Práctica 2
Human Interface Devices
XBox 360 Controller

Desarrollo de Sistemas Interactivos

Fuentes:

- Material MDV(UCM) de Pedro Pablo Gómez Martín
- “Game Engine Architecture” de Jason Gregory
- http://www.epanorama.net/documents/joystick/index.html

1 Human Interface Devices (HID)

Every game needs to process input from the player, obtained from various human interface device s (HIDs) including:

- the keyboard and mouse,
- joypad & joystick
- steering wheels,
- dance pads,
- the WiiMote, etc.

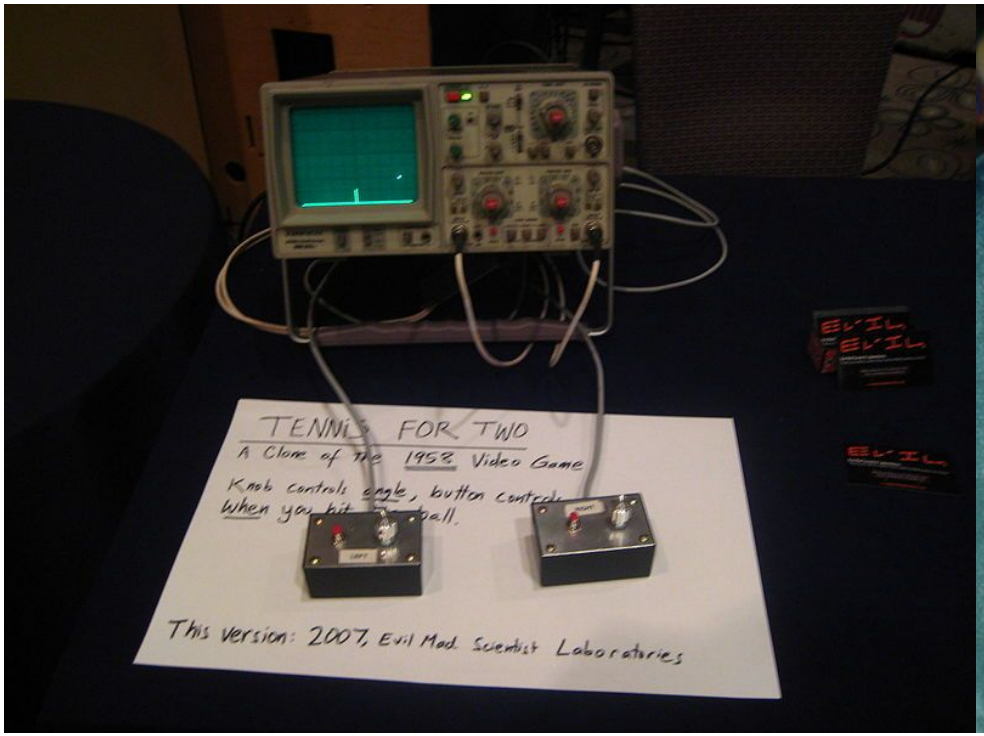
We sometimes call this component the player **I/O component, because we may also provide output to the player through the HID** , such as force feedback/rumble on a joypad or the audio produced by the WiiMote.

2. Historia

Paddles

En **1958** se desarrolla **“Tennis for Two”** controlado con dos potenciómetros.

Los dispositivos de entrada de ese tipo que se hicieron populares a partir de entonces suelen conocerse como “**paddles**”



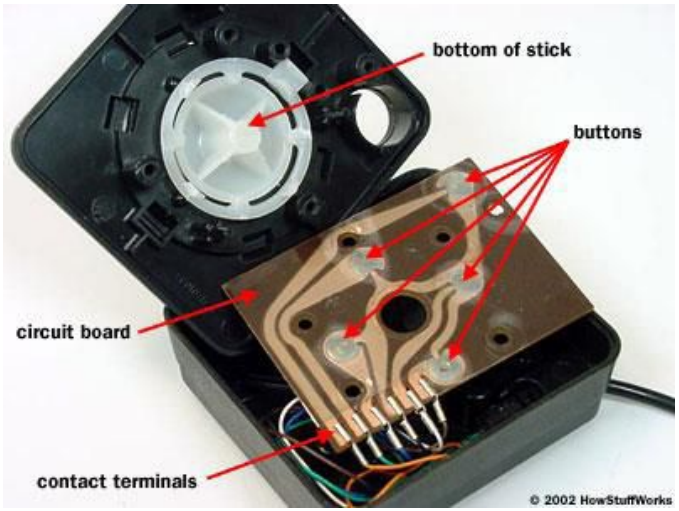
2. Historia

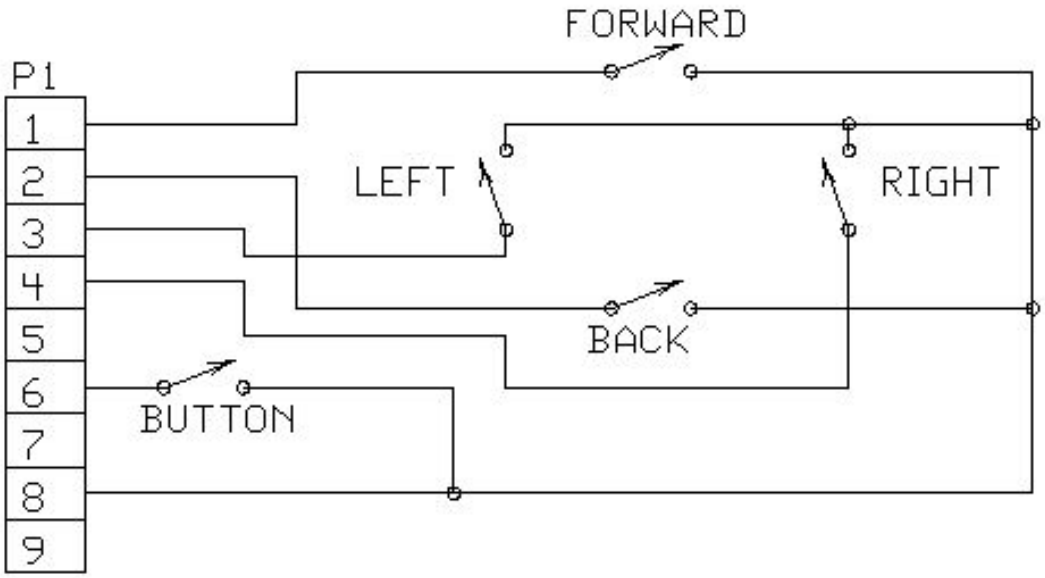
Joysticks digital

La idea se hizo popular y se convirtió en el dispositivo de juego habitual en las videoconsolas y ordenadores de 8 bits.

