



The Witch Shiinim

THE FORBIDDEN MAGIC STAFF



Game Design Document
MiinishBot

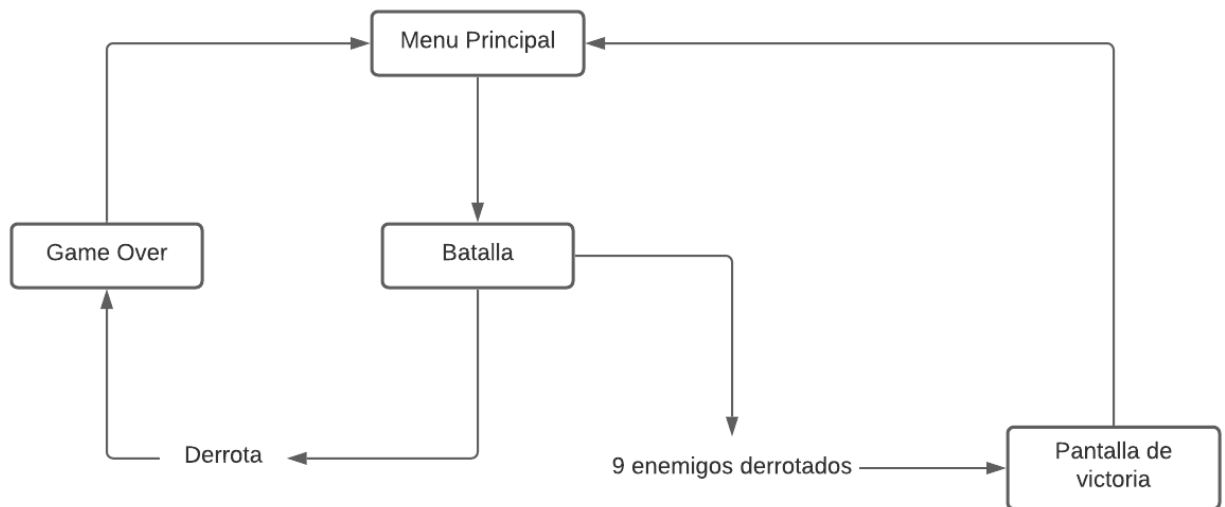
Contenido

Aplicación gráfica y artística	4
Flujo de programa.....	4
Menú	5
Script del menú	5
Escena de batalla.....	6
Script de la escena de pantalla	7
Unidades.....	14
Script Unidad.....	14
Interfaz de batalla	18
Script de la interfaz de batalla.....	18
Panel de Resultados	22
Script del panel de resultados	22
Game Over	24
Script de Game Over.....	24
Final.....	25
Script de victoria	25
Diseño de escenarios y accesorios.....	26
Terreno del menú principal.....	26
Terreno del campo de batalla.....	26
Objetos del terreno	27
Jugador.....	28
Enemigos.....	29
Programa de comunicaciones y háptico	32
Comunicación con sensores y elementos de salida.....	32
Componentes.....	32
Diagrama	33
Protocolo de comunicación entre la PC y el háptico.	34
Protocolo UART (recepción-transmisión asíncrona universal).....	34
Protocolo I2C	35
Programación en Arduino	36
Script.....	36

Programación en Unity	40
System.IO.Port	40
Script Arduino.cs	41
Script SistemaBatalla.css	42
Entradas generadas por el háptico	44
Movimiento del háptico	44
Salidas recibidas por el háptico	45
Encendido de leds	45
Vibración del motor	46

Aplicación gráfica y artística

Flujo de programa



Menú



El juego comienza con el menú principal, en él se pueden observar al personaje principal (La bruja), el nombre del juego, y dos botones; el primero para comenzar el juego y el segundo para cerrar el juego.

Script del menú

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour
{
    public void Jugar()
    {
        SceneManager.LoadScene("Batalla");
    }

    public void Salir()
    {
        Application.Quit();
    }
}
```

Escena de batalla



El juego tiene un sistema de combate por turnos, tanto el personaje del jugador como el enemigo tienen una barra de información donde se despliega la información básica de cada personaje como el nombre, nivel y vida actual.

En el caso del enemigo se mostrará una imagen dependiendo el tipo que este sea, y en la barra del jugador se mostrará una imagen del personaje que se esté usando.

Solo la barra del jugador despliega cuanta experiencia tiene y cuanta necesita para avanzar al siguiente nivel.

En la parte de abajo de la pantalla se tienen 4 iconos de tipos, estos indican hacia donde hay que mover el háptico para realizar un hechizo de cierto tipo, y a un lado de estos se encuentra un panel de mensajes, donde se redactará todo lo que esté sucediendo en el juego, suele indicar instrucciones para el jugador, que hechizo se realizó, si el enemigo ataca, si se sube de nivel, si se derrotó al enemigo o si fuimos derrotados.

Script de la escena de pantalla

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public enum BattleState
{
    INICIO,
    ELEGIRHECHIZO,
    TURNOJUGADOR,
    TURNOENEMIGO,
    GANA,
    PIERDE,
    RESULTADOS,
    SIGNIVEL,
    FINAL
}

public class SistemaBatalla : MonoBehaviour
{
    public GameObject jugadorPrefab;
    public List<GameObject> enemigoPrefab;

    public Transform spawnJugador;
    public Transform spawnEnemigo;

    Unidad entiWitch;
    Unidad enemigoUnidad;

    public Text mensajeTexto;

    public BatallaHUD jugadorHUD;
    public BatallaHUD enemigoHUD;
    public ResultadosHUD resultadoHUD;

    public Arduino arduino;
    float angleX;
    float angleY;

    public BattleState estado;
    Quaternion rotInicial;

    int fase;
    string tipoAtk;

    void Start()
    {
```

```

        estado = BattleState.INICIO;
        StartCoroutine(ConfigBatalla());
        rotInicial = entiWitch.transform.rotation;
        fase = 0;
    }

    void Update()
    {
        jugadorHUD.llenarExp(entiWitch);
        jugadorHUD.llenarVida(entiWitch);
        enemigoHUD.llenarVida(enemigoUnidad);

        angleX = arduino.leerX();
        angleY = arduino.leerY();
        if (estado == BattleState.ELEGIRHECHIZO)
        {
            if (angleX <= 70 && angleX >= 45)
            {
                BotonAtkArriba();
            }
            else if (angleX <= -45 && angleX >= -70)
            {
                BotonAtkAbajo();
            }
            else if (angleY <= 70 && angleY >= 45)
            {
                BotonAtkDer();
            }
            else if (angleY <= -45 && angleY >= -70)
            {
                BotonAtkIzq();
            }
        }
    }

    IEnumerator ConfigBatalla()
    {
        Debug.Log("Configurando Batalla");
        Debug.Log("Fase: " + fase);
        if (fase == 9)
            SceneManager.LoadScene("Victoria");

        if (estado == BattleState.INICIO)
        {
            GameObject jugadorGo = Instantiate(jugadorPrefab, spawnJugador);
            entiWitch = jugadorGo.GetComponent<Unidad>();
        }

        GameObject enemigoGo = Instantiate(enemigoPrefab[fase], spawnEnemigo);
        //Instanciar un enemigo al spawn
    }

