



Portafolio Optimización

Portafolio tercer trimestre

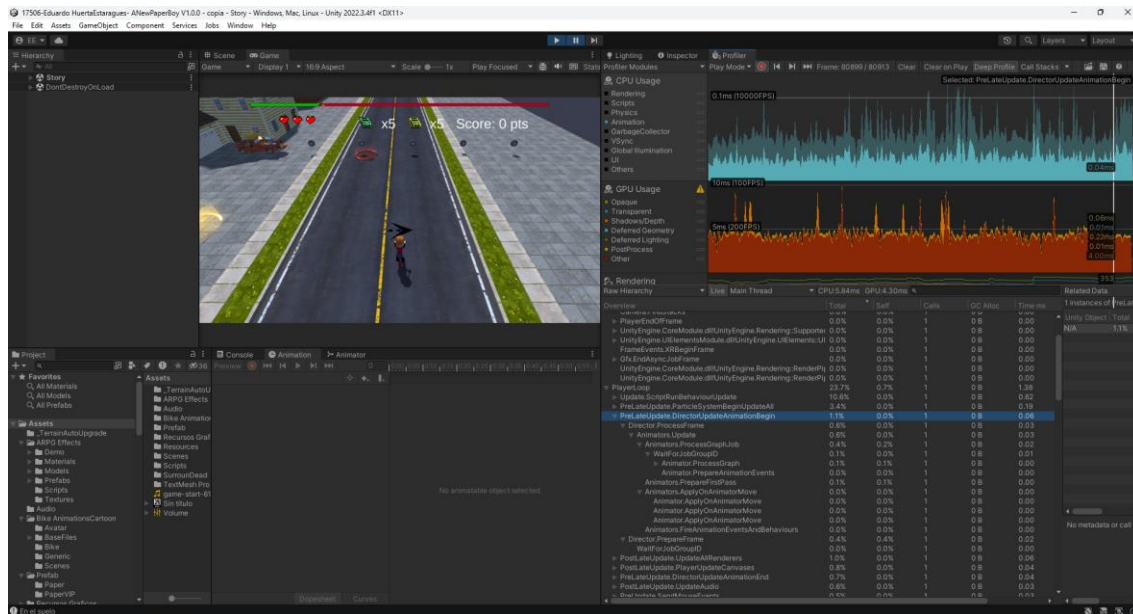
Nombre de los integrantes:

Huerta Estaragués Eduardo Damián

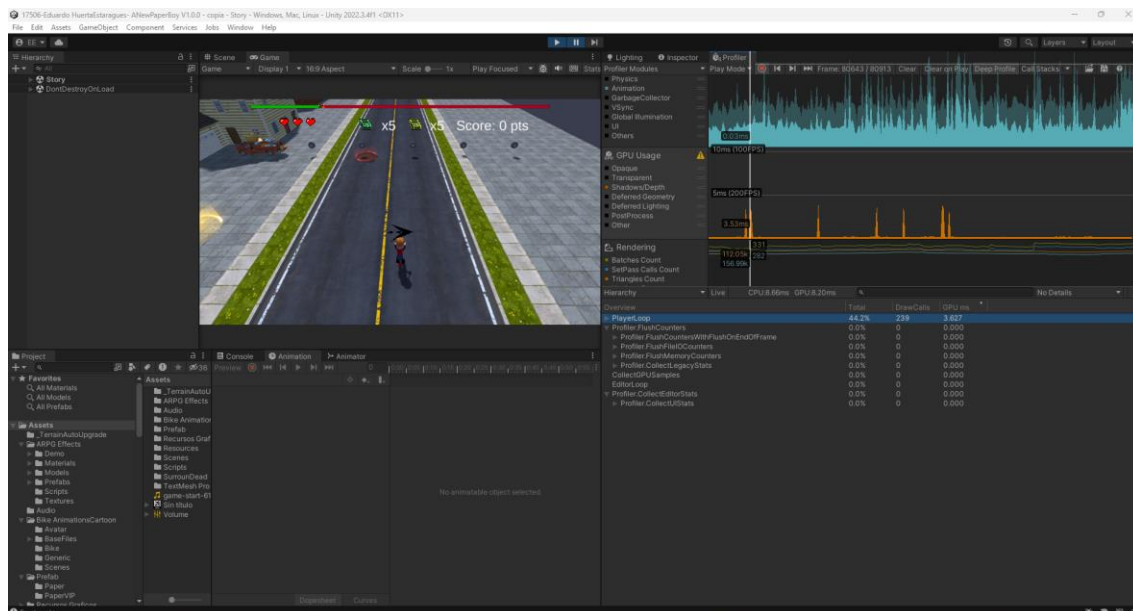
Fecha del Documento:

9/03/2023

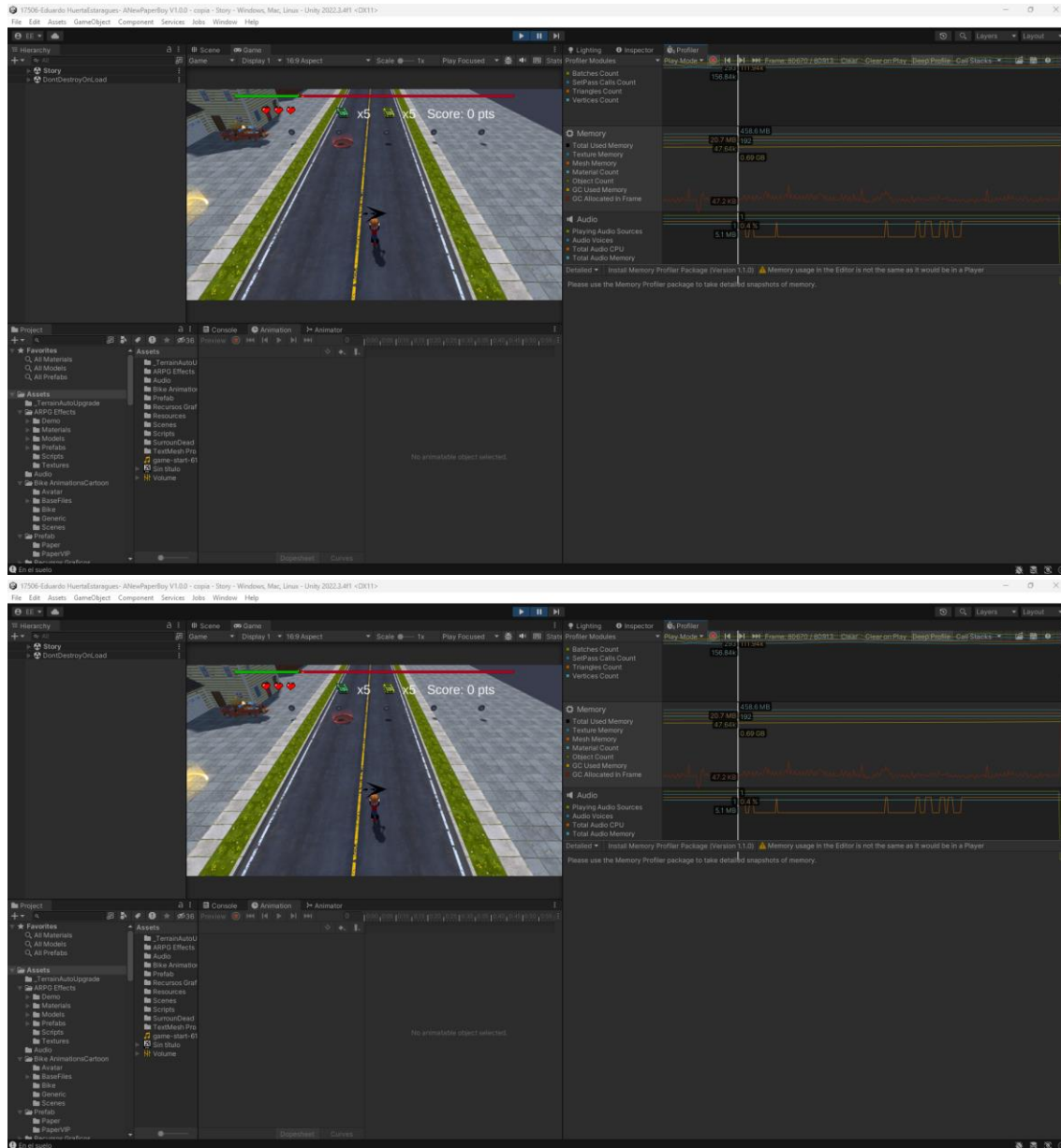
Desempeño - Proyecto Trimestre 1



Por lo que puedo ver, tengo un rendimiento, bastante adecuado de forma media, sin embargo, es posible mejorarlo optimizando procesos, pues por lo que veo en el análisis de recursos y procesos de UNITY, puedo resaltar que el “Update” de mi jugador es lo que está consumiendo la mayor parte de los recursos. Esto pues este se llama de forma constante y está haciendo el manejo de inputs y comunicar con los demás sistemas. Por lo que podría ser que separar algunos sistemas o el realizar un mejor manejo de estados, podría mejorar el desempeño, tal como la implementación de un switch o remover elemento que no se requieran llamar a cada fotograma.

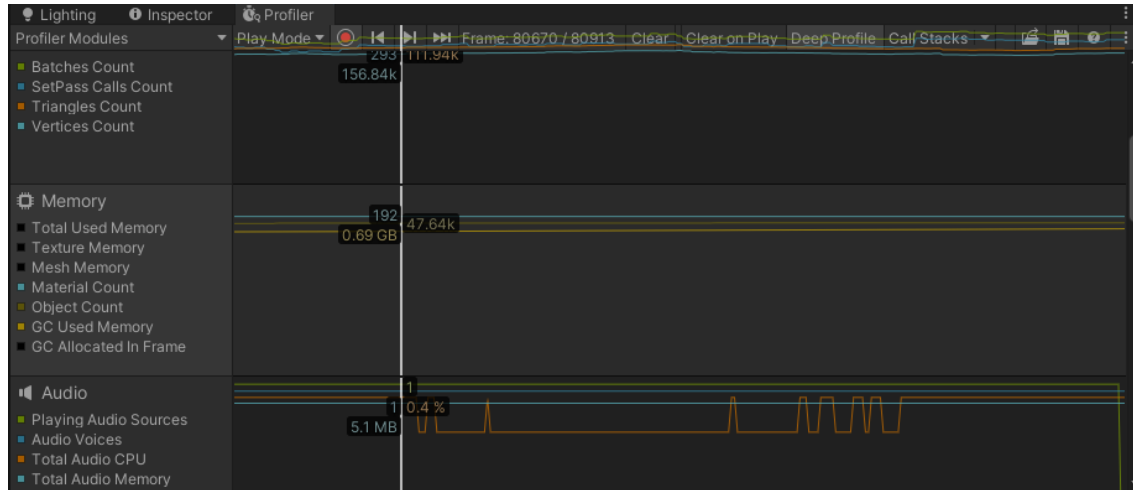




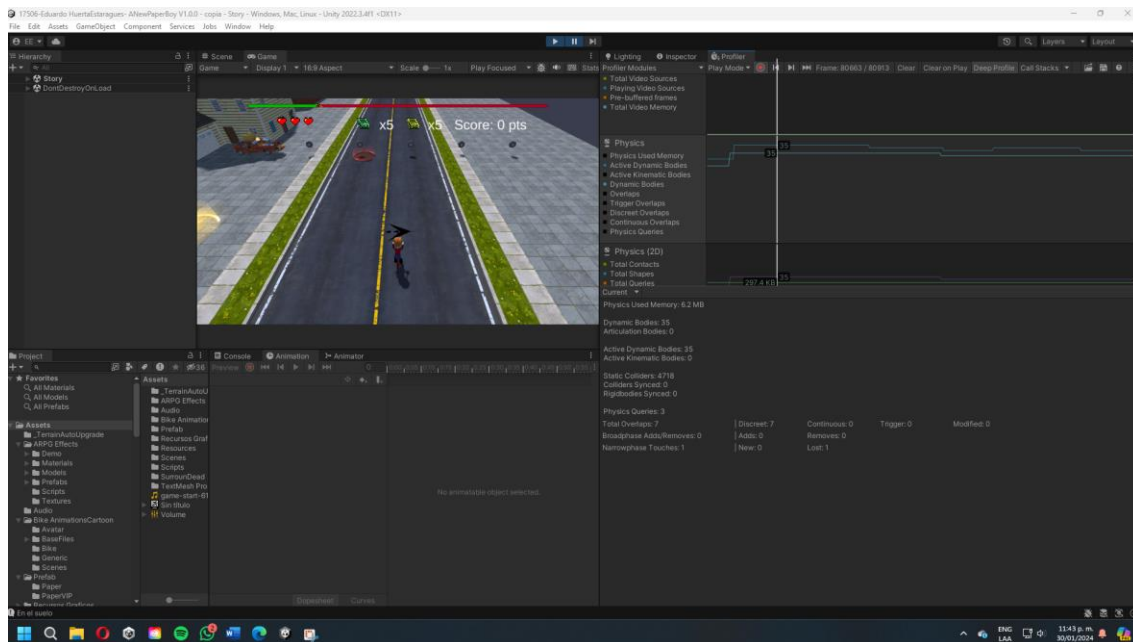


Igualmente, lo que concierne a memoria, este proyecto requiere una memoria medianamente grande, pues esta emplea, múltiples assets; modelos, texturas y audios. Los cuales se cargan al inicio de la escena, generando una carga inicial lenta y problemas sobre todo al usar almacenamientos más lentos.