2. Historia

Joysticks digital





2. Historia

Los **joysticks digitales** eran baratos de fabricar y de controlar. Eran apropiados para juegos tipo arcade, pero **no para simuladores de vuelo**, por ejemplo.

Los j**oysticks analógicos** son la mezcla de los joysticks digitales y los paddle. **Cada eje** en realidad está regulado por **un potenciómetro**.

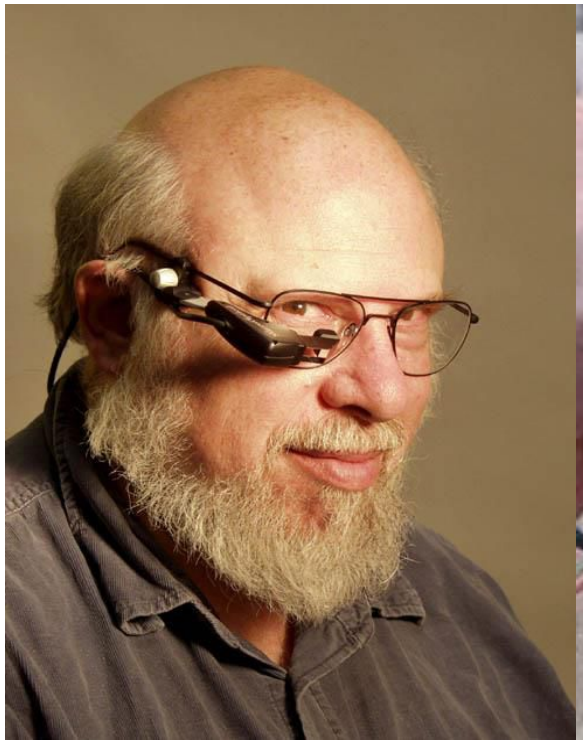
Los IBM PC “tuvieron” puerto para dos Joysticks analógicos desde el principio.