```



```

        enemigoUnidad = enemigoGo.GetComponent<Unidad>();

        mensajeTexto.text = "Ha aparecido " + "<b><color=#b40404>" + enemigoUnidad.nombre + "</color></b>" + ".";

        entiWitch.configExpSig();

        jugadorHUD.configHUD(entiWitch);

        enemigoHUD.configHUD(enemigoUnidad);

        resultadoHUD.configResultado(entiWitch.nombre, entiWitch.nivel, entiWitch.vidaMax, entiWitch.atk, entiWitch.def, entiWitch.vel);

        yield return new WaitForSeconds(2f);
        resultadoHUD.mostrar(false);

        estado = BattleState.ELEGIRHECHIZO;
        ElegirHechizo();
    }

    IEnumerator AtaqueJugador()
    {
        yield return new WaitForSecondsRealtime(4);

        bool estaMuerto = enemigoUnidad.RecibeDanioJ(entiWitch.atk, tipoAtk, enemigoUnidad.tipo.ToString());

        enemigoHUD.actVida(enemigoUnidad.vidaActual, enemigoUnidad.vidaMax);
        mensajeTexto.text = "Haz atacado a " + "<b><color=#b40404>" + enemigoUnidad.nombre + "</color></b>";

        yield return new WaitForSeconds(2f);

        if (estaMuerto)
        {
            StartCoroutine(MuereEnemigo());
            estado = BattleState.RESULTADOS;
            FinBatalla();
        }
        else
        {
            estado = BattleState.TURNOENEMIGO;
            StartCoroutine(TurnoEnemigo());
        }
    }

    IEnumerator MuereEnemigo()
    {
        Destroy(enemigoUnidad.gameObject);
    }

```

```

        yield return new WaitForSeconds(2f);
    }

    public void FinBatalla()
    {
        if (estado == BattleState.RESULTADOS)
        {
            mensajeTexto.text = "<b>Haz ganado la batalla!</b>";
            StartCoroutine(Resultados());
        }
        else if (estado == BattleState.PIERDE)
        {
            entiWitch.anim.SetTrigger(Animator.StringToHash("Muerte"));
            mensajeTexto.text = "<b>Haz sido derrotado.</b>";
            StartCoroutine(GameOver());
        }
    }

    public void Gana()
    {
        fase ++;
        StartCoroutine(ConfigBatalla());
    }

    IEnumerator GameOver()
    {
        yield return new WaitForSeconds(5f);
        SceneManager.LoadScene("Game Over");
    }

    IEnumerator TurnoEnemigo()
    {
        entiWitch.transform.rotation = rotInicial;
        mensajeTexto.text = "<b><color=#b40404>" + enemigoUnidad.nombre + "</color></b> ataca!";

        yield return new WaitForSeconds(1f);

        bool estaMuerto = entiWitch.RecibeDanioE(enemigoUnidad.atk);
        arduino.vibrarOn();

        jugadorHUD.actVida(entiWitch.vidaActual, entiWitch.vidaMax);

        yield return new WaitForSeconds(1f);

        if (estaMuerto)
        {
            estado = BattleState.PIERDE;
            FinBatalla();
        }
    }

```

```

        else
        {
            estado = BattleState.ELEGIRHECHIZO;
            ElegirHechizo();
        }
    }

    void ElegirHechizo()
    {
        mensajeTexto.text = "<b>Realiza un hechizo</b>";
    }

    public void BotonAtkArriba() //Ataque de Luz
    {
        tipoAtk = "luz";
        if (estado != BattleState.ELEGIRHECHIZO)
            return;

        arduino.amarilloOn();
        mensajeTexto.text = "Has realizado un <b><color=#f4be8a>hechizo de luz</color></b>";

        estado = BattleState.TURNOJUGADOR;

        entiWitch.anim.SetTrigger(Animator.StringToHash("Atk arriba"));
        entiWitch.transform.rotation = rotInicial;

        StartCoroutine(AtaqueJugador());
    }
    public void BotonAtkAbajo() //Ataque de Agua
    {
        tipoAtk = "agua";
        if (estado != BattleState.ELEGIRHECHIZO)
            return;

        arduino.blancoOn();
        mensajeTexto.text = "Has realizado un <b><color=#5b5bff>hechizo de agua</color></b>";

        estado = BattleState.TURNOJUGADOR;

        entiWitch.anim.SetTrigger(Animator.StringToHash("Atk abajo"));

        entiWitch.transform.rotation = rotInicial;

        StartCoroutine(AtaqueJugador());
    }
    public void BotonAtkIzq() //Ataque de planta
    {
        tipoAtk = "planta";

```

```

        if (estado != BattleState.ELEGIRHECHIZO)
            return;

        arduino.verdeOn();
        mensajeTexto.text = "Has realizado un <b><color=#008000>hechizo de planta</color></b>";

        estado = BattleState.TURNOJUGADOR;

        entiWitch.anim.SetTrigger(Animator.StringToHash("Atk izquierda"));
        entiWitch.transform.rotation = rotInicial;

        StartCoroutine(AtaqueJugador());
    }
    public void BotonAtkDer() //Ataque de fuego
    {
        tipoAtk = "fuego";
        if (estado != BattleState.ELEGIRHECHIZO)
            return;

        arduino.rojoOn();
        mensajeTexto.text = "Has realizado un <b><color=#ff3232>hechizo de fuego</color></b>";

        estado = BattleState.TURNOJUGADOR;

        entiWitch.anim.SetTrigger(Animator.StringToHash("Atk derecha"));

        entiWitch.transform.rotation = rotInicial;

        StartCoroutine(AtaqueJugador());
    }

    IEnumerator Resultados()
    {
        bool subeNivel;

        yield return new WaitForSecondsRealtime(2);
        mensajeTexto.text = "Haz obtenido <b>" + enemigoUnidad.exp + "</b> puntos de experiencia.";

        entiWitch.exp += enemigoUnidad.exp; //Se suma la exp al jugador sin importar que pueda subir de nivel

        if (entiWitch.exp >= entiWitch.expSig)
        {
            subeNivel = true;
            jugadorHUD.actExp(entiWitch.exp, entiWitch.expSig);
        }
        else
    
```

```

    {
        subeNivel = false;
        jugadorHUD.actExp(entiWitch.exp, entiWitch.expSig); //Se actualizo
la exp en pantalla
        yield return new WaitForSecondsRealtime(2);
    }

    while (subeNivel == true) //Se ejecuta mientras pueda seguir subiendo
de niveles
    {
        yield return new WaitForSecondsRealtime(1);

        mensajeTexto.text = "Has subido al nivel <b>" + (entiWitch.nivel +
1) + "</b>.";
        yield return new WaitForSeconds(1);

        resultadoHUD.mostrar(true); //Se muestra panel de resultados sin c
ambiar

        yield return new WaitForSeconds(2);

        entiWitch.nuevosStats(); //Se obtien los buffs
        resultadoHUD.actualizar(entiWitch.nombre, entiWitch.nivel, entiWit
ch.vidaMax, entiWitch.atk, entiWitch.def, entiWitch.vel, entiWitch.buffVida, e
ntiWitch.buffAtk, entiWitch.buffDef, entiWitch.buffVel);

        yield return new WaitForSeconds(2);

        subeNivel = entiWitch.subirNv(); //Aqui se sube el nivel y los sta
ts

        jugadorHUD.actExp(entiWitch.exp, entiWitch.expSig); //Se actualiza
la exp

        resultadoHUD.configResultado(entiWitch.nombre, entiWitch.nivel, en
tiWitch.vidaMax, entiWitch.atk, entiWitch.def, entiWitch.vel); //Los nuevos da
tos se sobre escriben en el panel
        yield return new WaitForSecondsRealtime(3);

        jugadorHUD.configHUD(entiWitch);
        resultadoHUD.mostrar(false); //Se esconde el panel de resultados
        yield return new WaitForSeconds(1);
    }
    estado = BattleState.SIGNIVEL;
    Gana();
}
}

```

Unidades



Las unidades son aquellos personajes que almacenan propiedades y métodos, en este caso son los enemigos y el personaje del jugador, ambos tipos de unidades almacenan nombre y stats como vida, experiencia, ataque, defensa, etc.

