiones pueden facilitar la asimilación de esta competencia.
- Etc.

Con ejemplos como los anteriormente descritos podríamos trabajar todas las competencias básicas.

Competencias como desarrollador

El conocimiento de estos entornos nos permite trabajar principalmente el aprender a aprender. No solo nos permite ejercitar la imaginación sino que además, hacer uso de todos nuestros conocimientos para la creación de aplicaciones con lo que trabajamos todas y cada una de las competencias básicas.

Conclusiones y líneas futuras

Al llegar al final de este proyecto en el que he trabajado con el lenguaje Scratch, he estudiado sus funciones de comunicación y he interactuado con software y hardware low cost y/o libre, me doy cuenta de lo enriquecedora que ha sido esta experiencia.

No solo he profundizado en un entorno de programación que uso en mi puesto de trabajo, sino que además, me ha permitido ampliar mis conocimientos gracias a las tecnologías Kinect y Arduino.

El uso de Kinect, un dispositivo que hasta hoy solo veía como periférico de juego, e ideas como la de Stephen Howell, dan acceso a todo un mundo de posibilidades e interactividad hasta ahora solo alcanzable a través de tecnologías que, a veces, son prohibitivas. La liberación del SDK de Kinect ha abierto un campo de investigación muy interesante al que, en un futuro, me gustaría dedicar más tiempo.

Al trabajar con Arduino la sorpresa ha sido mayúscula. La cantidad de aplicaciones posibles para este hardware es abrumadora. Tecnologías como ésta, al alcance de todo el mundo, permiten que el único límite al que nos enfrentemos sea nuestra imaginación. Fabricar nuestros propios sensores, utilizarlos en instalaciones de domótica o crear nuestros propios dispositivos de juego son solo unas pocas ideas que se me ocurren en este momento.

Por otro lado, iniciativas como las del grupo Citilab, que permiten el acercamiento de la robótica a jóvenes haciendo uso de Scratch4Arduino, me parecen una buena forma de trabajo. Por ello, durante este proyecto, he intentado reflejar los beneficios de aprender usando estas tecnologías.

La realización de un estudio sobre un pequeño conjunto de individuos me ha permitido ver cómo se trabaja en “el mundo real”, el laboral, y obtener un invaluable “feedback”. El uso de estos test junto con la experiencia de ver en primera persona las reacciones de los usuarios al probar las aplicaciones (frustración, alegría, satisfacción,...) ha sido un gran incentivo para profundizar en el estudio de estas tecnologías y mejorar las aplicaciones.

Creo que, tras estas reflexiones, queda patente el interés que estas investigaciones han suscitado en mí. Me gustaría seguir ampliando y profundizando mis conocimientos en estos campos: la creación de un driver propio para Scratch (LeapMotion, WiiMote), soluciones a pequeños problemas de la vida real haciendo uso de Arduino o conseguir aplicaciones más interactivas e intuitivas con Kinect.

Espero que en un futuro se me brinde de nuevo la oportunidad de trabajar de nuevo con alguno/s de estos entornos.

Bibliografía

Bibliografía digital

Enlaces Scratch

http://scratch.mit.edu/
http://wiki.scratch.mit.edu/wiki/Main_Page

Enlaces BYOB

http://byob.berkeley.edu/

Enlaces Arduino

http://www.arduino.cc/
http://playground.arduino.cc/

Enlace Kinect2Scratch

http://scratch.saorog.com/

Enlace Scratch4Arduino

http://seaside.citilab.eu/scratch/arduino

Otras webs

http://softwareybarralibre.org/
http://highperformanceitrobotica.blogspot.com.es/2011/12/comenzar-trabajar-con-scratch-para.html

Bibliografía física

ARDUINO COOCKBOOK. Recipes to Begin, Expand, and Enhance your projects. Michael Margolis. Ed. O'Reilly.

Anexo A

Test de esfuerzo

Se trata de un test denominado Test de índice de carga y es una herramienta de evaluación subjetiva que califica la carga de una tarea. Su objetivo consiste en evaluar la carga que supone un sistema, aplicación o tarea a un sistema. Ha sido desarrollado por el grupo de rendimiento humano del centro de investigación Ames de la NASA.

