**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | Desarrollo de videojuegos y entornos interactivos |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501090 - Probar el videojuego de acuerdo con el procedimiento técnico y herramientas de  Desarrollo. | RESULTADOS DE APRENDIZAJE | 220501090-01 - Planear el proceso de prueba del videojuego de acuerdo al diseño.  220501090-02 - Probar el videojuego de acuerdo a la planeación.  220501090-03 - Validar el videojuego de acuerdo con el proceso de prueba.  220501090-04 - Corregir el videojuego de acuerdo con el proceso de validación. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 011 |
| NOMBRE DEL COMPONENTE FORMATIVO | Técnicas de testeo y pruebas para videojuegos |
| BREVE DESCRIPCIÓN | El componente de formación hace énfasis en las pruebas del prototipo del juego las cuales comienzan desde la etapa de preproducción, para optimizar el videojuego y corregir los errores o bugs en dicho videojuego. |
| PALABRAS CLAVE | Department QA, prototipo, pruebas, *testing*. |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | 5 - ARTE, CULTURA, ESPARCIMIENTO Y DEPORTES |
| IDIOMA | Español |

1. **TABLA DE CONTENIDOS:**

**Introducción**

**1. ¿Por qué es necesario probar los juegos?**

1.1. Tipos de técnicas de pruebas de juegos

1.2. Métodos de prueba de juegos

1.3. ¿Qué es la automatización de pruebas de juegos?

1.4. ¿Qué hacen los QA *testers* de juegos?

1.5. ¿Cómo funciona la prueba de juegos?

1.6. Métricas clave de las pruebas funcionales de control de calidad

1.7. Informes de errores

1.8. Tipos principales de métodos de prueba de juegos

1.9. Cómo funcionan las pruebas en diferentes plataformas

**2. Cómo iniciar las pruebas o *testing* en un videojuego**

**3. Utilizando el *Framework Test Runner* *- Unit testing en unity***

**4. Pruebas del sistema**

1. **INTRODUCCIÓN**

Estimado aprendiz, continuando con este maravilloso aprendizaje, reflexione sobre lo siguiente: **Antes de que una mariposa pueda tomar su primer vuelo, debe empujar y luchar para salir de un capullo**. **Sin este desafiante obstáculo, sería demasiado débil para sobrevivir en el mundo exterior.** ¿Qué tiene esto que ver con las pruebas de juego? Bueno, como una mariposa, cada juego debe someterse a una prueba seria que lo fortalezca antes de lanzarlo al mundo. Por lo anterior le invitamos a revisar el siguiente video para identificar el contexto de aprendizaje de este componente formativo:

Video\_Introducción

1. **DESARROLLO DE CONTENIDOS:**
2. **¿Por qué es necesario probar los juegos?**

Las empresas de desarrollo de juegos deben invertir lo suficiente en las pruebas de los videojuegos para garantizar que el producto final no tenga errores o *bugs*. Eso haría decaer la experiencia del jugador y la parte central de los juegos es la experiencia del consumidor. Con una buena experiencia de usuario, un juego prosperará. Hay millones de juegos en la *App Store* y la *Play Store*, pero ser un ganador entre esta dura competencia requiere mucha paciencia y trabajo.

Con tantos juegos llegando al mercado diariamente, hay que ser diferente para destacar y dejar huella. Los niveles complejos no son suficientes para que los usuarios se enganchen a un juego y lo recomienden a sus amigos, ya que si el juego no funciona como los usuarios esperan, no tardarán en desinstalarlo y pasarse a otro.

**1.1. Tipos de técnicas de pruebas de juegos**

Los encargados de las pruebas de control de calidad en un estudio de videojuegos (QA, *Quality Assurance*) buscan los problemas y los puntos débiles que se pueden pulir antes de lanzar el producto al público, además, deben asegurarse, a través de diversas técnicas, que aspectos como la conectividad emocional con el juego o el nivel de diversión estén en óptimas condiciones.