También almacenan métodos como subir de nivel, recibir daño o aumentar stats.

Todo esto se encuentra en el script de Unidad que almacena dicha clase.

Script Unidad

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public enum listaTipo
{
    jugador,
    agua,
    fuego,
    planta,
    tierra,
    oscuridad,
    luz
}

public class Unidad : MonoBehaviour
{
    [Header("Info")]
    //nombre de la entidad
    public string nombre;
    //Stats
    [SerializeField]
    public listaTipo tipo;
```

```

[Header("Stats")]
public int nivel;
public int exp; //Experiencia actual
public int expSig {get; private set;} //Experiencia necesaria para el siguiente nivel

[Space(10)]
public int vidaMax;
public int vidaActual {get; private set;}
public int atk;
public int def;
public int vel;

//Bufs a usar en el panel de resultados
[HideInInspector]
public int buffVida;
[HideInInspector]
public int buffAtk;
[HideInInspector]
public int buffDef;
[HideInInspector]
public int buffVel;
[Space(20)]
//Aqui se guarda el Animator Controller
public Animator anim;

void Awake()
{
    vidaActual = vidaMax;
}

void Start()
{
    anim = GetComponent<Animator>();
}

public void configExpSig()
{
    float buff;
    buff = Mathf.Pow(nivel + 1, 3);
    buff *= 4;
    buff /= 5;
    expSig = (int) buff;
}

//Funcion que se ejecuta al momento de atacar
public bool RecibeDanioJ(int dmg, string tipoAtk, string tipoEnemigo)
{
    if (tipoAtk == tipoEnemigo)
    {
        dmg = 0;
    }
}

```

```

        else if(tipoAtk == "agua" && tipoEnemigo == "fuego")
        {
            dmg *= 2;
        }
        else if(tipoAtk == "planta" && tipoEnemigo == "agua")
        {
            dmg *= 2;
        }
        else if(tipoAtk == "fuego" && tipoEnemigo == "planta")
        {
            dmg *= 2;
        }
        else if(tipoAtk == "luz" && tipoEnemigo == "oscuridad")
        {
            dmg = 9999;
        }
        else if(tipoAtk == "agua" && tipoEnemigo == "tierra")
        {
            dmg *= 2;
        }
        vidaActual -= dmg;

        //Si esta muerto es true
        if (vidaActual <= 0)
        {
            vidaActual = 0;
            return true;
        }
        else
            return false;
    }

    public bool RecibeDanioE(int dmg)
    {
        vidaActual -= dmg;

        //Si esta muerto es true
        if (vidaActual <= 0)
        {
            vidaActual = 0;
            return true;
        }
        else
            return false;
    }

    public void nuevosStats()
    {
        buffVida = Random.Range(5,10);
    }

```



```
        buffAtk = Random.Range(5,10);
        buffDef = Random.Range(5,10);
        buffVel = Random.Range(5,10);
    }

    public bool subirNv() //Regresa false si ya no puede subir de nivel y true
    si puede seguir subiendo
    {
        nivel ++;
        exp -= expSig;
        configExpSig(); //Calcula la exp necesaria para el siguiente nivel

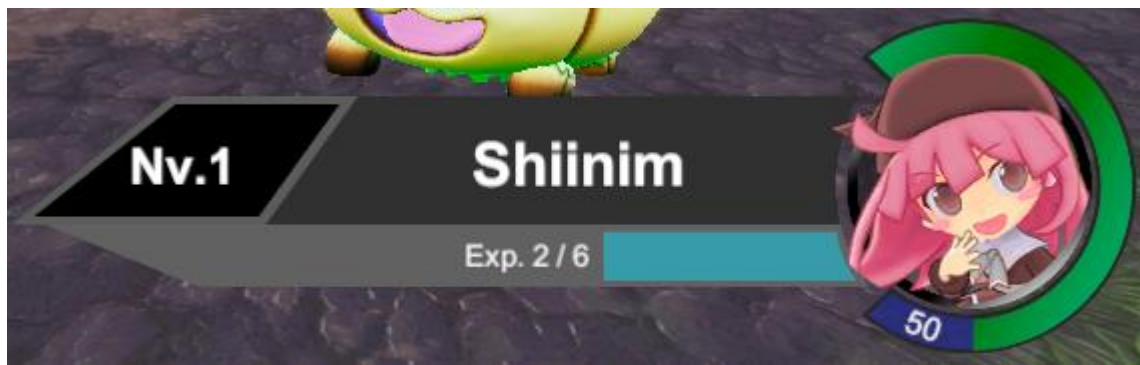
        vidaMax += buffVida;
        atk += buffAtk;
        def += buffDef;
        vel += buffVel;

        vidaActual = vidaMax; //Si sube de nivel su salud se recupera por comp
    }
}

leto

    if(exp >= expSig)
        return true; //Puede subir otro nivel
    else
        return false; //Ya no puede subir otro nivel
}
```

Interfaz de batalla



Estas barras de información comparten un script, el funcionamiento de este es almacenar las funciones que actualizarán la información de las barras a través de todo el juego, así como asignar el icono de los tipos de los enemigos.

Script de la interfaz de batalla

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class BatallaHUD : MonoBehaviour
{
    public Text txtNombre;
    public Text numVida;
    public Text numNivel;
    public Text numExp;
    public Image imgTipo;
    public List<Sprite> Tipos;
    //Nuevas
    public Image imgVidaC;
    public Image imgExpBar;
    float velocidad;

    public void configHUD(Unidad unidad)
    {
        txtNombre.text = unidad.nombre;
```

```
numVida.text = "" + unidad.vidaActual;

numNivel.text = "Nv." + unidad.nivel;

switch(unidad.tipo)
{
    case listaTipo.jugador:
        imgTipo.sprite = Tipos[0];
        break;

    case listaTipo.agua:
        imgTipo.sprite = Tipos[1];
        break;

    case listaTipo.fuego:
        imgTipo.sprite = Tipos[2];
        break;

    case listaTipo.planta:
        imgTipo.sprite = Tipos[3];
        break;

    case listaTipo.tierra:
        imgTipo.sprite = Tipos[4];
        break;

    case listaTipo.oscuridad:
        imgTipo.sprite = Tipos[5];
        break;

    case listaTipo.luz:
        imgTipo.sprite = Tipos[6];
        break;
}

//Solo el jugador muestra su exp
if(unidad.tipo == 0)
{
    numExp.text = "Exp. " + unidad.exp + " / " + unidad.expSig;
    if (unidad.exp != 0)
    {
        float buff = (float) unidad.exp / unidad.expSig;
        imgExpBar.fillAmount = buff;
    }
    else
        imgExpBar.fillAmount = 0;
}

//El enemigo y el jugador muestran su vida
if(unidad.vidaActual != 0)
```

```

        {
            float buff = (float) unidad.vidaActual / unidad.vidaMax;
            buff = 0.677f * buff;
            buff += 0.216f;
            imgVidaC.fillAmount = buff;
        }
        else
            imgVidaC.fillAmount = 0.215f;
    }

    public void llenarExp(Unidad unidad)
    {
        velocidad = 3f * Time.deltaTime;

        if(unidad.tipo == 0)
        {
            velocidad = 3f * Time.deltaTime;
            numExp.text = "Exp. " + unidad.exp + " / " + unidad.expSig;
            if (unidad.exp != 0)
            {
                float buff = (float) unidad.exp / unidad.expSig;
                imgExpBar.fillAmount = Mathf.Lerp(imgExpBar.fillAmount, buff, velocidad);
            }
            else
                imgExpBar.fillAmount = 0;
        }
    }

    public void llenarVida(Unidad unidad)
    {
        velocidad = 3f * Time.deltaTime;
        if(unidad.vidaActual != 0)
        {
            float buff = (float) unidad.vidaActual / unidad.vidaMax;
            buff = 0.677f * buff;
            buff += 0.216f;
            imgVidaC.fillAmount = Mathf.Lerp(imgVidaC.fillAmount, buff, velocidad);
        }
        else
            imgVidaC.fillAmount = 0.215f;
    }

    public void actVida(int vidaActual, int vidaMax)
    {
        numVida.text = "" + vidaActual;
    }

    public void actExp(int exp, int expSig)

