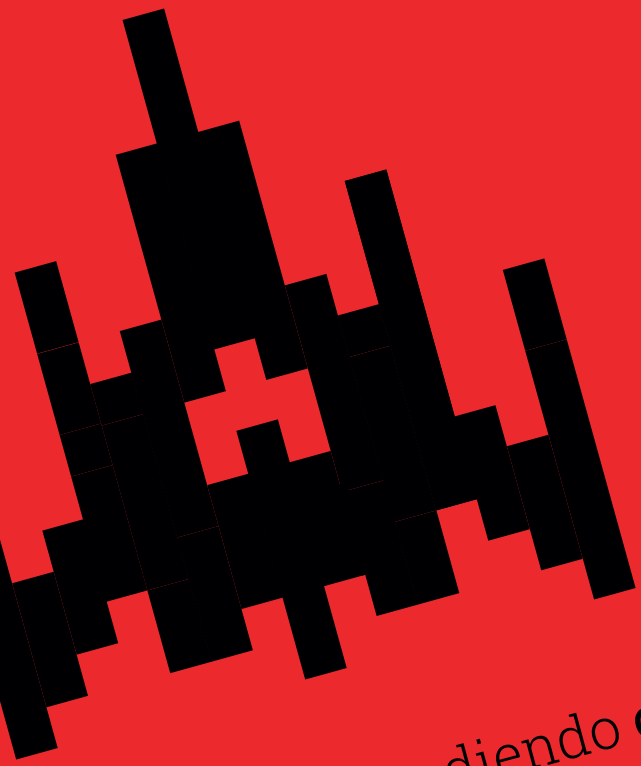
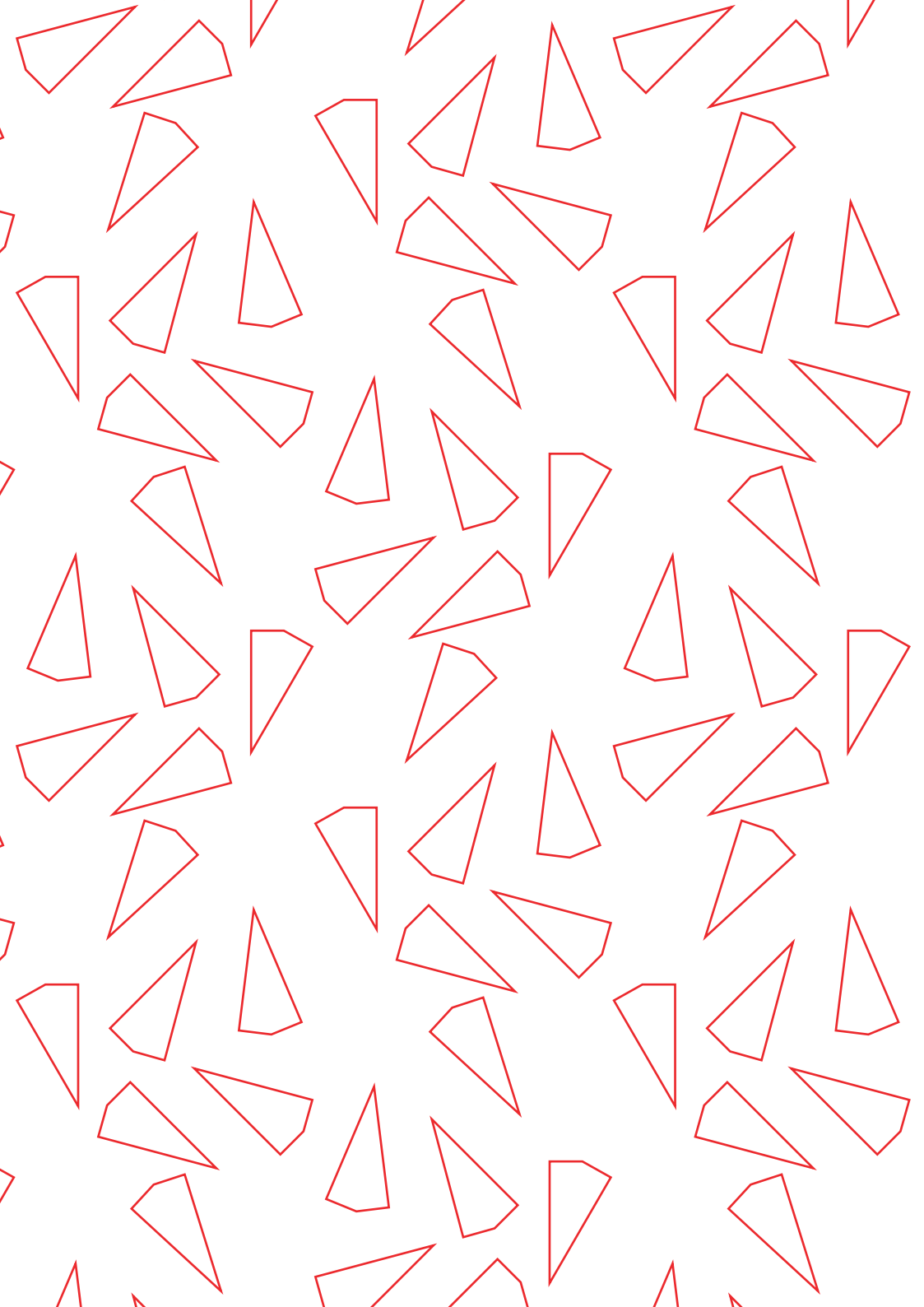
artem DISEÑO



Emprendiendo con **videojuegos**

TOMO 3

NIVELES, DESARROLLO Y PRUEBAS



Emprendiendo **con videojuegos**

TOMO 3:

Niveles, desarrollo y pruebas

EMPRENDIENDO CON VIDEOJUEGOS
Tomo 3: Niveles, desarrollo y pruebas

© del texto: Paúl Esteban Crespo Martínez, Roberto Landívar Feicán,
Verónica Rosales Moscoso, Oscar Vintimilla Ugalde, Andrés Patiño León, 2026
© de esta edición: Universidad del Azuay, Casa Editora, 2026

ISBN: 978-9942-54-187-1
e- ISBN: 978-9942-54-188-8
ePUB: 978-9942-54-189-5
ISBN de la Colección: 978-9942-54-183-3

Corrección de estilo: Bernardo López Bueno
Revisión pares: Salvador Bueno Ávila, María Dolores Gallego
Diseño y diagramación: David Jaramillo Carrasco-Cuarto Gráfico, Fernando Yukich.
Ilustración: Carlos Orellana, Sebastían Pesantez, Roberto Landívar y Esteban Crespo.

Impresión: PrintLab/Universidad del Azuay

Se prohíbe la reproducción total o parcial de esta obra, por cualquier medio, sin la autorización expresa del titular de los derechos

CONSEJO EDITORIAL/UNIVERSIDAD DEL AZUAY

Francisco Salgado Arteaga
Rector

Genoveva Malo Toral
Vicerrectora Académica

Raffaella Ansaloni
Vicerrectora de Investigaciones

Toa Tripaldi
Directora de la Casa Editora

Emprendiendo **con videojuegos**

TOMO 3:
Niveles, desarrollo y pruebas

Paúl Esteban Crespo-Martínez
Roberto Landívar-Feicán
Verónica Rosales-Moscoso
Oscar Vintimilla-Ugalde
Andrés Patiño-León

UNIVERSIDAD DEL AZUAY | Casa Editora


LOADING...

ÍNDICE

DESARROLLO DE VIDEOJUEGOS 11

LENGUAJES DE PROGRAMACIÓN 14

MOTORES DE JUEGO 20

COMPARACIÓN EN EL DESARROLLO DE VIDEOJUEGOS 22

CREACIÓN DE JUEGO USANDO PLANTILLAS DE UNITY 36

PRUEBAS 42

EL TESTING EN EL DESARROLLO DE VIDEOJUEGOS 44

TIPOS DE PRUEBAS 48

PRUEBAS DE RENDIMIENTO 57

VULNERABILIDADES 62

ESTRATEGIAS 66

PROTECCIÓN DE ACTIVOS 71

PROCESO DE PRUEBAS 75

BIBLIOGRAFÍA 91

DESARROLLO DE VIDEOJUEGOS

En los albores de la informática personal, los programadores independientes tenían pocas oportunidades de convertir sus ideas en videojuegos comerciales. Factores como el limitado acceso a la tecnología, la complejidad de los lenguajes de programación de la época, la ausencia de estándares de desarrollo o los costos asociados a la producción/distribución de los juegos; reducían el mercado a unas pocas compañías con incipientes recursos financieros. Esta situación fue cambiando a raíz de la popularización de las computadoras personales y la aparición de lenguajes de programación de alto nivel, lo que permitió a los desarrolladores emprender proyectos desde sus hogares, pequeñas compañías o incluso desde centros educativos, como sucedió con el afamado juego *Tetris* en el centro de computación Dorodnitsyn de la Academia de Ciencias de la Unión Soviética (Plank-Blasko, 2015).



Figura 1: *Shigeru Miyamoto*





Figura 2: *Crimson Desert y Grand Theft Auto V, desarrollados por Pearl Abyss y Rockstar Games*

La creación de un videojuego es un proyecto multidisciplinario que comprende una serie de características, entre las que podemos mencionar (McShaffry et al., 2013):

- Gráficos (Animación renderizado, gestión de formas, gestión de vectores, sombras, gestión de frames, gestión de escenas)
- Sonido
- Físicas (colisiones, gravedad, etc.)

- Gestión de memoria
- Interacción con sistemas de archivos y estructuras de datos
- Gestión de eventos
- Compatibilidad entre plataformas
- Rendimiento
- Procesamiento paralelo
- Networking/Streaming
- Inteligencia artificial
- Mensajería/Logging
- Mantenibilidad

En la actualidad (2023), los lenguajes de programación de alto nivel como C++, C#, Javascript, Python o Java; se utilizan en el desarrollo de videojuegos en conjunto con bibliotecas dedicadas o frameworks, sin embargo, se ha popularizado el uso de motores de juego como Unity o Unreal engine, los mismos que permiten abstraer características de videojuegos de forma que su código y elementos puedan ser reutilizados en nuevas aplicaciones (Goel & Semah, 2023). Componentes encargados de tareas como la renderización de gráficos, físicas de juego, gestión de redes o inteligencia artificial, son integrados en una capa de abstracción que proporciona un entorno integrado de desarrollo multiplataforma que satisface gran parte de los requerimientos de los desarrolladores (Andrade, 2015).

LENGUAJES DE PROGRAMACIÓN

C++

Desarrollado en 1979 en Bell Labs por Bjarne Stroustrup, C++ es un lenguaje compilado basado en C, que añade soporte para la gestión de clases y la programación orientada a objetos. Es un lenguaje ampliamente utilizado debido a su compatibilidad con distintas plataformas, así como la capacidad de gestión de recursos de hardware a bajo nivel, lo que le hace ideal para tareas con un uso exhaustivo de recursos. Al ser un lenguaje compilado, proporciona un mejor desempeño que lenguajes interpretados como LUA y Python.

C++ posee una gran cantidad de bibliotecas o frameworks para el desarrollo de videojuegos, entre las que se encuentran (Oliveira, 2022):

- **SFML (Simple and Fast Multimedia Library)**
  Proporciona una interface simple a los componentes de hardware para simplificar el desarrollo de juegos (2D) y aplicaciones multimedia. Está programado en C++ y proporciona interfaces oficiales a C y .NET, aunque su comunidad ha desarrollado versiones para Java, Ruby, Python y otros lenguajes. SFML está disponible en múltiples plataformas como Windows o Linux (SFML Team, n.d.)

- **SDL/SDL2 (Simple Direct Media Layer)**
  Es una biblioteca de desarrollo multi plataforma que proporciona un acceso de bajo nivel a componentes de hardware como audio, dispositivos de entrada o componentes gráficos a través de APIs como OpenGL o Direct3D. Tiene soporte para Windows, macOS, Linux, iOS

o Android. Está escrito en C y trabaja de forma nativa con C++, aunque existen versiones para otros lenguajes como Python (Lantinga, n.d.).

• **Raylib**
Es una biblioteca open source minimalista y de fácil aprendizaje, diseñada para la creación de videojuegos simples. Tiene soporte para plataformas compatibles con C/C++ y OpenGL, como Windows, Linux o MacOs, aunque su comunidad ha desarrollado versiones para distintos lenguajes de programación como C#, Lua o Python. El proyecto ha recibido varios premios por parte de compañías como Google y Epic (Santamaria, n.d.).

• **Allegro**
Es una biblioteca multiplataforma diseñada para la creación de videojuegos y multimedia. Permite controlar tareas de bajo nivel como la carga de datos, la gestión de imágenes o creación de ventanas. Soporta primitivas para la creación de gráficos usando OpenGL o Direct3D, además de que puede usarse con C++ y otros lenguajes. Su última versión estable fue lanzada en 2022 (Allegro Development Team, n.d.).

Debido a su capacidad para generar código nativo de cada plataforma, lo que incrementa la eficiencia en la ejecución de tareas que requieren un uso exhaustivo de recursos, C++ es ampliamente utilizado en el desarrollo de motores de juego. Entre algunos ejemplos tenemos:

• CryEngine
• Unreal Engine
• Torque3D
• Unity, en combinación con C#.

Generalmente estos motores gráficos, con excepción de Unity, utilizan C o C++ como lenguaje base para el desarrollo de aplicaciones o scripts dentro de los juegos.

Finalmente, otra tarea que demanda un gran consumo de recursos corresponde a la gestión de gráficos y la interacción con el hardware especializado. Es por eso que C++ representa una de las principales alternativas para la construcción de APIs (Application Programming Interface) orientadas a la gestión multimedia y la generación de gráficos en 2D y 3D. Bibliotecas como OpenGL (OpenGL - The Industry Standard for High Performance Graphics, n.d.) o Vulkan (Blanco, 2020; Vulkan, 2023), están programadas principalmente en este lenguaje.



Figura 3: Ejemplo de uso del motor gráfico REDEngine 3 con The Witcher III: Wild Hunt

JAVASCRIPT

Es un lenguaje de programación interpretado ligero, funcional y dinámico independiente de la plataforma, basado en prototipos y compatible con POO. Es principalmente usado para implementar características complejas en páginas web (Microsoft, 2023a).

Javascript, casi siempre en conjunto con HTML 5, permite generar aplicaciones para web que pueden publicarse fácilmente en otras plataformas móviles como Android e IOS. De allí que este lenguaje se ha popularizado para la construcción de videojuegos sencillos a través de motores como Godot (Godot Engine, n.d.), Construct (Scirra, n.d.), melon.js (MelonJS, n.d.) entre otros. También se apoya en WebGL para la gestión de gráficos en 3D.

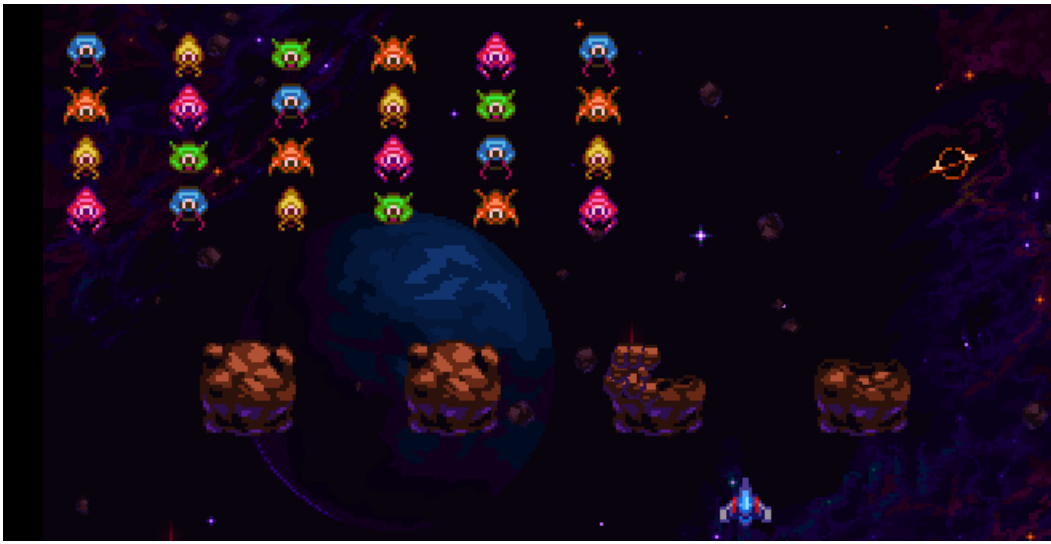


Figura 4: Ejemplo de juego generado con Construct
Fuente: De "Alien invation template" por Scirra, 2025 (https://www.construct.net/en/make-games/free-trial).

C#

Es un lenguaje de programación orientado a objetos, de tipeado seguro (type-safe) y con una estrecha relación con C, C++ y Java. El código generado por C# puede ejecutarse en el entorno .NET de Microsoft e interactuar con programas que usan las versiones .NET de Visual Basic, C++ y otros lenguajes (Microsoft, n.d.).

Una de las principales ventajas de C# es su compatibilidad multiplataforma, ya que sus programas pueden ejecutarse en varios sistemas operativos, lo que facilita su distribución. Por otra parte, es un lenguaje muy popular para los desarrolladores que trabajan en el entorno de Microsoft, lo que le ha permitido convertirse en uno de los lenguajes preferidos para el scripting en motores de juego como Unity (Unity Technologies, n.d.-a) .

