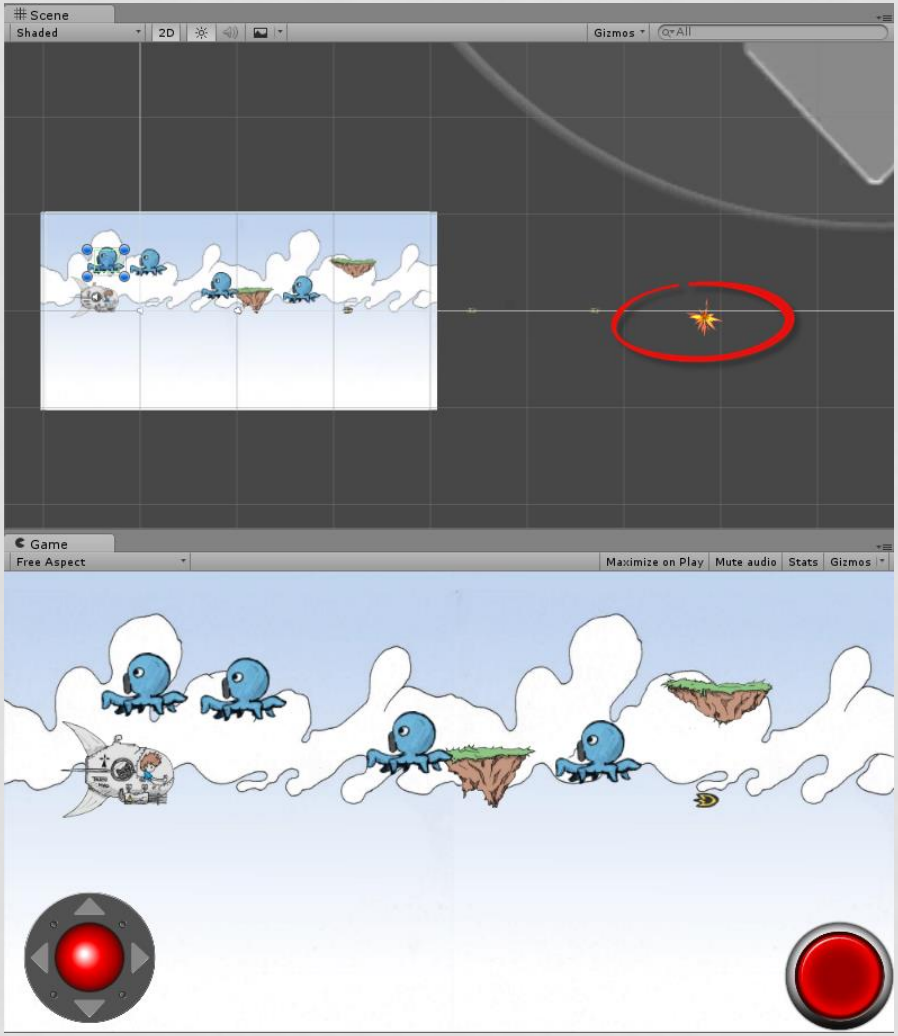
UNITY

PULPI – JOC 2D PER ANDROID

CONTROLANT LA VISIBILITAT

Exemple 1: si posem la interfície en mode 2 by 3, veiem que les bales poden eliminar els pops que encara no estan en escena.