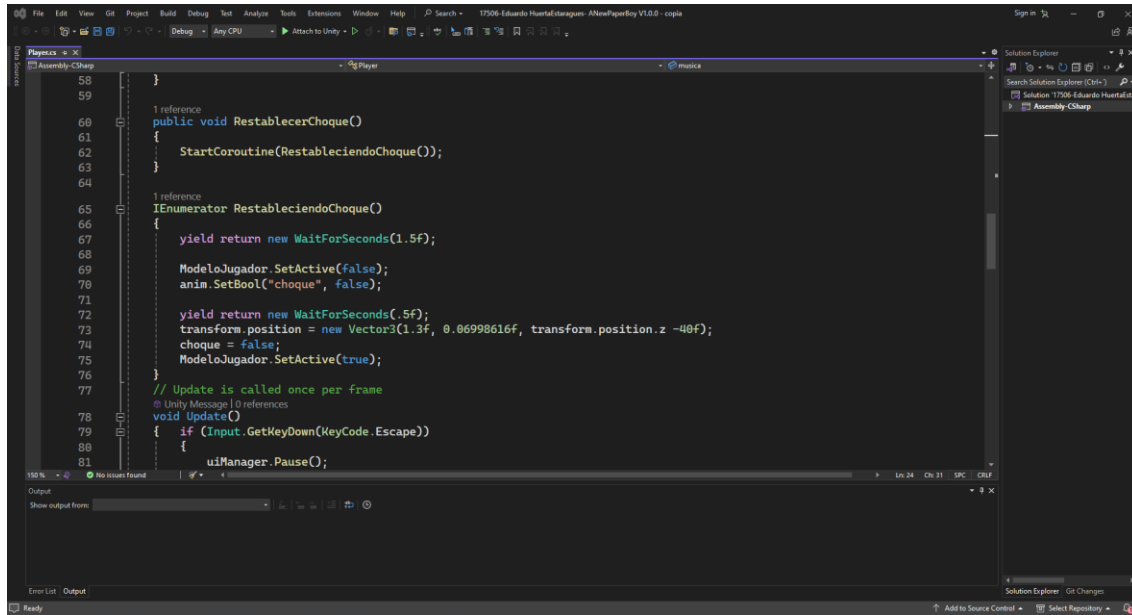
Para esto se puede generar un sistema de renderizado por proximidad al jugador, o reducir resolución de materiales, y más al pasar a un móvil donde estos no serán tan apreciables.



En general los assets requieren una gran cantidad de memoria, debido a su naturaleza 3D, texturizado y uso de sistemas de partículas y generación de elementos de prefabs. Por lo que una solución viable, sería activar y desactivar elementos sin instanciar nuevos, para reducir carga y movimiento de elementos en memoria, a la vez reducir número de partículas.



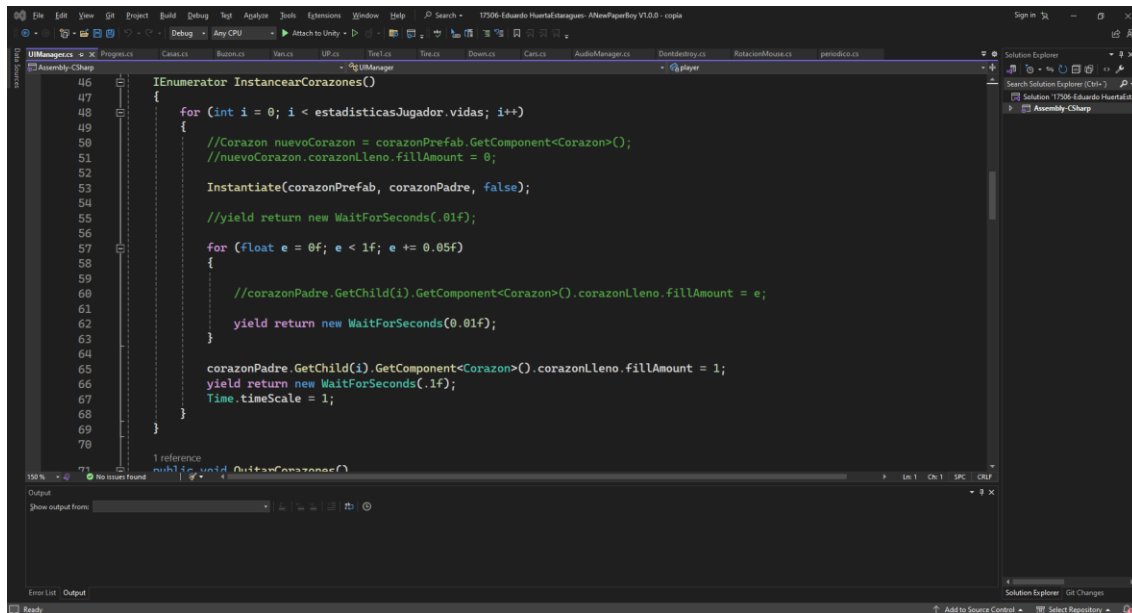
En el caso de sistemas de físicas, si bien no está requiriendo una gran cantidad de recursos, el uso de estos sistemas de forma tan amplia puede resultar en que sistemas más bajos tenga problemas o errores al ejecutarlo. Por lo que en caso de ser posible sería recomendable emplear otros métodos, o tener sistemas para limitar la cantidad de elementos que estén realizando estos procesos.

```

58 }
59
60 1 reference
61 public void RestablecerChoque()
62 {
63     StartCoroutine(RestableciendoChoque());
64 }
65
66 1 reference
67 IEnumerator RestableciendoChoque()
68 {
69     yield return new WaitForSeconds(1.5f);
70     ModeloJugador.SetActive(false);
71     anim.SetBool("choque", false);
72     yield return new WaitForSeconds(.5f);
73     transform.position = new Vector3(1.3f, 0.86998616f, transform.position.z -40f);
74     choque = false;
75     ModeloJugador.SetActive(true);
76 }
77 // Update is called once per frame
78 @ Unity Message 10 references
79 void Update()
80 {
81     if (Input.GetKeyDown(KeyCode.Escape))
82     {
83         uiManager.Pause();
84     }
85 }

```

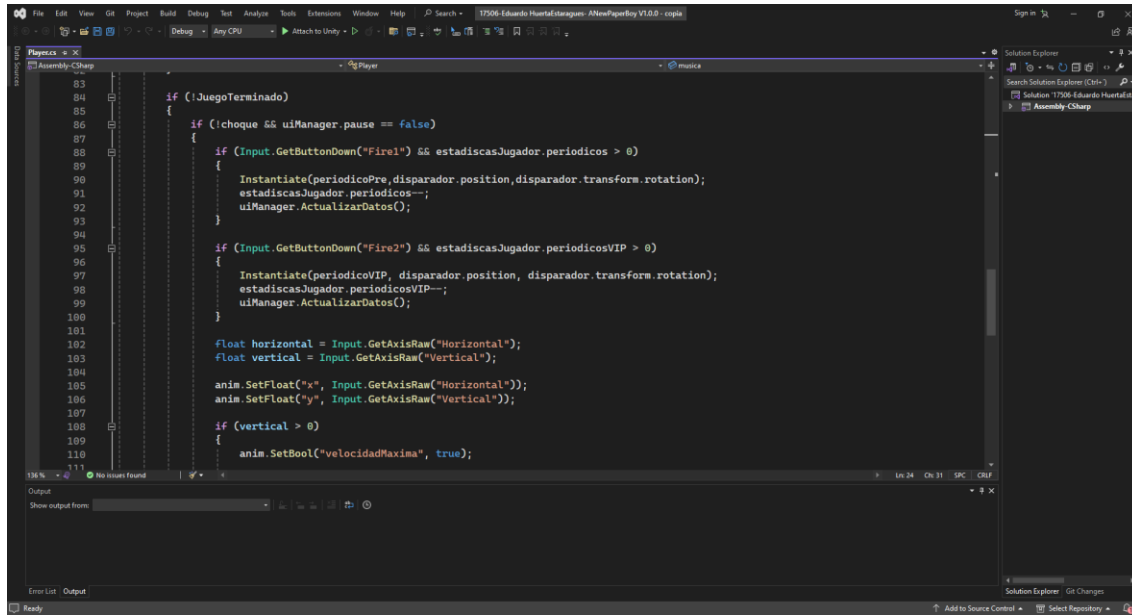


```

46 1 reference
47 IEnumerator InstancearCorazones()
48 {
49     for (int i = 0; i < estadisticasJugador.vidas; i++)
50     {
51         //Corazon nuevoCorazon = corazonPrefab.GetComponent<Corazon>();
52         //nuevoCorazon.corazonLleno.fillAmount = 0;
53         Instantiate(corazonPrefab, corazonPadre, false);
54
55         //yield return new WaitForSeconds(.01f);
56
57         for (float e = 0f; e < 1f; e += 0.05f)
58         {
59             //corazonPadre.GetChild(i).GetComponent<Corazon>().corazonLleno.fillAmount = e;
60             yield return new WaitForSeconds(0.01f);
61         }
62
63         corazonPadre.GetChild(i).GetComponent<Corazon>().corazonLleno.fillAmount = 1;
64         yield return new WaitForSeconds(.1f);
65         Time.timeScale = 1;
66     }
67 }
68
69 public void OuttarCorazones()
70 {
71 }

```

En este primer proyecto, se emplearon varias corrutinas para realizar acciones y procesos, como al llamar los cambios de animaciones y estados podrían cambiarse por sistemas más eficientes como async/await.

```

83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112

if (!JuegoTerminado)
{
    if (!choque && uiManager.pause == false)
    {
        if (Input.GetButtonDown("Fire1") && estadiscasJugador.periodicos > 0)
        {
            Instantiate(periodicoPre, disparador.position, disparador.transform.rotation);
            estadiscasJugador.periodicos--;
            uiManager.ActualizarDatos();
        }

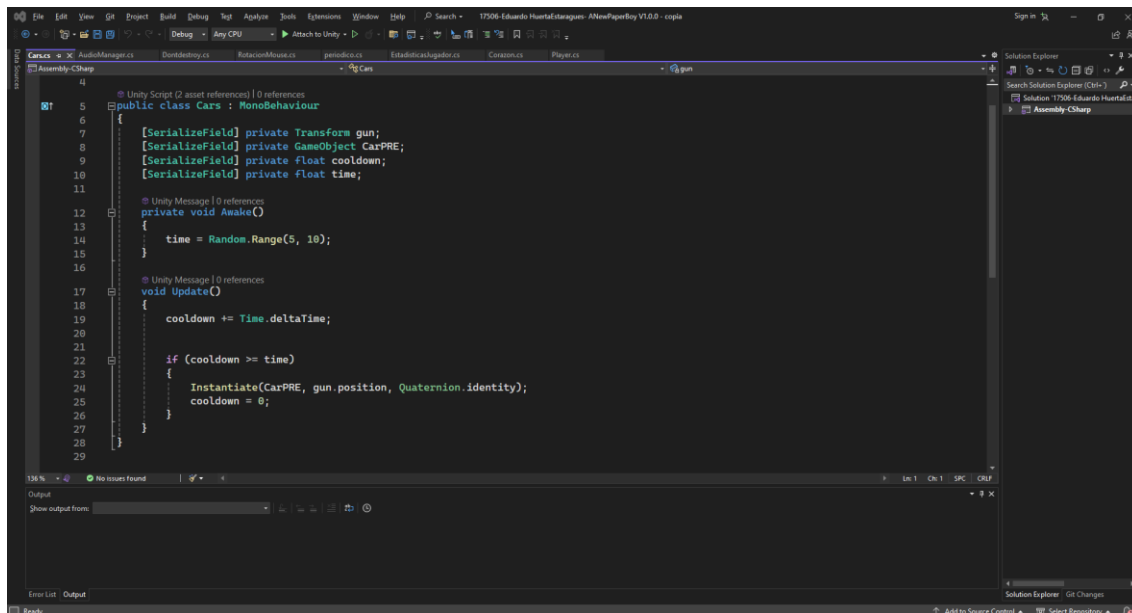
        if (Input.GetButtonDown("Fire2") && estadiscasJugador.periodicosVIP > 0)
        {
            Instantiate(periodicoVIP, disparador.position, disparador.transform.rotation);
            estadiscasJugador.periodicosVIP--;
            uiManager.ActualizarDatos();
        }

        float horizontal = Input.GetAxisRaw("Horizontal");
        float vertical = Input.GetAxisRaw("Vertical");

        anim.SetFloat("x", Input.GetAxisRaw("Horizontal"));
        anim.SetFloat("y", Input.GetAxisRaw("Vertical"));

        if (vertical > 0)
        {
            anim.SetBool("velocidadMaxima", true);
        }
    }
}

