Sistemas embebidos

JEDI

Campbell Eric Patricio
ercampbe@alu.itba.edu.ar

Rey Gonzalo Matías
gorey@alu.itba.edu.ar

TABLA DE CONTENIDOS

RESUMEN

Este trabajo describe todo el desarrollo del JEDI, el mando de una consola de video juegos basada en un circuito embebido. Aquí, se hará una extensa descripción del desarrollo del mismo, desde su diseño de hardware, especificaciones técnicas, software, juegos desarrollados, etc.

INTRODUCCIÓN

El JEDI (Juego Embebido Desarrollado por el ITBA) es el mando de una consola de juegos, cuyo manejo es similar al de una Wii™.
El desarrollo de la consola hizo uso de un circuito embebido, un acelerómetro, y diversos componentes electrónicos en cada uno de los controles para su completa implementación.
Los JEDI (forma en la que se llamó a cada uno de los controles) se comunican con una PC, la cual contiene una API que permite interfacear con los controles y a travéz de la cual se obtiene el punto de entrada para poder dearrollar juegos y aplicaciones que hagan uso de los JEDI.

DISEÑO

DISEÑO DE HARDWARE

El diseño del JEDI se realizó a partir de la definición y especificación de las funcionalidades con las que este debería contar, y una vez definidas, se procedió con la búsqueda de componentes de hardware.

Detección de movimiento

Primero y principal, la capacidad de sensado de movimiento del dispositivo, la cual lo dotaría de las capacidades de mando tipo Wii™. Habiendo varias opciones en el mercado, se decidió utilizar el ADXL330, el mismo utilizado por la Wii™ y gran cantidad de dispositivos que hacen uso de acelerómetros, de modo de usar un dispositivo ya conocido y altamente testeado el cual contara con una comunidad de desarrollo que pudiera darnos apoyo en caso de complicaciones. Teniendo en cuenta que sus pequeñas dimensiones hacian muy dificil el soldarla a una placa, se decidió usar una placa de montaje ADXL335 DKSB1002A Prototyping Board, la cual nos permitía manipular al acelerómetro con mas facilidad y soldarlo a la placa de manera adecuada.

Botones de función

Se tuvieron en cuenta botones que permitieran dar vesatilidad al manejo de los juegos, permitiendoles seleccionar eventos o realizar acciones puntuales como disparar o saltar. Para esto se utilzaron dos pulsadores pull-up.

Indicadores de estado

Para indicar distintos estados del JEDI, conexión y número de JEDI, se utilizaron 2 LEDS.

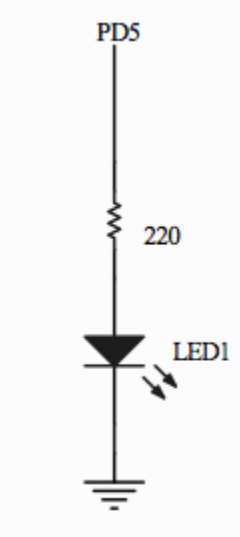
Microcontrolador del JEDI

Como sostén principal del dispositivo, se utilizó un microcontrolador ATmega644, el cual nos permitiera comunicarnos con cada uno de los sensores y tomar las acciones correspondientes para poder comunicar lo que hace el usuario hacia la consola.