Con C# es posible construir aplicaciones para Android y IOS, través de Xamarin, e interactuar con APIs gráficas como DirectX o Vulkan usando envoltorios (Microsoft, 2023b).

PYTHON

Es un lenguaje de programación interpretado, orientado a objetos y de propósito general; que se ha popularizado por su facilidad de aprendizaje, lectura simple y compatibilidad con distintas plataformas (Python Institute, n.d.). En la actualidad Python es el lenguaje más popular del mundo (TIOBE, 2023) y es ampliamente usado en áreas como la ciencia de datos, inteligencia artificial y programación web.

Debido a su naturaleza, los programas de Python tienen un menor rendimiento y velocidad que aquellos implementados en C++, por lo que generalmente no es un lenguaje usado para juegos con elevados requerimientos técnicos, sin embar-

go, es posible combinar ambos lenguajes ejecutando invocaciones desde Python hacia código de C++, para aprovechar las ventajas del último (Renard, 2022) (Kozacz, n.d.). Entre algunas de las herramientas disponibles para el desarrollo de juegos con Python tenemos a PyGame, Pyglet y Panda3D.

PYGAME

Es una de las principales bibliotecas para el desarrollo de videojuegos con Python e incluye un conjunto de módulos que permiten la creación de programas multimedia y juegos. Utiliza código ensamblador y C en funciones que requieren gran procesamiento, lo que permite ejecutar el código con una velocidad de entre 10 y 20 veces superior (Pygame.org, n.d.).

PYGLET

Biblioteca escrita íntegramente en Python que permite la creación de videojuegos y aplicaciones visuales complejas. Gestiona elementos como ventanas, imágenes, videos, sonido, dispositivos de control (teclado, mouse), etc. No requiere dependencias externas y ofrece enlaces directos a OpenGL para usuarios avanzados (Pyglet.org, n.d.).

PANDA3D

Inicialmente desarrollada por Disney, es una máquina 3D de código abierto y libre uso que permite el renderizado y la creación de juegos tridimensionales, así como simulaciones o experimentos. Aunque está programada en C++, pueden implementarse programas que exploten sus funcionalidades con

Python. Aún se utiliza para la creación de juegos comerciales y se enfoca en áreas como la velocidad, completitud y tolerancia a errores (Panda3D.org, n.d.).

MOTORES DE JUEGO

UNITY

En la actualidad, Unity es uno de los motores de juego más utilizados para el desarrollo de videojuegos multiplataforma, ya que proporciona a los programadores un conjunto de herramientas y elementos, tanto bidimensionales como tridimensionales, que agilitan la construcción de aplicaciones. Unity fue lanzado con el objetivo de democratizar el desarrollo de videojuegos, pues en épocas anteriores los motores eran de carácter propietario, lo que limitaba el acceso a los programadores independientes y compañías pequeñas (Foxman, 2019).

La primera versión de Unity fue lanzada en 2005 y estaba orientada al desarrollo con Mac, por lo que el soporte para Windows llegó años después. Sus creadores fueron David Helgason, Joachim Ante y Nicholas Francis. Actualmente ofrece soporte para otras tecnologías como realidad virtual o realidad aumentada y, para 2022, era la solución preferida por el 61% de desarrolladores (InformIT, n.d.; Unity Technologies, n.d.-b).

El lenguaje más utilizado para scripting en Unity es C#, aunque en los últimos años se ha fomentado el uso de scripting visual (Unity Visual Scripting), el mismo que permite la creación de procesos a través de un sistema basado en componentes gráficos (Unity Technologies, n.d.-c). Entre algunos de los títulos desarrollados con Unity están *Fall Guys, Cuphead y Hollow Knight.*

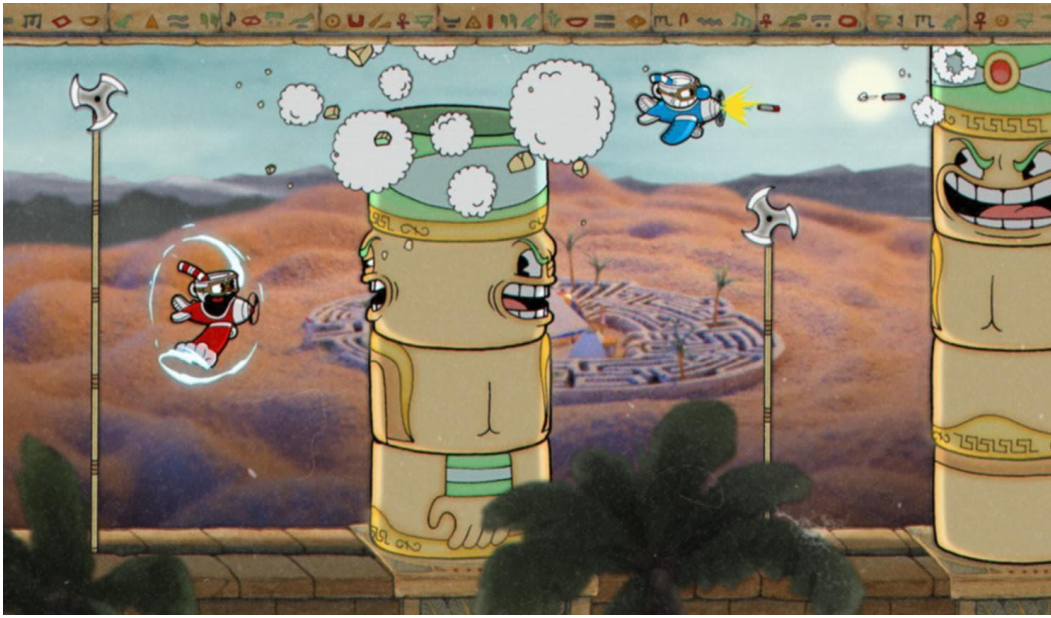


Figura 5: *Juego desarrollado en Unity-Cuphead*
Nota. De "Cuphead" por StudioMDHR, 2024 (https://unity.com/es/made-with-unity/cuphead).

UNREAL ENGINE

Unreal engine es un motor de juego creado en 1998 por Tim Sweeney, de Epic Games, para la aplicación Unreal, aunque posteriormente fue licenciado a otras compañías. Se enfocaba inicialmente en el desarrollo de juegos de computadora, pero desde la versión 2.0 se incluyó soporte a consolas como Playstation 2, X box y Gamecube. En la actualidad es uno de los motores más utilizados, con el que se han creado juegos "triple A" como *Fornite, **Gears 5** o Street Fighter 5 (Jensen, n.d.).*

Usa el entorno de desarrollo Unreal Editor, que está disponible para los principales sistemas operativos. Se puede acceder a su código o crear programas usando C++, aunque también soporta scripting con Python (Epic Games, 2022a, 2022b). Adicionalmente, Unreal utiliza *"Blueprint"* como un entorno de desarrollo gráfico que permite crear aplicaciones limitadas, sin necesidad de tener amplios conocimientos en estos lenguajes.

A pesar de que Unity y Unreal tienen amplias capacidades para el desarrollo de todo tipo de juegos, es más común que Unreal se use para grandes juegos comerciales, en tanto que Unity se utiliza en la implementación de juegos indie o para plataformas móviles (Tristem, 2022). Otra diferencia sustancial es que las versiones completas de Unity tienen un costo mensual, en tanto que la Unreal Engine es de código abierto (aunque se cobra el 5% en juegos monetizados a partir de cierto monto).

COMPARACIÓN EN EL DESARROLLO DE VIDEOJUEGOS

A continuación, se realizará una comparación entre el desarrollo de un prototipo de videojuego simple usando un lenguaje de programación y otro implementado con un motor de juego. Para este efecto seleccionamos el lenguaje Python y la máquina de juego Unity debido a su popularidad.

CREACIÓN DE JUEGO CON PYTHON

El objetivo es implementar una variación del juego Snake usando la biblioteca PyGame de Python y el IDE PyCharm (Community). En primera instancia instalamos los paquetes:

- Pygame
- Pywin32

Para el desarrollo del aplicativo, es necesario considerar los siguientes elementos:

- Definición de ventana
- Dibujar la cabeza y alimento
- Bucle para desplazar la cabeza
- Captura de eventos por teclado
- Control de impacto entre la víbora y el alimento
- Control de impacto contra las paredes

La creación de la ventana principal, y de los elementos gráficos en general, se realiza a través de comandos.

```
pygame.init()
ventana = pygame.display.set_mode((w,h))
pygame.display.set_caption('Prototipo vibora')
```

Figura 6: *Creación de la ventana principal*

En primera instancia inicializamos los elementos de pygame y establecemos la ventana del aplicativo. Luego creamos la función para dibujar a la víbora, que inicialmente tiene un solo elemento de tamaño.

```
def dibujar_vibora(xcabeza, ycabeza, ventana,color,listatamano, listadireccion,direccion):
    pygame.draw.rect(ventana, color, [xcabeza, ycabeza, 10, 10])
```

Figura 7: *Función para dibujar la víbora*

El siguiente aspecto a considerar es la comida, para lo cual calculamos unas coordenadas aleatorias e implementamos la función correspondiente:

```
def dibujar_comida(x,y,ventana,color):
    pygame.draw.rect(ventana, color, [x, y, 10, 10])
    pygame.display.update()
```

Figura 8: *Función para dibujar la comida*

De estos elementos gráficos, sólo la ventana permanecerá estática. En el caso de la víbora, la función que la dibuja debe colocarse en un bucle que se invocará en una cantidad determinada de tiempo, como se aprecia en el código adjunto.

```
while True:
    if direccion == "N":
        y = y - 10
    elif direccion == "S":
        y = y + 10
    elif direccion == "E":
        x = x + 10
    elif direccion == "O":
        x = x - 10
    dibujar_vibora(x, y, ventana, blue, listatamano, listadireccion, direccion)
    dibujar_comida(cx, cy, ventana, red)
```

Figura 9: *Función para graficar la víbora*

Podemos observar que la víbora se irá desplazando según la dirección establecida. La función clock.tick determina la cantidad de frames por segundo que pasarán (mientras más frames, mayor velocidad).

```
clock.tick(10)
```

Figura 10: *Función clock.tick*

El siguiente punto a considerar corresponde a la captura de los eventos de teclado, para lo que utilizamos la función pygame.event.get(). Identificamos la tecla presionada y cambiamos la dirección de la víbora.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:  #QUIT
        pygame.quit()
        quit()

    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_UP:
            direccion = 'N'
        elif event.key == pygame.K_DOWN:
            direccion = 'S'
        elif event.key == pygame.K_LEFT:
            direccion = 'O'
        elif event.key == pygame.K_RIGHT:
            direccion = 'E'
```

Figura 11: *Captura de eventos de teclado*

La constante pygame.QUIT se refiere a la posibilidad de cerrar la ventana usando el botón X de la ventana.



Figura 12: *Ventana principal*

El control de las colisiones, requerido para gestionar el impacto de la víbora con el alimento, debe realizarse a través del control de las coordenadas de los elementos. En primera instancia es importante discretizar los rangos de valores de las posiciones (x, y) para que no tengamos que realizar un control más complejo.

	Caso 1	**Caso 2**	**Observaciones**
Sin discretizar			El control de coordenadas es más complejo porque pueden existir superposiciones parciales
Discretizando			Solo pueden existir superposiciones totales,
			lo que simplifica el control

Tabla 1: *Efecto de la discretización*

Para discretizar, creamos listas con las coordenadas

```
coordenadasx = [x for x in range(10,600,10)]
coordenadasy = [x for x in range(10,400,10)]
```

Figura 13: Creación de listas de coordenadas

Lo que ocasiona que el control de colisión entre los elementos se simplifique.

```
if cx == x and cy == y:
    tamano = tamano + 1
    cx = coordenadasx[rd.randint(0, len(coordenadasx) - 1)]
    cy = coordenadasy[rd.randint(0, len(coordenadasy) - 1)]
    listatamano.append(tamano)
```

Figura 14: Control de colisión

En donde (cx y cy) son las coordenadas del alimento y (x,y) son las coordenadas de la víbora. La colisión ocasiona que la víbora crezca (añade elemento a la lista de coordenadas) y genera un nuevo alimento en una posición aleatoria.

Finalmente, al no existir un control de colisiones predefinido, es más sencillo no generar elementos gráficos que representen paredes, sino controlar si la víbora está fuera del rango de coordenadas de la pantalla.

```
if x <= 0 or x >= w or \
        y <= 0 or y >= h:
    print("Finaliza")
    pygame.quit()
    quit()
```

Figura 15: *Control del rango de pantalla*

Como se puede observar Python, junto con el paquete pygame, nos ayuda a implementar un juego en un corto tiempo. Sin embargo, consideraciones adicionales como la gestión gráficos, más complejos o el control de físicas, incrementan la complejidad de la aplicación, y por ende, el tiempo requerido para el desarrollo.

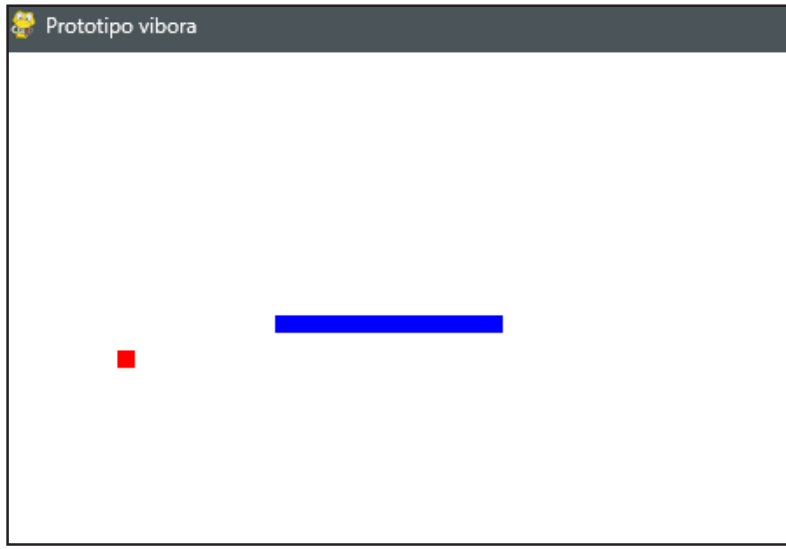


Figura 16: *Prototipo de juego*

CREACIÓN DE JUEGO CON UNITY

A continuación, se implementará un ejemplo similar al planteado anteriormente, pero usando el motor de juego Unity y el lenguaje de programación C#.

En primera instancia se generan los elementos gráficos. Para este efecto Unity posee un editor que nos permite configurar la ventana principal e ir agregando los elementos dentro de la carpeta de recursos de la aplicación (Assets).

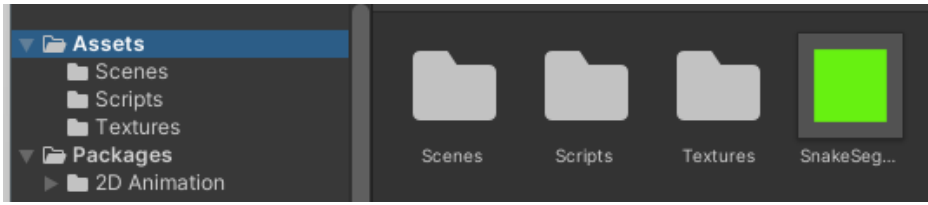


Figura 17: *Carpeta "Assets" de Unity*

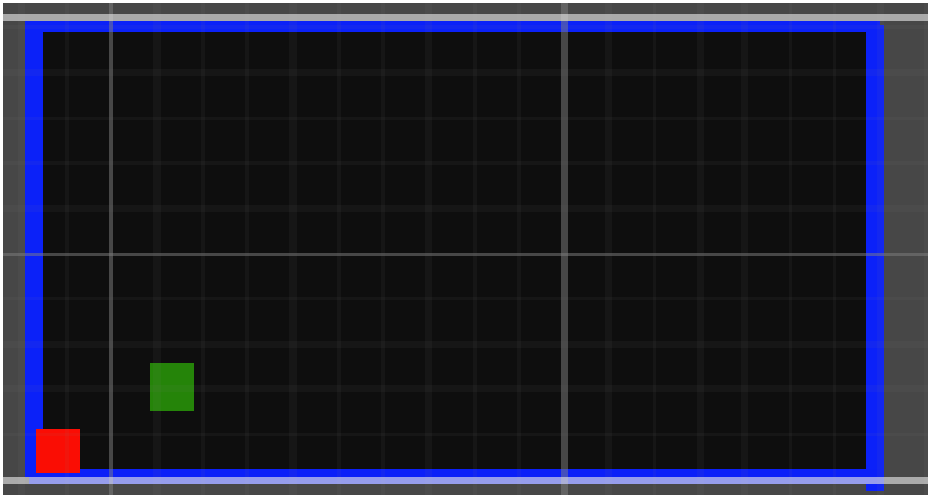


Figura 18: *Componente del cuerpo de la serpiente*

En el ejemplo observamos el gráfico que formará el cuerpo de la serpiente, así como la pantalla principal en donde se encuentran: a) el segmento inicial, b) el alimento (rojo) y c) las paredes. Cada elemento corresponde a un *GameObject.*



Figura 19: *GameObjects*

Para el cuerpo de la víbora configuramos los gestores de colisión (Box collider 2D y Rigibody 2D) y agregamos un script, al cual le pasamos un segmento como parámetro.

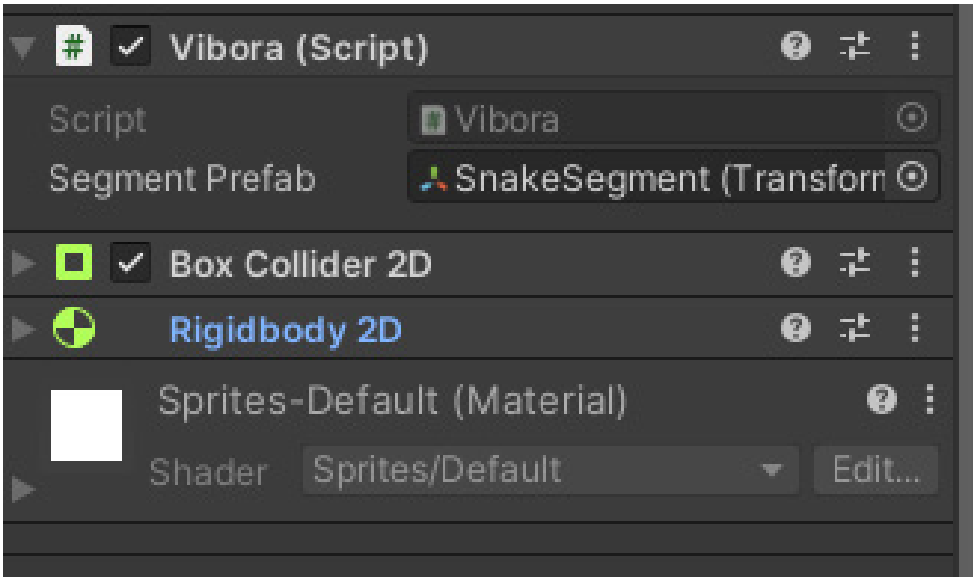


Figura 20: *Configurar colisión*

En el script de la víbora es necesario capturar la dirección de los movimientos y transformar las posiciones.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.UpArrow))
    { _direccion = Vector2.up; }
    else if (Input.GetKeyDown(KeyCode.DownArrow))
    { _direccion = Vector2.down; }
    else if (Input.GetKeyDown(KeyCode.LeftArrow))
    { _direccion = Vector2.left; }
    else if (Input.GetKeyDown(KeyCode.RightArrow))
    { _direccion = Vector2.right; }
```