CONTROLANT LA VISIBILITAT

Exemple 2: potser volem que la IA dels pops no es posi en marxa fins que no entrin en pantalla

Això ho podem controlar verificant si un objecte és visible o no:

- **Renderer.isVisible**: indica si s'està renderitzant en qualsevol càmera
- **void OnBecameVisible()**: s'executa quan un objecte es fa visible
- **void OnBecameInvisible()**: s'executa quan un objecte es fa invisible

PROBLEMA: també es te en compte la càmera de l'editor (verificar en (Layout 2 by 3) => dificulta disseny de nivells

EXTENSIÓ DE CLASSES EN C#

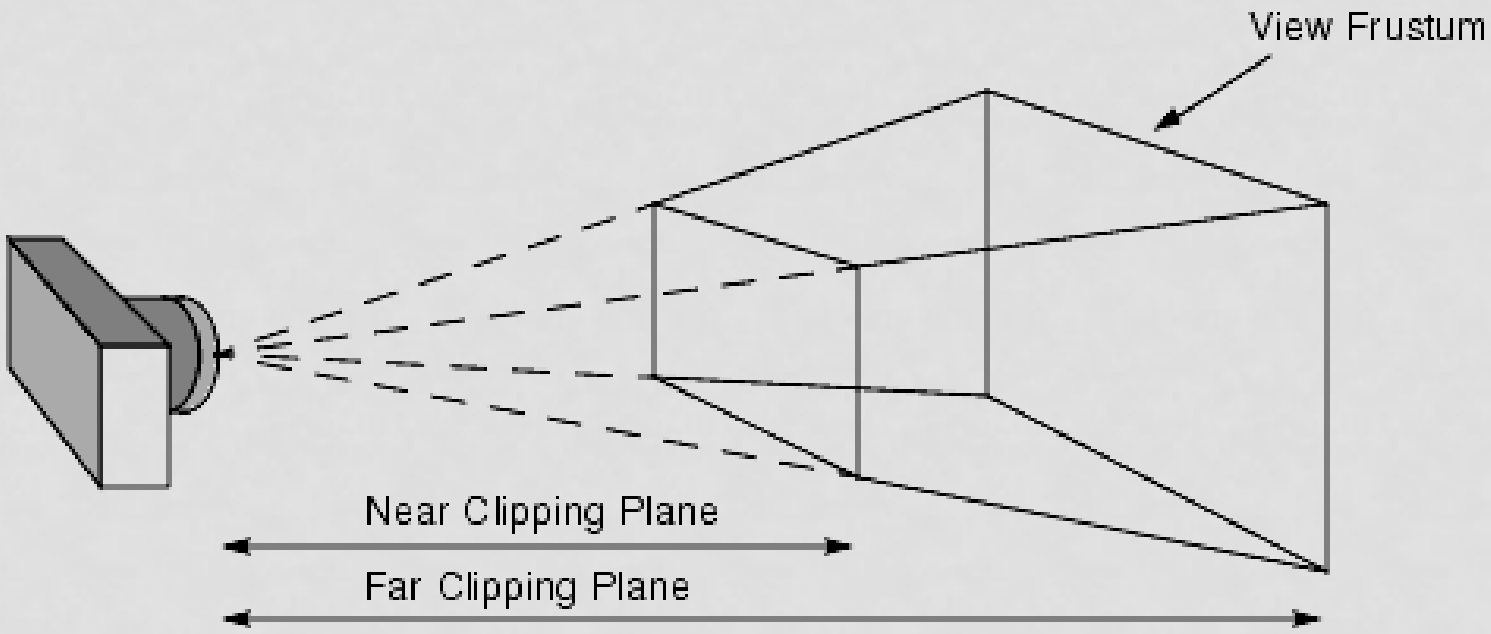
Afegir nous mètodes a un component ja existent (renderer) perquè ens digui si un objecte és visible o no en una determinada càmera. Creem el següent script (no cal associar-lo a cap objecte)

ScrExtRenderer.cs

```
public static class RendererExtensions
{
    public static bool EsVisibleDesde(this Renderer renderer, Camera camera)
    {
        Plane[] planes = GeometryUtility.CalculateFrustumPlanes(camera);
        return GeometryUtility.TestPlanesAABB(planes, renderer.bounds);
    }
}
```

Verifica si hi ha intersecció entre el render de l'objecte i l'espai definit pels quatre plans que conformen el frustum de la càmera