```

```
{  
    numExp.text = "Exp. " + exp + " / " + expSig;  
}  
}
```

Panel de Resultados



Este panel semitransparente solo se muestra cuando el jugador sube de nivel, en él se muestra el nombre del personaje, sus estadísticas actuales, lo que subirán las estadísticas y las estadísticas después de subir de nivel.

Script del panel de resultados

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

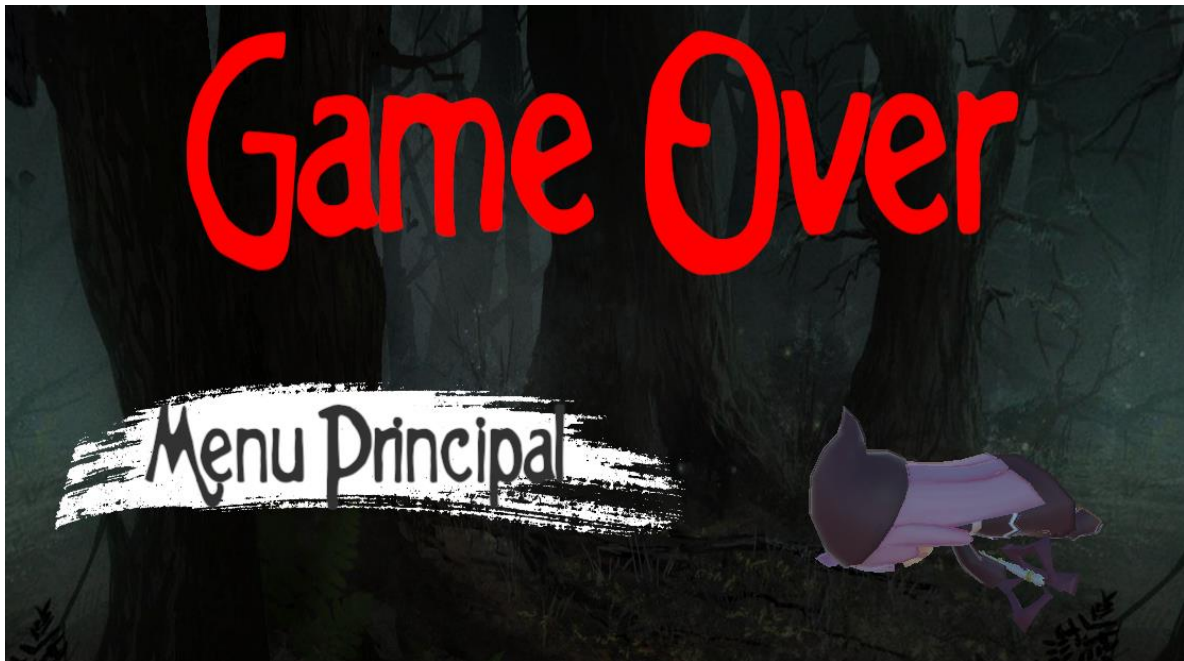
public class ResultadosHUD : MonoBehaviour
{
    public GameObject panel;
    public Text txtResultado;
    Vector3 iniPos;

    void start()
    {
        iniPos = panel.transform.position;
    }
    public void configResultado(string nombre, int nivel, int vida, int atk, int def, int vel) //Se ponen los valores iniciales de Shiinim al comenzar la pelea
    {
        txtResultado.text = nombre + "  Nv." + nivel + "\n \n"
        + "Vida: " + vida + "\n"
        + "Ataque: " + atk + "\n"
        + "Defensa: " + def + "\n"
        + "Velocidad: " + vel;
    }

    public void mostrar(bool visible) //El panel de resultados se hace visible
    {
        panel.SetActive(visible);
    }
}
```

```
public void actualizar(string nombre, int nivel, int vida, int atk, int def,
int vel, int buffVida, int buffAtk, int buffDef, int buffVel) //Se muestra la
info actual y lo que se le va a sumar
{
    txtResultado.text = nombre + "  Nv." + nivel + "\t<color=#f3d352>+ 1</colo
r>" + "\n \n"
    + "Vida: " + vida + "\t<color=#f3d352>+ " + buffVida + "</color>\n"
    + "Ataque: " + atk + "\t<color=#f3d352>+ " + buffAtk + "</color>\n"
    + "Defensa: " + def + "\t<color=#f3d352>+ " + buffDef + "</color>\n"
    + "Velocidad: " + vel + "\t<color=#f3d352>+ " + buffVel + "</color>";
}
}
```

Game Over



Cuando la vida del jugador llegue a 0 se contará como derrota y se mostrará la pantalla de Game Over, en esta pantalla se mostrará un bosque oscuro de fondo, al personaje del jugador tirado en él, unas letras grandes que dicen Game Over y un botón que nos llevará devuelta al menú principal.

Script de Game Over

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameOver : MonoBehaviour
{
    void Menu()
    {
        SceneManager.LoadScene("Menu Principal");
    }
}
```


Final



Al acabar con todos los enemigos y derrotar al jefe final se nos mostrará la pantalla del final del juego, en ella se nos muestra el título del juego, nuestro personaje principal, un panel con información del equipo de hápticos y un botón que nos llevará de vuelta al menú principal.