La primera parte del test consiste en seis preguntas que permiten evaluar de 0 a 100 (con escalas de 5 puntos) los siguientes factores:

1. Demanda mental
2. Demanda física
3. Demanda temporal
4. Rendimiento
5. Esfuerzo
6. Frustración

Una vez que tenemos evaluados estos puntos se plantean una sucesión de 15 preguntas para determinar la importancia que se debe dar a cada uno de los factores según la percepción del individuo.

Podemos encontrar este test en su página oficial en formato físico http://humansystems.arc.nasa.gov/groups/TLX/paperpencil.html y en formato digital en http://www.keithv.com/software/nasatlx/

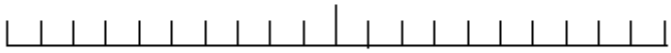
A continuación se presenta el test en formato físico.

NASA Task Load Index

Hart and Staveland's NASA Task Load Index (TLX) method assesses work load on five 7-point scales. Increments of high, medium and low estimates for each point result in 21 gradations on the scales.

Name	Task	Date

Mental Demand — How mentally demanding was the task?

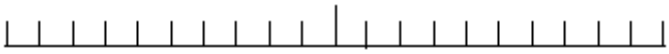


Very Low — Very High

Physical Demand — How physically demanding was the task?

Very Low — Very High

Temporal Demand — How hurried or rushed was the pace of the task?

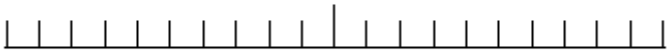


Very Low — Very High

Performance — How successful were you in accomplishing what you were asked to do?

Perfect — Failure

Effort — How hard did you have to work to accomplish your level of performance?



Very Low — Very High

Frustration — How insecure, discouraged, irritated, stressed, and annoyed wereyou?

Very Low — Very High

Anexo B

Test de usabilidad

En la página web http://www.usabilitynet.org perteneciente a una organización europea que pretende estandarizar el uso de esta clase de herramientas podemos encontrar el test de Brook entre otros.

A continuación el test de Brook usado para el estudio de la usabilidad durante este proyecto.

System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

La evaluación de este test se hace de la siguiente manera:

- Las preguntas 1, 3, 5, 7 y 9 se usa la puntuación de la casilla elegida menos uno.
- Para las preguntas 2, 4, 6, 8 y 10 se resta a 5 el valor de la casilla seleccionada.
- Una vez obtenido el valor se multiplica por 2.5 y se obtiene la nota final.

Proyecto Fin de Carrera

"Aprendizaje y desarrollo de aplicaciones multimedia e interactivas en entorno Scratch/BYOB con Arduino y Kinect"

Jorge Wederago Jiménez
Tutor: Alfredo Pina Calafí

Índice

- Razón del Proyecto
- Scratch
- Comunicación Externa
- Kinect2Scratch
- Scratch4Arduino
- Proyectos Educativos
- Pruebas con usuarios
- Competencias educativas
- Conclusiones y líneas futuras
- Demo