EXTENSIÓ DE CLASSES EN C#



QUE ELS POPS NO MORIN FORA DE CAMP

Amb això, el component renderer ja disposa del mètode **render.EsVisibleDesde (objeto_cámara)**

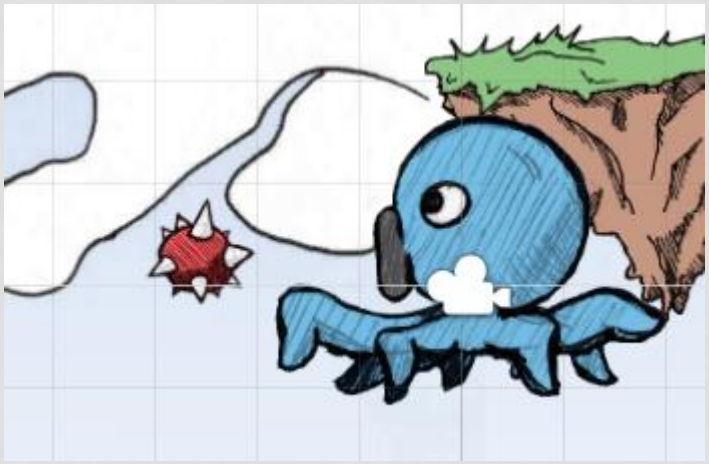
En ScrControlVida.cs

```
Renderer render;

void Start ()  {
        render = GetComponent<Renderer>();
}

void OnTriggerEnter2D(Collider2D otro){
   if (otro.tag == "shot1_player" && render.EsVisibleDesde(Camera.main))
        {
        ...
        }
}
```

SHOT DEL PULPI

Podem duplicar el shot del Player i canviem:

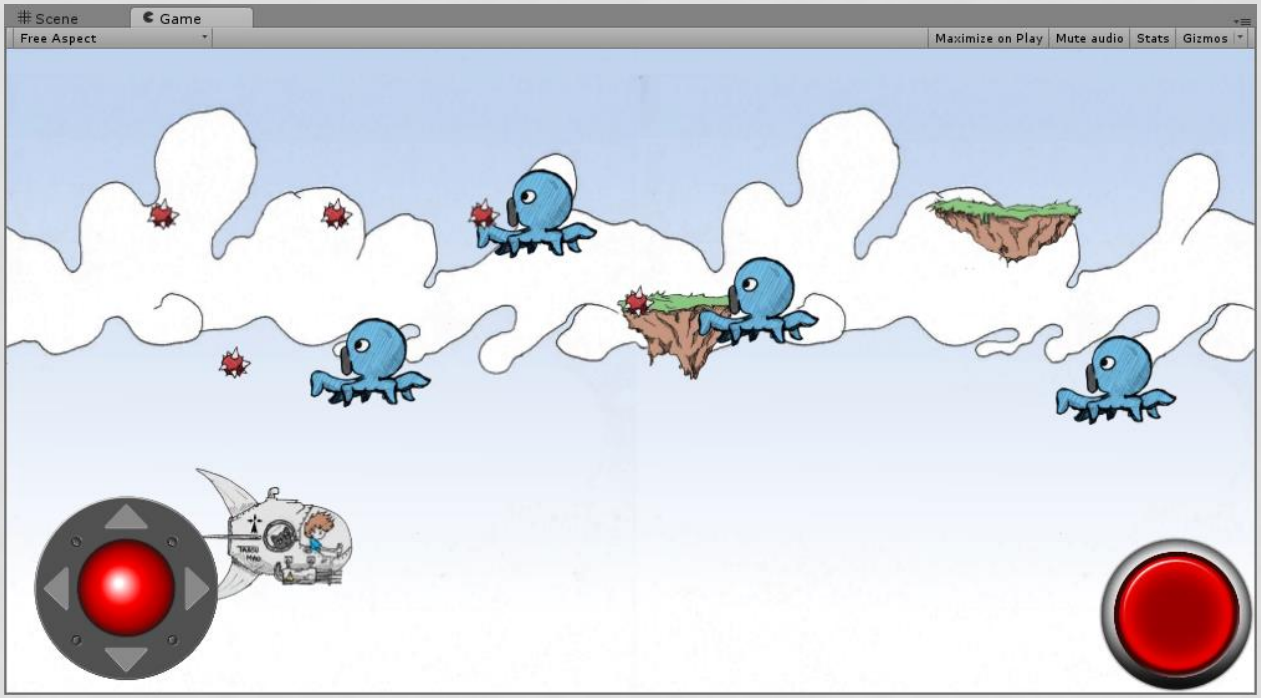
- Sprite
- Name i Tag (**shot_poulpi**)
- Velocitat