Script de victoria

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

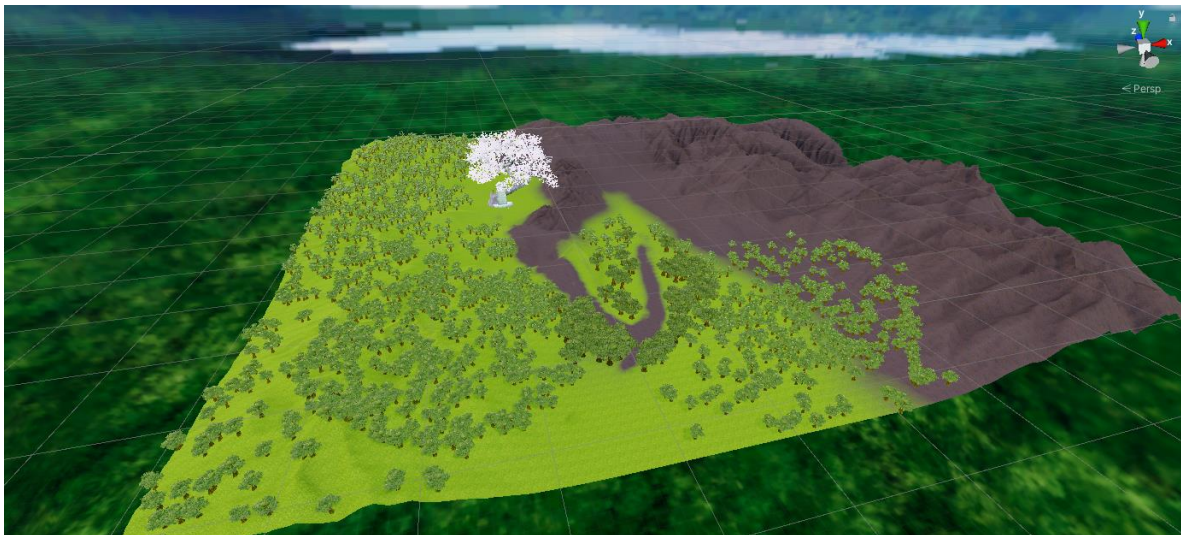
public class Victoria : MonoBehaviour
{
    public void Continuar()
    {
        SceneManager.LoadScene("Menu Principal");
    }
}
```

Diseño de escenarios y accesorios

Terreno del menú principal



Terreno del campo de batalla



Cada terreno pertenece a una escena diferente, pero ambos comparten características como textura del terreno, arboles, pasto y ambiente ya que simulan ser un frondoso bosque.

Objetos del terreno



Este árbol de cerezo se encuentra en el terreno de la batalla, se encuentra al fondo de la vista del jugado así que casi no se aprecia con claridad.



Esta textura se usa como detalle del terreno, sirve para dar la impresión de que el suelo esta cubierto por pasto grande.



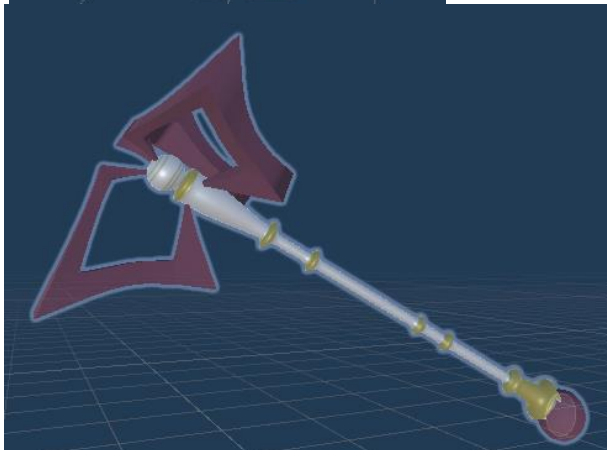
Este es el modelo de un árbol, y hay un gran número de ellos en ambos terrenos, de esta forma se puede crear un ambiente parecido al de un bosque.

Jugador

Los modelos mostrados del jugador se instancian en el script de sistema de batalla.



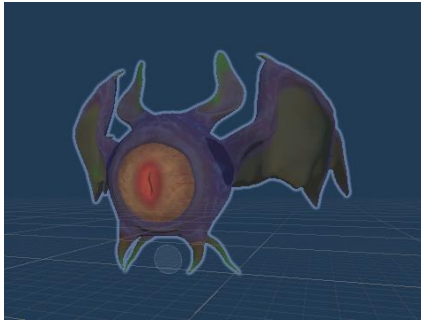
El personaje principal que usa el jugador tiene diferentes animaciones dependiendo del hechizo o acción realizados.



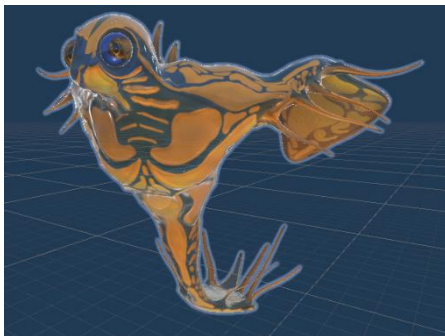
Este báculo es el que usa el personaje principal como su arma predeterminada, lo tiene sostenido en su brazo derecho.

Enemigos

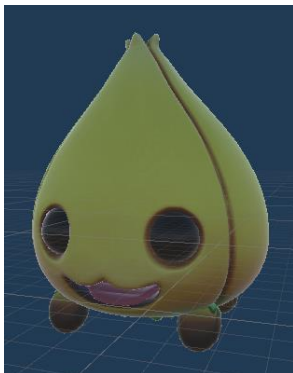
Los modelos mostrados de enemigos se instancian en el script de sistema de batalla.



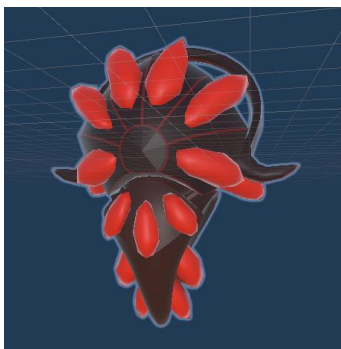
Enemigo con aspecto de murciélago, pero con solo un ojo, se mantiene volando en el escenario y es de tipo agua.



Esta criatura de color amarillo tiene rasgos de criaturas acuáticas por lo que su tipo es agua.



Tiene forma de cebolla por lo que su tipo es planta, de todos lo enemigos este es el más débil.



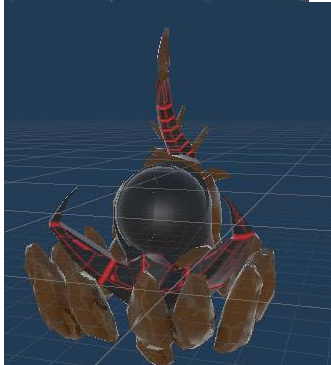
Este es el último enemigo de todos, tiene un aspecto robótico, durante el combate abre su armadura para mostrar su ojo, es de tipo oscuro.



Una planta carnívora en una maceta, su tipo es planta.



Esta criatura es un champiñón de color rojo que tiene una cara, su tipo es planta.



Esta criatura de aspecto robótico es similar al último enemigo, esta rodeado de rocas por lo que su tipo es de tierra.



Este roedor tiene una cara muy curiosa, parece como si hubiera sido pintada con marcadores, tiene cola y orejas de hojas de color verde o naranja, y el cuerpo color negro, su tipo es fuego.



Esta criatura cuadrúpeda tiene rasgos de perro y elefante, aunque sus orejas puntiagudas recuerdan a las de un duende, tiene una pequeña trompa, seis ojos color blanco y piel naranja, su tipo es tierra.

Programa de comunicaciones y háptico

Comunicación con sensores y elementos de salida.

Componentes



Arduino Nano: Arduino Nano es una placa pequeña, completa y compatible con placas de pruebas basada en la ATmega328P lanzada en 2008. Ofrece la misma conectividad y especificaciones de la placa Arduino Uno en un factor de forma más pequeño.