Joysticks analógico

2. Historia

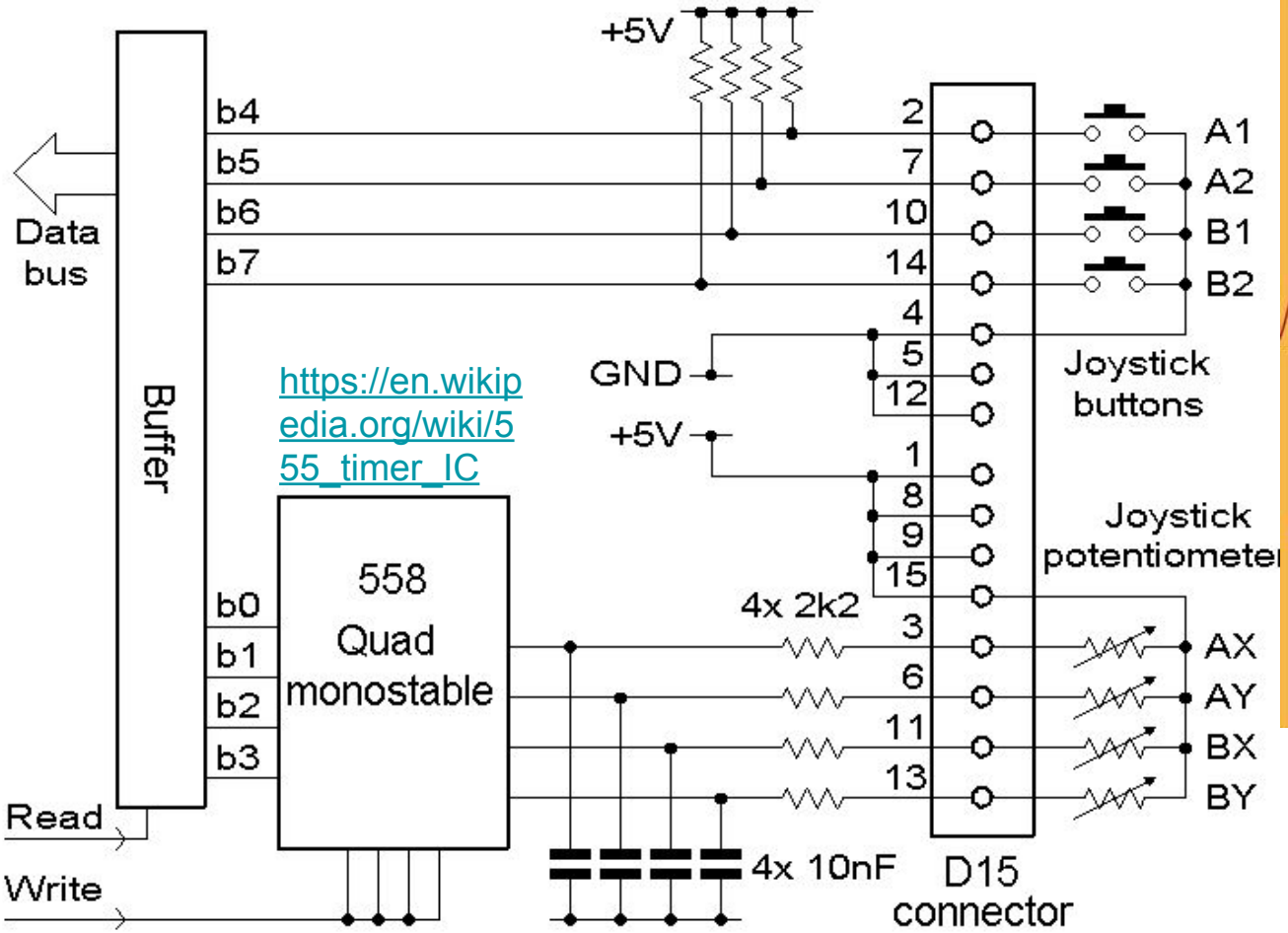
Joysticks analógico

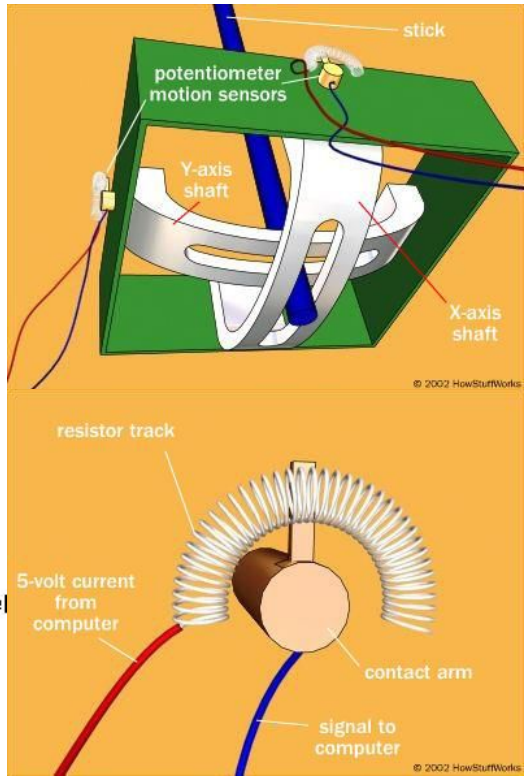
Jef Raskin creó el primer Joystick analógico para Apple II



2. Historia

Joysticks analógicos





STANDARD PC JOYSTICK INTERFACE (2 STICKS)

2. Historia Joysticks analógico

Requerían tarjetas específicas.

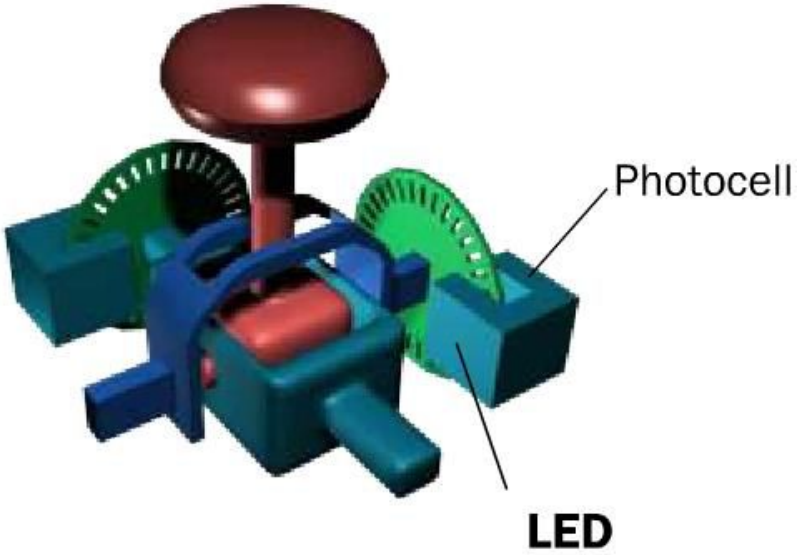
Algunas tarjetas sólo soportaban un joystick. Surgen interfaces alternativos para conectar joysticks con otras funcionalidades.

El puerto del Joystick era únicamente de entrada. Para añadirle Force Feedback había que utilizar algún truco.

- Algunos conectaban el joystick también al puerto serie, o al puerto del teclado para poder enviar información.
- Microsoft Sidewinder Force Feedback Pro utilizaba el puerto del Joystick habitual, haciendo uso de los conectores MIDI.

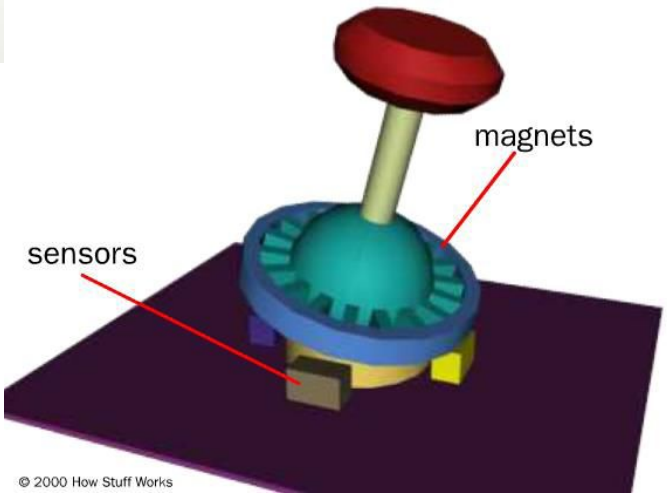
2. Historia

Mando de Nintendo64: óptico



2. Historia

Mando de Dreamcast: magnético

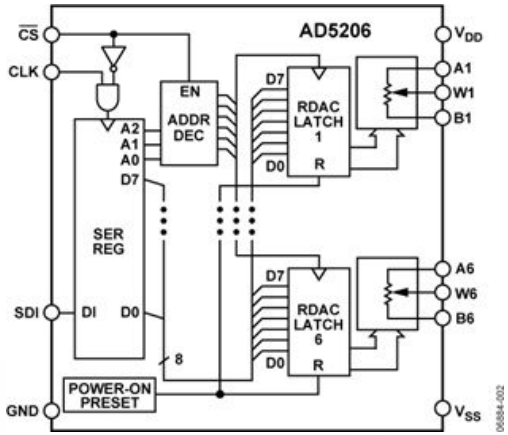


2. Historia XBox 360 y PS 3 Joypad Controllers

Consoles like the Xbox 360 and PS3 come equipped with joypad controllers:

- Botones digitales y analógicos
- Pads y Joysticks analógicos
- Vibradores
- Audio