Figura 21: *Eventos de teclado*

Se usan estructuras Vector2 y Vector3 para almacenar valores de coordenadas planas o espaciales. En el método FixedUpdate se va redibujando la víbora.

```
this.transform.position = new Vector3(
    Mathf.Round(this.transform.position.x) + _direccion.x,
    Mathf.Round(this.transform.position.y) + _direccion.y,
    0.0f
```

Figura 22: *Coordenadas planas*

Se implementa el método OnTriggerEnter2D, el cual se dispara al detectar una colisión con el cuerpo de la serpiente. Para identificar el elemento con el que se impacta, es necesario recuperar la etiqueta del objeto. En este caso se definen las etiquetas correspondientes a la comida, que harán crecer el cuerpo de la serpiente; o a la pared, en cuyo caso deberá finalizar el juego.

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Food")
    { Grow(); }
    if (other.tag == "wall")
    { EditorApplication.isPlaying = false; }
}
```

Figura 23: *Control de colisiones*

El disparador se define como una propiedad del *box collider 2D.*

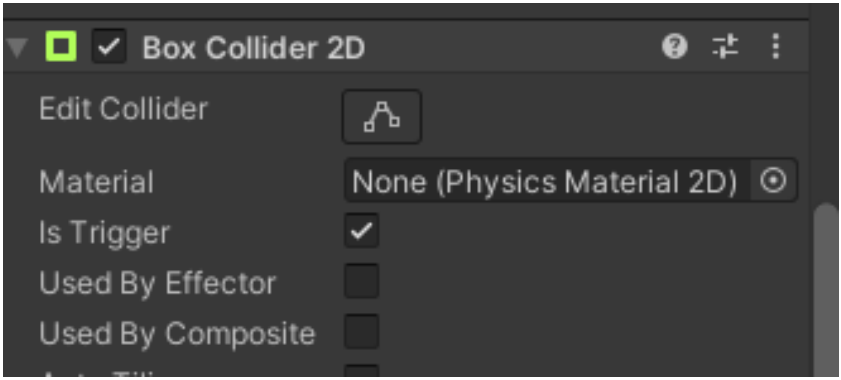


Figura 24: *Propiedades del box collider*

Posteriormente, se agrega un controlador de colisiones en el GameObject correspondiente a la comida.

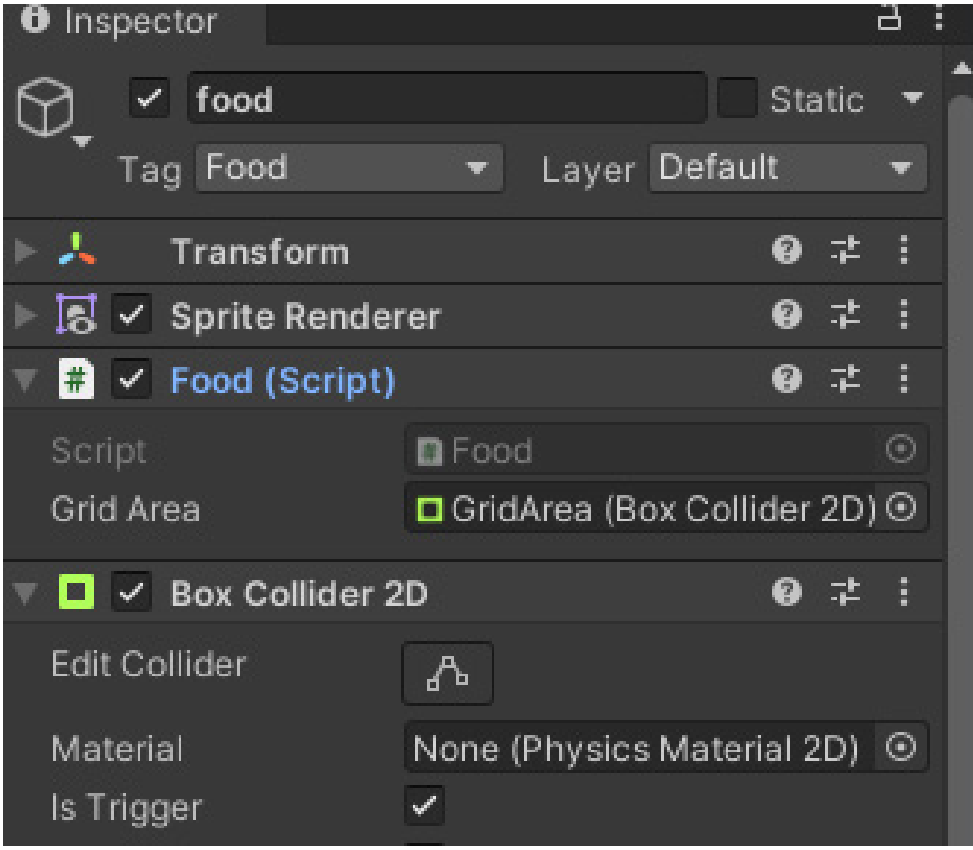


Figura 25: Controlador de colisiones para la comida

En el script asociado, cuando el alimento colisiona con la víbora, se generan nuevas coordenadas aleatorias para el *GameObject*.

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Player")
    {
        PosicionAleatoria();
    }
}

private void PosicionAleatoria()
{
    Bounds bounds = this.gridArea.bounds;
    float x = Random.Range(bounds.min.x, bounds.max.x);
    float y = Random.Range(bounds.min.y, bounds.max.y);

    this.transform.position = new Vector3(Mathf.Round(x), Mathf.Round(y), 0.0f);
}
```

Figura 26: *Script para generar coordenadas*

Para modificar la velocidad del bucle que controla el juego, es necesario configurar la característica "Time", que está disponible en el menú "Project settings".

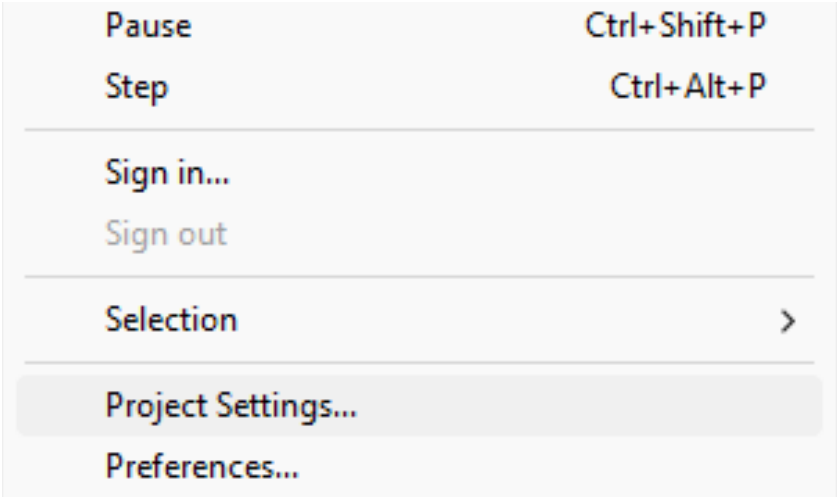


Figura 27: Propiedades del proyecto

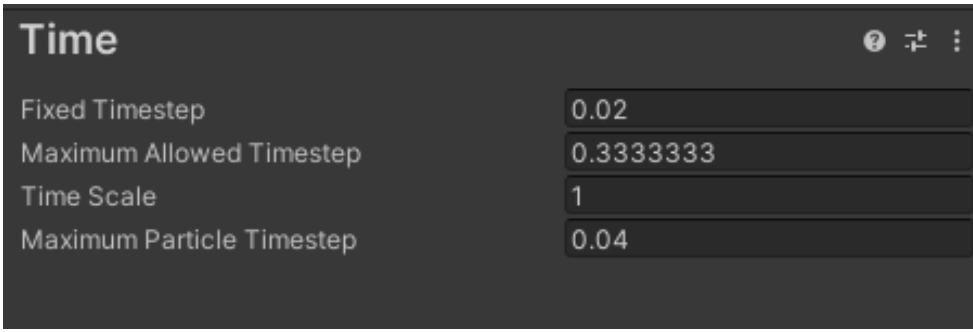


Figura 28: Configuración del tiempo

La propiedad Fixed Timestep determina el tiempo en que se dispara el método FixedUpdate() en los scripts.

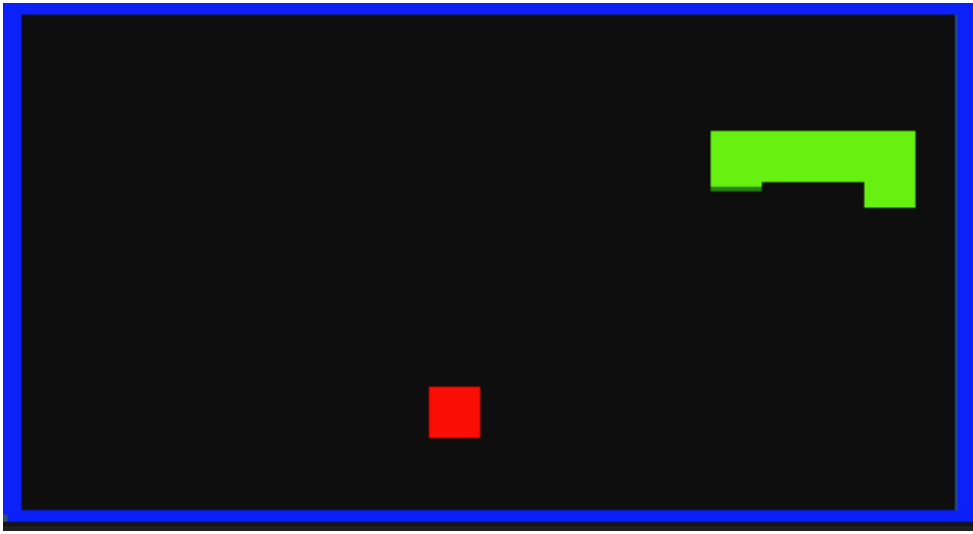


Figura 29: *Pantalla del juego*

Como se puede apreciar, la implementación a través de un motor de videojuegos proporciona una serie de características adicionales que simplifican la construcción de la aplicación, sin embargo, es importante determinar la idoneidad de la herramienta según el tamaño del proyecto, costo (recursos humanos, licencias, infraestructura) o curva de aprendizaje requerida.

Otro aspecto por considerar al utilizar motores de juego es que usualmente proporcionan plantillas para simplificar el desarrollo de juegos plataformas, shooters, entre otros. Adicionalmente, Unity posee soporte para aplicaciones de realidad virtual y realidad aumentada.

CREACIÓN DE JUEGO USANDO PLANTILLAS DE UNITY

Para el siguiente ejemplo, se creará una aplicación a partir de una plantilla de un videojuego en primera persona, usando el entorno Unity. En primer lugar, descargamos el recurso correspondiente y creamos el proyecto.

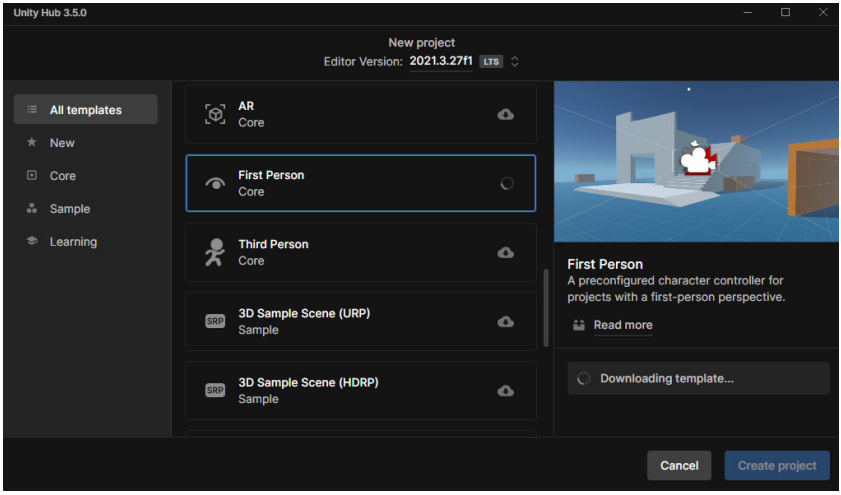


Figura 30: *Crear proyecto de Unity desde plantilla*

Podemos apreciar que se generan los elementos del juego, así como los recursos en la carpeta Assets.

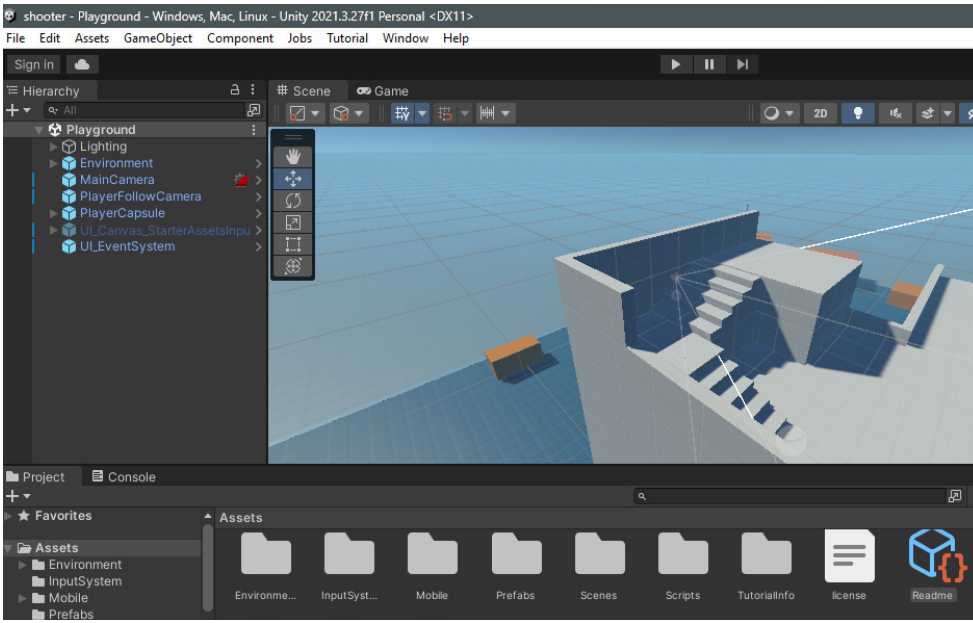


Figura 31: *Elementos del juego*

Al ejecutar la aplicación, podemos navegar por el entorno generado usando el teclado.