A continuación, detallamos algunas de las técnicas más comunes utilizadas por los estudios de videojuegos y los QA *testers* especialistas en pruebas de control de calidad de juegos:

|  |
| --- |
| **Slide Simple**  CF011\_1.1\_SlideSimple\_Tipos\_técnicas\_pruebas |

**1.2. Métodos de prueba de juegos**

Aquí es donde entramos en uno de los puntos más relevantes y nos centramos en los diferentes tipos de metodologías de prueba de control de calidad de juegos disponibles.

**Pruebas de funcionalidad del juego**

Las pruebas funcionales son una actividad que tiene como objetivo determinar si un juego funciona de acuerdo con las especificaciones e identificar cualquier error o problema que pueda afectar negativamente la experiencia del jugador. Esto se aplica tanto a las pruebas de juegos móviles como a las de otras plataformas, incluidas PC, consolas, web y VR/AR.

Podemos ordenar las pruebas en múltiples subtipos:

|  |  |  |
| --- | --- | --- |
| **Subtipos** | **Descripción** | **Cuándo se necesita** |
| **Pruebas de juego de interoperabilidad** | Se utiliza para probar cómo un juego interactúa con otras aplicaciones. Los probadores buscan cualquier problema de compatibilidad, incluida la inaccesibilidad de las funciones, el bajo rendimiento y los retrasos en la comunicación. | Se utiliza principalmente para juegos multijugador y aquellos que hacen uso de funciones avanzadas del dispositivo (por ejemplo, enlace ascendente de *Internet*, *Bluetooth*, cámara, etc.). |
| **Pruebas de juegos de regresión** | Este ciclo se aplica después de las principales actualizaciones de código para asegurarse de que la actualización no haya afectado negativamente la funcionalidad existente (por ejemplo, rompiendo una función, agregando numerosos errores nuevos, etc.). El objetivo es asegurarse de que el código aún funcione. | Esta es una práctica muy común tanto durante el desarrollo (cuando se crean nuevas compilaciones) como en el período posterior al lanzamiento, cuando se implementan el mantenimiento y las actualizaciones. Puede automatizar muchas de estas pruebas. |
| **Pruebas de juegos de humo** | La prueba de humo se trata de verificar la estabilidad después de una actualización de código. Sin embargo, tiende a ser una evaluación realmente simple, verificando cosas básicas como si el juego se inicia, si la interfaz de usuario sigue respondiendo. Puede ser realizado tanto por desarrolladores como por probadores. | La prueba de humo tiende a realizarse justo antes de la prueba de regresión, por lo que podemos decir que es igual de común y esencial para todos los proyectos. Se considera una buena práctica para realizar todos los días del proyecto. |
| **Pruebas de juego de localización** | Este método tiene como objetivo garantizar que un juego sea totalmente utilizable y agradable para los jugadores de diferentes países y regiones. En primer lugar, todas las mismas características y funcionalidades deben ser accesibles en diferentes ubicaciones (a menos que se planifique lo contrario). En segundo lugar, el contenido debe adaptarse a los idiomas y culturas donde se presenta. | Este tipo de prueba es vital si planeas lanzar tu juego en varios idiomas, o si planeas hacer que el contenido esté disponible en diferentes regiones. Por lo tanto, es preferible que los evaluadores tengan un amplio conocimiento de los idiomas/culturas asociados antes de comenzar a evaluar el software. |
| **Pruebas de juego de control de acceso de seguridad** | Una práctica muy importante que verifica si existen lagunas o puertas de enlace no autorizadas que puedan permitir que alguien acceda al *backend* del juego o a elementos/características que de otro modo estarían restringidas. Por ejemplo, algunos jugadores intentan piratear juegos para obtener recompensas o dinero gratis en el juego, hacerse invencibles, etc. y el control de acceso lo impide o al menos reduce la probabilidad. | Aunque esta forma de prueba rara vez conduce a cambios que afecten la experiencia del usuario o que sean incluso perceptibles, se recomienda realizarla al menos una vez antes del lanzamiento y cada actualización posterior al lanzamiento. Puede ser realizado por ingenieros generales de control de calidad del juego o probadores de pluma (penetración) experimentados |
| **Pruebas de juego de aceptación del usuario** | Esta es una de las últimas etapas de prueba antes de lanzar un juego en producción. A veces, se lleva a cabo como una prueba beta, atrayendo a jugadores fuera del equipo de desarrollo. Rara vez se encuentran grandes problemas, por lo que la atención se centra principalmente en pulir los errores menores restantes y realizar pequeñas mejoras en la experiencia del usuario. | Al involucrar a probadores beta externos, puede obtener una nueva perspectiva sobre la experiencia del jugador y recopilar comentarios valiosos sobre posibles mejoras que el equipo de desarrollo no conocía o no prestó suficiente atención. Por ejemplo, las pruebas beta de juegos móviles pueden atraer a un grupo de edad con ideas y deseos únicos que su equipo de pruebas de adultos no tuvo en cuenta. |