```



```

5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

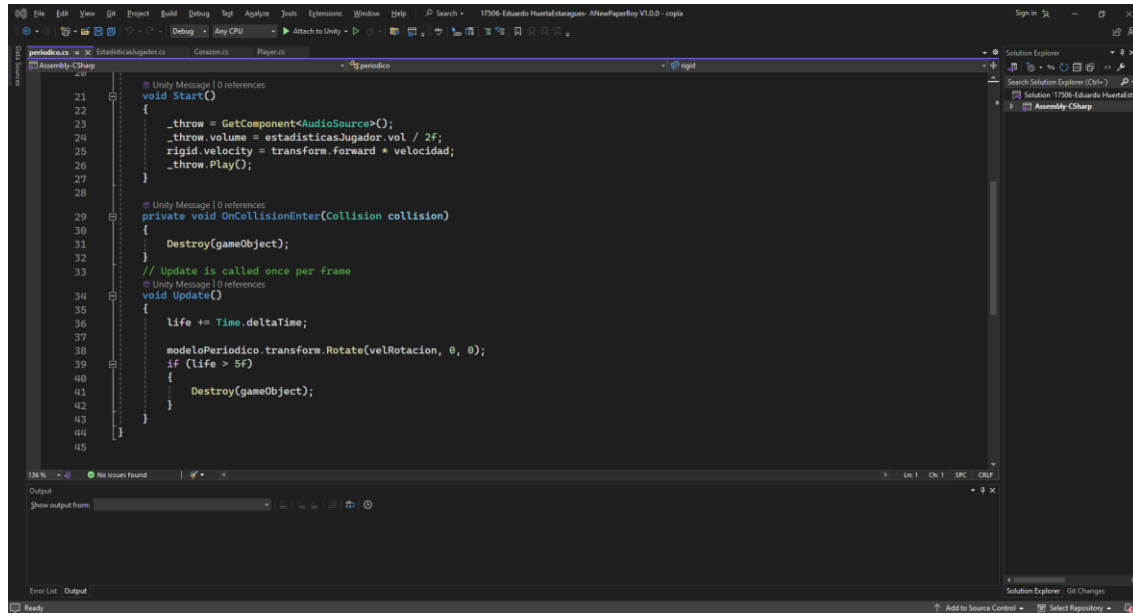
public class Cars : MonoBehaviour
{
    [SerializeField] private Transform gun;
    [SerializeField] private GameObject CarPRE;
    [SerializeField] private float cooldown;
    [SerializeField] private float time;

    private void Awake()
    {
        time = Random.Range(5, 10);
    }

    private void Update()
    {
        cooldown += Time.deltaTime;

        if (cooldown >= time)
        {
            Instantiate(CarPRE, gun.position, Quaternion.identity);
            cooldown = 0;
        }
    }
}

```

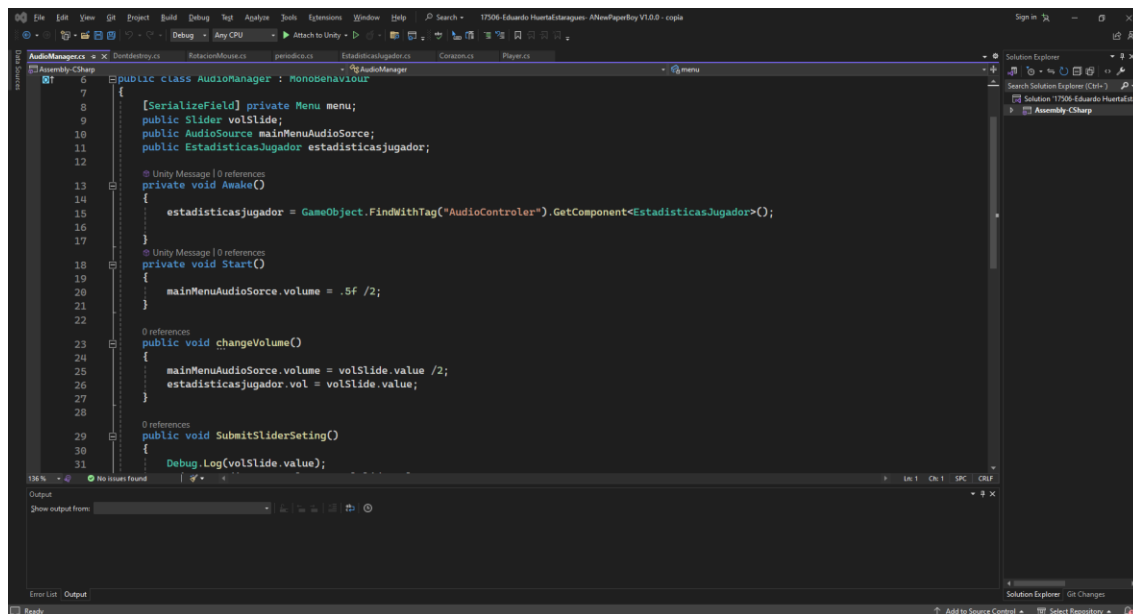



```

21 void Start()
22 {
23     _throw = GetComponent();
24     _throw.volume = estadisticasJugador.vol / 2f;
25     rigid.velocity = transform.forward * velocidad;
26     _throw.Play();
27 }
28
29 // Unity Message | 0 references
30 private void OnCollisionEnter(Collision collision)
31 {
32     Destroy(gameObject);
33 }
34 // Update is called once per frame
35 // Unity Message | 0 references
36 void Update()
37 {
38     life += Time.deltaTime;
39     modeloPeriodico.transform.Rotate(velRotacion, 0, 0);
40     if (life > 5f)
41     {
42         Destroy(gameObject);
43     }
44 }
45

```

Se destruyen los elementos, en vez de activar y desactivarlos. Generando mayor carga y tención en el sistema. Por lo que se plantea realizar un cambio a sistema de pull de obetos.

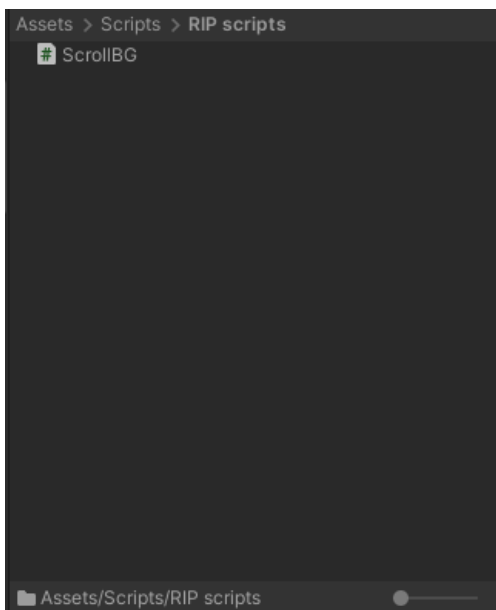
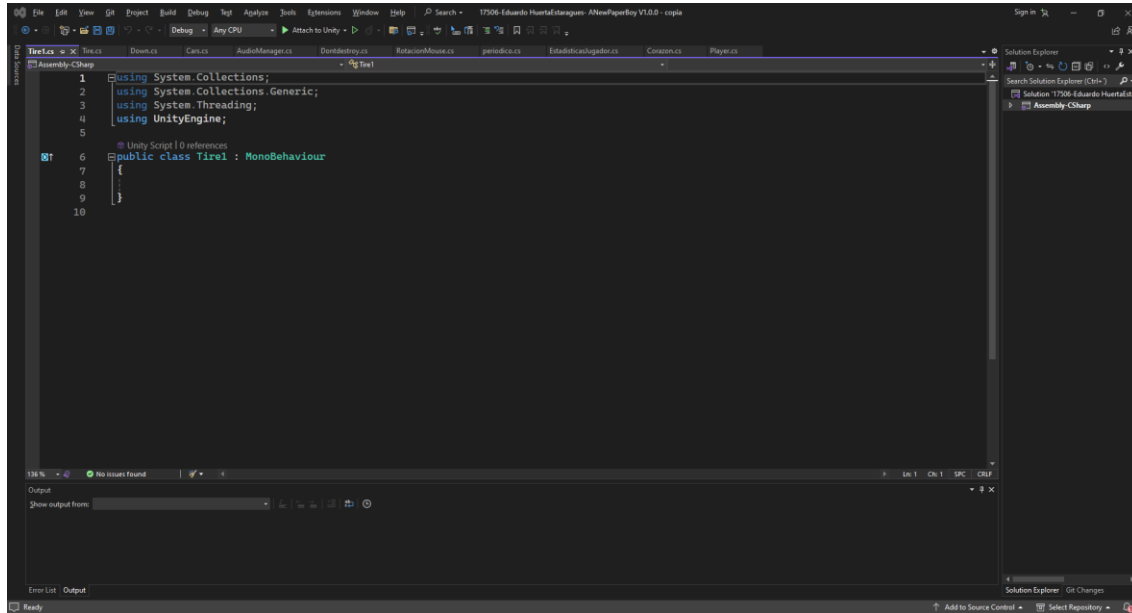


```

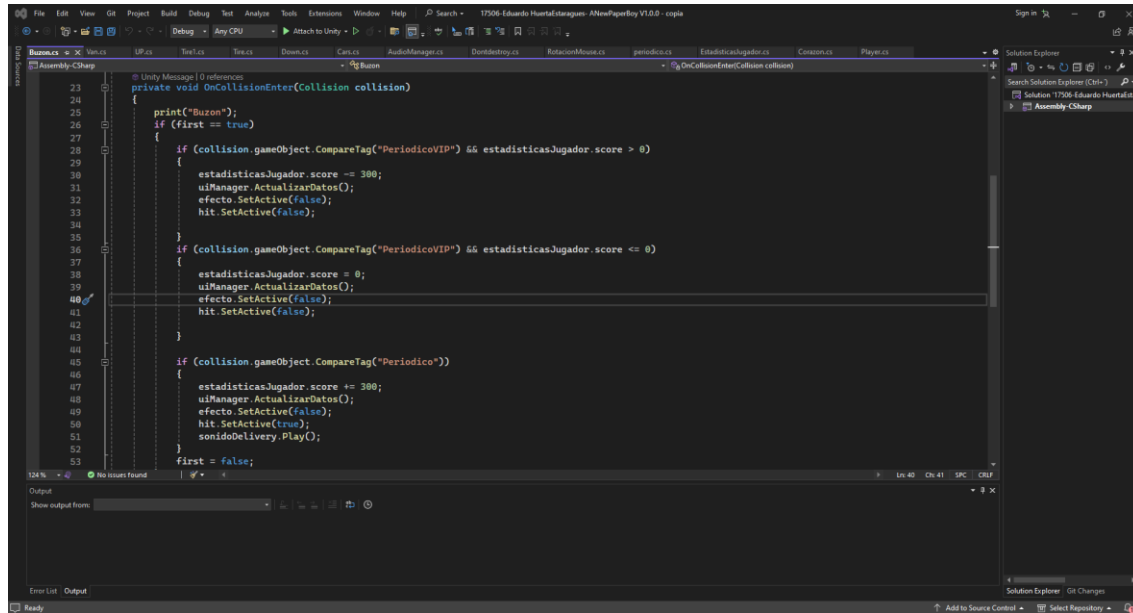
7 public class AudioManager : MonoBehaviour
8 {
9     [SerializeField] private Menu menu;
10    public Slider volSlide;
11    public AudioSource mainMenuAudioSource;
12    public EstadisticasJugador estadisticasJugador;
13
14    // Unity Message | 0 references
15    private void Awake()
16    {
17        estadisticasJugador = GameObject.FindWithTag("AudioController").GetComponent<EstadisticasJugador>();
18    }
19    // Unity Message | 0 references
20    private void Start()
21    {
22        mainMenuAudioSource.volume = .5f / 2;
23    }
24
25    // references
26    public void changeVolume()
27    {
28        mainMenuAudioSource.volume = volSlide.value / 2;
29        estadisticasJugador.vol = volSlide.value;
30    }
31
32    // references
33    public void SubmitSliderSetting()
34    {
35        Debug.Log(volSlide.value);
36    }
37

```

En esta parte se busca el manejador de audio, mediante solicitar el tag. El cual se llama una vez al inicializar el código mediante el awake, sin embargo, es in proceso algo pesado, por lo que en proyectos más grandes podría tomar mucho tiempo, por lo que sería mejor solicitar este recurso mediante otra forma más eficiente; como Types. En vista de brindar solución a problemas futuros.