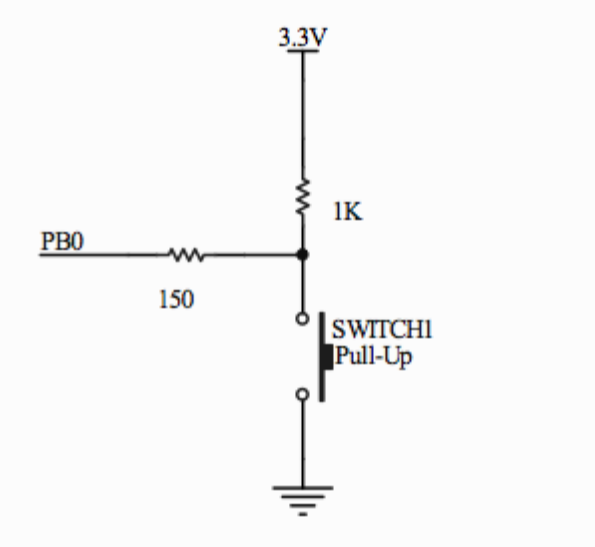
Esquemas del circuito

A continuación se encontrará una detallada descripción del circuito del JEDI, indicando la conexión de cada uno de sus componentes.

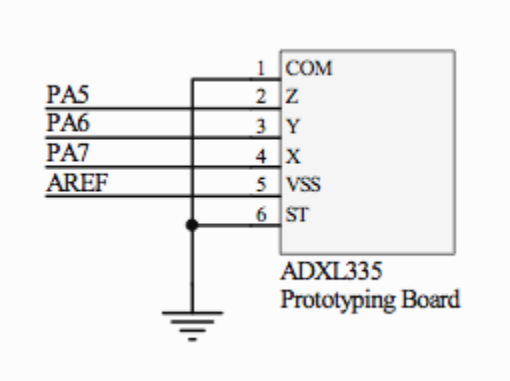
Diodos LEDs



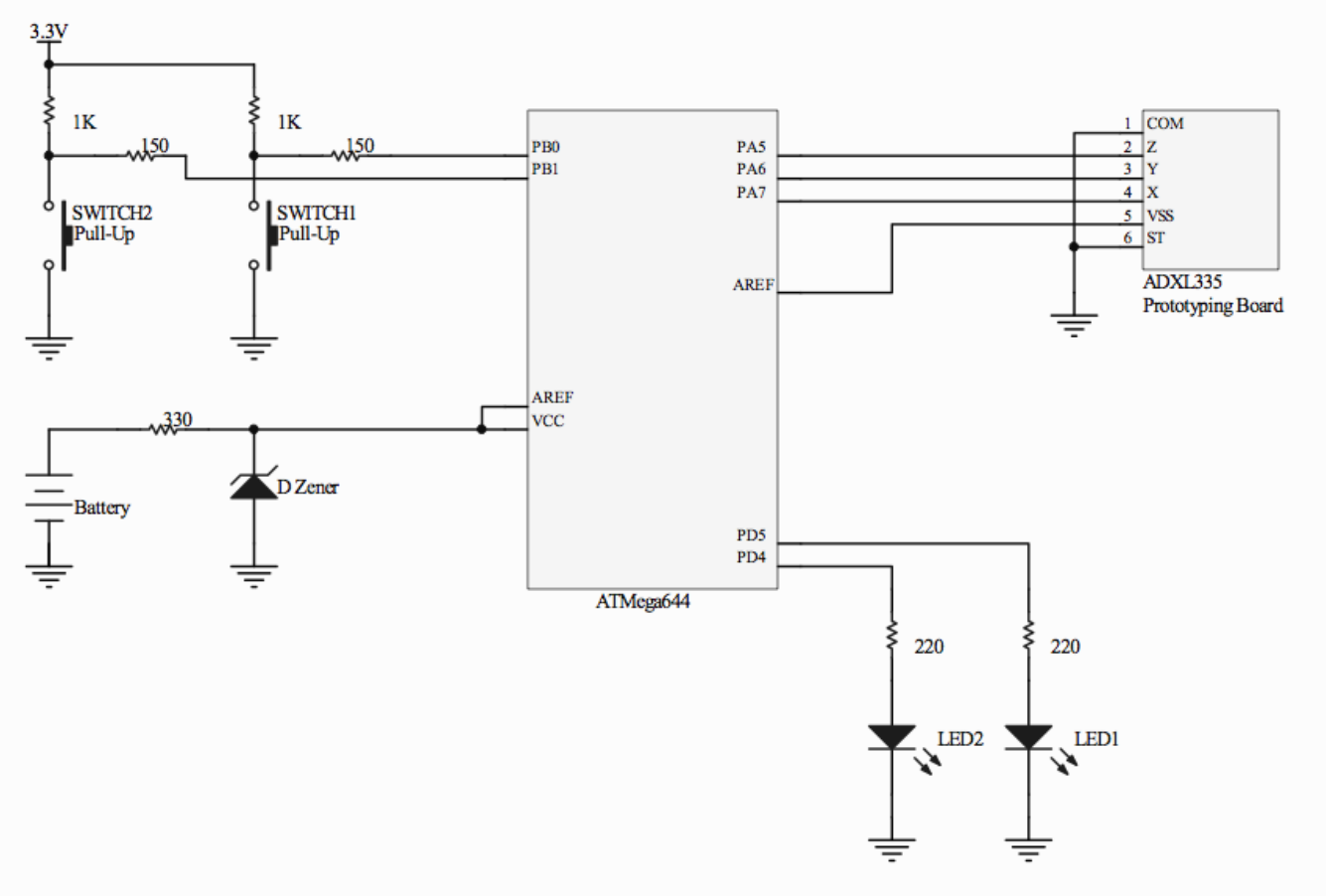
Pulsadores Pull-UP



Acelerómetro



Circuito completo



Lista de materiales

- 2 x ATmega 644 PDIP (1 x JEDI)
- 2 x Acelerómetro (1 x JEDI)
- 2 x ADXL335 DKSB1002A Prototyping Board (1 x JEDI)
- 4 x Switch (2 x JEDI)
- 4 x Led (2 x JEDI)
- 4 x Resistencia 220 Ω (2 x JEDI)
- 4 x Resistencia 150 Ω (2 x JEDI)
- 4 x Resistencia 10 kΩ (2 x JEDI)
- 2 x Resistencia 330 Ω (1 x JEDI)
- 2 x Diodo Zener 3.3 V (1 x JEDI)

DISEÑO DE SOFTWARE

El diseño de software constó de tres etapas fundamentales, divididas de acuerdo a la parte que involucraban dentro de la consola.
Estas se dividieron en *lógica del JEDI*, que permite hacer el procesamiento del comportamiento del JEDI y transmitirlo; la *API de desarrollo*, que permite obtener los distintos servicios que

comunica el JEDI y hacer uso de sus funcionalidades; y el *protocolo de comunicación*, que servía de nexo entre el JEDI y la API.

Lógica del JEDI

La lógica del JEDI se corresponde con el software dentro del microcontrolador, que se encarga de hacer las lecturas de las aceleraciones generadas por el acelerómetro, y transmitirlas a traves del puerto UART del que dispone, tomar los estados de los botones, y representar eventos mediante los LEDs.

Acelerómetro

Para la obtención de las aceleraciones, se conectaron a los puertos PA5, PA6, y PA7 del ADC del microcontrolador (pines 33, 34 y 35 respectivamente) las patas con las componentes X, Y y Z del acelerómetro, de modo de poder convertir la señal analógica generada por ellas a una digital que el pueda transmitir y procesar.
Los valores generados por el acelerómetro están comprendidos entre 0V y 3.3V para representar una aceleración que iba de -3g a 3g, los cuales son mapeados por el ADC a valores digitales que van de 0 a 255, con lo que nos daba una precisión de 0.02g, lo cual consideramos suficientemente alta.
El microcontrolador toma mediciones de los valores generados por el acelerómetro cada 6ms (2ms para cada componente) y los va almacenando en un buffer, sobre el cual aplica una operación de procesamiento que puede ser la media, la mediana o la moda de los valores almacenados en el mismo (configurable desde el lado del servidor) y los envía al servidor cada 40ms, lo que equivalen a 25 veces por segundo. Esto nos permitió por un lado tener una gran precisión a la hora de medir las aceleraciones leídas por el JEDI, y por otro, con el buffer, controlar gran parte del ruido generado por las mediciones.