upna Nafarroako Unibertsitate Publikoa

Razón del Proyecto

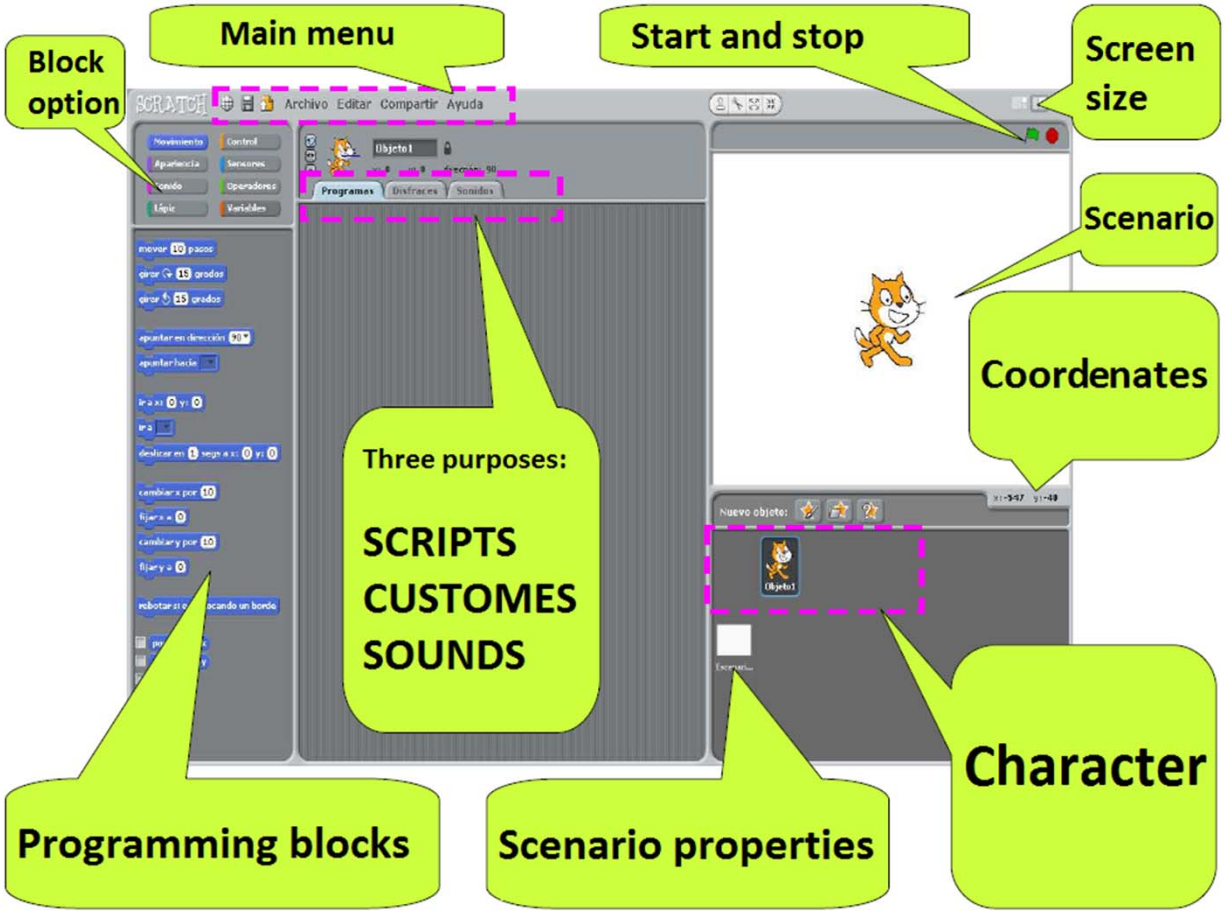
- ¿Por qué?
- ¿Objetivos?
- Materiales

upna Nafarroako Unibertsitate Publikoa

Scratch

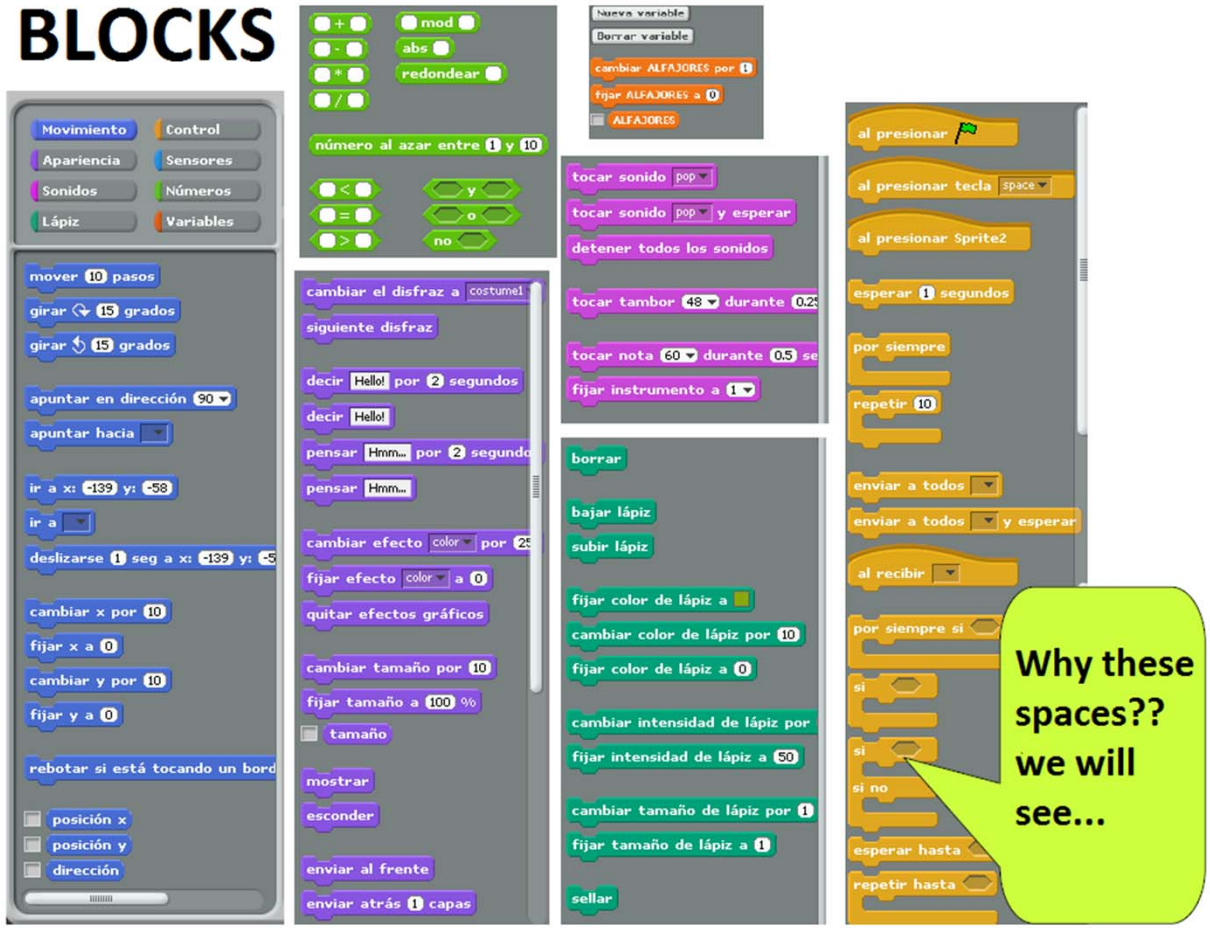
upna Nafarroako Unibertsitate Publikoa

Scratch



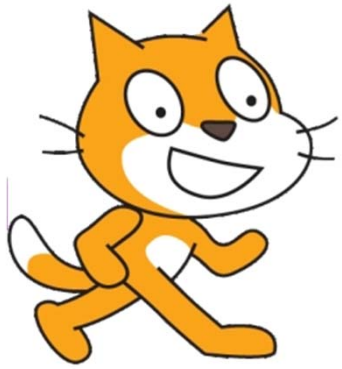
upna Nafarroako Unibertsitate Publikoa

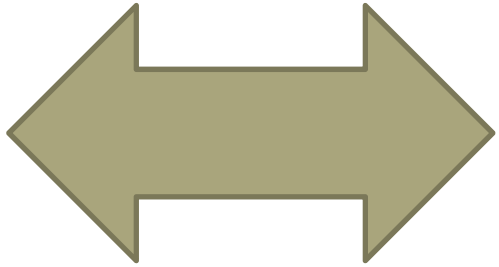
Scratch



upna Nafarroako Unibertsitate Publikoa

Comunicación Externa





python

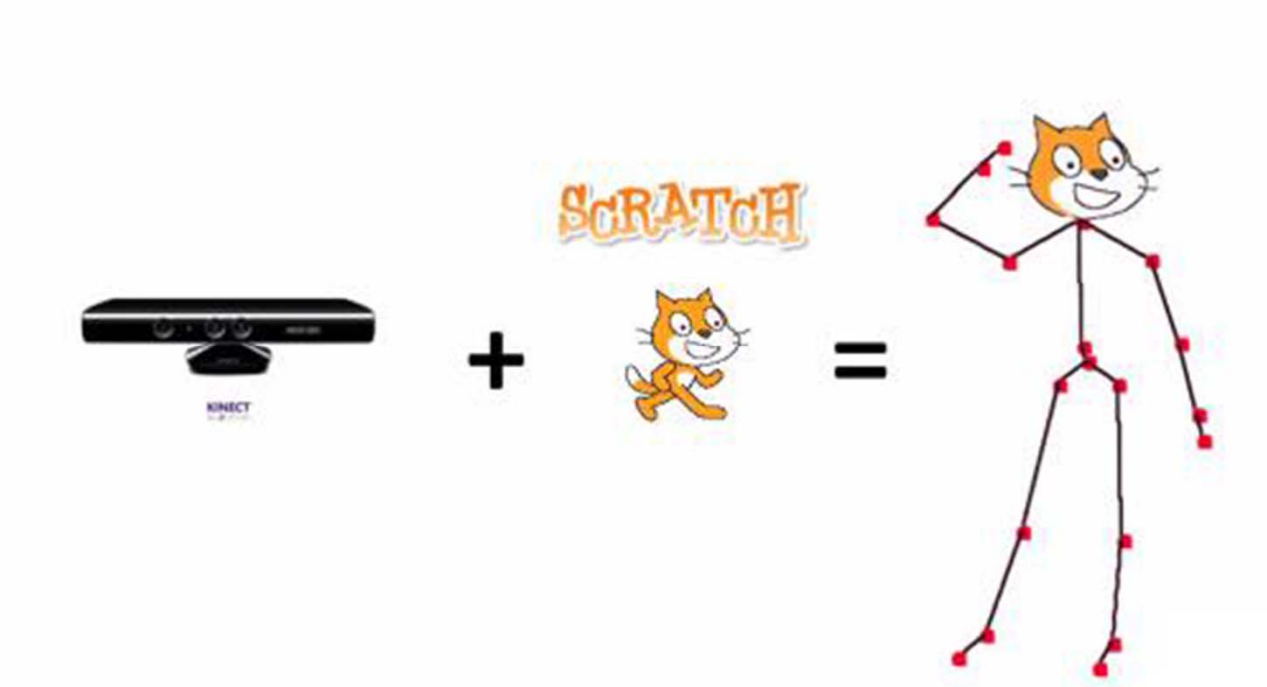
upna Nafarroako Unibertsitate Publikoa