Igual se cuenta con scripts vacíos o no empleados, los cuales requieren ser cargado y no permiten un funcionamiento optimo pues deben extraerse del almacenamiento para la ejecución. De igual forma assets no empleados o esenciales, llenan la memoria y generan más sobre carga del sistema y en caso de requerir buscar objetos generan que se requieran más iteraciones.

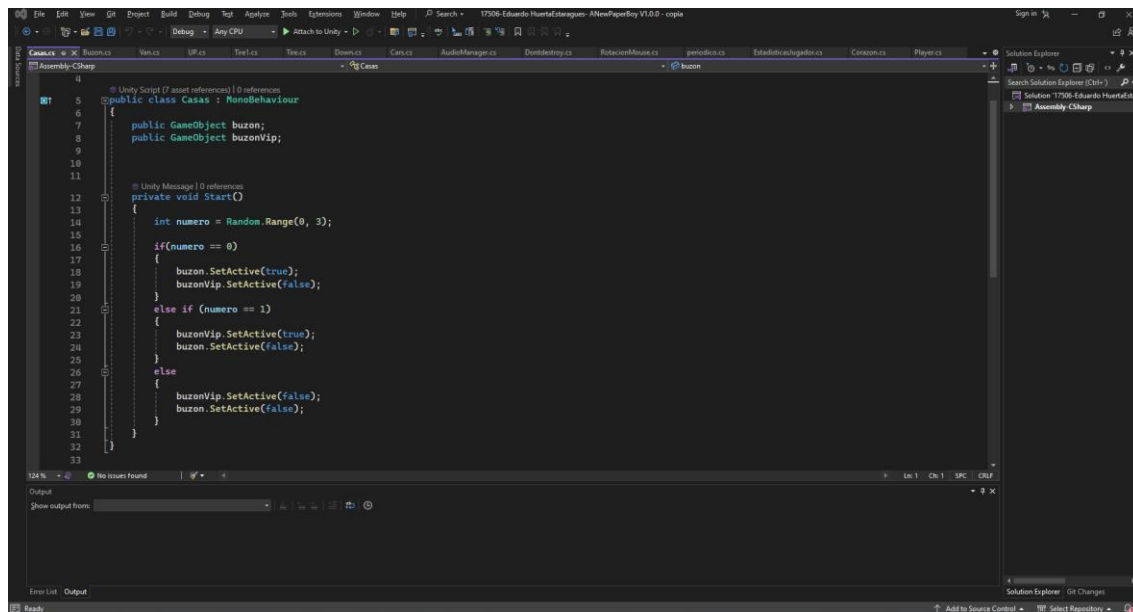


```

23 private void OnCollisionEnter(Collision collision)
24 {
25     print("Buzon");
26     if (first == true)
27     {
28         if (collision.gameObject.CompareTag("PeriodicoVIP") && estadisticasJugador.score > 0)
29         {
30             estadisticasJugador.score -= 300;
31             uiManager.ActualizarDatos();
32             efecto.SetActive(false);
33             hit.SetActive(false);
34         }
35     }
36     if (collision.gameObject.CompareTag("PeriodicoVIP") && estadisticasJugador.score <= 0)
37     {
38         estadisticasJugador.score = 0;
39         uiManager.ActualizarDatos();
40         efecto.SetActive(false);
41         hit.SetActive(false);
42     }
43     if (collision.gameObject.CompareTag("Periodico"))
44     {
45         estadisticasJugador.score += 300;
46         uiManager.ActualizarDatos();
47         efecto.SetActive(false);
48         hit.SetActive(true);
49         sonidoDelivery.Play();
50         first = false;
51     }
52 }
53

```

En estos, se tiene un sistema de control de colisiones, mediante puros tags, haciendo por ende que este código consuma gran parte de los recursos generales, como se podía ver en la gráfica de análisis de recursos y procesos.

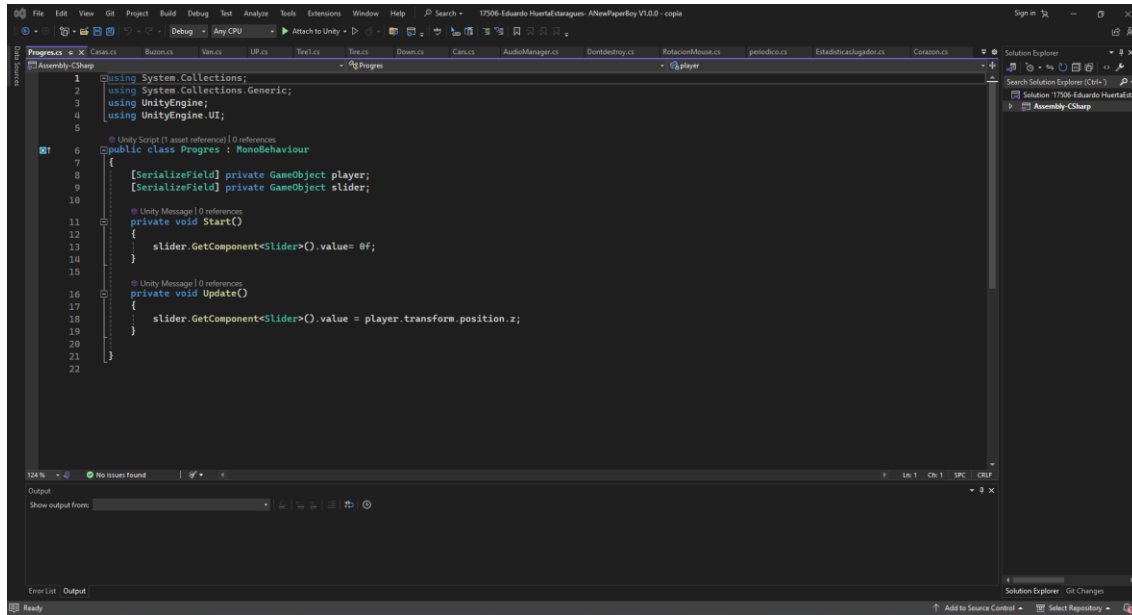


```

5 public class Casas : MonoBehaviour
6 {
7     public GameObject buzon;
8     public GameObject buzonVip;
9
10
11
12 private void Start()
13 {
14     int numero = Random.Range(0, 3);
15
16     if (numero == 0)
17     {
18         buzon.SetActive(true);
19         buzonVip.SetActive(false);
20     }
21     else if (numero == 1)
22     {
23         buzonVip.SetActive(true);
24         buzon.SetActive(false);
25     }
26     else
27     {
28         buzonVip.SetActive(false);
29         buzon.SetActive(false);
30     }
31 }
32
33

```

El emplear void Awake, en los sistemas de inicialización son importante para una generación más continua y acción más rápida.

```

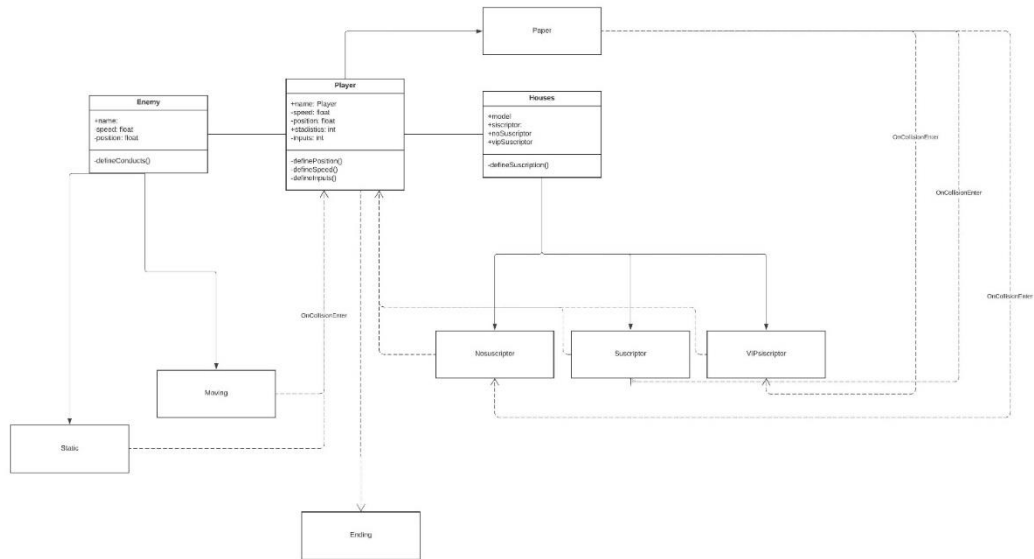
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.UI;
5
6 public class Progres : MonoBehaviour
7 {
8     [SerializeField] private GameObject player;
9     [SerializeField] private GameObject slider;
10
11     private void Start()
12     {
13         slider.GetComponent<Slider>().value = 0f;
14     }
15
16     private void Update()
17     {
18         slider.GetComponent<Slider>().value = player.transform.position.z;
19     }
20
21 }
22

```

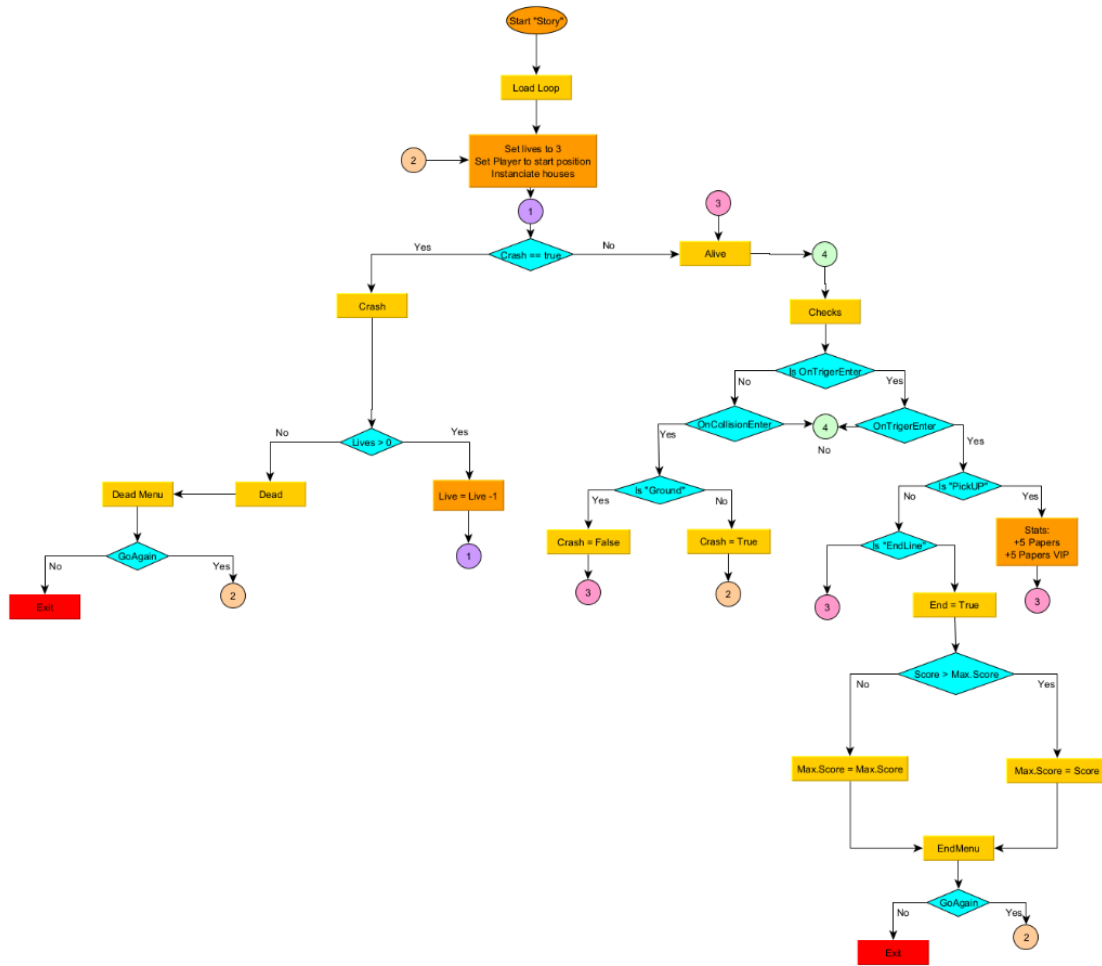
En este caso, se podría realizar un sistema, para actualizar esta interfaz cada cierto tiempo controlado, y no cada fotograma reduciendo la carga y ciclos por ejecución.