AD5206



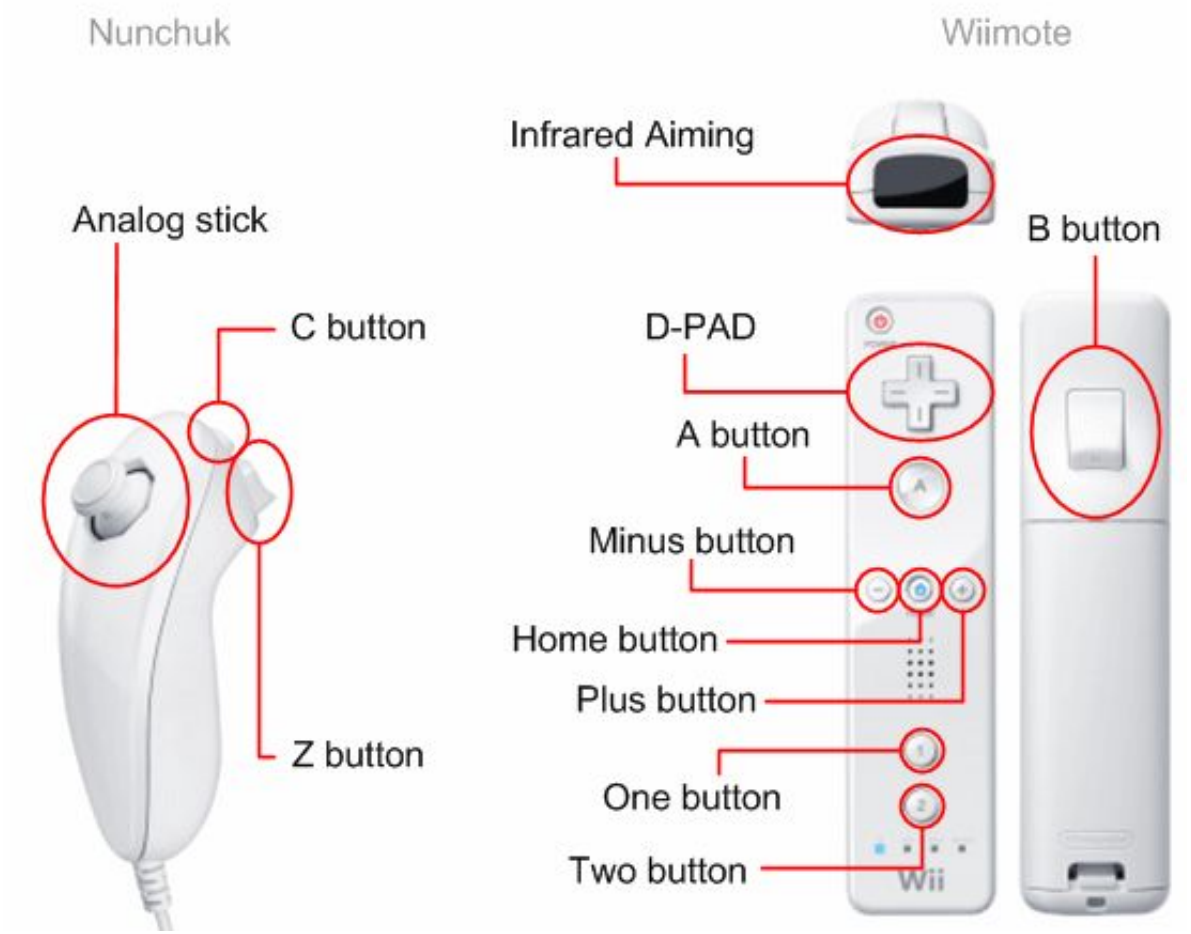
2. Historia

Nintendo’s Wii console is well known for its unique and innovative WiiMote controller:

Además:

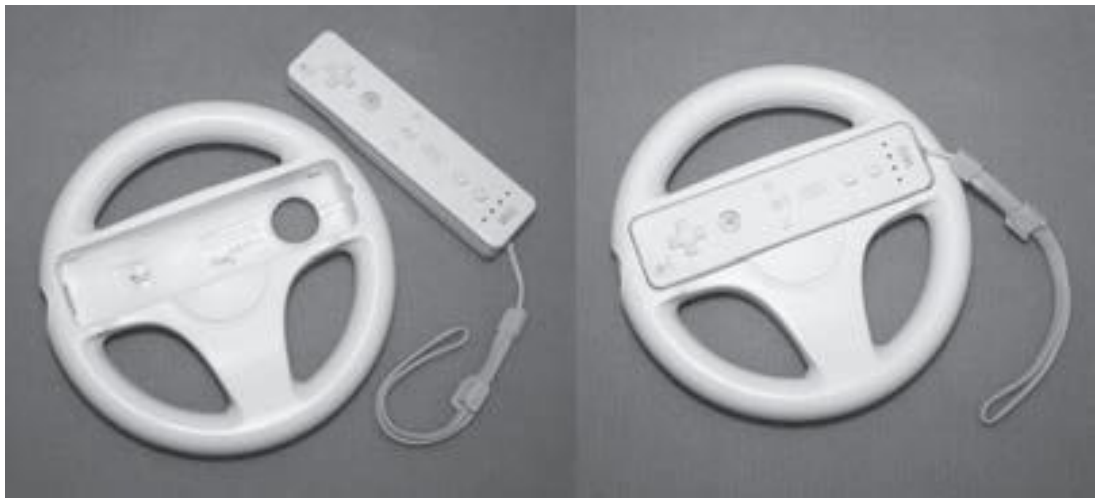
- acelerómetros
- girómetros
- cursor(cámara)
- altavoz
- vibración

Wiimote



2. Historia Otros, un mercado interesante

Many specialized input devices are available for use with consoles.



2 Historia Programación de HID

En MS-DOS

- En los XT no había soporte software para acceder a los joysticks. Los juegos implementaban las rutinas de lectura del estado de los ejes.
- La BIOS de los AT añadió una función para leer el estado.
- Es necesario realizar encuesta. No hay interrupción asociada al joystick.
- El modo de controlar a cada dispositivo es propietario, es necesaria la instalación de drivers para controlarlo.