SHOT DEL PULPI

Ara ha de ser el pop el que dispari els projectils:

- Creem Empty Game Object fill i ubiquem
- Modifiquem script ScrPulpi



SHOT DEL PULPI

```
// Disparo ************************************
public Transform arma;
public Transform bala;
public float cadenciaMin=1, cadenciaMax=3; // tiempo entre disparos
float crono=0f;
Renderer render;

void Start() {
render = GetComponent<Renderer> ();
crono = Random.Range(cadenciaMin,cadenciaMax); // Preparamos primer disparo
... }
void Update () {
   if (render.EsVisibleDesde (Camera.main)) { // Empieza a disparar si visible
      if (crono <=0) Dispara();
      crono-=Time.deltaTime;
   }
}

void Dispara()   {
   Transform b = Instantiate (bala,arma.position,arma.rotation) as Transform;
   b.Rotate(0,0,Random.Range(-10,10)); // modificamos trayectoria aleatoriamente
   crono = Random.Range(cadenciaMin,cadenciaMax); // Siguiente disparo
}
```

SHOT DEL PULPI

Si ens fixem en ScrControlVida.cs, serveix totalment excepte que ara hem de verificar si choquem contra un objecte que en comptes de dir-se **shot1_player** s'ha de dir **shot_poulpi**. Per resoldre-ho, parametritzarem aquet valor. A **ScrControlVida.cs**:

```
public string nomProjectil;
...
if (otro.tag == nomProjectil && render.EsVisibleDesde(Camera.main))
{...}
```

Recordar també actualitzar el prefab del pop per què la propietat **nomProjectil** tingui el valor **shot1_player**

SHOT DEL PULPI

Ara ja podem arrossegar ScrVida.cs al Player, i establir les variables públiques:

- **Vitalidad**: 10
- **Explosion**: CFX_SmokeExplosionAlt
- **Nom projectil**: shot_poulpi
- **Tocado**: small_hard_object_strike_large_metal_drum
- **Hundido**: explosion_soft_rnd_01 (8 bit Retro Rampage)

COL·LISIÓ PLAYER-POULPI

Si ens fixem en **ScrControlVida.cs**,

```
if (otro.tag == nomProyectil && ...
```

Només verifica si hem xocat amb un tipus d'objecte, però ara en volem controlar més d'un: pulpi, projectil del pulpi,... (podriem posar mines, per exemple)

Modifiquem l'script per que permeti controlar no un nom, sinó un **array de noms**.

COL·LISIÓ PLAYER-POULPI

```
using System;  // per poder utilitzar Array.IndexOf

public string[] nomProyectils; // que elementos nos matan
// recordar establir els valors per el poulpi

// busquem en tot l'array el tag del objecte col·lisionat
if (Array.IndexOf(nomProyectils,otro.tag)>-1  &&
render.EsVisibleDesde(Camera.main))
```

Per que funcioni, com a mínim un dels dos ha de ser Trigger

BALANCEJAT

Ara ja podem començar a balancejar les propietats:

- Temps de vida dels projectils
- Velocitats de tots els elements
- Poder de destrucció de les armes
- Cadències
- Nombre d'enemics i disposició
- Vitalitat de Player i NPCs
- ...