En general se debe buscar reducir los requisitos de memoria, como la cantidad de elementos por renderizar. Mejorar rendimiento mediante sistemas eficientes y sistemas de activación y desactivar elementos, en lugar de crearlos.

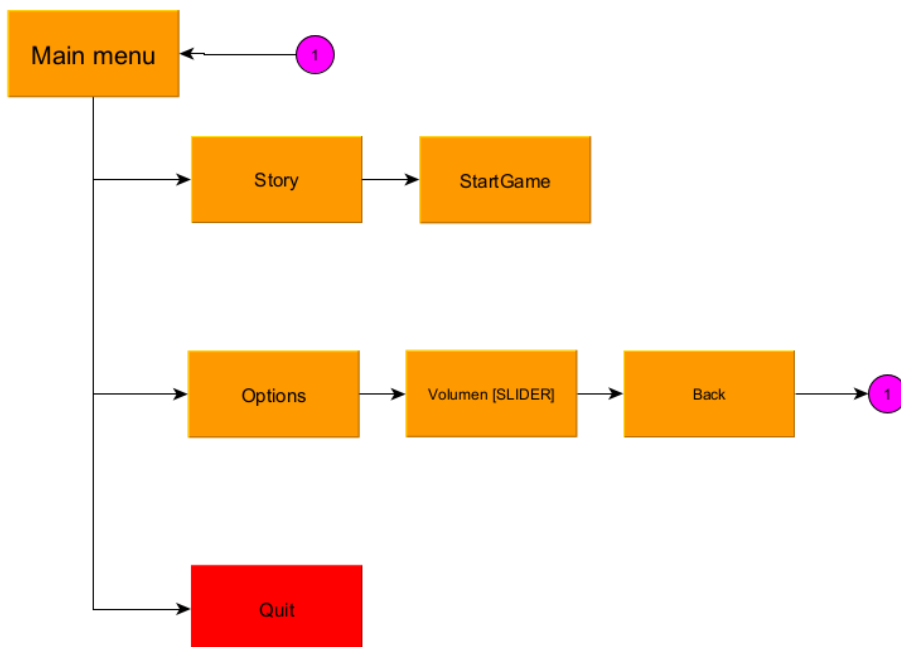
Este proyecto, corre alrededor de los 200fps, mientras que si llega a caer a los 60. Lo cual es algo que se debe checar, por el momento creo es mayormente cosa del animador, pero hay procesos catalogados como otros, que no sé porque consumen tantos recursos del sistema. En los siguientes diagramas se hace una representación visual del flujo del código, para mejor entendimiento y guía a futuro en la solución, al contar con conocimientos más amplios y mejor idea hacia la optimización de los programas.



(Huerta, 2023)

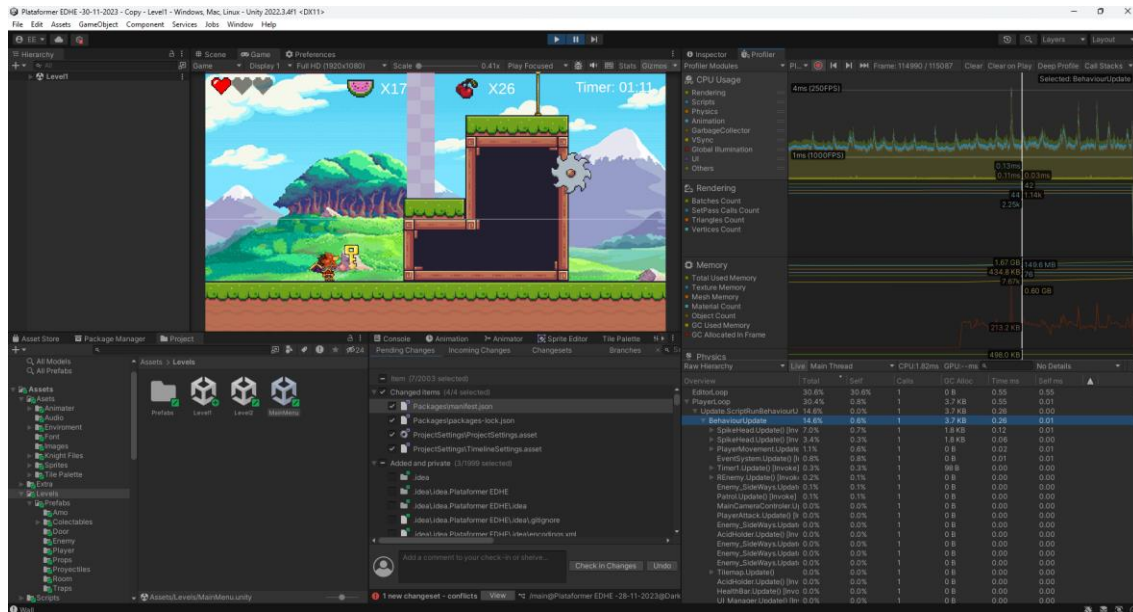


(Huerta, 2023)



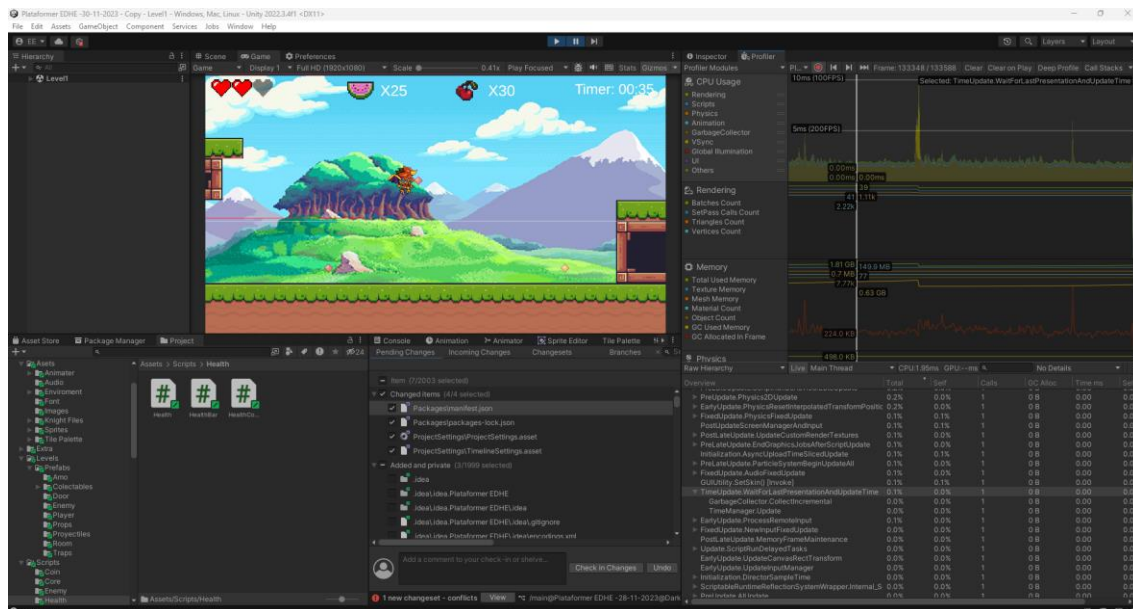
(Huerta, 2023)

Desempeño - Proyecto Trimestre 2

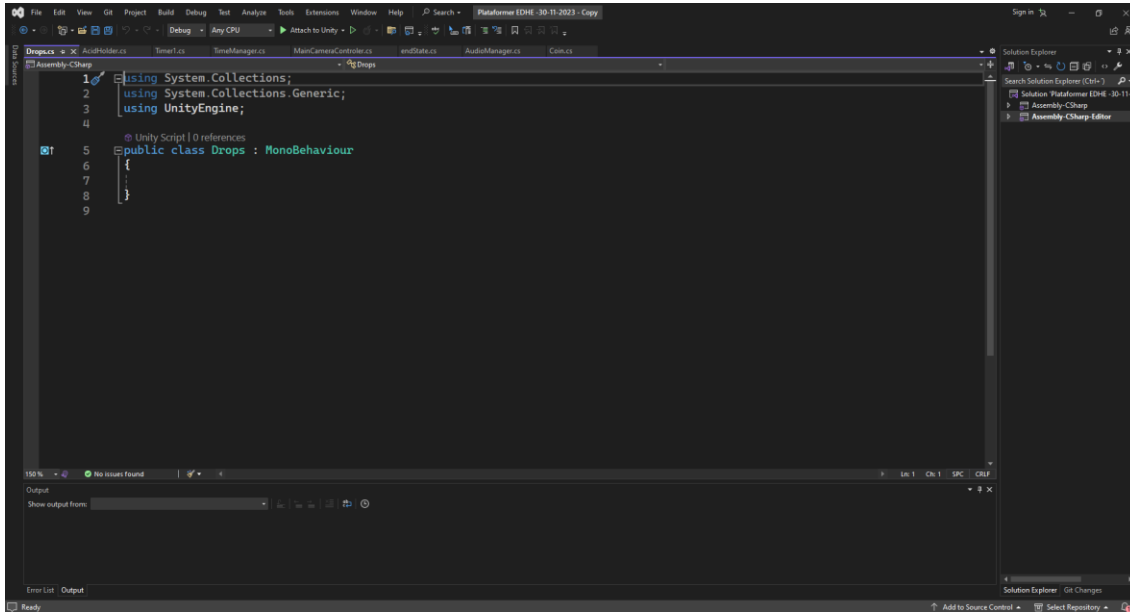


En este segundo proyecto se tienen sistemas mucho más simples a la vez que mucho más pulidos y controlados, resultando en un rendimiento mucho mayor, además del echo del proyecto ser 2D.

Para lograr este mejor desempeño se emplean lo que vienen siendo Tile Maps, en lugar de objetos, a la vez que se tiene más control sobre el cuándo se llaman las funciones de los objetos, evitando ciclos innecesarios.



Sin embargo, encontré este pico muy pronunciado que reduce los cuadros de cerca de los 1000 a 100. Sin embargo, no he podido encontrar de donde provienen este ciclo de Update.