**1.3. ¿Qué es la automatización de pruebas de juegos?**

La automatización se da en dos enfoques: en el control de calidad del juego y el otro es la prueba manual. Con la automatización, ciertas pruebas de rendimiento se pueden realizar muchas veces seguidas sin la participación del evaluador, esto ahorra tiempo y hace que sea mucho más probable detectar errores.

Aunque este enfoque es más conveniente que pedirles a los probadores que completen los mismos niveles muchas veces de forma manual, es bastante difícil crear herramientas de automatización adaptadas a una aplicación individual.

Según una encuesta de empresas de desarrollo de juegos, la mayoría prefiere probar manualmente sus juegos, citando la falta de recursos y tiempo para crear software de automatización.

**1.4. ¿Qué hacen los QA *testers* de juegos?**

Muchos jóvenes sueñan con convertirse en probadores de juegos porque pueden jugar todo el día, pero esto solo rasca la superficie de las responsabilidades del puesto. Aparte de revisar la mecánica del juego, tienen tareas menos románticas y dinámicas como:

1. Navegación y muestreo de todas las opciones del menú.
2. Ejecución de análisis de rendimiento y CPU.
3. Muestreo de todas las opciones de personalización.
4. Fallar intencionalmente en el juego y probar obstáculos.
5. Comprobación y análisis de las especificaciones del sistema.
6. Compilar informes y completar formularios de comentarios.
7. Ejecución de scripts de prueba y casos de prueba para aplicaciones de juegos.

Como se puede observar, hay muchas repeticiones involucradas en las pruebas de funcionalidad del juego y los expertos tienen que dedicar gran parte de su tiempo a otras tareas como hacer informes y trabajar con código y scripts. Para cualquiera que esté considerando este tipo de trabajo, no solo debe amar los juegos, sino también estar preparado para verlos desde un punto de vista analítico y constructivo.

**1.5. ¿Cómo funciona la prueba de juegos?**

Como mencionamos anteriormente, las pruebas de juegos implican más que solo iniciar un juego y jugarlo de principio a fin para asegurarse de que funcione sin problemas. Hay docenas de complejidades en el proceso de prueba del juego, así que echemos un vistazo más de cerca.

**Etapas de la prueba del juego**

La estructura general del proceso de control de calidad es simple, esto es:

Sin embargo, ¿dónde encaja la revisión de control de calidad en el gran esquema de desarrollo? Durante las fases iniciales de desarrollo (planificación y producción), los ingenieros de control de calidad tienden a tener muy poca participación. Estas etapas son manejadas principalmente por gerentes de proyecto, expertos en desarrollo comercial y desarrolladores. Aun así, se les puede pedir comentarios sobre cosas como la lógica y el flujo del juego, el diseño de UX y el diseño de interacción.

|  |  |
| --- | --- |
|  | La mayor parte del trabajo realizado por los ingenieros comienza cuando está lista una compilación jugable de la aplicación. Esto a veces se denomina **prueba alfa de videojuegos**.  La revisión se lleva a cabo dentro de los límites de una empresa, con el objetivo de solucionar la mayoría de los problemas y brindar una experiencia de juego completa y de alta calidad |

Algunas compañías optan por agregar también juegos de prueba beta al expediente. Con este enfoque, se pide a personas ajenas a la empresa que realicen una vista previa y una prueba beta de los juegos y lanzamientos de PC en otras plataformas (las consolas también son comunes). Esto brinda una comprensión mucho mejor de la impresión general que se puede esperar de los jugadores que la que una empresa podría obtener de unos pocos ingenieros de control de calidad.