PARALLAX SCROLLING

Moure les capes més allunyades més lentament. Primer hem de decidir l'estratègia:

- Cas 1: **movem el player i la càmera**, i la resta queda fix. En un joc 3D ja tindríem l'efecte de profunditat, però no serveix per jocs 2D amb càmera ortogràfica (el nostre cas).
- Cas 2: el player i la càmera romanen estàtics, i **el nivell va passant**

IMPORTANT: lligar la càmera al Player normalment no és una bona solució.

PARALLAX SCROLLING

Un bon exemple

http://www.youtube.com/watch?v=TCIMPYM0AQg

PARALLAX SCROLLING

Shoot-em-up (shmups)
Scrolling shuter

PARALLAX SCROLLING

ScrScroll.cs: s'ha d'aplicar a les diferents capes de profunditat: background, far middleground i near middleground.

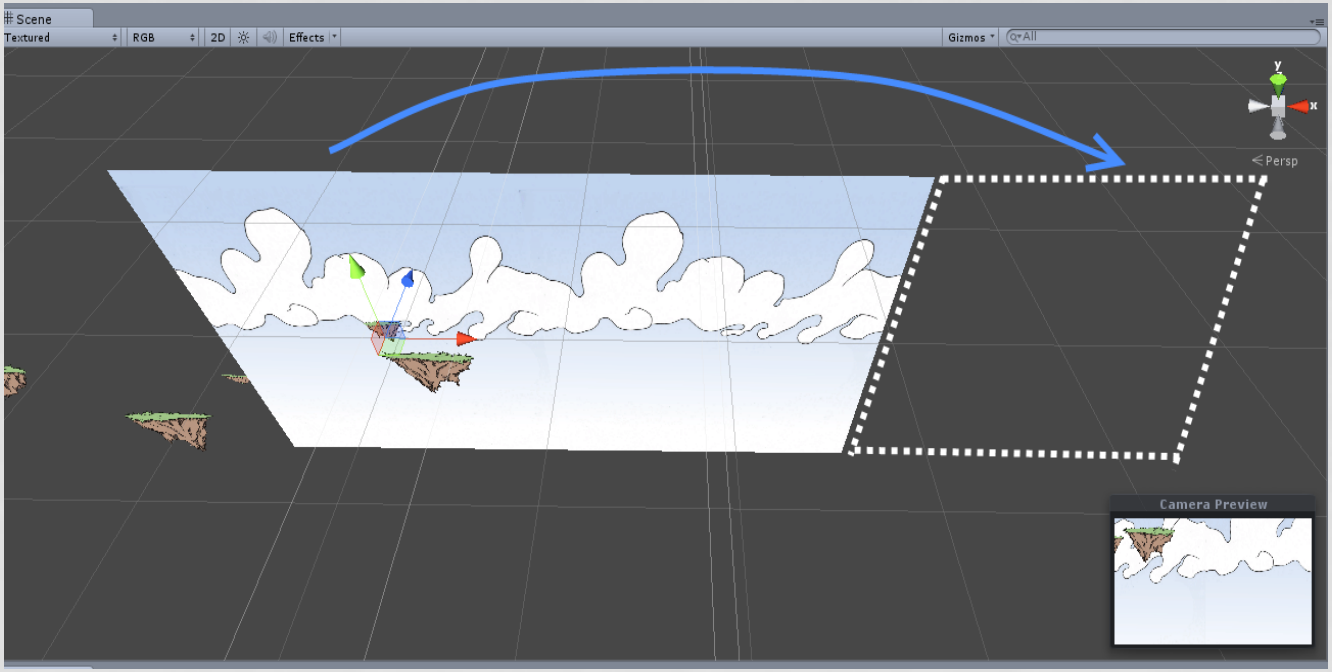
```
public float scrollSpeed = 0f;
void Update()
{
    transform.Translate(scrollSpeed*Time.deltaTime,0,0);
}
```

Exemple de scrollSpeed:

- Background: -1
- Far Middleground: -2
- Near Middleground: -3
- Foreground: no se li aplica l'scroll

INFINITE BACKGROUND SCROLLING

Necessitarem un mínim que cobreixi el camp visual de la càmera. Creem una tercera còpia dels núvols que posem a X=40.96



INFINITE BACKGROUND SCROLLING

ScrInfiniteLoop.cs (versió fàcil): fa que quan un fons desaparegui per l'esquerra, el traslladi a la dreta.