Leds: Un diodo emisor de luz o led es una fuente de luz constituida por un material semiconductor dotado de dos terminales.

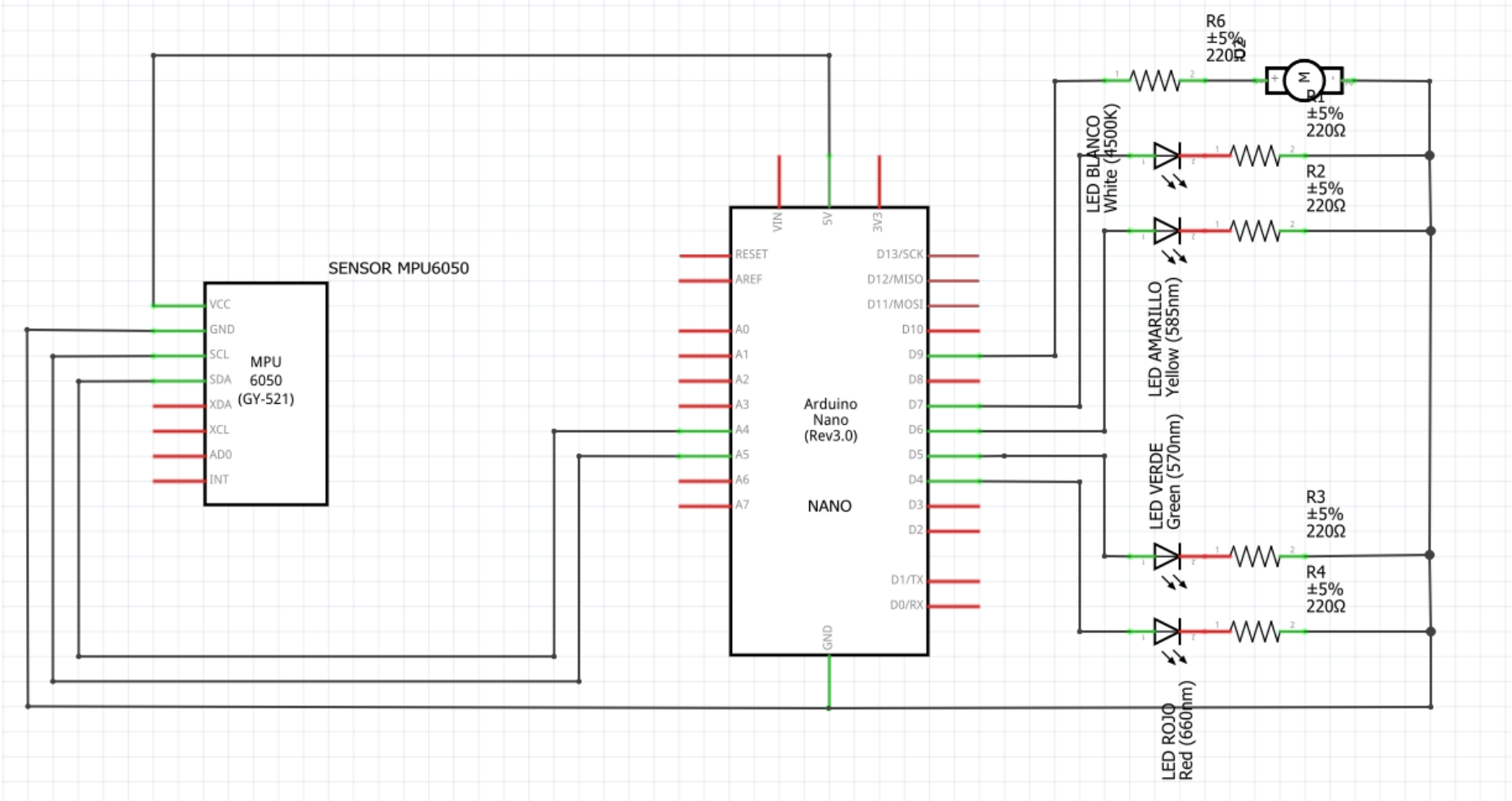


Motor de vibración: De formato plano muy pequeño, comienza a agitarse con solo 3V, llevan dos cables para aplicar tensión. Lleva pre-aplicado un adhesivo de doble cara, para que se adhiera a casi cualquier superficie.



MPU6050: Mide el movimiento en 6 grados de libertad, combinando un giroscopio de 3 ejes y un acelerómetro de 3 ejes en un mismo chip. Integra un DMP (Procesador digital de movimiento) capaz de realizar complejos algoritmos de captura de movimiento de 9 ejes.

Diagrama



Protocolo de comunicación entre la PC y el háptico.

Protocolo UART (recepción-transmisión asíncrona universal)

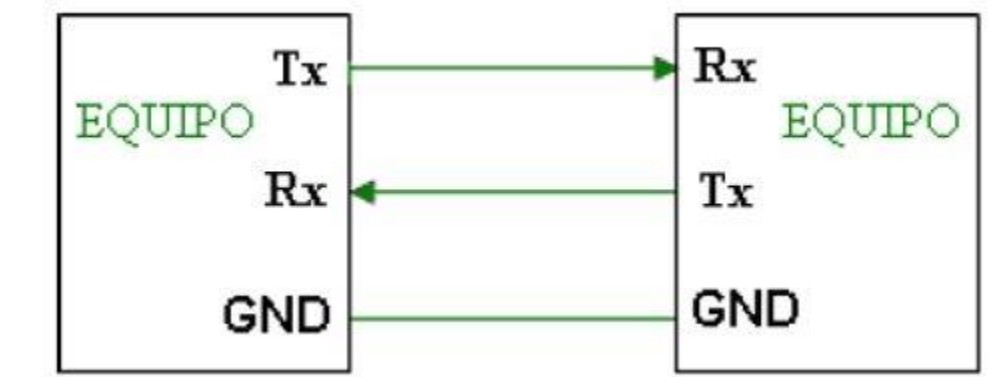
El UART está integrado en el Arduino UNO.

Usa una línea de datos simple para transmitir y otra para recibir datos.

Comúnmente, 8 bits de datos son transmitidos de la siguiente forma: un bit de inicio, a nivel bajo, 8 bits de datos y un bit de parada a nivel alto.

UART se diferencia de SPI y I2C en que es asíncrono y los otros están sincronizados con señal de reloj. La velocidad de datos UART está limitado a 2Mbps.

En el puerto serie tenemos dos líneas, Tx y Rx. Estas líneas se deben cruzar para comunicar dos dispositivos, es decir, el Tx del dispositivo 1 debe conectarse al Rx del dispositivo 2. El Tx del dispositivo 2 debe conectarse al Rx del dispositivo 1.



El UART se encargará de leer datos cuando lleguen, generar y gestionar interrupciones, enviar datos y gestionar los tiempos de bit. En general la UART se encarga de hacer todo el trabajo.

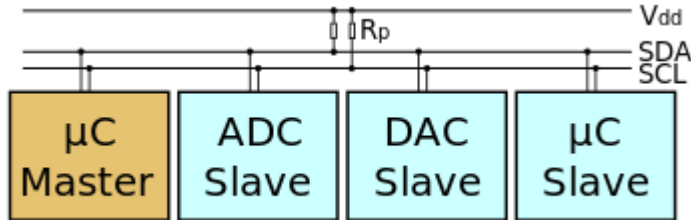
El Arduino se conecta a nuestro ordenador a través del puerto USB, pero el puerto USB se debe conectar al microcontrolador a través del puerto serie.

En el Arduino usamos el puerto USB para conectarnos al puerto Serie (UART) para comunicarnos durante la ejecución del programa.

El puerto serie conectado al USB lo usamos como puerto de consola o puerto de debug.

Protocolo I2C

MPU6050 se comunica con el protocolo I2C



I2C es un protocolo síncrono. I2C usa solo 2 cables, uno para el reloj (SCL) y otro para el dato (SDA). Esto significa que el maestro y el esclavo envían datos por el mismo cable, el cuál es controlado por el maestro, que crea la señal de reloj. I2C no utiliza selección de esclavo, sino direccionamiento.

Dos o más señales a través del mismo cable pueden causar conflicto, y ocurrirían problemas si un dispositivo envía un 1 lógico al mismo tiempo que otro envía un 0. Por tanto, el bus es “cableado” con dos resistencias para poner el bus a nivel alto, y los dispositivos envían niveles bajos. Si quieren enviar un nivel alto simplemente lo comunican al bus.

Programación en Arduino

Script

En el script del programa Arduino se establecerán las acciones a ejecutar con los componentes.

- Encendido y apagado de todos los leds.
- Encendido y apagado del motor de vibración.
- Obtención de valores del MPU6050.
- La conversión de los valores del MPU6050 en ángulos.
- El envío de estos ángulos a Unity.