**1.6. Métricas clave de las pruebas funcionales de control de calidad**

La retroalimentación sobre la calidad del juego no es algo formulado ampliamente como "El juego funciona sin problemas". Incluye cifras concretas, estadísticas y terminología que ofrece información mucho más útil.

Repasemos algunas de las métricas clave de control de calidad del juego:

|  |
| --- |
| **Tarjetas animadas**  CF011\_1.6\_TarjetasAnimadas\_Metricas\_pruebas\_funcionales |

**1.7. Informes de errores**

Los informes de errores son una parte indeleble de la revisión de control de calidad y representan la información/los comentarios recopilados en función de las pruebas de los ingenieros que se pueden utilizar para realizar mejoras adicionales.

Un informe de error tradicional incluirá los siguientes detalles:

**1.8. Tipos principales de métodos de prueba de juegos**

Cada empresa tiene sus propias percepciones sobre qué pruebas son necesarias para un juego, y la lista incluso puede variar de un proyecto a otro. Aun así, podemos delinear las pruebas más comunes, dividiéndolas en funcionales (pruebas de funcionalidad del juego) y no funcionales (asociadas con el rendimiento general y UX), veamos esto de una manera más detallada:

|  |  |
| --- | --- |
| Principales métodos de *testing* | |
| Pruebas funcionales   * Componente/módulo (comprobación del rendimiento de las unidades de *software* más pequeñas). * Integración (encontrar defectos en interfaces e interacciones de componentes). * Humo (determinando la estabilidad de construcción). * Regresión (verificación de correcciones de errores). * Localización (verificación de la consistencia del contenido traducido). * Seguridad y control de acceso (identificación de vulnerabilidades y verificación de permisos de usuarios). | Pruebas no funcionales   * Rendimiento (velocidad de carga del juego en tiempo real). * Carga/Estrés (comprobación del rendimiento en condiciones de gran actividad/tráfico de usuarios). * Instalación (qué tan bien se guarda el juego en diferentes dispositivos). * Usabilidad (conveniencia de los mecanismos de juego). * Recuperación (cómo funciona la aplicación después de fallar). |
|
|

**1.9. Cómo funcionan las pruebas en diferentes plataformas**

Los juegos a menudo se lanzan en varias plataformas a la vez e incluso dentro del alcance de una plataforma, los evaluadores deben asegurarse de que todo funcione en diferentes dispositivos. Por lo tanto, hay muchas peculiaridades en lo que respecta a la revisión de control de calidad en diferentes plataformas, esto se entenderá de manera más detallada a continuación:

|  |
| --- |
| **Pestañas horizontales sencillas**  CF011\_1.9\_PestañasHSencillas\_Pruebas\_diferentes\_Plataformas |

**2. Cómo iniciar las pruebas o *testing* en un videojuego**

En teoría, cualquiera puede aprender a probar un juego, incluso si no está capacitado para ser un ingeniero de control de calidad. Sin embargo, tener éxito y eficacia en las pruebas es otra cuestión, y los proyectos a gran escala generalmente necesitan evaluadores calificados para tener éxito. Por lo tanto, puede considerar contratar a estos profesionales o delegar las pruebas a una empresa con amplia experiencia en el campo.