NOTA: si tenim la vista Scene mentre juguem, no funcionarà

```
void OnBecameInvisible()
{

    // la imagen mide 2048 pixels, y hay 3
    transform.Translate(20.48f * 3, 0f, 0f);
}
```

Per testejar-ho ràpidament, podem accelerar la velocitat del background

INFINITE BACKGROUND SCROLLING

ScrInfiniteLoop.cs (versió difícil): aquesta versió utilitza **EsVisibleDesde**, per la qual cosa funcionarà sempre, encara que estem veient la vista Scene

```
bool visible;
Renderer render;
void Start() {
    render = GetComponent<Renderer>();
    if (render.EsVisibleDesde(Camera.main)) visible = true; // inicialmente visible?
    else visible = false;
}
void Update() {
    // si desaparece por la izquierda…
    if (visible && render.EsVisibleDesde(Camera.main) == false)
    {   // lo trasladamos a la derecha
        transform.Translate(20.48f * 3, 0f, 0f);
        visible = false;
    }
    // si aparece por la derecha
    if (!visible && GetComponent<Renderer>().EsVisibleDesde(Camera.main) == true)
                    visible = true;     // lo marcamos como visible
}
```

RESPAWN

Fer que quan els pops o les plataformes desapareguin per l'esquerra es 'teletransportin' a la dreta i així tornar a aparèixer de nou.

Les variables **minX** i **maxX** determinen el mínim i el màxim que es poden desplaçar horitzontalment

minY i **maxY**: determinen els límits verticals on s'ha de crear (recordem que les coordinades de la nostra càmera estaven entre -10,24 i 10.24)

RESPAWN

ScrRespawn.cs (versió fàcil): si tenim la vista Scene mentre juguem, no funcionarà

```
public bool respawnable = true; // hay que generarlo de nuevo?
public float minX = 50f, maxX = 65f, minY = -9, maxY = 9;

void OnBecameInvisible()
{
    if (respawnable)
    {
        float x = Random.Range(minX, maxX); // Cuanto tiramos hacia atras
        float y = Random.Range(minY, maxY); // A que altura
        Vector3 posi = new Vector3(transform.position.x + x, y,
transform.position.z);
        transform.position = posi;
    }
    else Destroy(gameObject);
}
```