Botones

El microcontrolador controla constantemente (dentro de un ciclo *while(1)*) el estado de los botones, seteando su estado y comunicandolo en cada paquete *inform* enviado al servidor. El botón A se encuentra conectado al puerto PB0 y el botón B al PB1 (pines 1 y 2 respectivamente).

LEDs

Los LEDs son utilizados para indicar el estado de *búsqueda y de configuración* del JEDI. Los mismos están conectados a los puertos PD5 para el L1 el y PD4 para el L2 (pines 19 y 18 respectivamente). Cuando el JEDI no está conectado a ningún servidor, está en un estado que llamamos *DISCOVERY*, en el cual ambos LEDs titilan (se prenden y apagan) dos veces por segundo. Una vez que el JEDI pasa a estar conectado, el mismo es informado que número de JEDI es (uno o dos para identificar al jugador) y prende el LED izquierdo o derecho de acuerdo a cual le haya correspondido.

Puerto UART

Para la comunicación con el servidor se realizan lecturas y escrituras sobre el puerto UART en los puertos RXD0 y TXD0 (pines 14 y 15 respectivamente). Las lecturas se realizan cada 1ms, mientras que las escrituras varían de acuerdo al estado en el que se encuentre el JEDI, *DISCOVERY* o *INFORMING*.

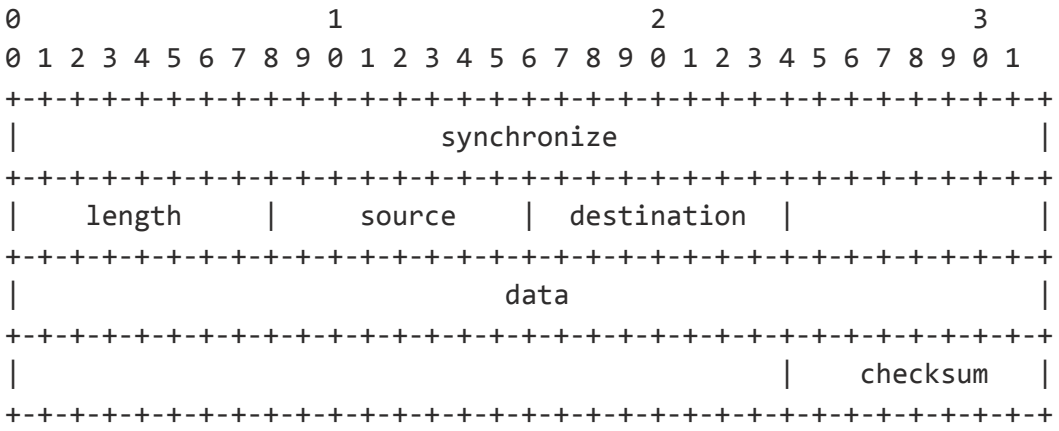
La configuracion de la UART es de 9600 baudios, 8 bits de datos, 2 bits de stop y sin paridad.

Protocolo de comunicación

Transporte

Se utilizó un protocolo especialmente diseñado para este sistema, extrayendo carácterísticas del protocolo Ethernet y del DHCP.

Dicho protocolo está formado por cinco elementos: padding, length, source, destination, data y finalmente checksum, dispuestos como se muestra a continuación.



Nombre	Tamaño	Descripción
`synchronize`	4 Bytes	Para lograr una sincronización entre paquetes, se envían 4 bytes en cero consecutivos, los cuales marcan el inicio de un paquete. Dicha característica fue tomada del protocolo de Ethernet, que realiza una sincronización similar.
`length`	1 Byte	Longitud del mensaje a recibir. En caso de que el contenido de `data` sea cero, `length` será 4.
`source`	1 Byte	ID del que está generando el mensaje.
`destination`	1 Byte	ID del destinatario del mensaje.
`data`	variable	Datos del paquete, de longitud variable. El mismo especifica

		el comando e información necesaria para su procesamiento, como por ejemplo, las aceleraciones X, Y y Z.
`checksum`	1 Byte	Validación del paquete, realizando un xor de cada Byte que lo comprende.