```
PruebaUnity | Arduino 1.8.13
File Edit Sketch Tools Help

PruebaUnity

#include<Wire.h>

int data;

const int motor=9;
const int led_rojo=4;
const int led_verde=5;
const int led_amarillo=6;
const int led_blanco=7;
const int MPU=0x68;//dirección I2C de la IMU
int16_t Gx,Gy,Gz,Temp,Ax,Ay,Az;
// Ratios de conversion especificados en la documentación
// Deberemos dividir los valores que nos dé el Giroscopio y el
// Acelerómetro entre estas constantes para obtener un valor
// coherente. RAD_A_DEG es la conversión de radianes a grados.
#define A_R 16384.0 // aceleracion
#define G_R 131.0 // giroscopo
//Ángulos
float Acc[2];
float GyY[2];
float Angle[2];

void setup() {
  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B); // PWM_MGMT_1 register
  Wire.write(0);
  Wire.endTransmission(true);
  Serial.begin(9600);
  pinMode(motor,OUTPUT);
  pinMode(led_rojo,OUTPUT);
  pinMode(led_verde,OUTPUT);
  pinMode(led_amarillo,OUTPUT);
  pinMode(led_blanco,OUTPUT);
}

void loop() {
  GetMpuValue(MPU);
  //Se calculan los ángulos Y, X respectivamente.
  Acc[1] = atan(-1*(Ax/A_R)/sqrt(pow((Ay/A_R),2) + pow((Az/A_R),2)))*RAD_TO_DEG;
  Acc[0] = atan((Ay/A_R)/sqrt(pow((Ax/A_R),2) + pow((Az/A_R),2)))*RAD_TO_DEG;
  //Calculo del angulo del Giroscopio
  GyY[0] = Gx/G_R;
  GyY[1] = Gy/G_R;
  //Aplicar el Filtro Complementario
  Angle[0] = 0.98 *(Angle[0]+GyY[0]*0.010) + 0.02*Acc[0];
  Angle[1] = 0.98 *(Angle[1]+GyY[1]*0.010) + 0.02*Acc[1];
  if(Serial.available()){
  }
}

1 Arduino Nano, ATmega328P (Old Bootloader) on COM3
```

```
PruebaUnity | Arduino 1.8.13
File Edit Sketch Tools Help

void loop() {

  GetMpuValue(MPU);
  //Se calculan los ángulos Y, X respectivamente.
  Acc[1] = atan(-1*(Ax/A_R)/sqrt(pow((Ay/A_R),2) + pow((Az/A_R),2))) *RAD_TO_DEG;
  Acc[0] = atan((Ay/A_R)/sqrt(pow((Ax/A_R),2) + pow((Az/A_R),2))) *RAD_TO_DEG;
  //Calculo del angulo del Giroscopio
  GyY[0] = Gx/G_R;
  GyY[1] = Gy/G_R;
  //Aplicar el Filtro Complementario
  Angle[0] = 0.98 * (Angle[0]+GyY[0]*0.010) + 0.02*Acc[0];
  Angle[1] = 0.98 * (Angle[1]+GyY[1]*0.010) + 0.02*Acc[1];
  if (Serial.available()){
    data=Serial.read();

    if (data=='A'){
      digitalWrite(led_rojo,HIGH);
      delay(2000);
      digitalWrite(led_rojo,LOW);
    }
    if (data=='B'){
      digitalWrite(led_verde,HIGH);
      delay(2000);
      digitalWrite(led_verde,LOW);
    }
    if (data=='C'){
      digitalWrite(led_amarillo,HIGH);
      delay(2000);
      digitalWrite(led_amarillo,LOW);
    }
    if (data=='D'){
      digitalWrite(led_blanco,HIGH);
      delay(2000);
      digitalWrite(led_blanco,LOW);
    }
    if (data=='E'){
      digitalWrite(motor,HIGH);
      delay(2000);
      digitalWrite(motor,LOW);
    }
    if (data=='F'){//Grados de giroscopio
      Serial.println(Angle[0]);
    }
    if (data=='G'){
      Serial.println(Angle[1]);
    }
  }
}
```

```
void GetMpuValue(const int MPU) {  
    Wire.beginTransaction(MPU);  
    Wire.write(0x3B);  
    Wire.endTransmission(false);  
    Wire.requestFrom(MPU, 14, true);  
    Ax=Wire.read()<<8|Wire.read();  
    Ay=Wire.read()<<8|Wire.read();  
    Az=Wire.read()<<8|Wire.read();  
    Tmp=Wire.read()<<8|Wire.read();  
    Gx=Wire.read()<<8|Wire.read();  
    Gy=Wire.read()<<8|Wire.read();  
    Gz=Wire.read()<<8|Wire.read();  
    delay(20);  
}
```

Programación en Unity

System.IO.Port

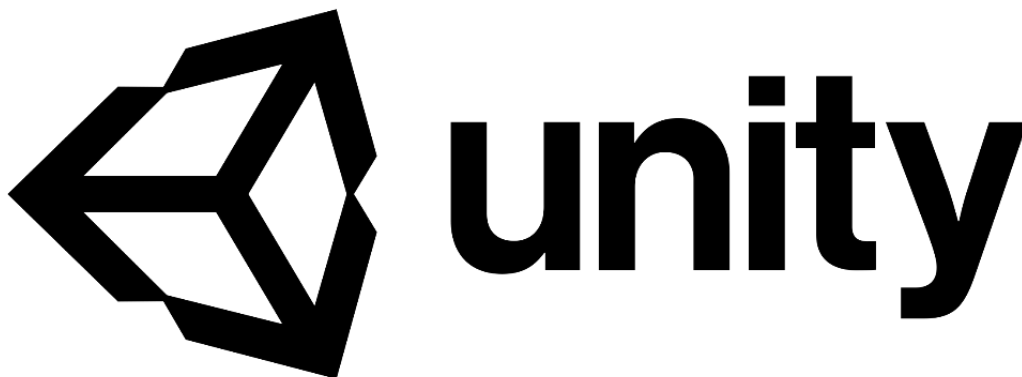
Contiene las clases necesarias para controlar los puertos serial. La clase mas importante es la clase SerialPort.

Para que Unity pudiera acceder a esta biblioteca se configuró su compatibilidad de API a 4.0.

La clase SerialPort proporciona una interfaz para entradas y salidas sincrónicas y controladores por eventos, acceso a estados de pin y rotura, y acceso a propiedades del controlador serial. Permite que las clases que usan secuencias puedan acceder al puerto serie.

Los eventos usados serán:

- serialPort.Open(): Esta función habilitará nuestro háptico para que pueda utilizarse en el proyecto.
- serialPort.ReadTimeout: Establece el número en milisegundos antes de que se produzca un tiempo de salida cuando una operación de lectura no finaliza.
- serialPort.IsOpen: Obtiene un valor que nos indica si el háptico ha sido habilitado.
- serialPort.Write(): Envía un valor al puerto serial.
- serialPort.ReadLine(): Lee el buffer de entrada que se envía desde el háptico hacia Unity.
- serialPort.Close(): Deshabilita el háptico cuando Unity ya no lo utiliza.



Script Arduino.cs



The screenshot shows the Unity 2020.2.6f1 Personal IDE. The top menu bar includes File, Edit, Assets, GameObject, Component, Window, and Help. The Inspector panel on the right displays the 'Arduino (Mono Script) Import Settings' for a script named 'Arduino'. The 'Imported Object' section shows 'Arduino (Mono Script)'. The 'Assembly Information' section shows 'Filename: Assembly-CSharp.dll'. The script code is as follows:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO.Ports;

public class Arduino : MonoBehaviour
{
    SerialPort serialPort = new SerialPort("COM3", 9600);

    void Start()
    {
        serialPort.Open();
        serialPort.ReadTimeout = 100;
    }

    public void rojoOn()
    {
        if (serialPort.IsOpen)
        {
            serialPort.Write("A");
        }
    }

    public void verdeOn()
    {
        if (serialPort.IsOpen)
        {
            serialPort.Write("B");
        }
    }

    public void amarilloOn()
    {
        if (serialPort.IsOpen)
        {
            serialPort.Write("C");
        }
    }

    public void blancoOn()
    {
        if (serialPort.IsOpen)
        {
            serialPort.Write("D");
        }
    }

    public void vibrarOn()
    {
        if (serialPort.IsOpen)
        {
            serialPort.Write("E");
        }
    }

    public float leerX()
    {
        {
            if (serialPort.IsOpen)
            {
                serialPort.Write("F");
                return int.Parse(serialPort.ReadLine());
            }
            return 0;
        }
    }

    public float leerY()
    {
        {
            if (serialPort.IsOpen)
            {
                serialPort.Write("G");
                return int.Parse(serialPort.ReadLine());
            }
            else
            {
                return 0;
            }
        }
    }

    public void cerrarPort()
    {
        {
            serialPort.Close();
        }
    }
}
```