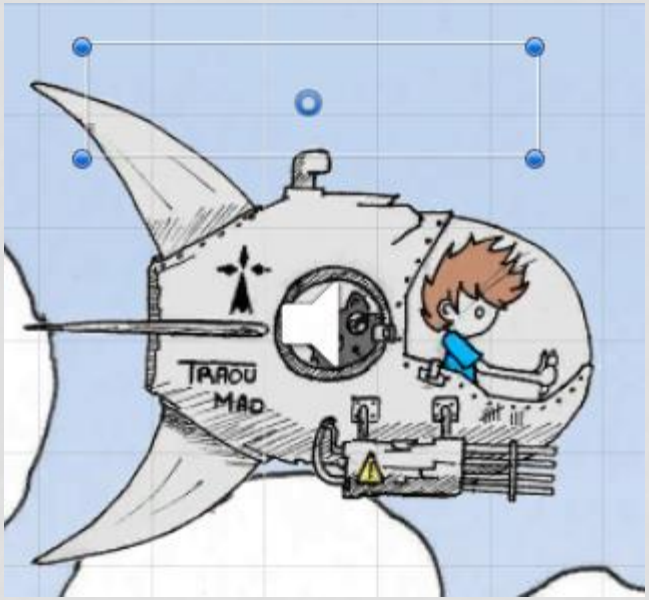
RESPAWN

ScrRespawn.cs (versió difícil): si volem que també funcioni amb la vista Scene haurem d'afegir:

```
bool visible;
Renderer render;
void Start()   {
    render = GetComponent<Renderer>();
    if (renderer.EsVisibleDesde(Camera.main)) visible = true;
    else visible = false;  // inicialmente es visible?
}
void Update()  {
    // ha desaparecido por la izquierda
    if (visible && renderer.EsVisibleDesde(Camera.main) == false)
    {
        visible = false;  // lo marcamos como invisible
        MiOnBecameInvisible(); // Cambiamos nombre función
    }
    // aparece por la derecha
    if (!visible && renderer.EsVisibleDesde(Camera.main) == true)
        visible = true;     // lo marcamos como visible
}
void MiOnBecameInvisible()
{ // aquí copiamos el código que antes pusimos en OnBecameInvisible anterior
}
```

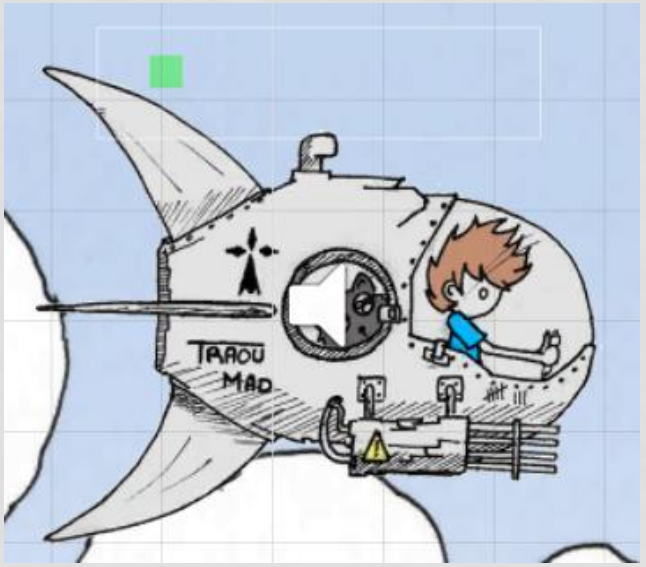
BARRA DE VIDA

- Creem canvas fill del Player
- Tipus World Space
- PosX=0, PosY=2
- Width=4, Height=1



BARRA DE VIDA

- Creem un Image
- Width=0.3, Height = 0.3
- Source Image = imatge verda
- Ubiquem a la posició del primer quadre
- Creem prefab
- Esborrem el quadre de l'escena



BARRA DE VIDA

Modifiquem **ScrControlaVida.cs**. Primer afegim les següents variables

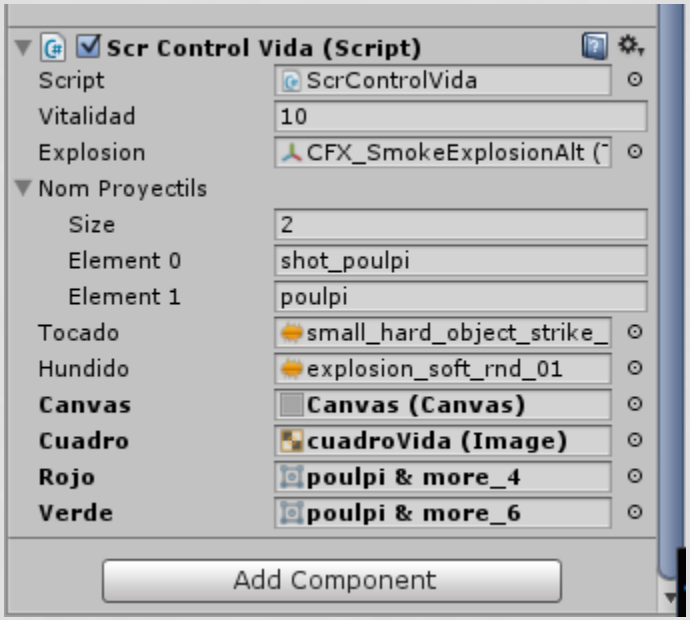
```
public Canvas canvas;    // si esta asignado, tiene barra de vida
public Image cuadro;     // imagen de un cuadro de vida
Image [] cuadrosVitalidad;  // todos los cuadros de vida sobre la nave
public Sprite rojo, verde;  // bitmaps
```