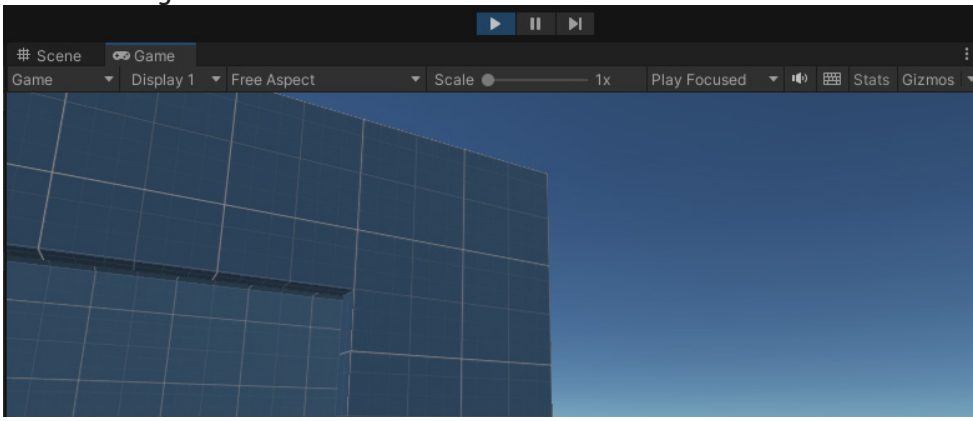


Figura 32: *Interfaz de la aplicación*

A continuación, agregamos un arma personalizada. Accedemos a la tienda de Unity (Asset Store), en donde podemos encontrar una gran cantidad de contenido gratuito y pagado.

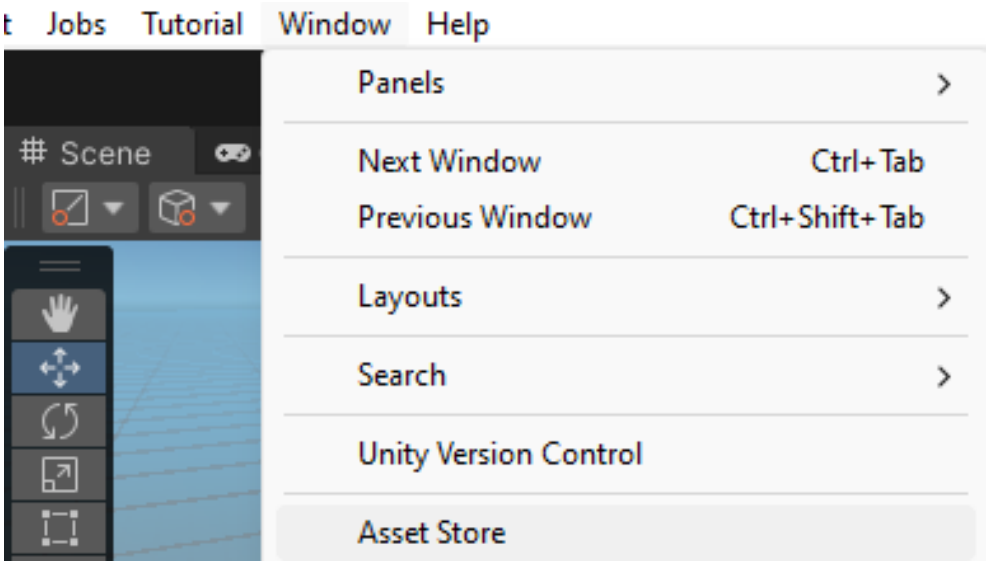


Figura 33: *Acceder a la "Asset Store"*

Descargamos e importamos un paquete de armas, el mismo que se agrega en la carpeta de assets.

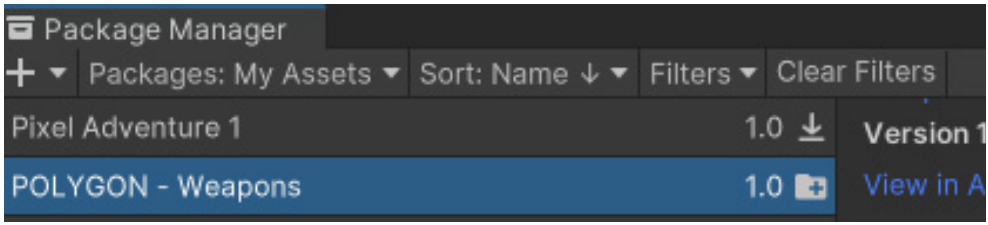


Figura 34: *Descargar paquete de armas*

En algunos casos es necesario convertir las texturas hacia URP (Universal Render Pipeline), que es un formato renderizado multiplataforma.

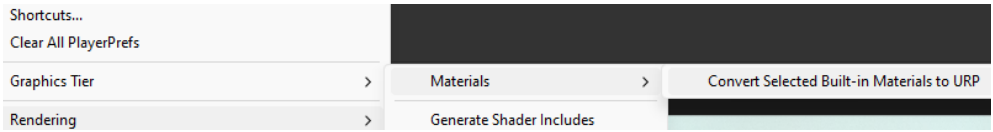
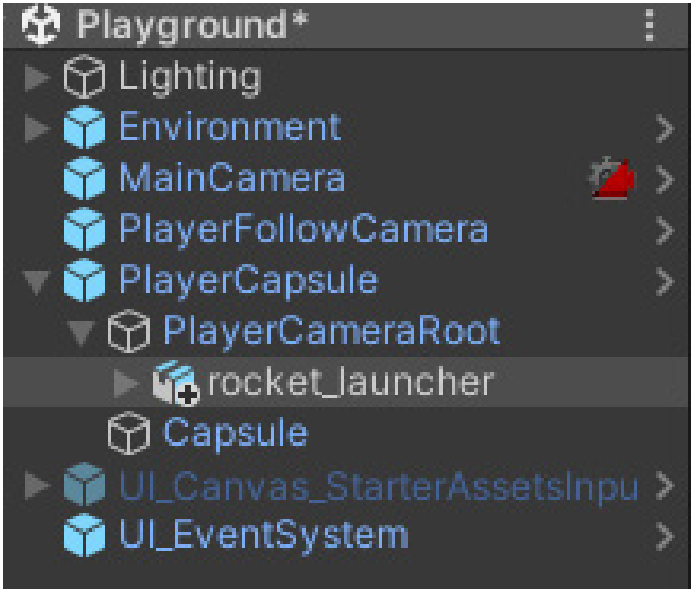


Figura 35: *Convertir texturas a URP*

Posteriormente agregamos el arma dentro del proyecto (en el elemento Player Camera Root) y modificamos la posición inicial



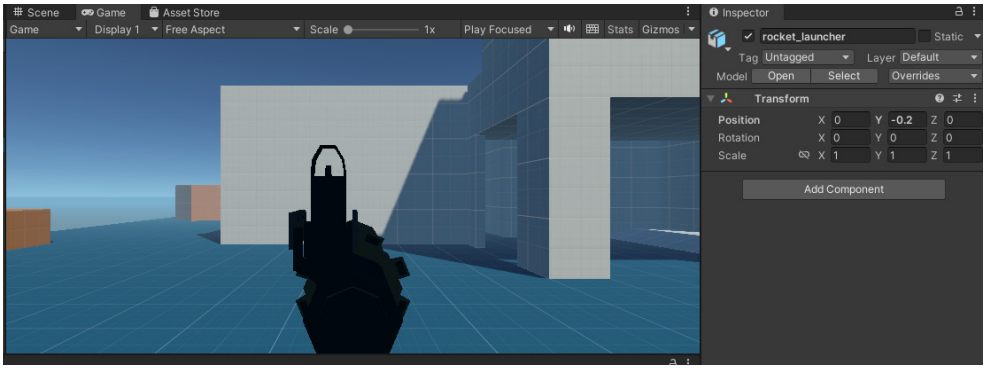


Figura 36: *Agregar nuevo elemento al proyecto*

Para agregar comportamientos, por ejemplo, disparar con el botón del ratón, seleccionamos el elemento StarterAssets (carpeta Assets) y añadimos la acción correspondiente.

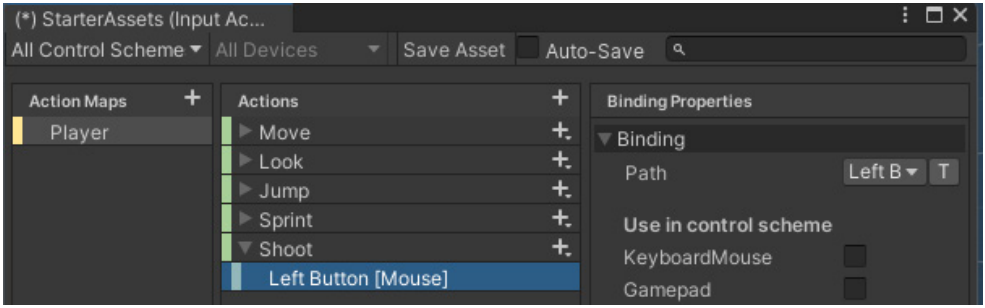


Figura 37: *Comportamiento del elemento*

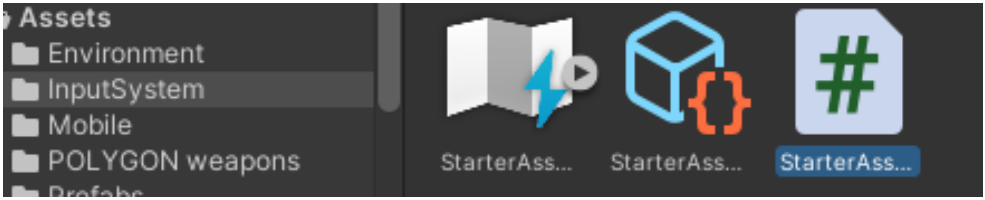


Figura 38: *Creación del script asociado*

Agregamos el código del programa asociado en el método OnShoot, el mismo que se dispara al presionar el botón izquierdo del ratón.

```
namespace StarterAssets
{
    public class StarterAssetsInputs : MonoBehaviour
    {
        [Header("Character Input Values")]
        public Vector2 move;
        public Vector2 look;
        public bool jump;
        public bool sprint;


        public void OnShoot(InputValue value)
        { print("Dispara"); }
```

Figura 39: *Configuración del script*

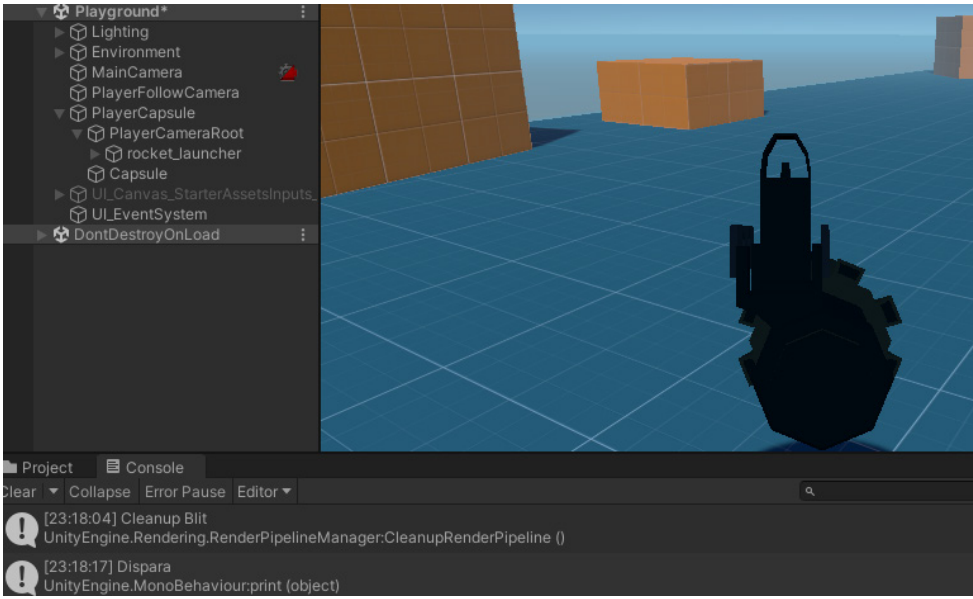


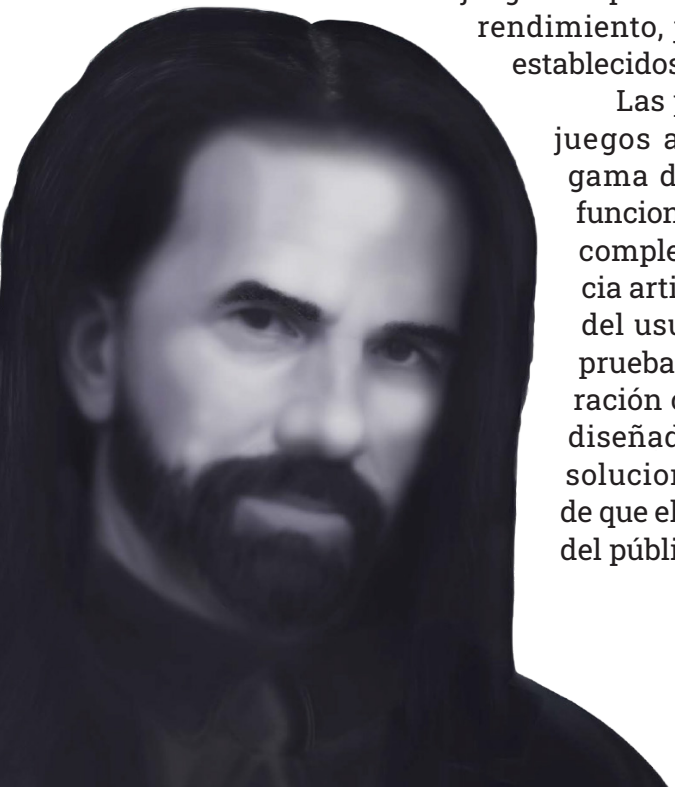
Figura 40: *Prueba de la aplicación*

Como se puede apreciar, el desarrollo de una aplicación base para un juego de shooter se realiza de forma rápida utilizando un motor de juegos, situación que sería mucho más compleja en el caso del desarrollo directo usando un lenguaje de programación.

PRUEBAS

Ya has llegado casi al final. Ahora que tienes tu videojuego, es importante probarlo, y no solamente debes hacerlo tú, sino que, como se había revisado anteriormente, los probadores o testers son quienes te ayudarán a evaluar tu obra.

La calidad de un videojuego es fundamental para garantizar una experiencia de juego satisfactoria para los usuarios. En este sentido, las pruebas, también conocidas como testing, juegan un papel crucial en el desarrollo de videojuegos. Estas pruebas no solo buscan identificar y corregir errores técnicos, sino que también se centran en asegurar que el juego cumpla con los estándares de rendimiento, jugabilidad y diseño establecidos.



Las pruebas en los videojuegos abarcan una amplia gama de aspectos, desde la funcionalidad básica hasta la complejidad de la inteligencia artificial y la experiencia del usuario. Los equipos de pruebas trabajan en colaboración con desarrolladores y diseñadores para detectar y solucionar problemas antes de que el juego llegue a manos del público.

Figura 41: *Billy Mitchell*

Las pruebas de funcionalidad buscan garantizar que todas las características del juego, desde los controles hasta los elementos de la interfaz de usuario, funcionen de manera coherente y libre de errores. Las pruebas de rendimiento se centran en la optimización del juego para diferentes plataformas, asegurando un rendimiento fluido y una experiencia sin problemas, ya sea en una consola, una PC o dispositivos móviles.

Además, las pruebas de compatibilidad se realizan para asegurarse de que el juego funcione correctamente en una variedad de configuraciones de hardware y software. Las pruebas de seguridad buscan identificar posibles vulnerabilidades que podrían comprometer la integridad del juego o la privacidad del usuario.

En última instancia, las pruebas en los videojuegos desempeñan un papel crucial en la creación de productos finales de alta calidad. Al abordar problemas desde las etapas iniciales del desarrollo hasta el lanzamiento, contribuyen significativamente a la satisfacción del jugador y al éxito comercial del juego. Con el continuo avance de la tecnología y la complejidad de los videojuegos modernos, la importancia de las pruebas sigue creciendo, asegurando experiencias de juego cada vez más inmersivas y libres de problemas técnicos.

Vamos a revisar algunos temas con mayor profundidad en la siguiente sección.

EL TESTING EN EL DESARROLLO DE VIDEOJUEGOS

DEFINICIÓN DEL TESTING EN EL DESARROLLO DE VIDEOJUEGOS

En el contexto de los videojuegos, el *testing* o pruebas se refiere al proceso sistemático de evaluación y verificación de un juego en todas sus facetas, desde la funcionalidad básica hasta aspectos más complejos como la jugabilidad, rendimiento y seguridad. Este proceso tiene como objetivo identificar y corregir posibles errores, defectos o vulnerabilidades que podrían afectar la calidad y la experiencia del usuario. Al abarcar diversas áreas que van desde la mecánica del juego hasta la estabilidad técnica en diferentes plataformas, el *testing* se convierte en un componente esencial para asegurar que el juego cumpla con las expectativas del público y los estándares de la industria.



Figura 42: *El testing en videojuegos*

¿POR QUÉ ES IMPORTANTE EL TESTING EN EL DESARROLLO DE VIDEOJUEGOS?

La relevancia del testing en el desarrollo de videojuegos es innegable en un entorno donde la competencia es feroz y las expectativas de los jugadores son cada vez más exigentes. Un juego sin un proceso de pruebas adecuado corre el riesgo de presentar defectos que podrían afectar la experiencia del usuario, desde pequeños errores visuales hasta problemas de rendimiento que pueden resultar en críticas negativas y pérdida de confianza por parte de los jugadores. Además, en un mercado en constante evolución, donde la diversidad de plataformas y dispositivos es la norma, las pruebas se vuelven esenciales para garantizar la compatibilidad y la funcionalidad óptima en todas las configuraciones posibles. El testing no solo es un paso necesario en el desarrollo de videojuegos, sino que es un pilar fundamental para la creación de productos exitosos y la construcción de relaciones sólidas con la comunidad de jugadores.