Script SistemaBatalla.css

Giroscópio

En la función Update de este script si el juego está en el turno del jugador Unity ejecutará las funciones para obtener los ángulos del háptico en tiempo real, una vez que el turno del jugador este activo comparará los ángulos del háptico con los hechizos, en caso de coincidir con uno cambia el estado de batalla y se realiza la acción de ataque.

```
void Update()
{
    jugadorHUD.IlenarExp(entiWitch);
    jugadorHUD.IlenarVida(entiWitch);
    enemigoHUD.IlenarVida(enemigoUnidad);

    angleX = arduino.leerX();
    angleY = arduino.leerY();
    Debug.Log("Angulo X: " + angleX);
    Debug.Log("Angulo Y: " + angleY);
    if (estado == BattleState.ELEGIRHECHIZO)
    {
        if (angleX <= 70 && angleX >= 45)
        {
            estado = BattleState.TURNOJUGADOR;
            BotonAtkArriba();
        }
        else if (angleX <= -45 && angleX >= -70)
        {
            estado = BattleState.TURNOJUGADOR;
            BotonAtkAbajo();
        }
        else if (angleY <= 70 && angleY >= 45)
        {
            estado = BattleState.TURNOJUGADOR;
            BotonAtkDer();
        }
        else if (angleY <= -45 && angleY >= -70)
        {
            estado = BattleState.TURNOJUGADOR;
            BotonAtkIzq();
        }
    }
}
```

Leds

Dependiendo del hechizo es la función para llamar que hará que cierto led se encienda.

```
arduino.amarilloOn();
mensajeTexto.text = "Has realizado un ataque de luz";
```

```
arduino.blancoOn();
mensajeTexto.text = "Has realizado un ataque de agua";
```

```
arduino.verdeOn();
mensajeTexto.text = "Has realizado un ataque de planta";
```

```
arduino.rojoOn();
mensajeTexto.text = "Has realizado un ataque de fuego";
```

Motor de vibración

Dentro de la función del Turno Enemigo se mandará a llamar la función del Arduino que activará el motor de vibración, esto se realizará justo después de que el jugador reciba daño.

Si el jugador recibe daño por parte del enemigo entonces se manda a llamar la función `vibrarOn()`, eso hará que el motor de vibración se active por unos segundos.

```
IEnumerator TurnoEnemigo()
{
    Debug.Log("Se ha ejecutado la funcion Turno del Enemigo");
    entiWitch.transform.rotation = rotInicial;
    mensajeTexto.text = enemigoUnidad.nombre + " ataca!";

    Debug.Log("Esperando 1 segundo");
    yield return new WaitForSeconds(1f);

    Debug.Log("Reduciendo vida del jugador");
    bool estaMuerto = entiWitch.RecibeDanio(enemigoUnidad.atk);
    arduino.vibrarOn();

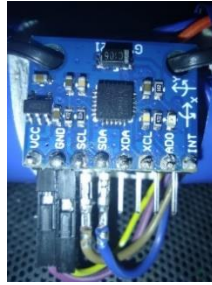
    Debug.Log("Actualizando HUD del jugador");
    jugadorHUD.actVida(entiWitch.vidaActual, entiWitch.vidaMax);

    Debug.Log("Esperando 1 segundo");
    yield return new WaitForSeconds(1f);

    if (estaMuerto)
    {
        Debug.Log("Actualizando estado a Pierde");
        estado = BattleState.PIERDE;
        FinBatalla();
    }
    else
    {
        Debug.Log("Actualizando estado a Elegir hechizo");
        estado = BattleState.ELEGIRHECHIZO;
        ElegirHechizo();
    }
}
```

Entradas generadas por el háptico

Movimiento del háptico



El giroscopio MPU 6050 envía valores en tiempo real al juego, cuando estos valores coincidan con los de un hechizo este se ejecuta solo cuando la pantalla te indique cuando deberás accionar un hechizo.

Dentro del Arduino los valores que vaya recolectando el MPU 6050 se estarán convirtiendo en ángulos, de esta manera se facilita las operaciones dentro del juego.

Hechizo de luz 45° a 70° en X

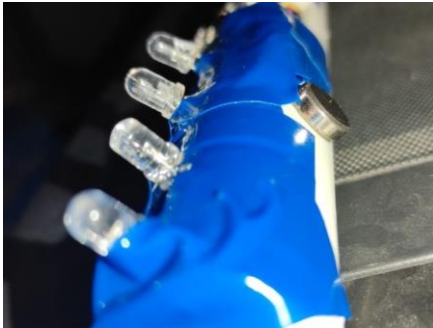
Hechizo de agua -45° a -70° en X

Hechizo de planta -40° a -70° en Y





Hechizo de fuego 45° a 70° en Y

Salidas recibidas por el háptico

Encendido de leds



Cuando el jugador realice un hechizo dependiendo de que tipo sea se encenderá un led de diferente color.

-  Si es de fuego se encenderá el led color de rojo.
-  Si es de agua se encenderá el led color de blanco.
-  Si es de planta se encenderá el led color de verde.
-  Si es de luz se encenderá el led de color amarillo.

Vibración del motor



Si durante el turno del enemigo el jugador recibe daño el motor de vibración se activará y se agitará durante 2 segundos, así mismo se verá en la pantalla como se reduce el círculo de vida del jugador en la pantalla.