En Windows

- Primero se proporcionó un API dentro del “Platform SDK Windows Multimedia” para acceso a los joystick.
- Después Direct Input como parte de DirectX
- **Ahora es un COM en XInput** (Es como lo programaremos en prácticas)

2 Historia Programación de HID en la actualidad

The definition of **HID started** in the late 1990s **as a device class over USB**. The goal at that time was to define a **replacement to PS/2** and create an interface over USB, allowing the creation of a **generic driver for HID devices like keyboards, mice, and game controllers**.

It was originally designed for low latency, low bandwidth devices but is flexible, and the rate is specified by the underlying transport.

Today, HID has a standard protocol over multiple transports, and the **following transports are supported natively in Windows 8 & 10 for HID**: **USB, Bluetooth, Bluetooth LE, I²C**

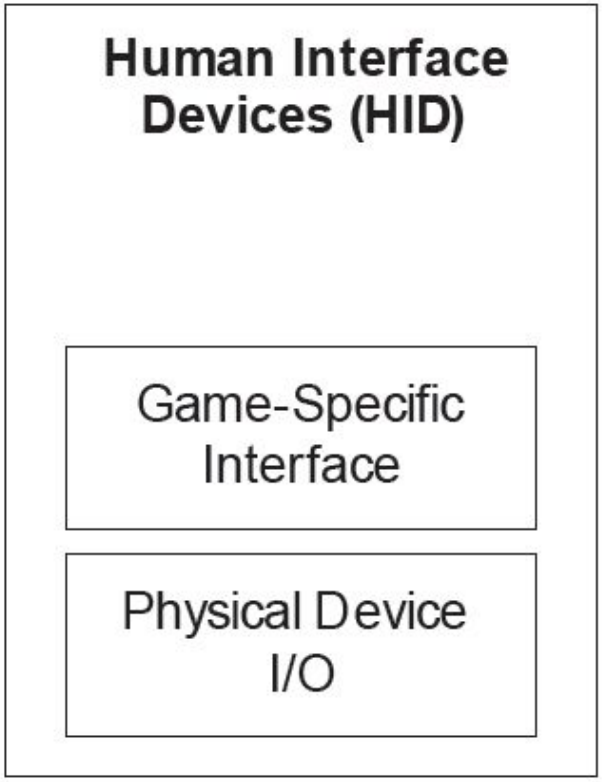
https://msdn.microsoft.com/es-es/windows/hardware/drivers/hid/index

https://developer.apple.com/library/content/documentation/DeviceDrivers/Conceptual/HID/new_api_10_5/tn2187.html

3.1 HID Abstracción

The HID engine component is sometimes architected to divorce the low-level details of the game controller(s) on a particular hardware platform from the high-level game controls:

- It massages the raw data coming from the hardware
- mapping between physical controls and logical game functions
- detecting chords, sequences and gestures



3.1 HID Programación Dispositivo

Every game needs to process input-output from-to the player, using various human interface device s (HIDs) in various ways:

- **Polling the hardware periodically.**
  "Microsoft 's XInput API, for use with Xbox 360 game pads on both the Xbox 360 and Windows PC platforms, is a good example of a simple polling mechanism"
- **Interrups or Menssages**: Some HIDs only send data to the game engine when the state of the controller changes in some way.
  "A mouse spends a lot of its time just sitting still on the mouse pad. There's no reason to send a continuous stream of data between the mouse and the computer when the mouse isn't moving"
- **Wireless Devices and USB**: The software must "talk" to the device via the Wireless/wired protocol.
  The inputs and outputs of a Bluetooth device, like the WiiMote, the DualShock 3 and the Xbox 360 wireless controller.

3.2 Programación XInput

XInput API soporta 4 controladores.

El número se asocia al puerto.

Para capturar el estado *XInputGetState*:

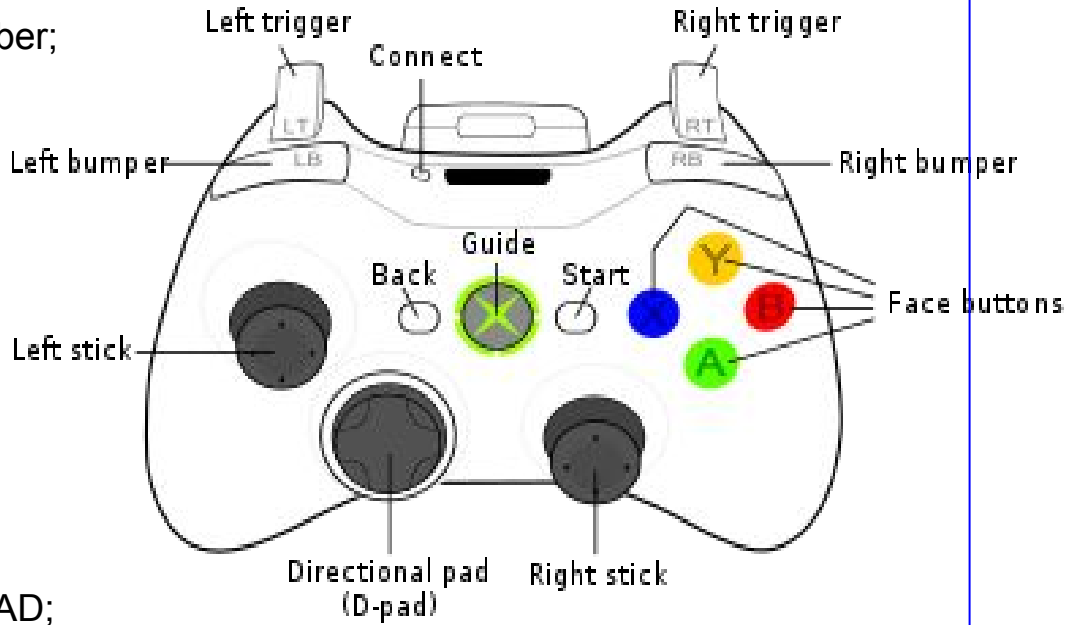
```
struct CONTROLER_STATE
{
    XINPUT_STATE state;
    bool bConnected;
};

CONTROLER_STATE
g_Controllers[MAX_CONTROLLERS];
```

```
HRESULT UpdateControllerState()
{
    DWORD dwResult;
    for( DWORD i = 0; i < MAX_CONTROLLERS; i++ )
    {
        // Simply get the state of the controller from XInput.
        dwResult = XInputGetState( i, &g_Controllers[i].state );

        if( dwResult == ERROR_SUCCESS )
            g_Controllers[i].bConnected = true;
        else
            g_Controllers[i].bConnected = false;
    }

    return S_OK;
}
```