CONTRIBUCIÓN A LA CALIDAD DEL JUEGO

El proceso de pruebas en el desarrollo de videojuegos se convierte en una actividad muy relevante para asegurar la calidad integral del producto final. Al someter el juego a rigurosos escenarios de prueba, se logra identificar y corregir una amplia gama de posibles errores, desde bugs técnicos hasta

problemas de jugabilidad. La detección temprana y resolución de estos problemas contribuye de manera significativa a la coherencia y estabilidad del juego, permitiendo que los jugadores disfruten de una experiencia fluida y libre de interrupciones. Además, las pruebas son esenciales para garantizar que las características anunciadas funcionen según lo previsto, brindando a los desarrolladores la confianza necesaria para presentar un producto final que cumple con las expectativas tanto internas como del público.

IMPACTO EN LA SATISFACCIÓN DEL JUGADOR

La satisfacción del jugador es el corazón mismo de la industria de los videojuegos, y las pruebas desempeñan un papel clave en este aspecto. Al abordar y eliminar errores antes del lanzamiento, se reduce significativamente la probabilidad de que los jugadores encuentren problemas molestos que puedan afectar negativamente su experiencia. Una jugabilidad fluida, una interfaz intuitiva y un rendimiento óptimo son resultados directos de un proceso de pruebas minucioso. La confianza del jugador en la calidad de un juego se refuerza cuando este funciona como se espera, sin frustraciones ni sorpresas desagradables. En última instancia, las pruebas no solo buscan la corrección de errores, sino que trabajan activamente para elevar la satisfacción del jugador, fomentando la lealtad a la marca y construyendo una reputación positiva en la industria del desarrollo de videojuegos.



Figura 43: *El testing en videojuegos: Un paso fundamental para medir el grado de satisfacción del jugador.*

TIPOS DE PRUEBAS

Cuando se prueba un videojuego, varias pruebas se pueden realizar.

PRUEBAS DE FUNCIONALIDAD

Las pruebas de funcionalidad en el desarrollo de videojuegos son un componente esencial para garantizar que todas las características y sistemas del juego funcionen de manera coherente y sin problemas. Este tipo de pruebas se centra en evaluar la ejecución de las funciones básicas del juego, desde la interacción de los controles hasta la implementación correcta de mecánicas de juego clave. Se exploran casos de uso variados para asegurar que todas las características respondan de manera predecible a las acciones del jugador. La detección y corrección de errores en esta fase temprana no solo mejora la estabilidad técnica del juego, sino que también contribuye a una experiencia de juego más inmersiva y satisfactoria.



Figura 44: *Los videojuegos deben ser probados en su mercado objetivo*

ESCENARIOS DE PRUEBA DETALLADOS

Durante las pruebas de funcionalidad, se diseñan escenarios de prueba que abarcan situaciones típicas y excepcionales. Desde la navegación por menús hasta la ejecución de acciones específicas dentro del juego, cada aspecto es evaluado de manera minuciosa. Los probadores buscan identificar posibles conflictos en la lógica del juego, problemas de respuesta de los controles o cualquier anomalía que pueda afectar la jugabilidad. Además, las pruebas de funcionalidad son fundamentales para asegurar que las características específicas del juego, como habilidades especiales de personajes o mecánicas únicas, se ejecuten de acuerdo con el diseño original, evitando desviaciones que podrían comprometer la experiencia del usuario.



Figura 45: *El testing en videojuegos ayuda a mejorar la experiencia del usuario*

IMPORTANCIA EN EL CICLO DE DESARROLLO

Las pruebas de funcionalidad se llevan a cabo en varias fases del desarrollo del juego, desde las etapas iniciales hasta las versiones más avanzadas. Esto permite a los desarrolladores abordar problemas a medida que surgen, evitando acumulaciones de errores que podrían ser más difíciles de resolver en etapas posteriores. Al integrar las pruebas de funcionalidad en el ciclo de desarrollo, se establece una base sólida para construir características más complejas y avanzadas, asegurando que cada elemento contribuya de manera positiva a la experiencia global del juego.

DETALLES SOBRE CÓMO SE PRUEBAN LAS FUNCIONES BÁSICAS DEL JUEGO.

Debes considerar que las pruebas de las funciones básicas de un juego son un proceso meticuloso que implica explorar y evaluar cada elemento esencial que contribuye a la jugabilidad. Los probadores se sumergen en la experiencia del jugador, abordando aspectos que van desde los controles simples hasta las mecánicas fundamentales que definen la esencia del juego. La evaluación de estas funciones básicas comienza con la interacción directa con el juego, utilizando todos los medios posibles para garantizar que los controles respondan de manera intuitiva y precisa.

ESCENARIOS DE PRUEBA REALISTAS

Los escenarios de prueba se diseñan para replicar situaciones de juego comunes y específicas. Esto implica explorar diversas combinaciones de acciones del jugador, desde movimientos simples hasta interacciones más complejas. Los probadores también se enfocan en situaciones límite y casos extremos para identificar posibles fallos en la lógica del juego o en la respuesta de las funciones básicas. Estos escenarios realistas permiten no solo identificar posibles problemas, sino también evaluar la coherencia y la fluidez de la experiencia de juego en diferentes contextos.



Figura 46: *Escenarios reales de simulación–Fuente Eyeware Tech*

PRUEBAS DE RETROALIMENTACIÓN Y RESPUESTA DEL JUGADOR

Además de la ejecución directa de acciones, las pruebas de funciones básicas incluyen la evaluación de la retroalimentación proporcionada al jugador. Esto abarca desde efectos visuales y auditivos hasta mensajes informativos en la interfaz de usuario. Los probadores buscan asegurarse de que la retroalimentación sea clara, informativa y estéticamente coherente con el estilo del juego. La respuesta del jugador a estas señales es cuidadosamente observada para identificar posibles puntos de confusión o áreas donde se puede mejorar la comunicación entre el juego y el usuario.



Figura 47:
La retroalimentación del jugador

ANÁLISIS DE DATOS Y RESULTADOS

Los datos recopilados durante estas pruebas son analizados de manera exhaustiva. Los desarrolladores revisan informes detallados que destacan áreas de mejora, errores identificados y sugerencias para optimizar las funciones básicas. Este enfoque basado en datos no solo asegura que los errores se corrijan, sino que también proporciona información valiosa para la evolución continua del juego a lo largo del desarrollo. Así, las pruebas de funciones básicas se convierten en una herramienta esencial para perfeccionar la esencia del juego y ofrecer una experiencia de juego sólida y envolvente.

EJEMPLOS DE ESCENARIOS DE PRUEBA Y CASOS DE USO

Sin dejar de lado otros escenarios y casos de uso,
te contaremos algunos que pueden resultar interesantes.



Figura 48: *Testing automatizado funcional en Unreal Engine 4.*

ESCENARIOS DE PRUEBA

1. PRUEBAS DE JUGABILIDAD:

Las pruebas de jugabilidad son esenciales para evaluar la experiencia del jugador. Estos escenarios de prueba incluyen la interacción del usuario con el juego, la respuesta a los controles, la fluidez del movimiento del personaje y la usabilidad general de la interfaz de usuario. Por ejemplo, los testers pueden evaluar cómo se comporta el juego ante diferentes estilos de juego, asegurándose de que los controles sean intuitivos y que no haya problemas de rendimiento durante las acciones del jugador.

2. PRUEBAS DE RENDIMIENTO:

Los escenarios de prueba de rendimiento se centran en garantizar que el juego funcione de manera eficiente en términos de velocidad de fotogramas, tiempos de carga y rendimiento general en diversas plataformas. Los testers pueden simular condiciones de carga pesada para verificar la estabilidad del juego y su capacidad para manejar múltiples situaciones, como escenas intensas con gráficos complejos o grandes multitudes en pantalla.

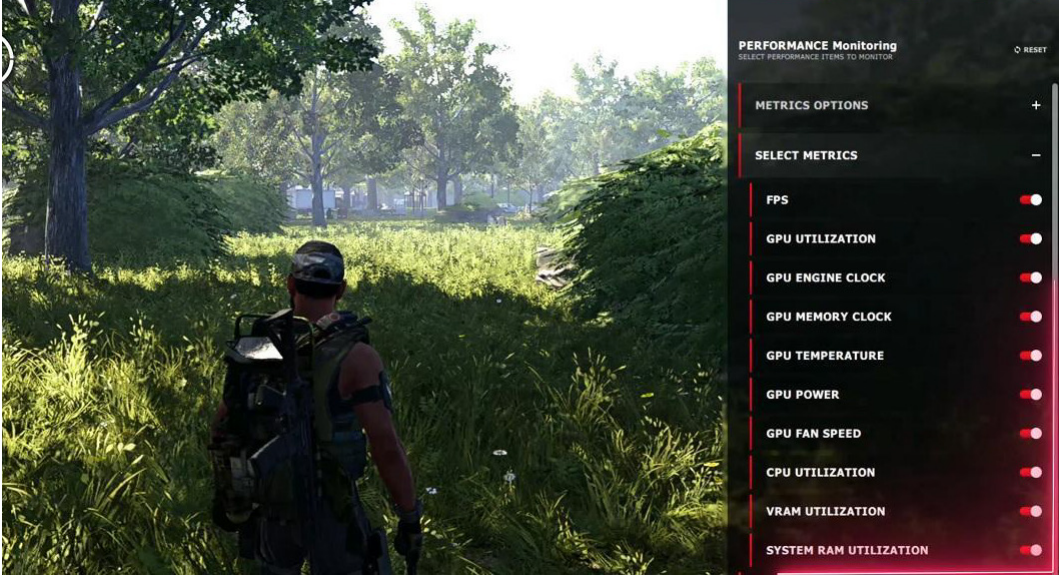


Figura 49: *Pruebas de rendimiento de videojuegos*

CASOS DE USO

1. ESCENARIO DE COMBATE

En un juego de acción o aventuras, un caso de uso podría ser el escenario de combate. Se evaluaría cómo el sistema responde a las acciones del jugador durante una pelea, verificando la precisión de los movimientos, la efectividad de las habilidades y la coherencia en la representación visual y auditiva de las secuencias de combate.

2. CASO DE USO DE MULTIJUGADOR EN LÍNEA

Para juegos con modos multijugador en línea, el caso de uso podría incluir la creación y participación en partidas en línea. Los testers evaluarían la estabilidad del servidor, la calidad de

la conexión, la ausencia de lag[1] y la capacidad del juego para manejar múltiples jugadores simultáneos. También se verificaría la implementación efectiva de funciones como chat, emparejamiento y sistemas de puntuación.

Las pruebas de compatibilidad aseguran que el aplicativo pueda ejecutarse sin problemas en una amplia gama de dispositivos, permitiendo identificar cualquier inconveniente sobre la experiencia del jugador. Se consideran aspectos como:

- **Hardware:** Ejecución sobre varias configuraciones de procesadores, memorias, tarjetas gráficas, etc.

- **Software:** Comprobar que la aplicación funcione adecuadamente sobre un sistema operativo o navegador. Adicionalmente, valida la interacción adecuada con bibliotecas de terceros.

- **Plataformas cruzadas:** Comprobar que el juego se ejecute sin problemas en diferentes entornos como Android, iOS, PC.

- **Dispositivos de entrada:** Es importante que valides los dispositivos de entrada, tales como el teclado o un joystick, pues es un elemento muy importante y dará más realce a la interacción con el juego.

- **Ubicación:** Incluye adaptaciones culturales

- **Red y protocolos:** Es importante considerar que los juegos actuales buscan la competencia y el trabajo en

[1] Retardos en la comunicación

equipo. Por ello, el uso de protocolos de red se torna importante para lograr este objetivo.

Las pruebas de compatibilidad se ejecutan de forma manual, automatizada, usando emuladores o a través de servicios en la nube. Es importante priorizar las plataformas hacia las que el juego está orientado, debido a que cada entorno tiene requisitos específicos de hardware. Otro aspecto relevante corresponde al tipo de juego en desarrollo, ya que los juegos casuales podrían no requerir características complejas como soporte a micro transacciones o multijugador en línea (Testbytes, 2023).

Existen algunas herramientas orientadas a validar la compatibilidad de una aplicación, que combinan procesos de evaluación comparativa (bajo distintas cargas de trabajo) y una base de datos con los requisitos de algunos juegos populares. Entre estos programas se incluyen: *Can you run it*, *PC Game Benchmark*, *Game debate*, *Valve Steram game compatibility checker tool*, *Speccy*, entre otros.

PRUEBAS DE RENDIMIENTO

EXPLORACIÓN DE PRUEBAS DE VELOCIDAD, EFICIENCIA Y OPTIMIZACIÓN

En el proceso de desarrollo de videojuegos, la exploración de pruebas de velocidad, eficiencia y optimización desempeña un papel crucial para garantizar que el jue-

go ofrezca una experiencia fluida y envolvente. Las pruebas de velocidad se centran en evaluar el rendimiento del juego, asegurándose de que mantenga una velocidad de fotogramas constante y evite cualquier retraso que pueda afectar la experiencia del jugador. Este tipo de pruebas también examina la carga y descarga eficiente de recursos, para minimizar los tiempos y optimizar el uso de la memoria.

La eficiencia del código es un aspecto fundamental de las pruebas en el desarrollo de videojuegos. Los testers exploran cómo se utilizan los recursos del sistema, identificando posibles cuellos de botella y asegurándose de que el código esté optimizado para funcionar de manera eficiente en una variedad de plataformas. Esto incluye la revisión de algoritmos, estructuras de datos y procesos en segundo plano para mejorar la eficiencia y maximizar el rendimiento del juego.

La optimización va más allá de la velocidad y la eficiencia del código; también se enfoca en la utilización efectiva de los activos del juego como texturas, modelos 3D y efectos visuales. Las pruebas exploratorias en este sentido buscan garantizar que los recursos gráficos se utilicen de manera óptima para mantener una calidad visual impresionante sin comprometer el rendimiento del juego.

CONSIDERACIONES ESPECÍFICAS PARA DISTINTAS PLATAFORMAS

Desde la adaptación de juegos de máquinas recreativas hacia consolas o computadoras personales, los videojuegos han sufrido una continua evolución en cuanto a las plataformas hacia las que se dirigen y la forma en la que se consumen. No existe una clasificación estricta con respecto a los tipos de videojuegos según su plataforma destino, sin embargo, es posible distinguir dos escenarios principales: videojuegos orientados a dispositivos fijos (PCs o consolas) y videojuegos orientados a dispositivos móviles.

Pese a que los videojuegos que se consumen en consolas, como PlayStation o Xbox, guardan ciertas diferencias con sus versiones para computadoras personales, comparten una cantidad significativa de atributos que permiten relacionarlos. En general este software presenta características como:

- El control del juego requiere de mandos físicos o periféricos como teclados y ratones. Son muy pocos los juegos que utilizan métodos alternativos como pulsaciones en la pantalla o reconocimiento de movimientos, pese a que este último fue muy popular en los últimos años de la década del 2010 en dispositivos como la Nintendo Wii o el Kinect de Microsoft.

- El juego se proyecta en una pantalla o monitor, con un tamaño que frecuentemente supera las 14 pulgadas, y una resolución que puede llegar a 4K (2023).

- Requieren una fuente continua de electricidad y no están orientadas a mantener un bajo consumo de energía.

• El precio de los juegos suele ser significativamente mayor al que se encuentra en dispositivos móviles.

• La potencia computacional de los dispositivos es significativamente superior a la encontrada en teléfonos inteligentes y tabletas.

• El tamaño de los juegos es más grande, pues generalmente incluye una gran cantidad de texturas y cinemáticas.

Los teléfonos inteligentes, y en menor medida las tabletas, se han convertido en la principal fuente de acceso de los jóvenes al contenido de internet. Esta situación es similar para el caso de los videojuegos, pues los usuarios pueden desarrollar una partida rápida de su aplicación favorita mientras esperan en la cola de un comercio o en el tráfico de la ciudad. Algunas de las características de los juegos para dispositivos móviles incluyen:

• El control del juego se basa en pulsaciones sobre una pantalla táctil

• Debe soportar distintas resoluciones de gráficos, pues se puede instalar en una amplia gama de dispositivos con características heterogéneas.

• La mayoría de los juegos tiene precios inferiores a sus contrapartes de escritorio o suelen tener una versión gratuita.

• Los dispositivos no requieren una fuente de continua de alimentación.

• Aunque existen juegos que requieren características técnicas considerables, la gran mayoría corresponden

a juegos casuales que no utilizan gran potencia computacional.

• La mayoría de juegos tiene un tamaño reducido, aunque existen juegos excepcionales que requieren varios gigabytes de almacenamiento (Galehantomo, P.S., 2015).

Finalmente, los juegos en web se presentan como una especialización de los juegos de escritorio, con la diferencia de que requieren un navegador para su ejecución. Aunque existen juegos con escenarios tridimensionales, como Roblox o Minecraft, la mayoría no requieren de grandes capacidades de procesamiento. Se destacan por su compatibilidad y por el uso de tecnologías estándares como Javascript. Pueden ejecutarse sobre dispositivos que posean un navegador, como una tableta, pero la experiencia no está optimizada o suele tener limitaciones.

PRUEBAS DE SEGURIDAD

En los primeros años de los videojuegos de uso doméstico, aspectos como la protección de la propiedad intelectual eran prioritarios para las compañías desarrolladoras, pues de esta forma protegían sus finanzas y limitaban la distribución de copias piratas con posibles programas maliciosos. La difusión de estas amenazas informáticas no se realizaba a gran escala, principalmente por las limitaciones de distribución de la época.