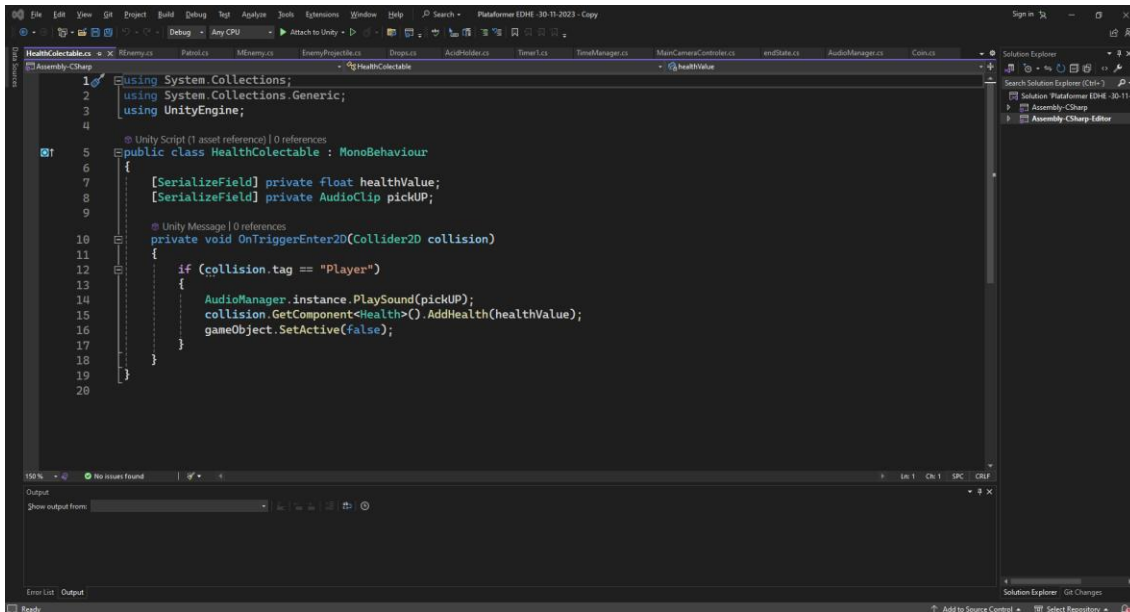


```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Drops : MonoBehaviour
6 {
7 }
8
9

```

De igual forma se cuentan con scripts y ascetas que no están siendo activamente empleados, resultando en un consumo de carga y o recursos y necesario. Pese a ser ligeros, es importante mantener el sistema libre de este tipo de archivos para poder brindar la mejor experiencia y rendimiento a la vez que genera la buena práctica, al momento de trabajar en proyectos más grandes.



```

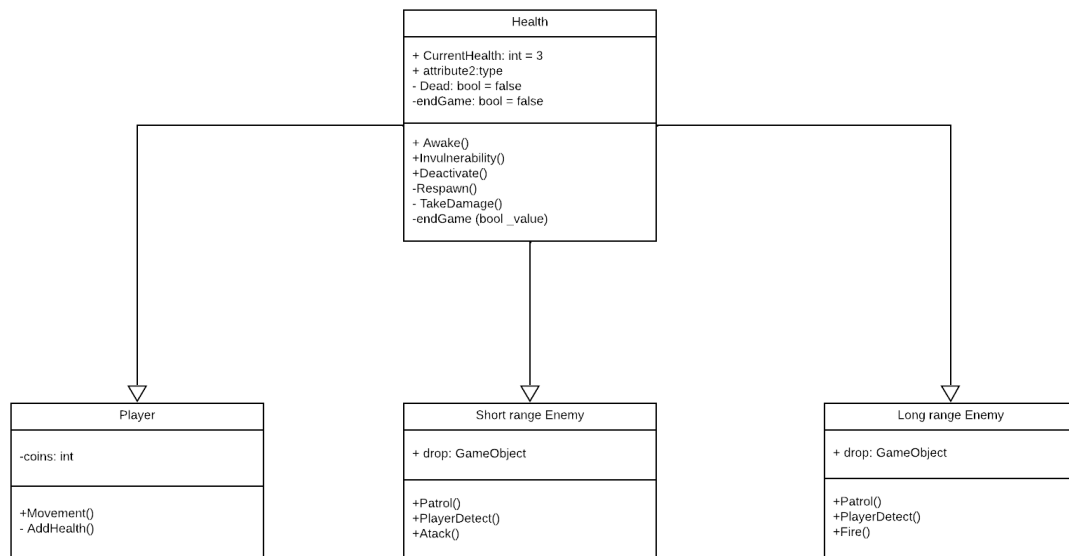
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class HealthCollectable : MonoBehaviour
6 {
7     [SerializeField] private float healthValue;
8     [SerializeField] private AudioClip pickUP;
9
10    private void OnTriggerEnter2D(Collider2D collision)
11    {
12        if (collision.tag == "Player")
13        {
14            AudioManager.instance.PlaySound(pickUP);
15            collision.GetComponent<Health>().AddHealth(healthValue);
16            gameObject.SetActive(false);
17        }
18    }
19 }
20
21

```

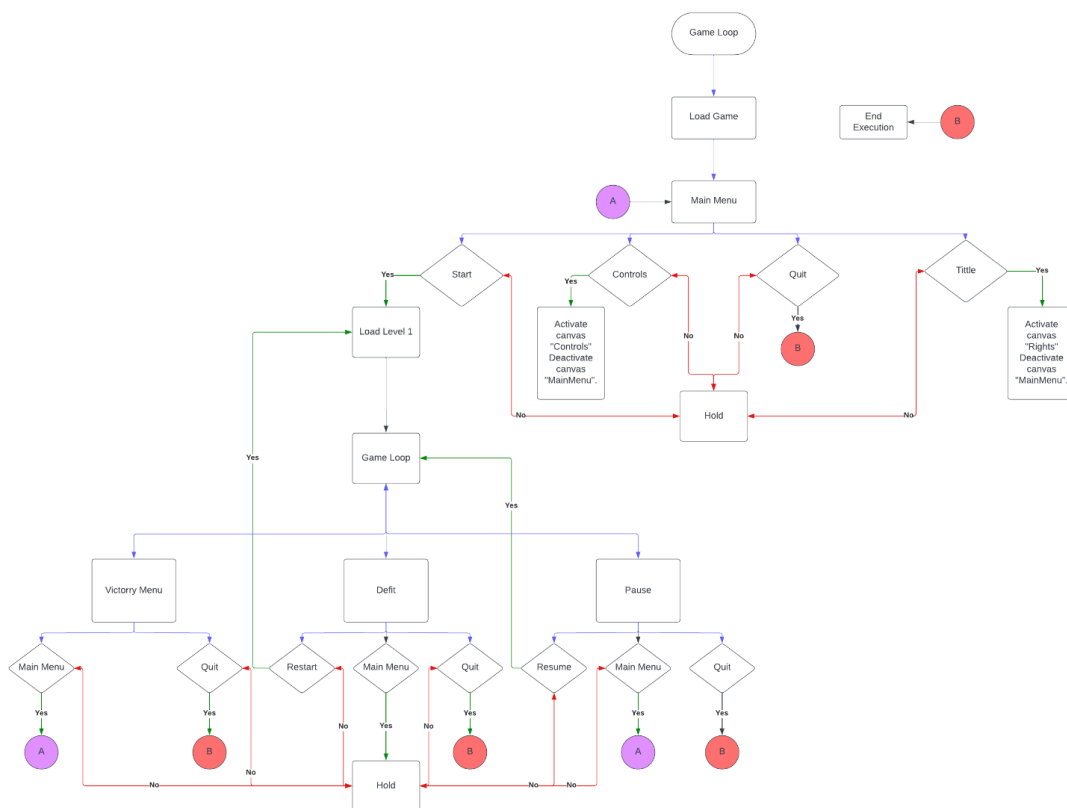
Se emplean en ocasiones sistemas de detección de colisión por tags, los cuales no parecen afectar de forma notable el desempeño, sin embargo, seria optimo buscar y consultar si existe una mejor solución que el uso de tags.



De forma general, este es un proyecto ampliamente pulido, a la vez que este es mucho más simple y menos ambicioso en algunos sentidos. Sin embargo, tiene cosas que podrían estar deteniendo su rendimiento y no son aparentes en este punto del análisis. En general este proyecto corre próximo a los 1000fps, sin embargo, llega a tener caídas hasta los 100, siendo una diferencia muy grande a la cual se le debe indagar más, con más conocimiento.



(Huerta, 2023b)



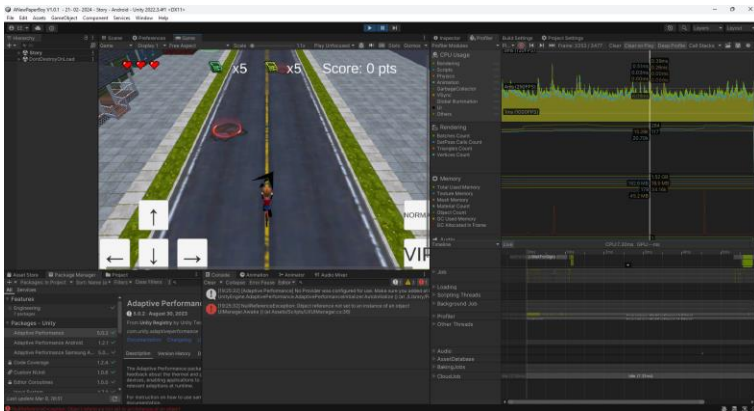
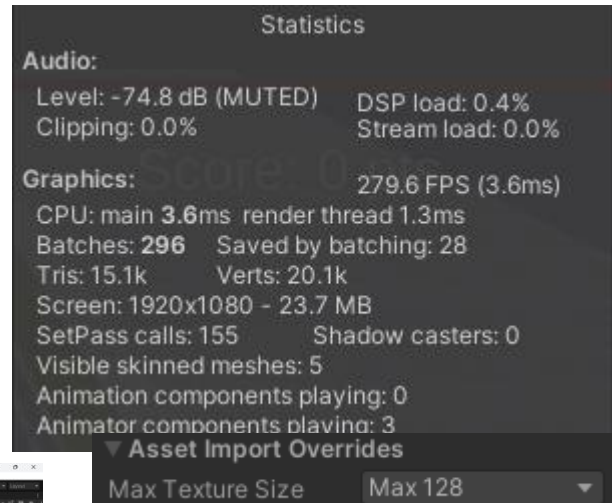
(Huerta, 2023b)

Resultados - Proyecto Trimestre 1

FPS:

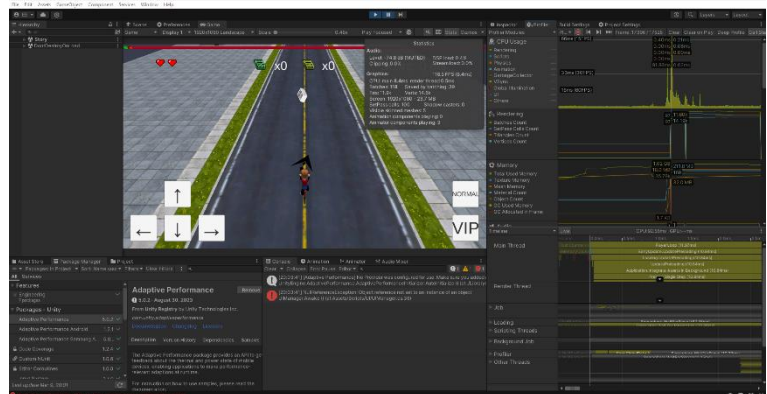
En los puntos más bajos el juego ronda a los 136FPS, sin embargo, en promedio está en los 279fps, Por lo que se esta teniendo un 90%+ de mejora respecto a la versión pasada. Esto se logro alterando ligeramente la lógica de los sistemas del Jugador, removiendo partes que no aportaban al sistema, a la vez que retirar sistemas que tomaba muchos pasos y preguntas para ser ejecutados, mediante el uso de "returns".

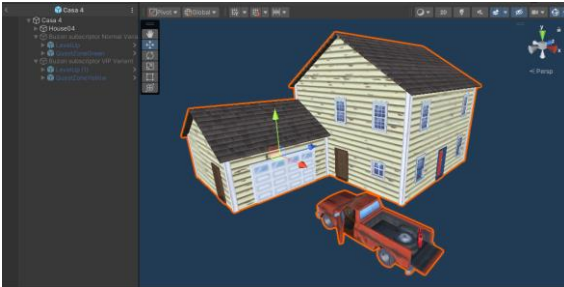
Otra parte que se implemento fue remover o reducir elementos gráficos que no aportaban al jugador o que en un juego móvil no son útiles. Como lo son sombras en tiempo real, reducir número de partículas, reducir tamaño de texturas, etc...