Cualquiera que sea la opción que elija, el proceso de prueba será más o *menos* el mismo. Entendemos que las docenas de tipos de pruebas que enumeramos son únicas y requieren software, metodología y conocimientos diferentes, pero las etapas de las pruebas suelen ser uniformes como se observa a continuación:

|  |
| --- |
| **Pasos vertical**  CF011\_2\_PasosVertical\_etapas\_pruebas\_videojuegos |

**Evaluación y pruebas**

Este apartado tiene como objetivo comentar las pruebas realizadas en el juego desarrollado. Las pruebas son un elemento fundamental en un desarrollo *software* y deben realizarse siempre para evitar fallos no deseados. Existen los siguientes tipos de pruebas:

|  |
| --- |
| **Tarjetas conectadas**  CF011\_2\_TarjetasConectadas\_Tipos de pruebas |

**3. Utilizando el *Framework Test Runner* - *Unit testing en unity***

Se ha decidido crear un solo epígrafe para las pruebas tanto unitarias como de integración dado que, por la naturaleza del propio programa, *Unity* y de los juegos como *software* desarrollado es casi imposible encontrar elementos aislados de código que funcionen sin depender de otros.

Es importante mencionar que en este apartado aparte de las pruebas realizadas también se han considerado como pruebas los *logs* generados durante el desarrollo del proyecto, así como las ejecuciones en el editor como parte de la integración porque al tratarse de un juego, los elementos debían coordinarse a la perfección para no obtener *bugs* indeseados.

De esta manera se subsanan errores que aparecían al producirse excepciones en el código. Para llevar a cabo estas pruebas se ha hecho uso de una funcionalidad implementada por *Unity* llamada *Test Runner* que permite su configuración y ejecución.

|  |  |
| --- | --- |
| Foto la empresaria apuntando a la prueba de palabras contra la pared gris | Algo que la mayoría de las veces solemos dejar completamente de lado a la hora de desarrollar nuevas *features* en nuestros juegos son, sin duda, los *Unit Tests*.  El *Unity Test Runner* hace uso de la librería *NUnit* que es una librería de pruebas de software libre para lenguajes .*Net*. El *Test Runner* permite ejecutar los test tanto en modo editor (*edit mode*) como en modo de juego *(play mode*). |

Sin embargo, el uso de estas herramientas que proporciona el software de Unity 3D se deja casi siempre de lado ya sea por poco conocimiento sobre el uso de este *framework,* ya que este posee muchas bondades. De esta manera cabe resaltar que la implementación de estos *test* en nuestros proyectos siempre nos va a demandar un buen gasto de tiempo y esfuerzo extra que no siempre podemos o estamos dispuestos a perder, sumado a la falsa sensación de pérdida de tiempo por estar invirtiendo una parte importante de nuestro trabajo en desarrollar código que ni siquiera pertenece al producto final, puede que el hacer un repaso de algunos de sus pros y contras nos haga verlo de otro modo. Al respecto:

|  |  |
| --- | --- |
| ***Ventajas***   * Aplicar *test* en nuestro código nos da seguridad y confianza en el comportamiento esperado de las lógicas que implementamos. * Algo muy importante es que sirve como documentación para nuevos desarrolladores que trabajen en tu código. * Nos obliga a implementar las cosas enfocándonos siempre a su testeo. * Nos ayuda a anticipar y localizar posibles bugs de manera más rápida y eficiente. * Nos convierte en mejores profesionales. | ***Desventajas***   * Inversión de tiempo extra de desarrollo (a veces puede llegar a ser demasiado). * No siempre es fácil testear todo (por ejemplo, comportamientos de UI). * No implementarlos de manera adecuada puede crearnos una falsa seguridad que puede ser peligrosa. |

La clave está en mantener el equilibrio que las propias circunstancias de nuestro proyecto nos permitan. No se trata de hacer *testing* de todo o de nada, siempre va a ser mejor hacerlo de algunas partes que no hacerlo de nada.

Por ello, creo que es posible buscar siempre un balance que nos permita llevar adelante el desarrollo de nuestro proyecto de manera que encaje en tiempos/planificaciones y, a su vez, tenga en cuenta la implementación de *test* en la medida de lo posible (un enfoque correcto sería por ejemplo priorizar *Unit* *testing* para aquellas mecánicas que consideremos más importantes o prioritarias).