Comandos

La especificación del protocolo comprende la descripción de los distintos comandos que usa:

Nombre	ID	Tamaño	Descripción
DISCOVER_REQUEST	11	5 Byte	Contiene 1 Byte de `data`, con el ID.
DISCOVER_RESPONSE	12	7 Bytes	Contiene 3 Bytes de `data`. Uno con el ID, otro con el número de JEDI que es, y otro con el método de procesamiento de las aceleraciones a obtener.
LISTENING	20	5 Bytes	Contiene 1 Byte de `data`, con el ID.
INFORM	30	10 Bytes	Contiene 6 Bytes de `data`, uno con el ID, los ejes X, Y y Z y el estado de los botones A y B.

Secuencia típica

La secuencia típica de una comunicación entre el JEDI y el server es:

- Inicialmente el JEDI comienza broadcasteando su mensaje DISCOVER_REQUEST a la espera de ser descubierto por algún servidor.
- Una vez que un servidor recibe un DISCOVER_REQUEST, se responde al que envió dicho paquete con un DISCOVER_RESPONSE.
- Una vez establecida la comunicación, el JEDI comienza a transmitir paquetes LISTENING cada 50 ms siempre y cuando se sigan recibiendo paquetes INFORM; si se dejan de recibir, y el tiempo de espera vence, el JEDI vuelve a pasar a estar en estado de DISCOVER_REQUEST.
- INFORM: Paquete enviado una vez que la comunicación ya fue establecida, el cual contiene el estado de los botones A y B, y de las componentes X, Y y Z.

Implementación

Para poder abstraerse lo mas posible de la forma en la que se maneja la comunicación, ya sea con la UART, bluetooth, USB, etc, se realizó una interfaz CommHandler, la cual proponía los métodos mas genéricos para poder comunicarse con el JEDI, y la misma implementación del JEDI_api decidiría que implementación de la interfaz CommHandler usaría. En nuestro caso, usamos la SerialCommHandler, que hacía uso de la comunicación con el puerto serie.

Para el manejo del puerto serie, se utilizó una librería open source basada en javax.comm, llamada RXTXcomm, la cual prometía ofrecer los mejores métodos de comunicación a traves de puertos serie, USB, etc. Sin embargo, luego de trabajar durante muchos meses con esta librería, ya teniendo todo construido en base a ella, surgió como inconveniente que al querer comunicar el servidor con dos JEDI conectados en dos puertos serie distintos simultaneamente, la comunicación con el segundo que se conectara al servidor anulaba por completo la comunicación del primero, por lo que impedía la comunicación con ambos JEDI a la vez. De esta manera, para poder simular la jugabilidad entre dos jugadores, se hace uso de una implementación de la JEDI_api que se comunicara con el teclado en lugar de con el puerto serie llamada FakeJEDI, logrando una experiencia un poco mas agradable que la de un solo jugador.

FakeJEDI hace una estimación de las aceleraciones en base a las teclas que el usuario precione en el teclado, conecta y desconecta al jugador mediante el uso de otras dos teclas, y asigna dos teclas mas para la representación de los botones.

La real implementación de la api del JEDI es mediante la clase JEDI, la cual establece la comunicación por el puerto serie con el JEDI conectado a el. La misma usa la clase SerialCommHandler para generar los paquetes necesarios y enviarlos por el puerto serie al JEDI conectado, establecer la comunicación, su estado, y mantenimiento de la conexión.
La aceleración se obtiene mediante packetes INFORM generados por el JEDI, y estos son almacenados en una cola a partir de la cual retorna el último elemento ingresado. Dicha cola se utiliza para poder estimar los valores de las velocidad y la posición en un instante, a partir de la integración de la aceleración a lo largo del tiempo transcurrido entre las mediciones. Esta estimación genera muchísimo error, por lo que hace falta realizar una calibración del dispositivo cada cierto tiempo transcurrido.