Tiempos de carga:

En la versión anterior tomaba alrededor de 8s o más, el paso del menú inicial al juego. A la vez que se tenían errores gráficos al seguir cargando elementos. Para solventar esta situación se recurrió a: LoadSceneAsync, a la vez que gran parte del proceso era debido al colider de los assets empleados. Por lo que se recurrió a remover los colideres iniciales y colocar colideres generales a los objetos al no tener una interacción cercana con ellos. A la vez que se mantienen apagados algunos objetos asta que se requieren para reducir la carga de elementos. Con esto se logró reducir a una carga virtualmente inexistente en PC y de unos 5s en móvil.





```
public void Story ()
{
    SceneManager.LoadSceneAsync("Story");
}
```

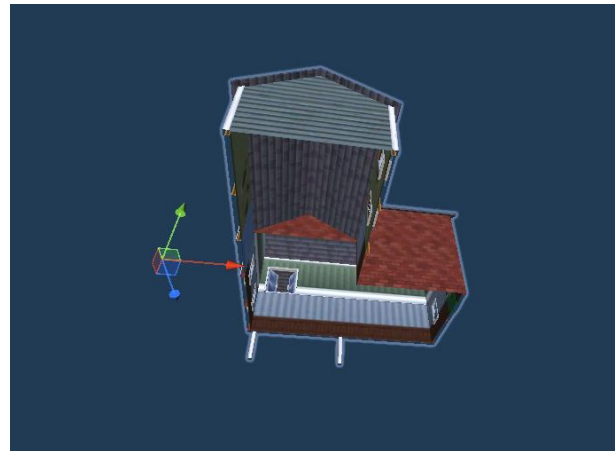
Estabilidad:

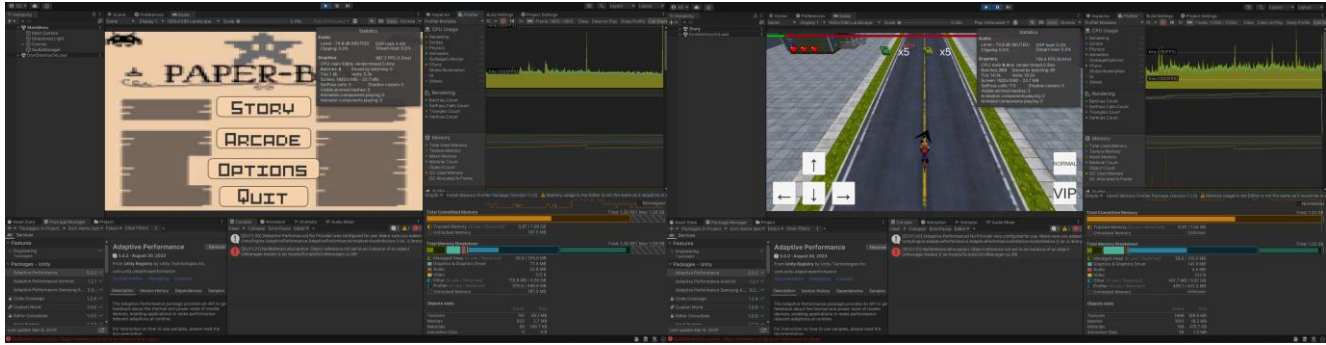
En la versión anterior, los FPS no eran para nada estables a la vez que el consumo de memoria era muy variante. Debido a que todos los elementos que se lanzaban o aparecían en escena, eran Instanciados al momento, lo cual requiere un llamado a memoria, agregar el elemento y todos sus componentes. Por lo que se cambio a un Pull, el cual enciende y mueve elementos existentes en la escena, siendo mas eficiente, constante y poniendo menos estrés en los sistemas.



RAM:

Para este apartado se logro reducir el consumo de memoria debajo de 1G, gracias a eliminar elementos no vivibles en escena, remover elementos no usados, remover scripts vacíos, eliminar Assets no empleados, reducir resolución de texturas. Esto permite un mayor rendimiento a la vez que va a permitir a futuro realizar proyectos mas grandes con este tipo de factores en mente.





0.67GB En menu

0.97G In game

Código:

Si bien, este es menos pesado y recursivo. Sigue siendo difícil de leer, pero es un problema de leer. Sin embargo, este es un problema de estructura que viene desde concepción del proyecto. Hecho por el cual se debería separar más en sistemas como lo es el segundo proyecto. Pues es difícil de leer, modificar e incluso de plasmar en diagramas pues por como esta todo pende de Player, generando un ligero código espagueti.

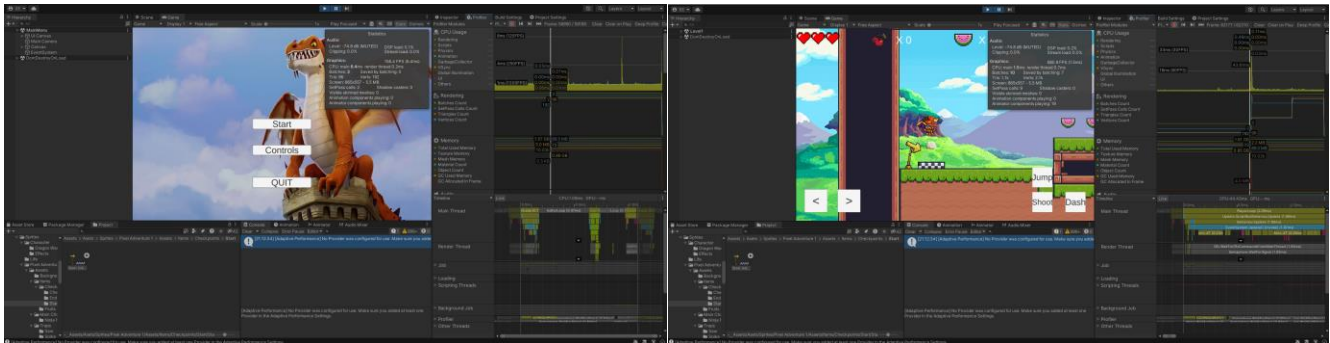
Resultados - Proyecto Trimestre 2

El segundo proyecto, es el que es un verdadero enigma y problema para mi

Pues este está bastante optimizado para empezar, de lo único que pecaba, era de tener aseos sin emplear y scripts redundantes. Sin embargo, no tenían un gran impacto en el desempeño. Hecho por el cual, pese a haber removido todo eso aun así el juego mantiene prácticamente el mismo desempeño y al ser pixel art el intentar comprimir las texturas lo hace más pesado.

No termino de comprender que es lo que está consumiendo todo el almacenamiento. Sin embargo, los demás sistemas al solo llamarse al ser requeridos resultan muy eficientes y rápidos. Permitiendo mantener el juego sobre los 500FPS, con cargas inmediatas y con un consumo de recursos muy reducidos (exceptuando memoria).

Siento en este caso, el sistema está a la altura de mis conocimientos, sin embargo, no descarto el poder mejorarlo más con más conocimientos y en un futuro intentarlo. Pues ambos proyectos ya integran lo visto y aprendido del sistema de prueba realizado para la segunda entrega.



Al cargar y ejecución tiene picos de parte del player, al este ser el centro o check de muchos métodos, sin embargo, el separarlo en otros elementos resulta en cambiar más elementos de lógica, y por el tiempo dado no valía la pena el trabajo implicado por la posible ganancia. El modificar la lógica completa de ninguno de los proyectos fue una opción viable ni mucho menos que valiera la pena dada el tiempo que se le pudo dedicar al proyecto.



Conclusión

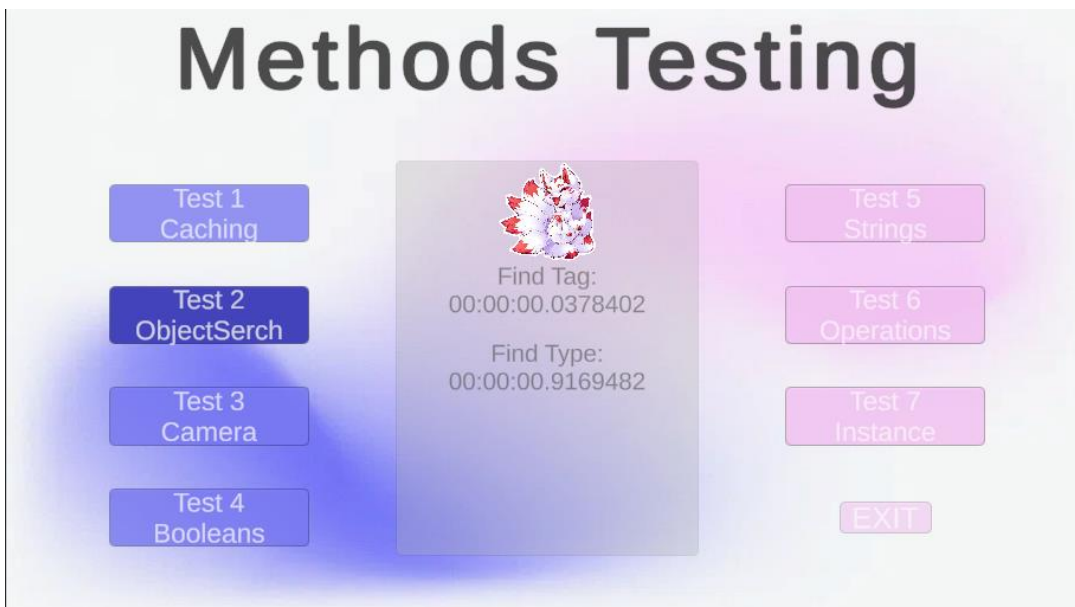
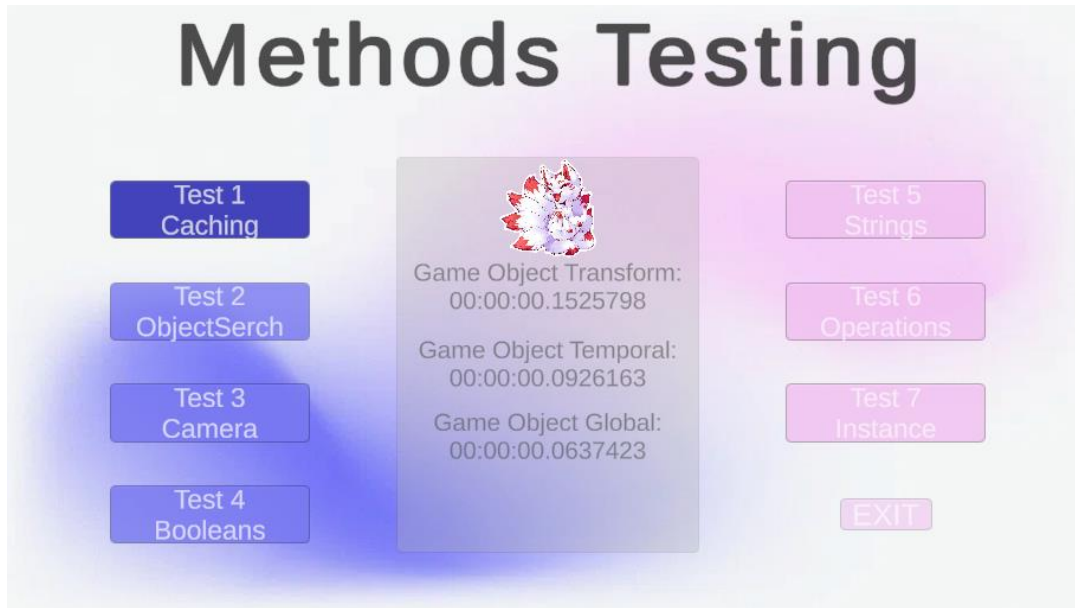
En videojuegos no importa solo la potencia de tu sistema, ni el motor gráfico. Sino que el entendimiento de código, programas, estructuras, patrones y el mismo motor gráfico. Pues como vimos, incluso el orden de operación afecta el tiempo de ejecución de un código. A la vez que hay muchas formas de hacer una misma cosa sin embargo no son equivalentes.

Cosas tan simples como tener elementos activos o no en jerarquía tienen un gran impacto, a la vez que hechos tal como lo son la forma de llamado de una función puede cambiar debido al funcionamiento de ciertos métodos o procesos.

A si mismo me sorprendió como incluso el orden operativo tiene un impacto, al estos correr izquierda a derecha, multiplicar por vectores requiere 3 operaciones por lo que es mejor operar primero eneros y posterior esos elementos especiales.

Si quisiera visitar estos proyectos en un futuro y poder mejorarlos de tal manera que sean mas eficientes a la vez que verlos con una mente más fría y no únicamente viendo por la fecha límite.