En las últimas décadas, la popularidad de los juegos en línea y las micro transacciones han convertido a la industria de los videojuegos en un blanco atractivo para los ciberdelincuentes, lo que ha ocasionado ataques a través de software malicioso, fraudes, robo de dinero, entre otros.

VULNERABILIDADES

TROYANOS

El troyano es un software malicioso que puede monitorear una red y capturar información sensible, como contraseñas, para transmitirla a servidores de terceros. Usualmente se presenta como un software inocuo, es decir inofensivo, pero en su interior contiene rutinas que pueden alterar el funcionamiento del computador desencadenando una serie de eventos

HOMBRE EN EL MEDIO (MAN IN THE MIDDLE)

En este caso, los hackers pueden hacerse pasar por personal de la compañía y ofrecer premios o recompensas. Es usual que generen copias de los portales de las empresas para capturar la información de acceso, con la que pueden ingresar al portal original y adueñarse de información personal, bienes digitales o números de tarjetas de crédito de los propietarios.

ATAQUES DE NEGACIÓN DE SERVICIO

Estos ataques están dirigidos principalmente a los servidores de las aplicaciones y buscan dejarlos inactivos, con lo que los jugadores no podrían acceder a sus cuentas y juegos. A menudo provienen de grupos organizados, que tienen afinidad con algún tipo de activismo (Chen et al., 2016).

FALSIFICACIÓN DE BIENES VIRTUALES

En los videojuegos actuales, es muy común que se agreguen tiendas virtuales en donde se pueden adquirir activos digitales como personajes, mapas, armas, etc. Adicionalmente, muchos juegos soportan el desarrollo de complementos o *plugins* que consisten en pequeñas aplicaciones que agregan funcionalidades al juego. Es común que estos programas no tengan un control de calidad adecuado, lo que provoca que algunos puedan modificar activos adquiridos legalmente para potenciarlos, o simplemente permitan agregar otros elementos propietarios de forma gratuita. A menudo estos activos se venden a otros jugadores, quienes pueden perder su inversión o incluso el acceso a sus cuentas, si la compañía matriz detecta estas irregularidades (Zhao, 2018).

TRAMPAS EN LAS MECÁNICAS DEL JUEGO

Relacionado al punto anterior, en este caso las modificaciones del juego están orientadas violentar sus mecánicas originales para proporcionar una ventaja injusta a un jugador. Esto es particularmente relevante en torneos de e-sports, donde el dinero de patrocinios o de casas de apuestas, puede representar un factor determinante (Bogdan et al., 2021).

PROTECCIÓN DE DATOS PERSONALES

Aunque gran parte de las compañías actualmente tienen políticas estrictas de protección de datos personales, como el cifrado de la información y el uso de validaciones de dos pasos para el acceso a las cuentas, han existido empresas que han reaccionado sólo después de un incidente significativo. Errores como el uso de protocolos de comunicación sin cifrado de extremo a extremo o la distribución de bases de datos sin encriptar, se han producido con relativa frecuencia, afectando la seguridad y privacidad de los consumidores.



Figura 50: *Protección de datos personales, una actividad clave en el entor no de videojuegos*

ACTIVISMO Y AMENAZAS

Finalmente, es importante señalar que el contenido mismo del juego puede estar relacionado con la difusión de ideologías, adoctrinamiento o incluso, generar situaciones de acoso dentro de su comunidad. Es por ello imprescindible establecer políticas que prevengan o permitan denunciar estas amenazas.

Las pruebas de seguridad dentro de los videojuegos se han convertido en un aspecto sumamente relevante que requiere una amplia coordinación entre el equipo de desarrollo y de control de calidad. Esta revisión puede involucrar desde cambios en las prácticas de programación hasta el uso de frameworks o herramientas especializadas.

ESTRATEGIAS

A continuación, se detallan algunas estrategias de seguridad que pueden aplicarse.

MINIMIZAR PRIVILEGIOS

En el desarrollo de videojuegos es importante evitar que las aplicaciones usen privilegios administrativos, ya podrían ser utilizadas para ejecutar ataques contra los clientes.

USAR INTERFACES DE PROGRAMACIÓN DE APLICACIONES (API) SEGURAS

Es común que algunos desarrolladores trabajen con versiones obsoletas ciertas APIs, lo que puede desembocar en la aparición de brechas de seguridad. Actualmente, muchos entornos integrados de desarrollo (IDE), como Visual Studio, incluyen herramientas que permiten validar la calidad de estos elementos.

Cuando se utilizan interfaces o complementos de terceros, debe realizarse una valoración de la reputación del fabricante, frecuencia de actualización, estadísticas de incidentes de seguridad, soporte, compatibilidad y facilidad de integración.

MODELADO DE AMENAZAS

Consiste en un modelo que permite identificar, divulgar y entender las distintas amenazas de seguridad y sus estrategias de mitigación, enfatizando la idea de la evaluación integral del sistema de seguridad en las distintas etapas del proceso de desarrollo. No está ligado a una tecnología específica e involucra procesos de evaluación de los protocolos de comunicación, fuentes de datos, envío de información, etc. Algunas de sus técnicas relacionadas son STRIDE y DREAD.

PREVENCIÓN DE EJECUCIÓN DE DATOS

Permite validar si un segmento de código tiene derecho de ejecución, o no. La revisión se realiza a nivel de procesador, por lo que, si un código no incluye ese derecho, el programa puede ser finalizado. En Windows puede implementarse con el switch /NX.

REVISIÓN DE LA SEGURIDAD DEL BUFFER

Usando banderas específicas del compilador, se ejecuta una validación de algunas formas de desbordamiento de buffer que pueden ser explotadas de forma maliciosa. Aunque la evaluación no es infalible, constituye una buena práctica de seguridad.



Figura 51: *Evaluación de seguridad del buffer*

VALIDACIÓN DEL CÓDIGO

Involucra el uso de herramientas que permitan validar la integridad del contenido ejecutable, lo que permite identificar posibles alteraciones en el código original. En plataformas como Windows, el método "Authenticode" permite validar que los ejecutables y archivos DLL hayan permanecido sin modificaciones, para lo cual utiliza una combinación de estándares de la industria y técnicas de criptografía.

OFUSCACIÓN DEL CÓDIGO

Consiste en volver el código de un programa más complejo y confuso, de manera que sea más difícil comprenderlo y copiarlo. Varios complementos de herramientas como Unity agregan esta funcionalidad.

PROTECCIÓN DEL PUNTAJE

Involucra la protección del puntaje del jugador, tanto de manera local como remota, para evitar la manipulación de este atributo que podría proporcionar acceso a material restringido o influir sobre los resultados de competencias. A nivel local se usan técnicas como validaciones de consistencia, cifrado y ofuscación; en tanto que en servidores se implementan validaciones, comunicaciones seguras (SLL/TLS), monitoreo de registros y restricciones en la ejecución de cálculos por parte de clientes.



Figura 52: *Un jugador que participa en torneos requiere que su sistema de puntuación esté intacto*

MONITOREO DE EVENTOS Y ACCIONES

Se refiere al análisis de registros y patrones de juego para detectar cualquier comportamiento sospechoso. Por ejemplo: identificar que un jugador ha ganado una gran cantidad de monedas o diamantes en un tiempo considerablemente corto, cuando antes requería semanas para obtener estos elementos.

SEGURIDAD DE LAS COMUNICACIONES

Involucra la implementación de técnicas de cifrado de la información, el uso de protocolos seguros para la transmisión/recepción de datos y la verificación del material transmitido.



Figura 53: *La seguridad en las comunicaciones. Un aspecto clave frente a los ataques de Hombre en el medio*

PROTECCIÓN DE ACTIVOS

Es importante proteger elementos, como gráficos o sonidos, de su robo o divulgación no autorizada. Se pueden agregar aplicaciones de validación de derechos de autor.

HERRAMIENTAS

La revisión de la seguridad de las aplicaciones se apoya en una multitud de herramientas que pueden ejecutar pruebas sobre el sistema operativo, evaluar temas específicos (por ejemplo, comunicaciones) o simular situaciones de ataque a servidores. Entre algunas, se puede mencionar:

• Ingest and make sense of any data — regardless of source or structure.

• Support insights and analytics across functional domains, leveraging commonly shared data.

• Build integrated analytics and connect service experience to the network and infrastructure.

• Incorporate machine learning and a rich array of capabilities as your business moves forward.

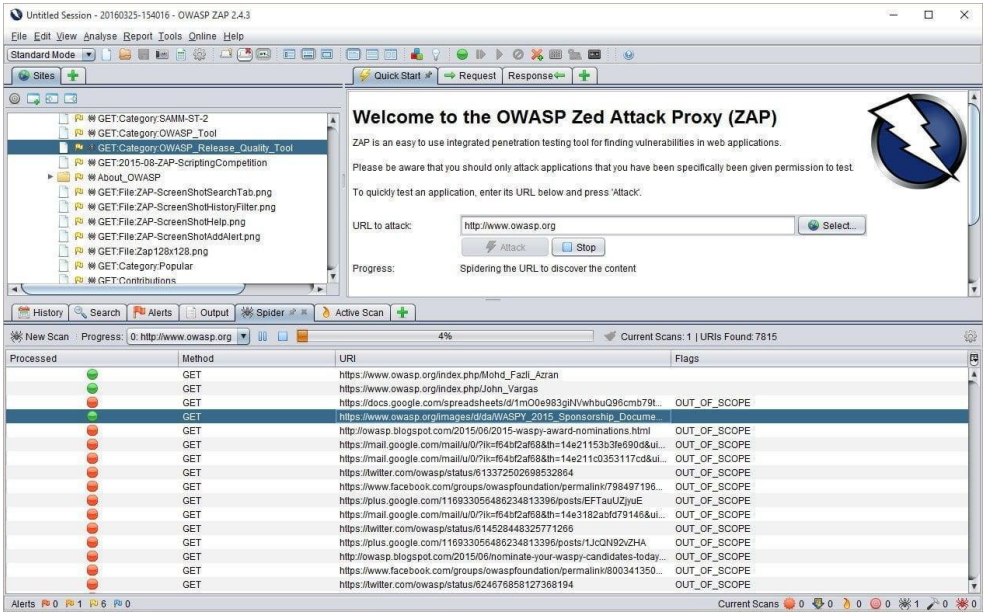


Figura 54: *OWASP ZAP*

• **OWASP ZAP:** Se enfoca en la revisión de vulnerabilidades basada en web (Ackroyd, n.d.).

• **Splunk:** Es una solución de monitoreo de datos empresariales que permite identificar riesgos y amenazas. Incorpora herramientas de análisis a través de dominios funcionales y de aprendizaje automático (Splunk, n.d.).

• **Veracode:** Es una plataforma que incorpora pruebas de seguridad estáticas, dinámicas, de composición de software, entre otras (Veracode, n.d.).

• **Metasploit:** Verifica problemas de seguridad y ejecuta pruebas de penetración o pentesting (Metasploit, n.d.).

• **Kali Linux:** Incluye una multitud de herramientas de seguridad informática.

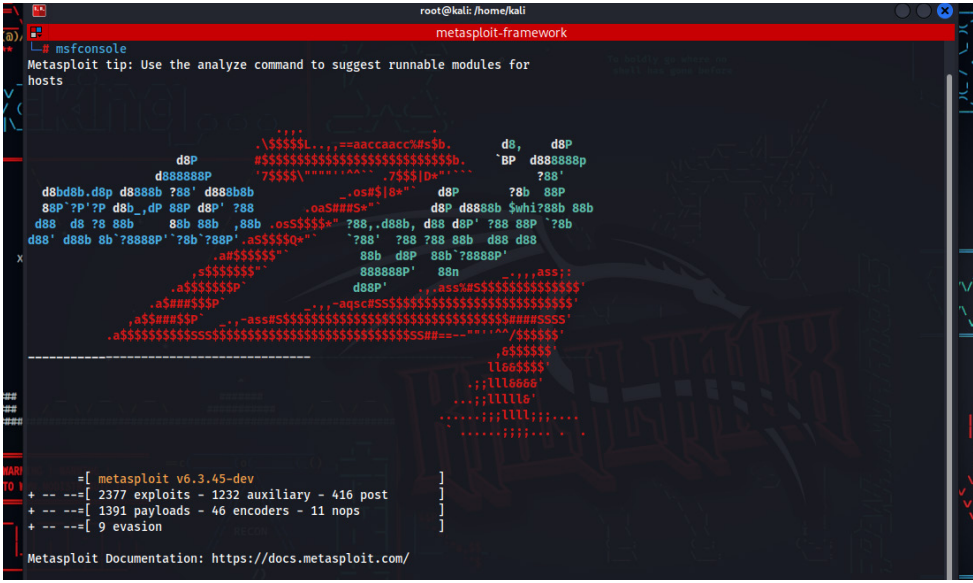


Figura 55: *Entorno de Metasploit*

Es importante señalar que programas como Unity, Unreal Engine o cualquier IDE avanzado, poseen funcionalidades nativas para ejecutar pruebas de seguridad, a la vez que son compatibles con extensiones orientadas a satisfacer estos requisitos. Otro aspecto por considerar está relacionado con la revisión de la calidad del código, para lo que utilizamos herramientas como:

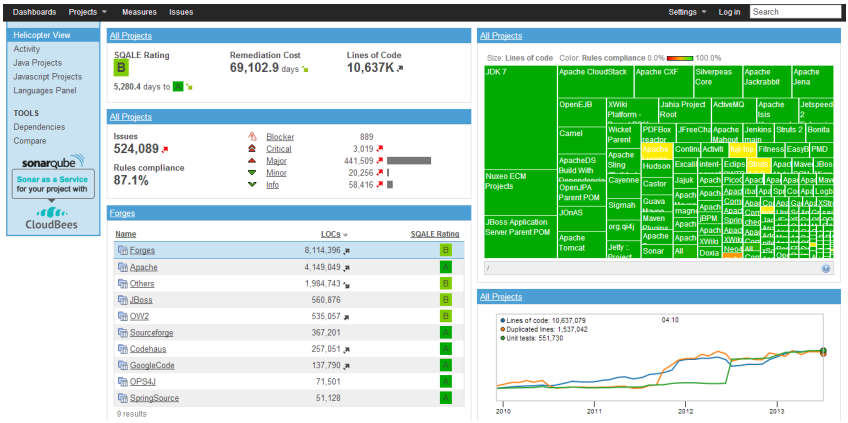


Figura 56: *Plataforma Sonar Qube*

• **SonarQube:** Se enfoca en la inspección continua de calidad de código usando herramientas de análisis estático (Sentrio, 2021).

• **Windows Application Verifier:** Ejecuta pruebas sobre el código para asegurar su calidad. Por ejemplo, revisar el manejo de excepciones o verificaciones sobre el control de lista de acceso (Microsoft, 2022).

• **Deepsource:** Realiza una revisión del código generado antes de integrarlo en un proyecto principal, a fin de encontrar y corregir problemas de calidad (DeepSource, 2023).

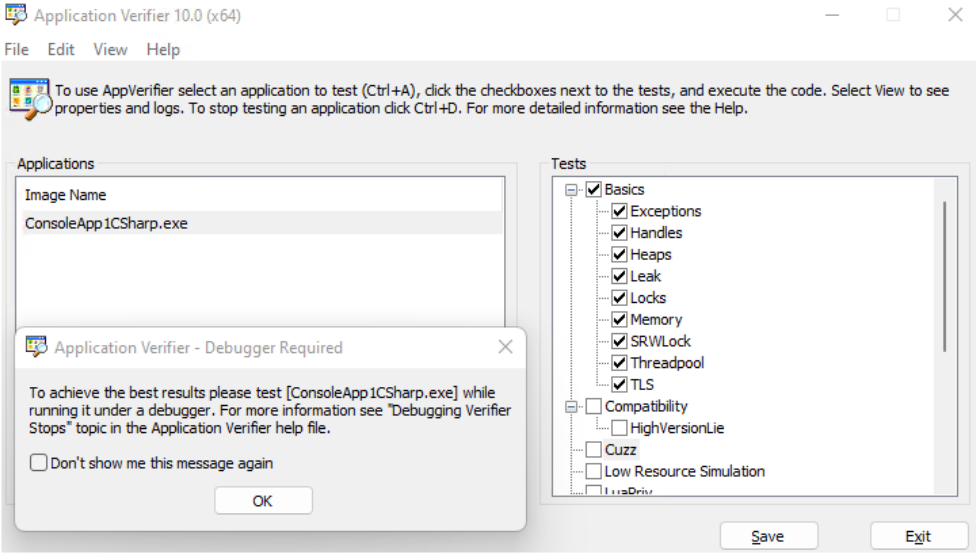


Figura 57 : *Microsoft Application Verifier*

Las herramientas mencionadas involucran una pequeña parte de las aplicaciones disponibles para la gestión de la seguridad en los juegos. Finalmente, existen algunos estándares de seguridad como OWASP (OWASP Foundation, n.d.) o NIST, que proporcionan abundante información y orientación acerca de mejores prácticas y recomendaciones de seguridad.

PROCESO DE PRUEBAS

El proceso de pruebas de los videojuegos debe ser incluido en todas las etapas de desarrollo, desde el diseño hasta la puesta en producción. Los métodos tradicionales generalmente incluyen una sección para la ejecución de pruebas, sin embargo, las metodologías ágiles proponen ciclos de desarrollo más cortos, lo que permite ejecutar planes de pruebas puntuales para requisitos más acotados.