La idea del método *calibrate* es que el usuario deje quieto el JEDI, mientras esta toma las aceleraciones durante un cierto tiempo y luego ajusta las aceleraciones seteando un nuevo cero, y así, disminuir el error en las mediciones.

API de desarrollo

La API de desarrollo incluye los métodos, clases e interfaces que el usuario debe implementar para poder interactuar con los JEDI y aprovechar sus características.
En primer lugar, para la creación de un juego, el usuario debería implementar una clase que extienda de JEDIGame, la cual abarca todos los métodos e interfaces para comunicarse con el JEDI. Luego, para interfacear con el, el usuario debe utilizar las implementaciones de la clase JEDI_api, JEDI y FakeJEDI que hacen de controles.

JEDI_api.java

```
package api;
```

```
import data.Buttons;
import data.Axis;

public interface JEDI_api {

        /**
          * Enum that indicates the possible JEDI numbers
          */
         public static enum JoystickNumbers{JEDI_ONE, JEDI_TWO};

         /**
          * Gets the number of the current JEDI, setted as JEDI_ONE or JEDI_TWO
          *
          * @return An enum with the number of the JEDI
          */
         JoystickNumbers getJediNumber();

         /**
          * Gets the ID of the JEDI
          *
          * @return An integer with the id of the current JEDI
          */
         Integer getJediID();

         /**
          * Get the acceleration component of each axis
          * @return An object Axis with the JEDI accelerationsw
          */
         Axis getAcceleration();

         /**
          * Get the velocity component of each axis
          * @return An object Axis with the JEDI velocity
          */
         Axis getVelocity();

         /**
          * Get the position component of each axis
          * @return An object Axis with the JEDI position
          */
         Axis getPosition();
```

```
	/**
	 * Get the direction component of each axis
	 * @return An object Axis with the JEDI directions
	 */
	Axis getDirection();

	/**
	 * Get the rotation component of each axis
	 * @return An object containing the states of the JEDI rotation
	 */
	Axis getRotation();

	/**
	 * Get the buttons
	 * @return An object containing the states of the JEDI buttons
	 */
	Buttons getPressedButtons();

	/**
	 * Calibrate the JEDI
	 * @param milliseconds Lapse where the calibration will take place
	 * @param calibrationCallback Callback function to be called at the beginning and
	 * at the end of the calibration process
	 */
	void calibrate(int milliseconds, CalibrationInterface calibrationCallback);
}
```

JEDIGame.java

```
package api;

import javax.swing.JPanel;

public abstract class JEDIGame extends JPanel implements ButtonListenerInterface,
ConnectionListenerInterface, CalibrationInterface {

	private static final long serialVersionUID = 1L;

	/**
	 * JEDI_api instances that the game will handle
	 */
	protected JEDI_api jedi1;
	protected JEDI_api jedi2;
```

```
	/**
	 * Callback interface to be called at the start and ending of the game
	 */
	private GameCallbackInterface callback;

	/**
	 * Constructor of a Game with two JEDIs
	 *
	 * @param jedi1 Instance of the JEDI_ONE
	 * @param jedi2 Instance of the JEDI_TWO
	 */
	public JEDIGame(JEDI_api jedi1, JEDI_api jedi2) {
		this.jedi1 = jedi1;
		this.jedi2 = jedi2;
	}

	/**
	 * Constructor of a Game with just one JEDI
	 *
	 * @param jedi1 Instance of the JEDI_ONE
	 */
	public JEDIGame(JEDI_api jedi1) {
		this.jedi1 = jedi1;
	}

	/**
	 * Run the game with the current callback function
	 *
	 * @param callback interface to be exectuted at the start and ending of the game
	 */
	public void run(GameCallbackInterface callback){
		callback.onStart();
		start();
		this.callback = callback;
	}

	/**
	 * Gets the callback reference
	 *
	 * @return gets the callback reference
	 */
	public GameCallbackInterface getCallback(){
		return callback;
	}

	/**
	 * Method to be implemented by the caller of the function
	 */
	protected abstract void start();
}
```