Test – Impacto en desempeño



Methods Testing

Test 1
Caching

Test 2
ObjectSerch

Test 3
Camera

Test 4
Booleans



Camera Temporal:
00:00:00.0329160

Camera Tag:
00:00:00.0853157

Camera GameObject:
00:00:00.2061977

Camera Global:
00:00:00.0179530

Test 5
Strings

Test 6
Operations

Test 7
Instance

EXIT

Methods Testing

Test 1
Caching

Test 2
ObjectSerch

Test 3
Camera

Test 4
Booleans



Bool For:
00:00:00.0039139

Bool While:
00:00:00.0032905

Test 5
Strings

Test 6
Operations

Test 7
Instance

EXIT



Methods Testing

Test 1
Caching

Test 2
ObjectSerch

Test 3
Camera

Test 4
Booleans



String Add:
00:00:00.0008841

String Builder:
00:00:00.0000614

Test 5
Strings

Test 6
Operations

Test 7
Instance

EXIT

Methods Testing

Test 1
Caching

Test 2
ObjectSerch

Test 3
Camera

Test 4
Booleans



F*F*V:
00:00:00.0044174

F*V*F:
00:00:00.0093852

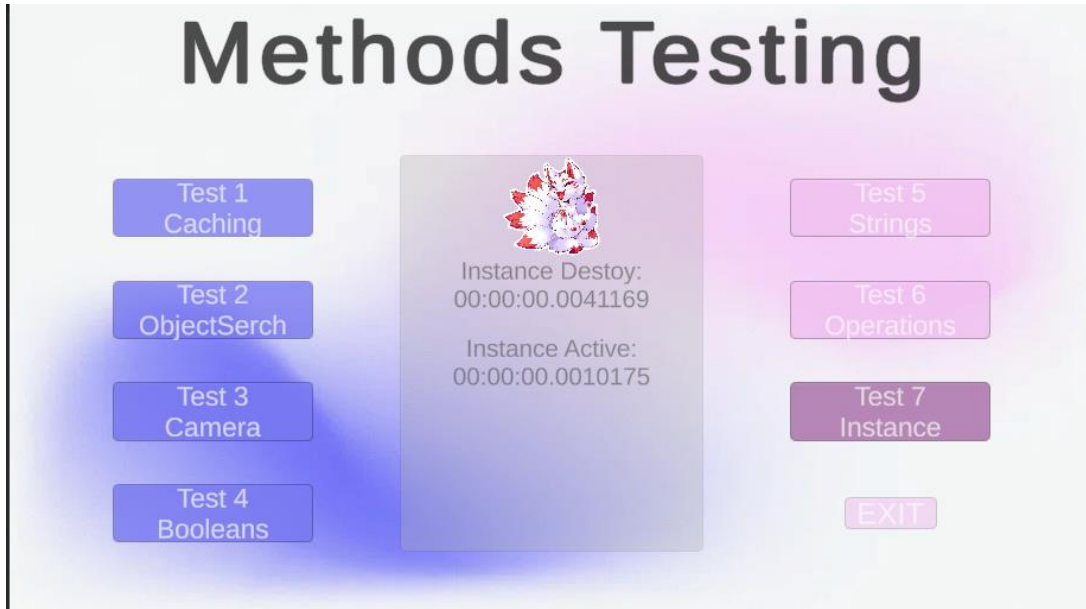
V*F*F:
00:00:00.0094354

Test 5
Strings

Test 6
Operations

Test 7
Instance

EXIT



Si bien, ninguno de estos da una solución a algo que salve o mate un proyecto. Da a ver como la forma en la que se hacen las cosas tiene un impacto en el resultado final, si bien en proyectos pequeños no hay tanto problema, conforme crecen nos proyectos, estas cosas tienen un gran impacto en el resultado final.

De los más importantes podemos ver:

- Destruir objetos toma más tiempo y recursos que solo activarlos y desactivarlos.
- Orden de operar con Vectores afecta el tiempo de operación.
- Un string bulider es mucho mas rápido que agregar a un string.
- Variables temporales o globales para lectura de varios datos de un objeto, es mucho más rápido que hacerlo cada llamado.
- Etc

Como estos ejemplos hay cientos, que muestran como todo tiene un impacto directo en el producto final.



APA:

A New Paper Boy:

- Huerta, E. (2023). *A new Paper boy: Portafolio Compilado*.
- Fuentes GDD:
 - Creately. (n.d.). <https://app.creately.com/d/start/dashboard>
 - slobulus. (2022, June 18). *PAPERBOY (Megadrive / SNES / Arcade) - Gameplay en Español || MORRALLA CLÁSICA* [Video]. YouTube. https://www.youtube.com/watch?v=ZGRyQw4g_Ck
 - GDC. (2019, May 1). *Classic Game Postmortem: Paperboy* [Video]. YouTube. <https://www.youtube.com/watch?v=MREsJsRZqu0>
 - Games Library. (n.d.). Arcade Club. <https://www.arcadeclub.co.uk/games/paperboy/>
 - 1.1: Quizizz — *The world's most engaging learning platform*. (n.d.). <https://quizizz.com/admin/quiz/606b0d98a81f0e001b7c5b88/divide-rectangulos-en-mitades-tercios-y-cuartos>

Recursos Graficos:

- Imagenes:
 - Leyton, A. (2020, June 11). *Corazón*. Pinterest. Retrieved September 8, 2023, from <https://www.pinterest.com.mx/pin/841891724081783771/>
 - Wiki, C. T. Z. S. RingsLord. (n.d.). *Reddit - Dive into anything*. https://www.reddit.com/r/PixelArt/comments/aoemaf/newspaper_for_a_game_i_am_working_on/
 - (n.d.). Wheel. *Zero Sievert Wiki*. <https://zero-sievert.fandom.com/wiki/Wheel>
 - Ramos, N. (n.d.). *Isométrica pixel pila de periódico*. 123RF. https://es.123rf.com/photo_70666641_isom%C3%A9trica-pixel-pila-de-peri%C3%B3dico.html
 - *Tire Tracks*. (n.d.). Creative Fabrica Buscar: Retrieved September 8, 2023, from <https://www.creativefabrica.com/es/product/tire-tracks-2/>
 - xXdigital_insanityXx. (n.d.). *Reddit - Dive into anything*. https://www.reddit.com/r/PixelArt/comments/8u9hbm/road_tile_for_my_game_what_do_you_guys_think_i/
 - Kinggod. (2022). Descargar icono de flecha png, signo de flechas png, flechas negras png gratis. Vecteezy. <https://es.vecteezy.com/png/9351319-icono-de-flecha-png-flechas-signo-png-flechas-negras-png>
 - Yo. (2015, March 14). *Vector Bike*. Pinterest. <https://www.pinterest.es/pin/610237818251912564/>
 - WASD. (n.d.). WASD. <https://www.wasdlive.com/>
 - *Download Free PC MOUSE PNG transparent background and clipart*. (n.d.). TransparentPNG. <https://www.transparentpng.com/cats/pc-mouse-932.html>
 - Flaticon. (2020, May 6). *todas las direcciones Icon - 2926322*. https://www.flaticon.es/icono-gratis/todas-las-direcciones_2926322



- Efectos
 - *Action RPG FX | VFX Particles* | *Unity Asset Store*. (2015, June 30). Unity Asset Store. <https://assetstore.unity.com/packages/vfx/particles/action-rpg-fx-38222>
- 3D Models
 - *SurroundDead - Survival Game Assets | 3D Environments* | *Unity Asset Store*. (2020, March 24). Unity Asset Store. <https://assetstore.unity.com/packages/3d/environments/surrounddead-survival-game-assets-76276>
 - Tsunami. (2021). *Bike Animations Cartoon* – Free download. *Unity Asset Collection*. <https://unityassetcollection.com/bike-animations-cartoon-free-download/>
- Audios
 - Pixabay. (n.d.). *Free Beep Sound Effects download* - Pixabay. <https://pixabay.com/es/sound-effects/search/beep/>
 - Pixabay. (2023, April 14). *8-Bit Powerup* | *Música sin regalías*. Pixabay. <https://pixabay.com/es/sound-effects/8-bit-powerup-6768/>
 - Pixabay. (2023b, June 9). *8bit Music for Game* | *Música sin regalías*. Pixabay. <https://pixabay.com/es/sound-effects/8bit-music-for-game-68698/>
 - Pixabay. (2023b, April 17). *beep3* | *Música sin regalías*. Pixabay. <https://pixabay.com/es/sound-effects/beep3-98810/>
 - *Download free Bomb Sound Effects* | *MixKit*. (n.d.). <https://mixkit.co/free-sound-effects/bomb/>
 - Pixabay. (2023a, February 17). *Whoosh* | *Música sin regalías*. Pixabay. <https://pixabay.com/es/sound-effects/whoosh-6316/>



Dragon Plataformer:

- Huerta, E. (2023b). *Portafolio Platformer: GDD - Postmortem*.
- Metroidover, C. T. (s. f.). Metroid (videojuego). *Metroidover*. [https://metroid.fandom.com/es/wiki/Metroid_\(videojuego\)](https://metroid.fandom.com/es/wiki/Metroid_(videojuego))
- *DOOM eternal en steam*. (s. f.). <https://store.steampowered.com/agecheck/app/782330/?l=spanish>
- Porcel, F. (2017). Bestiary. *Calavera Studio*. <https://calavera.studio/en/bestiary/>
- *Figure 1: Metroid © Early Concept Art*. (s. f.). ResearchGate. https://www.researchgate.net/figure/Metroid-C-early-concept-art_fig1_235745133
- Camero, C. (2019, 24 diciembre). *Samus From Metroid Kandi Pattern*. Pinterest. <https://ar.pinterest.com/pin/36521446965143090/>
- Pngtree. (s. f.). *Elemento de efecto bola de energía de dibujos animados PNG*. <https://es.pngtree.com/so/bola-de-energia>
- *Entrega 6 del informe de Metroid Dread: Los entresijos del nuevo tráiler*. (s. f.). Nintendo of Europe GmbH. <https://www.nintendo.es/Noticias/2021/agosto/Entrega-6-del-informe-de-Metroid-Dread-Los-entresijos-del-nuevo-trailer-2030683.html>
- Metroidover, C. T. (s. f.-a). Golpe en carrera. *Metroidover*. https://metroid.fandom.com/es/wiki/Golpe_en_Carrera
- *Icono de vector de proyectil guiado de misiles balísticos de pixel art para juego de 8 bits sobre fondo blanco Premium Vector*. (2022, 27 junio). Freepik. https://www.freepik.es/vector-premium/icono-vector-proyectil-guiado-misiles-balisticos-pixel-art-juego-8-bits-sobre-fondo-blanco_28763297.htm
- Araújo, S. (2018). Esta web te permite crear divertidas burbujas de texto animadas 8-bit. *Genbeta*. <https://www.genbeta.com/web/esta-web-te-permite-crear-divertidas-burbujas-texto-animadas-8-bit>

Sprites, Arte y audio:

- *Pixel Adventure 1 | 2D Characters | Unity Asset Store*. (2019, October 30). Unity Asset Store. <https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360>
- *Pixel Art Platformer - Village Props | 2D Environments | Unity Asset Store*. (2020, September 18). Unity Asset Store. <https://assetstore.unity.com/packages/2d/environments/pixel-art-platformer-village-props-166114>
- *Knight Sprite Sheet (Free) | 2D Characters | Unity Asset Store*. (2018, February 15). Unity Asset Store. <https://assetstore.unity.com/packages/2d/characters/knight-sprite-sheet-free-93897>
- *Dragon Warrior (Free) | 2D Characters | Unity Asset Store*. (2023, August 8). Unity Asset Store. <https://assetstore.unity.com/packages/2d/characters/dragon-warrior-free-93896>
- Angsurat, P. (2023, August 21). *Download the Pixel art speech bubble 5 27517467 royalty free PNG from Vecteezy for your project and explore o. . . Vecteezy*. <https://www.vecteezy.com/png/27517467-pixel-art-speech-bubble-5>



- *Voice_sans.wav.* (n.d.). Dropbox.
https://www.dropbox.com/s/xz6gpiwszwy41rh/voice_sans.wav?dl=0
- *Voice_Papyrus.wav.* (n.d.). Dropbox.
https://www.dropbox.com/s/1o8ale36xk0vz1r/voice_Papyrus.wav?dl=0
- *Collecting resources and references for the Undertale community!* (n.d.). Tumblr.
<https://undertale-resources.tumblr.com/tagged/voices>
- Myinstants. (n.d.). *I have the power of god and anime - Botón de sonido.* Myinstants.
<https://www.myinstants.com/es/instant/i-have-the-power-of-god-and-anime-61345/>
- *Pixel Keyboard Keys - for UI by Dream Mix.* (n.d.). itch.io.
<https://dreammix.itch.io/keyboard-keys-for-ui>
- pixabay. (n.d.). *dorm door opening.* Pixabay. Retrieved December 1, 2023, from
<https://pixabay.com/es/sound-effects/dorm-door-opening-6038/>
- *RetroNator.* (n.d.). Tumblr. <https://www.retronator.com/>