3.2 Programación XInput

El estado del mando se guarda en una estructura XINPUT_GAMEPAD:

```
typedef struct _XINPUT_STATE
{
    DWORD                   dwPacketNumber;
    XINPUT_GAMEPAD          Gamepad;
} XINPUT_STATE, *PXINPUT_STATE;

typedef struct _XINPUT_GAMEPAD
{
    WORD                    wButtons;
    BYTE                    bLeftTrigger;
    BYTE                    bRightTrigger;
    SHORT                   sThumbLX;
    SHORT                   sThumbLY;
    SHORT                   sThumbRX;
    SHORT                   sThumbRY;
} XINPUT_GAMEPAD, *PXINPUT_GAMEPAD;
```



3.2 Programación Digital Buttons

This struct contains a **16-bit unsigned integer (WORD)** variable named ***wButtons*** that holds the state of all buttons. The following masks define which physical button corresponds to each bit in the word.

```
#define XINPUT_GAMEPAD_DPAD_UP 0x0001 // bit 0
#define XINPUT_GAMEPAD_DPAD_DOWN 0x0002 // bit 1
#define XINPUT_GAMEPAD_DPAD_LEFT 0x0004 // bit 2
#define XINPUT_GAMEPAD_DPAD_RIGHT 0x0008 // bit 3
#define XINPUT_GAMEPAD_START 0x0010 // bit 4
#define XINPUT_GAMEPAD_BACK 0x0020 // bit 5
#define XINPUT_GAMEPAD_LEFT_THUMB 0x0040 // bit 6
#define XINPUT_GAMEPAD_RIGHT_THUMB 0x0080 // bit 7
#define XINPUT_GAMEPAD_LEFT_SHOULDER 0x0100 // bit 8
#define XINPUT_GAMEPAD_RIGHT_SHOULDER 0x0200 // bit 9
#define XINPUT_GAMEPAD_A 0x1000 // bit 12
#define XINPUT_GAMEPAD_B 0x2000 // bit 13
#define XINPUT_GAMEPAD_X 0x4000 // bit 14
#define XINPUT_GAMEPAD_Y 0x8000 // bit 15
```

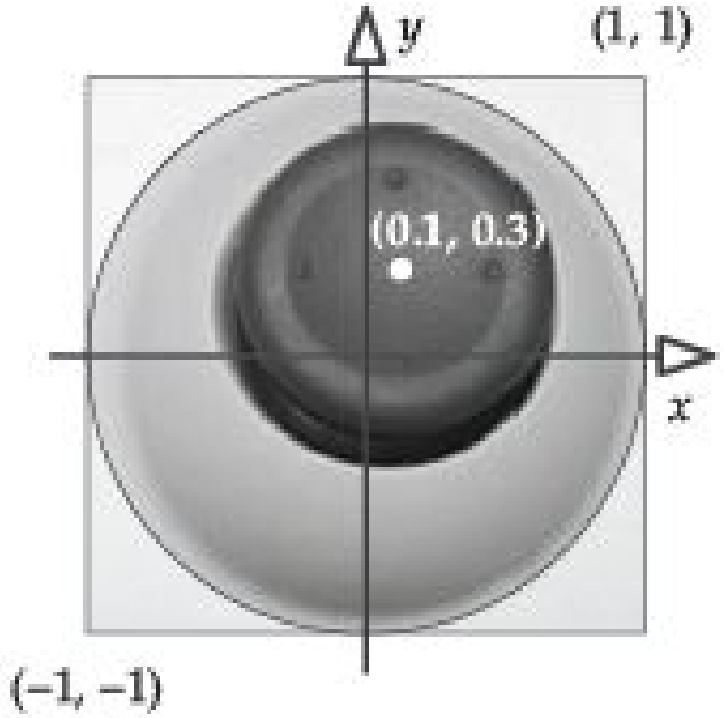
```
bool IsButtonADown(const XINPUT_GAMEPAD& pad)
{// Mask off all bits but bit 12 (the A button).
        return ((pad.wButtons & XINPUT_GAMEPAD_A) != 0);
}
```

3.2 Programación Analog Axes and Buttons

These kinds of inputs are often used to represent the degree to which a trigger is pressed, or the two-dimensional position of a joystick (which is represented using two analog inputs, one for the x-axis and one for the y-axis.

An analog input signal is usually digitized, meaning it is quantized and represented using an integer in software.

For example, an analog input might range from –32,768 to 32,767 if represented by a 16-bit signed integer. Sometimes analog inputs are converted to floating-point the values might range from –1 to 1, for instance.



3.2 Programación Rumble

In addition to getting the state of the controller, you may also send vibration data to the controller to alter the feedback provided to the user of the controller. The controller contains two rumble motors that can be independently controlled by passing values to the XInputSetState function.

```
XINPUT_VIBRATION vibration;
ZeroMemory( &vibration, sizeof(XINPUT_VIBRATION) );
vibration.wLeftMotorSpeed = 32000; // use any value between 0-65535 here
vibration.wRightMotorSpeed = 16000; // use any value between 0-65535 here
XInputSetState( i, &vibration );
```

3.2 Programación Audio

The headset that can be connected to an Xbox 360 Controller has two functions: it can record sound using a microphone, and it can play back sound using a headphone. It is accomplished using the IDirectSound8.

```
XInputGetDSoundAudioDeviceGuids( i, &dsRenderGuid, &dsCaptureGuid );
// Create IDirectSound8 using the controller's render device
if( FAILED( hr = DirectSoundCreate8( &dsRenderGuid, &pDS, NULL ) ) )
   return hr;
// Set coop level to DSSCL_PRIORITY
if( FAILED( hr = pDS->SetCooperativeLevel( hWnd, DSSCL_NORMAL ) ) )
   return hr;
// Create IDirectSoundCapture using the controller's capture device
if( FAILED( hr = DirectSoundCaptureCreate8( &dsCaptureGuid, &pDSCapture,
NULL ) ) )
   return hr;
```

3.2 Programación Typical Requirements

A game engine’s HID system usually provides some or all of the following features:

- dead zones,
- analog signal filtering,
- event detection (e.g., button up, button down),
- detection of button sequences and multibutton combinations (known as chords),
- gesture detection,
- management of multiple HIDs for multiple players,
- multiplatform HID support,
- controller input re-mapping,
- context-sensitive inputs,
- the ability to temporarily disable certain inputs.