Como primer paso, se debe realizar una planificación de las pruebas, lo que involucra la definición del alcance, objetivos, estrategias, requisitos del proyecto, recursos disponibles y audiencias objetivo. Posteriormente, de acuerdo a la madurez del producto, se contemplan distintos estilos de pruebas (QA-TestLab, n.d.) (Kevuru Games, 2023).

CONCEPTO/PROTOTIPO

Las pruebas se ejecutan en etapas tempranas de desarrollo, cuando aún no se implementa gran parte de la funcionalidad. Involucran tests de humo (smoking test) o de capacidad.

ALPHA

Pruebas diversificadas de alta calidad. Se realizan sobre un entorno controlado para identificar errores, problemas de usabilidad y asegurar el cumplimiento de metas funcionales y de diseño.

BETA

Las pruebas son ejecutadas por un grupo limitado de usuarios externos, con el objetivo de validar el juego en escenarios reales

ETAPA PREPRODUCCIÓN

Revisión final de errores, considerando características de estabilidad, desempeño y compatibilidad.

Luego del lanzamiento inicial, es necesario ejecutar pruebas continuas mientras el juego esté activo, para asegurar la calidad de las actualizaciones, corrección de errores, mejoras de rendimiento, etc.

HERRAMIENTAS Y TECNOLOGÍAS DE PRUEBAS

Entre algunas de las herramientas de pruebas de videojuegos más populares podemos mencionar:

UNITY TEST FRAMEWORK

El *Unity Test Framework (UTF)* es un marco de pruebas de Unity, orientado a garantizar la calidad de aplicaciones para diversos entornos, como Android o iOS. Permite la generación y ejecución de scripts de pruebas, así como la definición de llamadas y puntos de entrada de un proceso (Unity Technologies, n.d.).

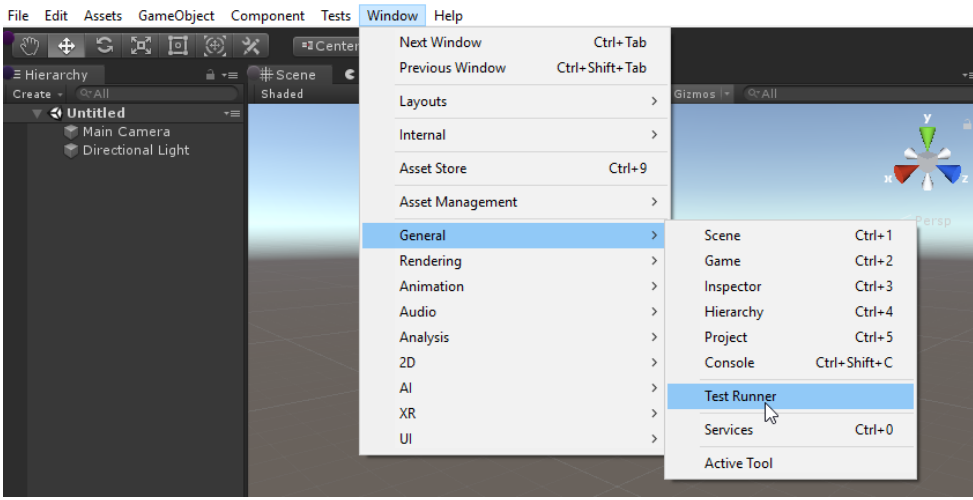


Figura 58: *Unity Test Framework*

Se integra con *NUnit*, que es un marco código abierto usado para la ejecución de pruebas unitarias de múltiples programas o segmentos de código, desarrollado para las tecnologías de .NET (como C#, Visual Basic, etc.). Soporta la creación y ejecución de tests automatizados, además de tener varios atributos para configurar la ejecución de pruebas en distintos entornos (Mehta, 2023) (Murari, n.d.).

TESTCOMPLETE

Herramienta automatizada de pruebas para interfaces de usuario (IU) que soporta aplicaciones de escritorio, web o para dispositivos móviles. Permite generar scripts y exportarlos a varios lenguajes o marcos de pruebas, como MSTest o Junit. Soporta la creación de scripts complejos basados en grabaciones, reconocimiento de objetos de la interfaz gráfica a través de inteligencia artificial y pruebas orientadas a datos (Data driven). Es compatible con otras herramientas como Git o Jenkins (Smart Bear, n.d.).

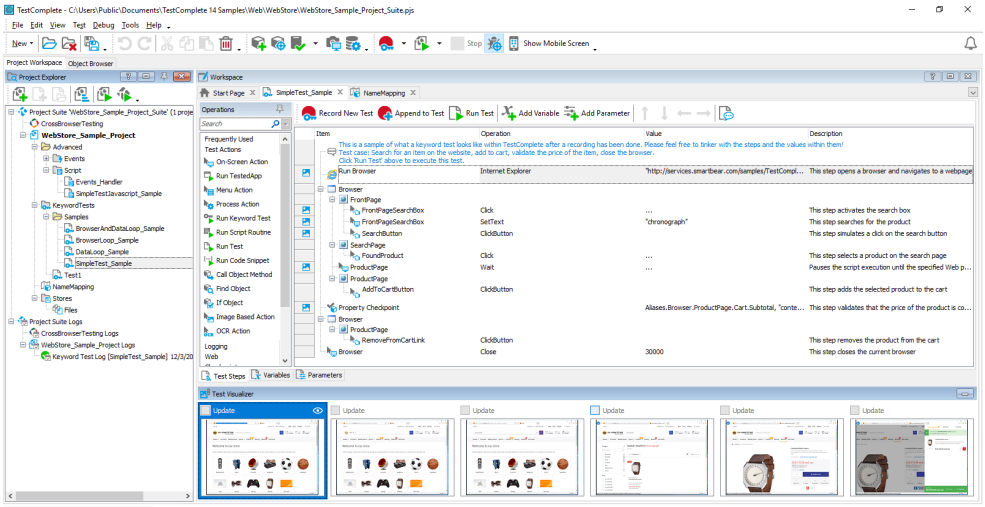
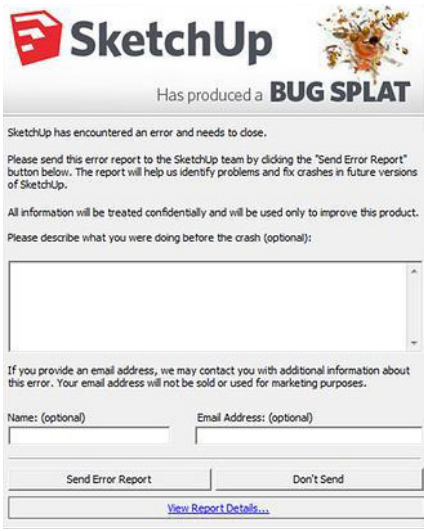


Figura 59: *TestComplete Framework*

BUGSPLAT

Es una herramienta multiplaforma compatible con diversos entornos que incluyen Unity, Unreal, PlayStation, Python, Angular, React, entre otros. Rastrea los errores y caídas del sistema para corregirlos, simplifica el proceso de

soporte automatizado y genera reportes de evaluación de desempeño de juegos, monitoreando su rendimiento (BugSplat, n.d.).



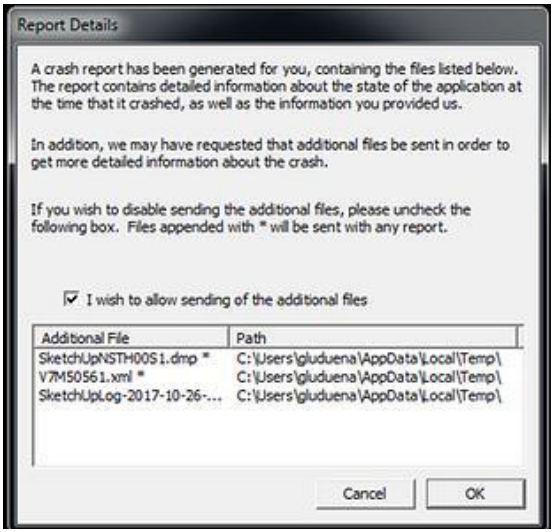


Figura 60: *Bugsplat*

GAMEBENCH – STUDIO SDK

Aplicación de gestión de desempeño que monitorea, alerta y analiza aplicaciones móviles en cada etapa de su desarrollo (Android o iOS). Ofrece una integración con motores de juegos, como Unity, a la vez que evalúa métricas como ocupación del CPU, GPU, tasa de cuadros, uso de memoria y batería.

Ofrece características como pruebas de regresión, pruebas automatizadas, pruebas de características y contenido, etc. (GameBench, 2022).

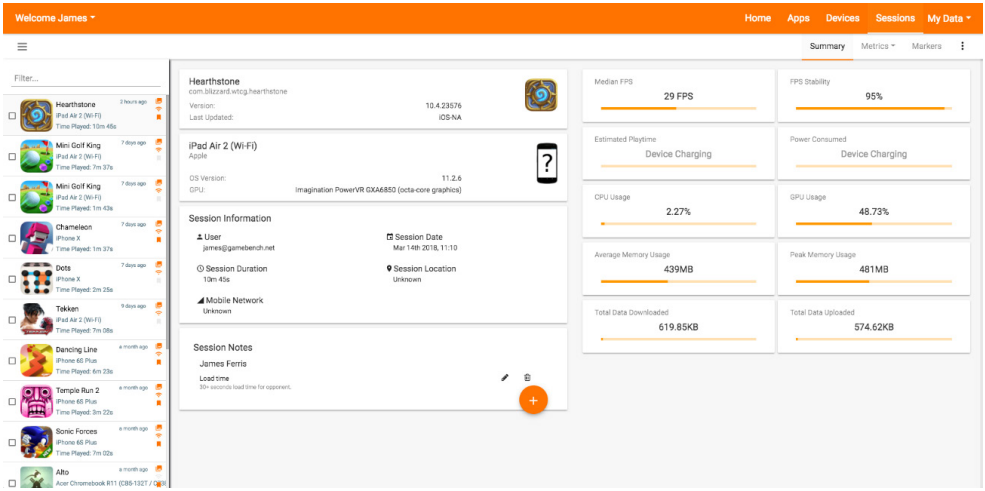


Figura 61: *Gamebench – Studio SDK*

APPIUM

Es un framework de pruebas automatizadas para aplicaciones móviles nativas e híbridas. Usa JSONWireProtocol para interactuar con aplicaciones de iOS y Android, pudiendo ejecutar scripts de pruebas en ambos ambientes. Soporta lenguajes como Java, Objective-C, Javascript y Python, entre otros (iXie Gaming, n.d.).

Adicionalmente, otros motores de juego o entornos de desarrollo, como Unreal, poseen sus propias herramientas de pruebas o automatización.

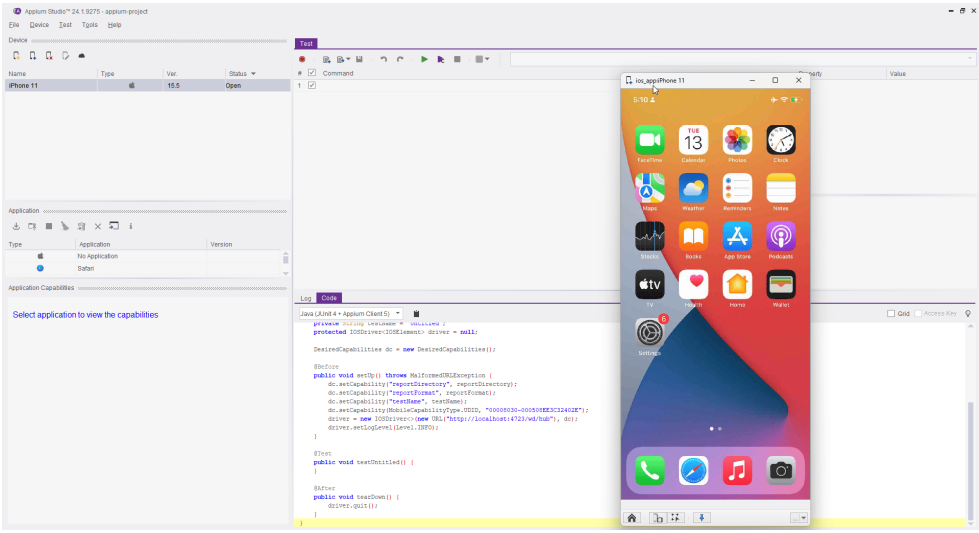


Figura 62: Appium

DESAFÍOS Y SOLUCIONES EN LAS PRUEBAS DE VIDEOJUEGOS

La ejecución de pruebas en los videojuegos es un proceso complejo, cuyas particularidades impiden que las técnicas de validación tradicionales puedan obtener los resultados deseados. Una evaluación funcional, como la correcta reproducción de un sonido o un acierto en la ejecución de transacciones en línea, evitará que el juego contenga errores que degraden la experiencia, pero no garantiza que llegue a ser divertido o capture la atención del jugador. La naturaleza de estas aplicaciones hace necesaria la implementación de metodologías de desarrollo que soporten cambios dinámicos y prototipado.

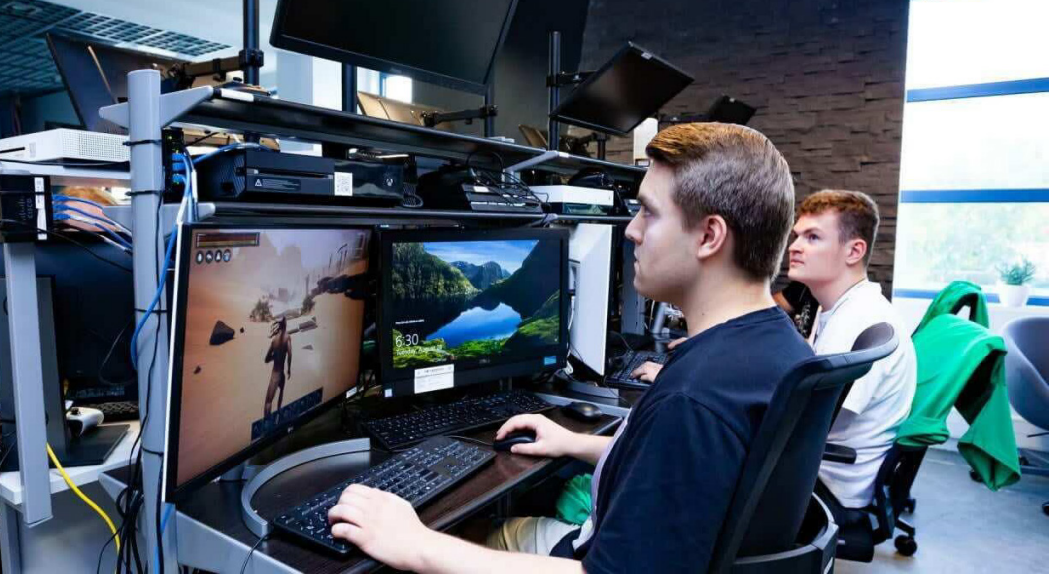


Figura 63: *Probando videojuegos. Fuente: Argentics*

Otro aspecto por considerar se relaciona con la plataforma hacia la que va dirigida el videojuego. Los juegos para dispositivos móviles no se diseñan para venderse una vez, sino que van agregando características paulatinamente para mantenerse en la mente de su público. Esto se puede apreciar en juegos como Clash Royale, Candy Crush o Free Fire, que continuamente añaden nuevos items o paquetes promocionales, por lo que los cambios son más acotados y el ciclo de desarrollo es más corto. Aproximaciones similares se tienen dentro de las aplicaciones de escritorio a través de los contenidos descargables (DLC), sin embargo, es más usual que estos juegos lancen una nueva versión cada cierto tiempo, en lugar de extender la versión original indefinidamente.



Figura 64: *Probando videojuegos. Fuente: Argentics*

De acuerdo con (Politowski et al., 2021), algunos de los problemas relacionados a las pruebas en los videojuegos incluyen:

- Pruebas insuficientes
- Inconvenientes con el plan de pruebas
- Requisitos específicos
- Problemas en la retroalimentación
- Alcance
- Reproducibilidad de errores

Los tres primeros elementos constituyen más del 70% de los problemas detectados. Entre las posibles causas se citan: subestimar la cantidad de unidades de prueba requeridas, la incorporación tardía de nuevos equipos de prueba, falencias en pruebas de regresión, uso de herramientas que no incorporan planes de prueba con facilidad, limitaciones al reproducir errores en juegos en línea—multijugador, falta de automatización de los procesos, limitaciones de hardware o interacción



Figura 65: *Probando videojuegos. Fuente: Business Insider*

La conformación del equipo de pruebas constituye otro problema representativo, ya que muchas compañías separan a los "testers" del personal con conocimientos informáticos por su costo inferior. Esto provoca que cualquier cambio no se comunique directamente y sea difícil de reproducir, lo que afecta la calidad en la corrección de los errores. Por otra parte, la incorporación de herramientas automáticas puede ayudar a la ejecución de pruebas funcionales más minuciosas y rápidas, generando una documentación que permite comprender mejor los errores, mientras delimita el alcance de las correcciones de forma más clara. Con ello se evita una cadena de problemas causada por una corrección mal ejecutada, sin embargo, este proceso tiene sus propias limitaciones ya que la evaluación del factor de diversión es una tarea subjetiva que generalmente se asigna a personal humano.