Des de l'inspector assignem:

- **Canvas**: el canvas fill del Player
- **Cuadro**: el prefab que hem creat anteriorment
- **Rojo**, **Verde**: els 2 sprites del spritsheet

BARRA DE VIDA

Ha de quedar quelcom semblant al següent:



BARRA DE VIDA

- A la funció **Start** afegim:

```
if (!canvas) return;  // Si no tiene barra de vida, no continuamos

// inicializa vides. Está programado para vitalidad del Player = 10
cuadrosVitalidad = new Image[10]; // inicializamos array
for (int i=0;i<10;i++)  // creamos la barra de vida
{
   cuadrosVitalidad[i] = Instantiate(cuadro); // instanciamos
   // Hacemos hijo del Canvas.
   cuadrosVitalidad[i].transform.SetParent (canvas.transform, false);
   // Desplazamos a la derecha
   cuadrosVitalidad[i].transform.Translate (0.33f * i, 0,
0,Space.World); //.33: tamaño del cuadro + un poco de separación
}
```

BARRA DE VIDA

Dins d'**OnTriggerEnter2D**:

```
float danyo = otro.GetComponent<ScrDanyo>().danyo;
vitalidad -= danyo;
if (canvas) PintaVidas(); // si tiene barra de vida, actualizamos
if (vitalidad <= 0)     Destruye();
```

BARRA DE VIDA

I finalment creem la funció **PintaVidas()**

```
void PintaVidas()
{
   int v = (int)vitalidad; // porque vitalidad era float

   if (v<0) v=0;  // para no tenir vida en negativo

   for (int i=0; i<v;i++) // pintamos los verdes
           cuadrosVitalidad[i].sprite = verde;

   for (int i=v;i<10;i++) // y despues los rojos
        cuadrosVitalidad[i].sprite = rojo;
}
```

LIMITAR MOVIMENT PLAYER

Farem ús de la funció

```
Camera.main.WorldToScreenPoint(transform.position);
```

Torna en quina coordinada de la pantalla apareix representada una coordinada del món.

LIMITAR MOVIMENT PLAYER

A ScrPlayer.cs:

```
// márgenes del espacio de movimiento de la nave
public int mLeft = 25, mRight = 225, mTop = 25, mBottom = 25;

void Update () {

   movi.x = ETCInput.GetAxis("Horizontal") * velocidad;
   movi.y = ETCInput.GetAxis("Vertical") * velocidad;

   // Calculamos coordenada en pantalla
   Vector3 posi = Camera.main.WorldToScreenPoint(transform.position);
   // y restringimos movimiento
   if (posi.x < mLeft && movi.x < 0) movi.x = 0;
   if (posi.x > Screen.width-mRight && movi.x > 0) movi.x = 0;
   if (posi.y < mBottom && movi.y < 0) movi.y = 0;
   if (posi.y > Screen.height-mTop && movi.y > 0) movi.y = 0;
```

IDEES PER IMPLEMENTAR

- Mostrar el nombre d'enemics
- Power-ups
- **Implementar vida infinita**
- **Next Level**
- Diferents armes, enemics
- Boss
- Més moviments d'IA