Comunicación Externa

- Protocolo TCP/IP
- Puerto 42001
  - Salida
    - cambios en variables
    - señales broadcast
  - Entrada
    - broadcast nombre_señal
    - sensor nombre_sensor valor

upna Nafarroako Unibertsitate Publikoa

Kinect2Scratch

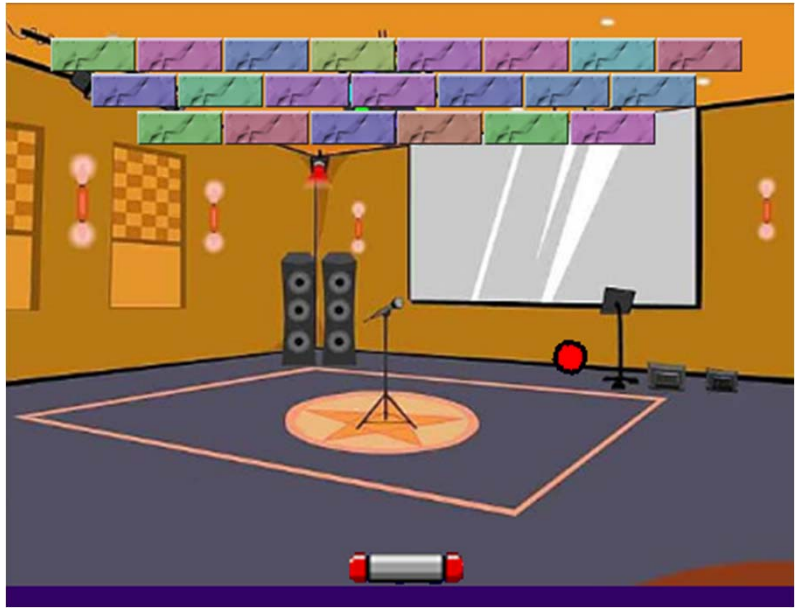


upna Nafarroako Unibertsitate Publikoa

Scratch4Arduino

upna Nafarroako Unibertsitate Publikoa

Proyectos Educativos



upna Nafarroako Unibertsitate Publikoa

Pruebas con usuarios

- Grupo de 8 personas
- Test de esfuerzo NASA TXL
- Test de Brook de usabilidad
- Resultados
  - 31/100
  - 88/100

upna Nafarroako Unibertsitate Publikoa

Competencias Educativas

- Comunicación lingüística
- Matemática
- Conocimiento e interacción con el mundo físico
- Tratamiento de información y competencia digital
- Social y Ciudadana
- Cultural y Artística
- Aprender a aprender
- Autonomía e iniciativa personal

upna Nafarroako Unibertsitate Publikoa

Conclusiones y líneas futuras

- Ampliación de Scratch
- Interactividad
- Versatilidad de Arduino
- Test
- Driver
- Iniciativas

upna Nafarroako Unibertsitate Publikoa

Preguntas

Y ahora la demo…

upna Nafarroako Unibertsitate Publikoa