Figura 66: *Probando videojuegos. Fuente: (nyfa, 2015)*

PARA SABER MÁS...

El artículo titulado T*hree Devastating Problems To Look Out For While Game Testing de NYFA (nyfa, 2015),* Game Design habla sobre los tres problemas devastadores que los desarrolladores de videojuegos deben tener en cuenta durante las pruebas de juego.
Estos problemas pueden afectar negativamente la experiencia del jugador y la calidad del juego antes de su lanzamiento, los cuales son:

Loopholes (lagunas): Problemas en el juego que permiten a los jugadores obtener una ventaja injusta. Estos exploits deben ser corregidos para evitar que el juego sea frustrante para otros jugadores.

Gameplay Problems (problemas de jugabilidad): Problemas con las reglas del juego que pueden frustrar a los jugadores. Estos problemas pueden surgir debido a la complejidad del juego y deben ser identificados y solucionados durante las pruebas.

Bugs y glitches: Errores y fallos técnicos que pueden interrumpir el juego y deben ser detectados y corregidos antes del lanzamiento.

Finalmente, los videojuegos deben considerar muchos más parámetros de prueba que sólo los funcionales o los relacionados al factor de diversión. Características como el desempeño, usabilidad o facilidad de reverso ante fallas, también constituyen elementos críticos. Definir un balance adecuado en la dificultad evita la frustración del jugador, mientras que una revisión lingüística y cultural exhaustiva, previene que el juego utilice expresiones o plantee situaciones que podrían resultar inapropiadas en diferentes regiones.

CASOS DE ESTUDIO

A continuación, revisaremos algunos casos de estudio sobre la gestión de errores y su resolución en el ámbito de los videojuegos.

CYBERPUNK 2077

Cyberpunk 2077 es un juego RPG de acción de mundo abierto desarrollado por CD Projekt Red, lanzado en 2020 para PC, PlayStation 5, Xbox Series X|S, Playstation 4 y Xbox One. Luego de su lanzamiento, las versiones para las plataformas más antiguas fueron ampliamente criticadas debido a la cantidad de errores que contenían, lo que hacía que la aplicación prácticamente no se pudiera jugar, incluso en consolas más potentes no era extraño que surgieran problemas que hacían que el juego colapse. La aplicación se retiró de las tiendas de Sony y Microsoft, además de que tuvo que afrontar un desplome de las acciones de la compañía, así como una gran cantidad de pedidos de reembolso (Baisden, 2021; Montero, 2021).

Se considera que la empresa desarrolladora no tenía la suficiente experiencia, generó expectativas poco realistas y se enfocó en realizar el desarrollo/pruebas principalmente para la versión de PC, dejando de lado a consolas con recursos más limitados. Entre sus alegatos, CD Projekt Red mencionó que la compañía externa encargada del aseguramiento de calidad había mentido sobre la cantidad de personal asignado, su experiencia y que frecuentemente se enviaban reportes de errores, que no eran críticos, solo para cumplir una cuota; lo que complicaba el proceso de corrección antes del lanzamiento (Ampoloquio, 2022).



Figura 67: *Uno de los numerosos bugs en Cyberpunk 2077*

En la actualidad, el juego se ha estabilizado en sus versiones para plataformas con mayores recursos, sin embargo, a la compañía le tomó mucho tiempo y dinero alcanzar esta etapa.

ASSASSIN'S CREED UNITY

Desarrollado por Ubisoft y lanzado en noviembre de 2014, es un juego de franquicia que involucró en su desarrollo a nueve estudios y a más de mil personas, en un lapso de tres años. El nivel de detalle llegó al punto de que se generó un modelo digital completo de la Catedral de Notre Dame, el mismo que luego sería utilizado en las labores de reconstrucción tras su incendio en 2019.

La cantidad de personal involucrado trabajando en paralelo en cada estudio, un motor gráfico nuevo e inmaduro, así como la incorporación de micro transacciones, provocó que la aplicación colapsara frecuentemente, la cantidad de cuadros por segundo bajara e incluso, ciertas texturas de las caras de los personajes no se cargaban adecuadamente haciendo que se muestren "transparentes".



Figura 68: *Error en Assassin's Creed Unity (Partridge, 2016)*

BIBLIOGRAFÍA

Ackroyd, R. (n.d.). Zed Attack Proxy—An overview. ScienceDirect Topics. https://www.sciencedirect.com/topics/computer-science/zed-attack-proxy

Allegro Development Team. (n.d.). Allegro: A game programming library. https://liballeg.org/index.html

Altbyte. (n.d.). MelonJS. https://melonjs.org/

Ampoloquio, R. (2022, June 27). New report suggests QA company to blame for Cyberpunk 2077 bugs. Xfire. https://www.xfire.com/report-qa-company-blame-cyberpunk-2077-bugs/

Andrade, A. (2015). Game engines: A survey. EAI Endorsed Transactions on Serious Games, 2(6). https://doi.org/10.4108/eai.5-11-2015.150615

Baisden, A. (2021, January 8). How developers can learn from the mistakes of Cyberpunk 2077. DEV Community. https://dev.to/andrewbaisden/how-developers-can-learn-from-the-mistakes-of-cyberpunk-2077-1pi4

Blanco, V. (2020). VulkanGuide. Vulkan Guide. https://www.vkguide.dev/

Bogdan, L., Maletska, M., Khrypko, S., Leonid, K., Olga, D., & Pasko, K. (2021). Videogames in cybersecurity: Philosophical and psychological review of possible impact. International Journal of Computer Science and Network Security, 21(7), 249–256. https://doi.org/10.22937/IJCSNS.2021.21.7.29

BugSplat. (n.d.). BugSplat: Crash and error reporting for busy developers. https://www.bugsplat.com

Chen, L., Shashidhar, N., Rawat, D., Yang, M., & Kadlec, C. (2016). Investigating the security and digital forensics of video games and gaming systems: A study of PC games and PS4 console. In Proceedings of the 2016 International Conference on Computing, Networking and Communications (ICNC) (pp. 1–5). https://doi.org/10.1109/ICCNC.2016.7440557

DeepSource. (2023). DeepSource: The code health platform. https://deepsource.com/

Epic Games. (2022a). Unreal Engine: Funciones. https://www.unrealengine.com/es-ES/features

Epic Games. (2022b). Utilizing Python for editor scripting in Unreal Engine. https://dev.epicgames.com/community/learning/courses/wk4/utilizing-python-for-editor-scripting-in-unreal-engine

Esains, I. (2020, May 5). Juegos condenados: Assassin's Creed Unity y los bugs más memorables de todos los tiempos. Cultura Geek. https://culturageek.com.ar/juegos-condenados-assassins-creed-unity-y-los-bugs-mas-memorables-de-todos-los-tiempos/

Foxman, M. (2019). United we stand: Platforms, tools and innovation with the Unity game engine. Social Media + Society, 5(4). https://doi.org/10.1177/2056305119880177

Galehantomo P. S., G. (2015). Platform comparison between games console, mobile games, and PC games. SISFORMA, 2(1), 23–26. https://doi.org/10.24167/sisforma.v2i1.407

GameBench. (2022, March 29). GameBench. https://www.gamebench.net/for-publishers/

Godot Foundation. (n.d.). Godot Engine. https://godotengine.org/

Goel, A., & Semah, B. (2023). 7 best programming languages for game development in 2023. https://hackr.io/blog/best-programming-language-for-games

InformIT. (n.d.). Why you should be using the Unity game engine. https://www.informit.com/articles/article.aspx?p=2031153

iXie Gaming. (n.d.). Game test automation. https://www.ixiegaming.com/whitepapers/

Jensen, T. (n.d.). 25 years later: The history of Unreal and an epic dynasty. PCMag. https://www.pcmag.com/news/25-years-later-the-history-of-unreal-and-an-epic-dynasty

Kevuru Games. (2023, June 29). Main stages of video game testing: From concept to perfection. https://kevurugames.com/blog/main-stages-of-video-game-testing-from-concept-to-perfection/

Khronos Group. (n.d.). OpenGL: The industry standard for high performance graphics. https://www.opengl.org/

Kozacz, H. (n.d.). Python for game development: Should you develop games in Python? https://www.stxnext.com/blog/python-for-game-development/

Lantinga, S. (n.d.). Simple DirectMedia Layer (SDL). https://libsdl.org/

McShaffry, M., Graham, D., & Graham, D. R. (2013). Game coding complete (4th ed.). Delmar Learning.

Mehta, P. (2023, October 16). Unit testing in .NET Core using NUnit. Positiwise. https://positiwise.com/blog/unit-testing-in-net-core-using-nunit

Metasploit. (n.d.). Metasploit: Penetration testing software. https://www.metasploit.com/

Microsoft. (2022, June 24). Application Verifier: Overview. https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/application-verifier

Microsoft. (2023a). JavaScript. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript

Microsoft. (2023b). Game development ecosystem. https://dotnet.microsoft.com/en-us/apps/games/ecosystem

Microsoft. (n.d.). C# docs. https://learn.microsoft.com/en-us/dotnet/csharp/

Montero, J. (2021). Cyberpunk 2077: El videojuego que pasó del "hype" al fracaso en una semana. Expreso. https://www.expreso.ec/ciencia-y-tecnologia/cyberpunk-2077-videojuego-paso-hype-fracaso-semana-105760.html

Murari, K. (n.d.). Selenium NUnit tutorial: A comprehensive guide with examples and best practices. LambdaTest. https://www.lambdatest.com/learning-hub/nunit-tutorial

NYFA. (2015, September 29). Three devastating problems to look out for while game testing. https://www.nyfa.edu/student-resources/three-devastating-problems-to-look-out-for-while-game-testing/

Oliveira, M. (2022, September 24). Top 5 libraries for game development with C++. https://terminalroot.com/top-5-libraries-for-game-development-with-cpp/

OWASP Foundation. (n.d.). Threat modeling. https://owasp.org/www-community/Threat_Modeling

Panda3D.org. (n.d.). Introduction to Panda3D. https://docs.panda3d.org/1.10/python/introduction/index

Partridge, J. (2016, August 2). The 10 most awkward bugs in gaming history. Red Bull. https://www.redbull.com/in-en/the-10-most-awkward-bugs-in-gaming-history

Plank-Blasko, D. (2015). "From Russia with fun!": Tetris, Korobeiniki and the ludic Soviet. The Soundtrack, 8(1), 7–24. https://doi.org/10.1386/st.8.1-2.7_1

Politowski, C., Petrillo, F., & Gueheneuc, Y.-G. (2021). A survey of video game testing. In 2021 IEEE/ACM International Conference on Automation of Software Test (AST) (pp. 90–99). https://doi.org/10.1109/AST52587.2021.00018

Pygame.org. (n.d.). About: Pygame Wiki. https://www.pygame.org/wiki/about

Pyglet.org. (n.d.). Pyglet. https://pyglet.org/

Python Institute. (n.d.). About Python. https://pythoninstitute.org/about-python

QATestLab. (n.d.). Video game testing guide. https://qatestlab.com/resources/whitepapers/video-game-testing-guide/

Renard, E. (2022, April 18). Python vs C++: Which one should you use? Udemy Blog. https://blog.udemy.com/python-vs-c-plus-plus/

Santamaria, R. (n.d.). Raylib. https://www.raylib.com

Scirra. (n.d.). Construct 3. https://www.construct.net

Sentrio. (2021, December 15). Qué es SonarQube: Verifica y analiza la calidad de tu código. https://sentrio.io/blog/que-es-sonarqube/

SFML Team. (n.d.). SFML. https://www.sfml-dev.org/

SmartBear. (n.d.). TestComplete automated UI testing. https://smartbear.com/product/testcomplete/features/

Splunk. (n.d.). Splunk for gaming and software companies [PDF]. https://www.splunk.com/en_us/pdfs/solution-guide/splunk-for-gaming-and-software-companies.pdf

Testbytes. (2023). 9 different types of game testing techniques. Testbytes Softwares. https://www.testbytes.net/blog/types-of-game-testing/

TIOBE. (2023). TIOBE Index. https://www.tiobe.com/tiobe-index/

Tristem, B. (2022, May 6). Unity vs. Unreal: Which one is best for you? Udemy Blog. https://blog.udemy.com/unity-vs-unreal-which-game-engine-is-best-for-you/

Unity Technologies. (n.d.-a). Learn how to code in C# for beginners. https://unity.com/how-to/learning-c-sharp-unity-beginners

Unity Technologies. (n.d.-b). 2021 Gaming Report. https://create.unity.com/2021-game-report

Unity Technologies. (n.d.-c). Unity visual scripting. https://unity.com/features/unity-visual-scripting

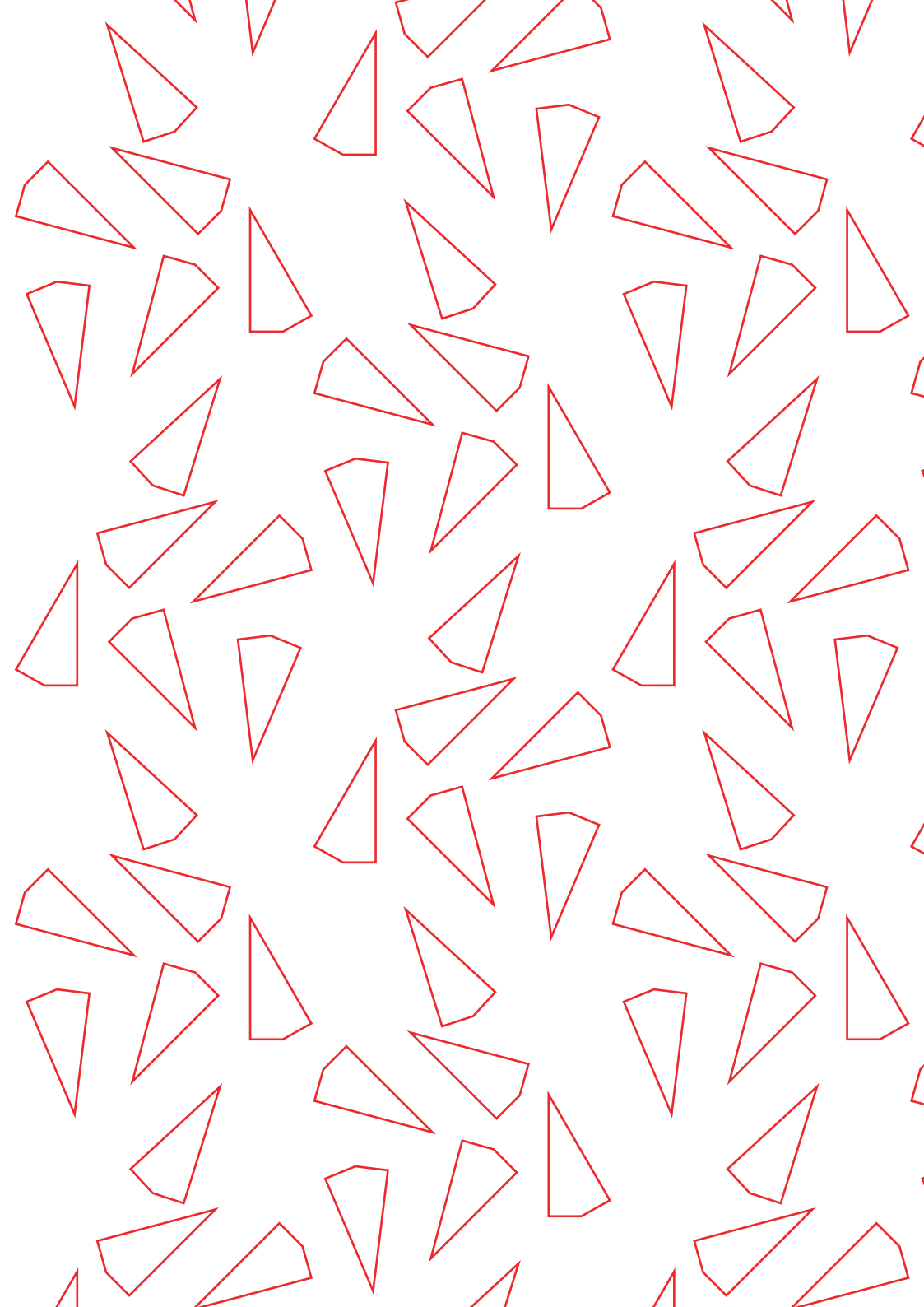
Unity Technologies. (n.d.). Unity test framework for video game development. https://unity.com/how-to/unity-test-framework-video-game-development

Veracode. (n.d.). Software testing. https://www.veracode.com/security/software-testing

Vulkan. (2023, August 22). Vulkan: Cross-platform 3D graphics. https://www.vulkan.org/

Zhao, C. (2018). Cyber security issues in online games. AIP Conference Proceedings, 040015. https://doi.org/10.1063/1.5033679

Este libro se terminó de imprimir y encuadernar en octubre de 2026 en el PrintLab de la Universidad del Azuay, en Cuenca del Ecuador.



Lleva tu gran idea del papel al código fuente. En este volumen comprenderás cómo estructurar niveles y objetivos claros que mantengan la atención del jugador. Al igual que un autor da vida a su obra, aprenderás técnicas esenciales de desarrollo y la importancia crítica de testear rigurosamente cada detalle antes de lanzar tu videojuego al mercado.

UNIVERSIDAD DEL AZUAY | Casa Editora

ISBN: 978-9942-54-188-8

PRO EDIT