3.2 Programación Dead zone

La zona muerta que propone DirectX no es muy efectiva.

Problemas: No separa ejes ni descuenta zona muerta.

```
if( g_bDeadZoneOn )
        {
            // Zero value if thumbsticks are within the dead zone
            if( ( g_Controllers[i].state.Gamepad.sThumbLX < INPUT_DEADZONE &&
                  g_Controllers[i].state.Gamepad.sThumbLX > -INPUT_DEADZONE ) &&
                ( g_Controllers[i].state.Gamepad.sThumbLY < INPUT_DEADZONE &&
                  g_Controllers[i].state.Gamepad.sThumbLY > -INPUT_DEADZONE ) )
            {
                g_Controllers[i].state.Gamepad.sThumbLX = 0;
                g_Controllers[i].state.Gamepad.sThumbLY = 0;
            }

            if( ( g_Controllers[i].state.Gamepad.sThumbRX < INPUT_DEADZONE &&
                  g_Controllers[i].state.Gamepad.sThumbRX > -INPUT_DEADZONE ) &&
                ( g_Controllers[i].state.Gamepad.sThumbRY < INPUT_DEADZONE &&
                  g_Controllers[i].state.Gamepad.sThumbRY > -INPUT_DEADZONE ) )
            {
                g_Controllers[i].state.Gamepad.sThumbRX = 0;
                g_Controllers[i].state.Gamepad.sThumbRY = 0;
            }
        }
```

3.2 Programación Dead zone

Separar ejes y descontar zona muerta.

```
//Zona muerta por cada eje
if (XBox.State.Gamepad.sThumbLX > INPUT_DEADZONE)
	XBox.State.Gamepad.sThumbLX -= INPUT_DEADZONE;
else if (XBox.State.Gamepad.sThumbLX < -INPUT_DEADZONE)
	XBox.State.Gamepad.sThumbLX += INPUT_DEADZONE;
else
	XBox.State.Gamepad.sThumbLX = 0;
```

Esto permite hacer movimiento horizontales y verticales.

3.2 Programación Analog signal filtering

This noise can sometimes cause the in-game behaviors controlled by the HID to appear jerky or unnatural. For this reason, many games fi lter the raw inputs coming from the HID.

A discrete first-order low-pass filter can be implemented by combining the current unfiltered input value with last frame's filtered input.

$$f(t) = (1-a)f(t-\Delta t) + au(t), \qquad a = \frac{\Delta t}{RC + \Delta t}.$$

where the parameter a is determined by the frame duration Δt and a filtering constant RC= Tiempo que tarda en alcanzar el 63%.

```
F32 lowPassFilter(      F32 unfilteredInput, F32 lastFramesFilteredInput,
                        F32 rc, F32 dt)
{
      F32 a = dt / (rc + dt);
      return (1 – a) * lastFramesFilteredInput + a * unfilteredInput;
}
```

3.2 Programación

Detecting Input Events

The easiest way to detect a change in button state is to keep track of the buttons' state bits as observed last frame and compare them.

We can use simple bit-wise operators to detect button-down and button-up events.

```
class ButtonState
{
 U32 m_buttonStates; // current states
 U32 m_prevButtonStates; // previous states
 U32 m_buttonDowns; // 1 = button pressed
 U32 m_buttonUps; // 1 = button released

 void DetectButtonUpDownEvents()
      {// First determine which bits have changed via XOR.
      U32 buttonChanges=m_buttonStates^ m_prevButtonStates;
      // Use AND to mask off only the bits that are DOWN.
      m_buttonDowns = buttonChanges & m_buttonStates;
      // Use AND-NOT to mask off only the bits that are UP.
      m_buttonUps = buttonChanges & (~m_buttonStates);
      }
      // ...
};
```

3.2 Programación Chords

A chord is a group of buttons that, when pressed at the same time, produce a unique behavior in the game. Here are a few examples:

- Super Mario Galaxy's start-up screen requires you to press the A and B buttons on the WiiMote together in order to start a new game.
- Pressing the 1 and 2 buttons on the WiiMote at the same time put it into Bluetooth discovery mode (no matter what game you're playing).
- The "grapple" move in many fighting games is triggered by a two-button combination.
- For development purposes, holding down both the left and right triggers on the DualShock 3 in Uncharted: Drake's Fortune allows the player character to fly anywhere in the game world, with *collisions turned off** .

3.2 Programación Chords

Detecting chords is quite simple in principle.

Some problems:

- Humans aren't perfect, and they often press one or more of the buttons in the chord slightly earlier than the rest.
  - You can introduce a delay between when an individual butt on-down event is seen and when it "counts" as a valid game event
- We must take care not to perform both the actions of the individual buttons and the action of chord when it is pressed.
  - You can design your button inputs such that a chord always does the actions of the individual buttons plus some additional action.
  - You can detect the chord when the buttons are pressed, but wait to trigger the effect until the buttons are released again.

3.2 Programación Sequences&Gesture Detection

A gesture is a sequence of actions performed via a HID by the human player over a period of time.

P.E: Rapid Button Tapping, Multibutton Sequence, Thumb Stick Rotation

Usually a sequence or gesture is only considered to be valid if it is performed within some maximum time-frame.