ButtonListenerInterface.java

```
package api;

import event.ButtonEvent;

public interface ButtonListenerInterface {

	/**
	 * Event that triggers when a button is pressed
	 * @param e Event sent to the handler
	 */
	public void buttonPressed(ButtonEvent e);

	/**
	 * Event that triggers when a button is released
	 * @param e Event sent to the handler
	 */
	public void buttonReleased(ButtonEvent e);
}
```

CalibrationInterface.java

```
package api;

public interface CalibrationInterface {

	/**
	 * Function to be called before the calibration starts
	 */
	public void onStart();

	/**
	 * Function to be called when the calibration ends
	 */
	public void onEnding();
}
```

ConnectionListenerInterface.java

```
package api;

import event.ConnectionEvent;

public interface ConnectionListenerInterface {
```

```
        /**
         * Event that triggers when the connection starts
         * @param event Event sent to the handler
         */
        public void connectionStarted(ConnectionEvent event);

        /**
         * Event that triggers when the connection ends
         * @param event Event sent to the handler
         */
        public void connectionEnded(ConnectionEvent event);
}
```

GameCallbackInterface.java

```
package api;

public interface GameCallbackInterface {

        /**
         * Method to be called before the start of a game
         */
        public void onStart();

        /**
         * Method to be called after the game finished
         */
        public void onFinish();
}
```

Implementación de un juego

Para la implementación de un juego, es necesaria que la clase que lo implemente extienda de JEDIGame, que abarca todas las interfaces y métodos a implementar para poder comunicarse con un JEDI.
En nuestro servidor, se hicieron tres clases que extiendan de JEDIGame, Menu, Spacecraft, y Pong. Todas ellas hacen uso de las características del JEDI para poder manifestarse, y se controlan mediante objetos que implementan JEDI_api, de modo de mantener la mayor abstracción posible, y así poder comunicarnos indistintamente con JEDI y FakeJEDI, logrando así la abstracción deseada.
A la vez, se usó un objeto llamado Dashboad, el cual muestra mediante una serie de gráficos, lo que está haciendo cada uno de los JEDI en el juego.

CONCLUSIÓN

Este trabajo nos enfrentó a problemas que no nos habíamos encontrado previamente en la carrera. En situaciones previas, partíamos de una premisa, una aplicación u objetivo claro, restringido a una consigna; sin embargo, en este caso se sabía meramente el objetivo, el cual era desarrollar una consola de videojuegos similar a la Wii™ junto con dos juegos.
El proceso de investigación tomó mucho tiempo, para la decisión de componentes a utilizar, investigar su uso, ponernos a probar, etc.

Problemas encontrados y posibles extensiones

Como problemas, el primero con el que nos encontramos fue bien temprano, a la hora de tratar de implemetar la comunicación por bluetooth que estaba dentro de las condiciones inicialmente propuestas para el trabajo, la cual resultó de muy alta complejidad, tanto por el lado de software como de hardware, por lo que se decidió por usar una comunicación serie, abstrayendola lo mas posible del resto del sistema, de modo de poder agregar una comunicación bluetooth mas adelante.
Luego, el mayor de los problemas con el que nos topamos fue con la limitación de la librería RXTXComm, que nos impedía comunicarnos con dos JEDI simultáneamente, a traves del puerto serie. Esto, obviamente es tenido en cuenta como futura extensión, debido a que por cuestión de tiempos no se pudo concretar esa funcionalidad, ya tenida en cuenta desde el comienzo del proyecto.