Gesture detection is generally **implemented by keeping a brief history** of the HID actions performed by the player:

- When the first component of the gesture is detected, it is stored in the history buffer, along with a time stamp indicating when it occurred.
- ....
- If the entire sequence is completed within the allotted time (i.e., the history buffer is filled), an event is generated telling the rest of the game engine that the gesture has occurred.

3.2 Programación Multiple HIDs

Managing Multiple HIDs for Multiple Players:Most game machines allow two or more HIDs to be attached for multiplayer games:

- Mapping controllers to players
- Robust to various exceptional conditions, such as the controller being accidentally unplugged or running out of batteries
- On systems with battery-operated HIDs, the game or the operating system is responsible for detecting low-battery conditions

Se deben poder gestionar varios tipos de HID a la vez:

- Controller: Mandos de Videojuegos.
- KeyState: Estado Teclado
- MouseState: Estado completo del ratón

3.2 Programación Cross-Platform

Many game engines are cross-platform. One way to handle HID inputs and outputs in such an engine would be to sprinkle conditional compilation directives all over the code, wherever interactions with the HID take place, as shown below. This is clearly not an ideal solution, but it does work.

```
#if TARGET_XBOX360
        if (ButtonsJustWentDown(XB360_BUTTONMASK_A))
#elif TARGET_PS3
        if (ButtonsJustWentDown(PS3_BUTTONMASK_TRIANGLE))
#elif TARGET_WII
        if (ButtonsJustWentDown(WII_BUTTONMASK_A))
#endif
{
// do something...
}
```

A **better** solution is to provide some kind of **hardware abstraction layer**, thereby insulating the game code from hardware-specific details.

3.2 Programación Hardware Abstraction Layer

For example, if our game is to ship on Xbox 360 and PS3, the layout of the controls (buttons, axes and triggers) on these two joypads are almost identical.

The controls have different ids on each platform, but we can come up with generic control ids that cover both types of joypad quite easily.

Our abstraction layer can translate between the raw control ids on the current target hardware into our abstract control indices.

```
enum AbstractControlIndex
{       // Start and back buttons
        AINDEX_START,               // Xbox 360 Start, PS3 Start
        AINDEX_BACK_PAUSE,      // Xbox 360 Back, PS3 Pause
        // Right "pad" of four buttons
        AINDEX_RPAD_DOWN,       // Xbox 360 A, PS3 X
        AINDEX_RPAD_UP,         // Xbox 360 Y, PS3 Triangle
        AINDEX_RPAD_LEFT,       // Xbox 360 X, PS3 Square
        AINDEX_RPAD_RIGHT,      // Xbox 360 B, PS3 Circle
....
```

3.2 Programación

Input Re-Mapping

Los volantes utilizan:

- el eje X para el volante
- el Y para los pedales
- los botones para las marchas.

3.2 Programación Input Re-Mapping

Many games allow the player some degree of choice with regard to the functionality of the various controls on the physical HID.

We assign **a simple table** which maps **each physical** or abstract control index **to** a **logical** function in the game.

Different controls produce different kinds of inputs.

- Analog axes may produce values ranging from –32,768 to 32,767, or from 0 to 255, or some other range.
- The states of all the digital buttons on a HID are usually packed into a single machine word.

Therefore, we must be careful to only permit control mappings that make sense.

3.2 Programación Input Re-Mapping

A reasonable set of classes for a standard console joypad and their normalized input values might be:

- **Digital buttons**. States are packed into a 32-bit word, one bit per button.
- **Unidirectional absolute axes** (e.g., triggers, analog buttons). Produce floating-point input values in the range [0, 1].
- **Bidirectional absolute axes** (e.g., joysticks). Produce floating-point input values in the range [–1, 1].
- **Relative axes** (e.g., mouse axes, wheels, track balls). Produce floating-point input values in the range [–1, 1] , where ±1 represents the maximum relative off set possible within a single game frame (i.e., during a period of 1/30 or 1/60 of a second).

3.2 Programación Context-Sensitive Controls

A single physical control can have different functions, depending on context.

> A simple example is the ubiquitous “use” button. If pressed while standing in front of a door, the “use” button might cause the character to open the door. If it is pressed while standing near an object, it might cause the player character to pick up the object, and so on.
>
> Another common example is a modal control scheme. When the player is walking around, the controls are used to navigate and control the camera. When the player is riding a vehicle, the controls are used to steer the vehicle, and the camera controls might be different as well.

Context-sensitive controls are reasonably straightforward to implement via **a state machine**. Depending on what state we’re in, a particular HID control may have a different purpose.

Another related concept is that of control ownership.

Some game engines introduce the concept of a logical device.

3.2 Programación Disabling Inputs

In most games, it is sometimes necessary to disallow the player from controlling his or her character.

> For example, when the player character is involved in an in-game cinematic, we might want to disable all player controls temporarily; or when the player is walking through a narrow doorway, we might want to temporarily disable free camera rotation.

Two approaches:

- A heavy-handed approach is to use a bit mask to disable individual controls on the input device itself. We must be particularly cautious when disabling controls, however. If we forget to reset the disable mask, the game can get itself into a state where the player looses all control forever, and must restart the game.
- A better approach is probably to put the logic for disabling specific player actions or camera behaviors directly into the player or camera code itself.

3.2 Programación Implementación Multicapa

- Realmente son tres capas software:
  - **Capa HID-Hardware:** Encargada de leer/escribir el dispositivo.
    - Debe implementarse para cada dispositivo.
    - Puede depender de librerías particulares.
    - Puede detectar y eliminar zonas muertas.
    - Puede soportar la calibración y eliminar offsets
  - **Capa HID-Abstracta:** Encargada de contener la información en un formato normalizado, independiente del dispositivo y de la aplicación.
    - Puede asignar/configurar un mapeo de ejes.
    - Reconocer gestos y combinaciones
    - Debe ser reutilizable para otros proyectos
  - **Capa Aplicación-HID:** Encargada de utilizar la información del HID en la aplicación. Como tiene acceso al estado se encarga de:
    - Aplicar las acciones
    - Acciones dependientes del contexto, como habilitar/deshabilitar controles.