**Configurar *Test Runner***

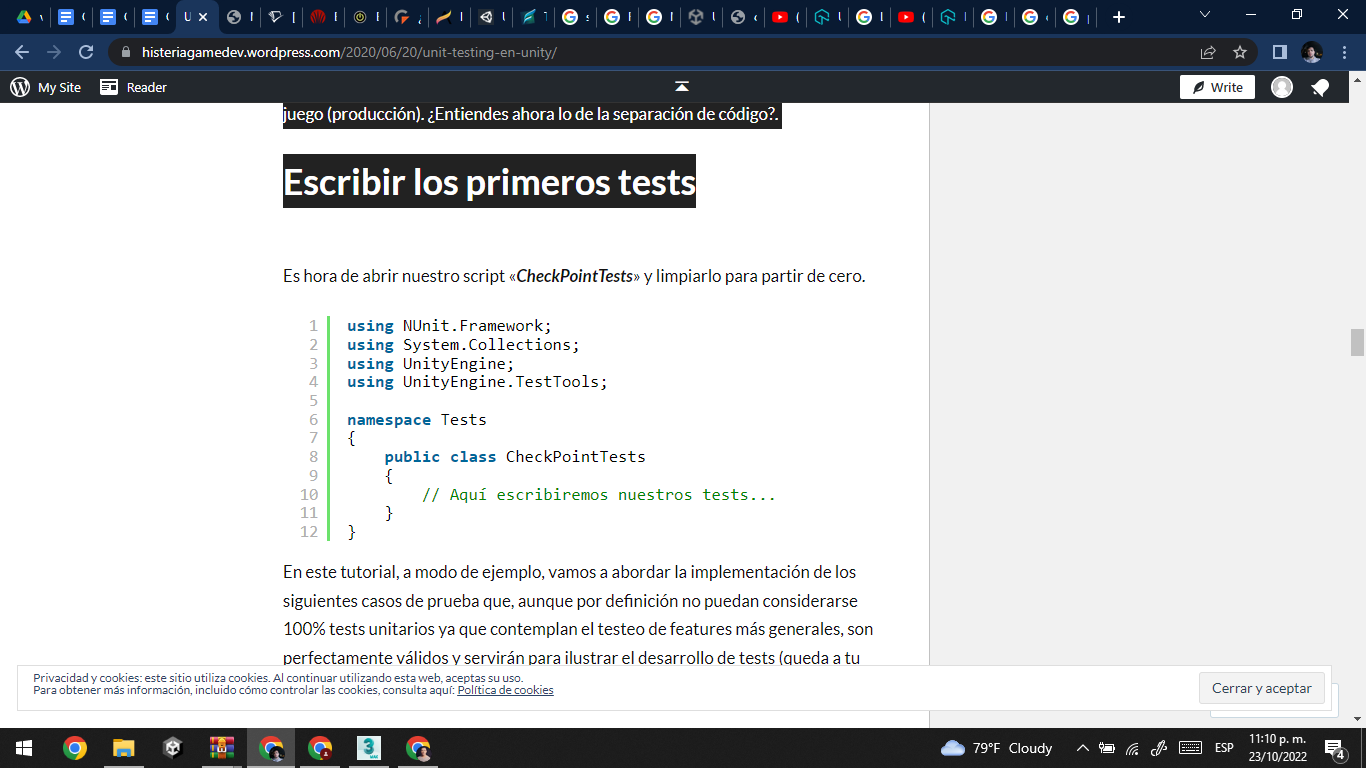
Antes de comenzar a desarrollar el código para los respectivos *tests*, se debe inicialmente configurar *Test Runner,* que es la herramienta que viene integrada con *Unity* para gestionar todo lo relacionado con *Unit testing* y la cual utiliza el *framework NUnit*. Veamos los siguientes pasos:

**Video tutorial *Framework Test Runner***

Con el fin de contextualizarlo de una mejor manera sobre este tema, le invitamos a que revise el siguiente video tutorial, se recomienda su descarga:

**Ejercicio práctico:** querido aprendiz,con elfin de iniciar la práctica de los temas que se abordan y las herramientas, le invitamos a desarrollar este ejercicio, atendiendo los siguientes puntos:

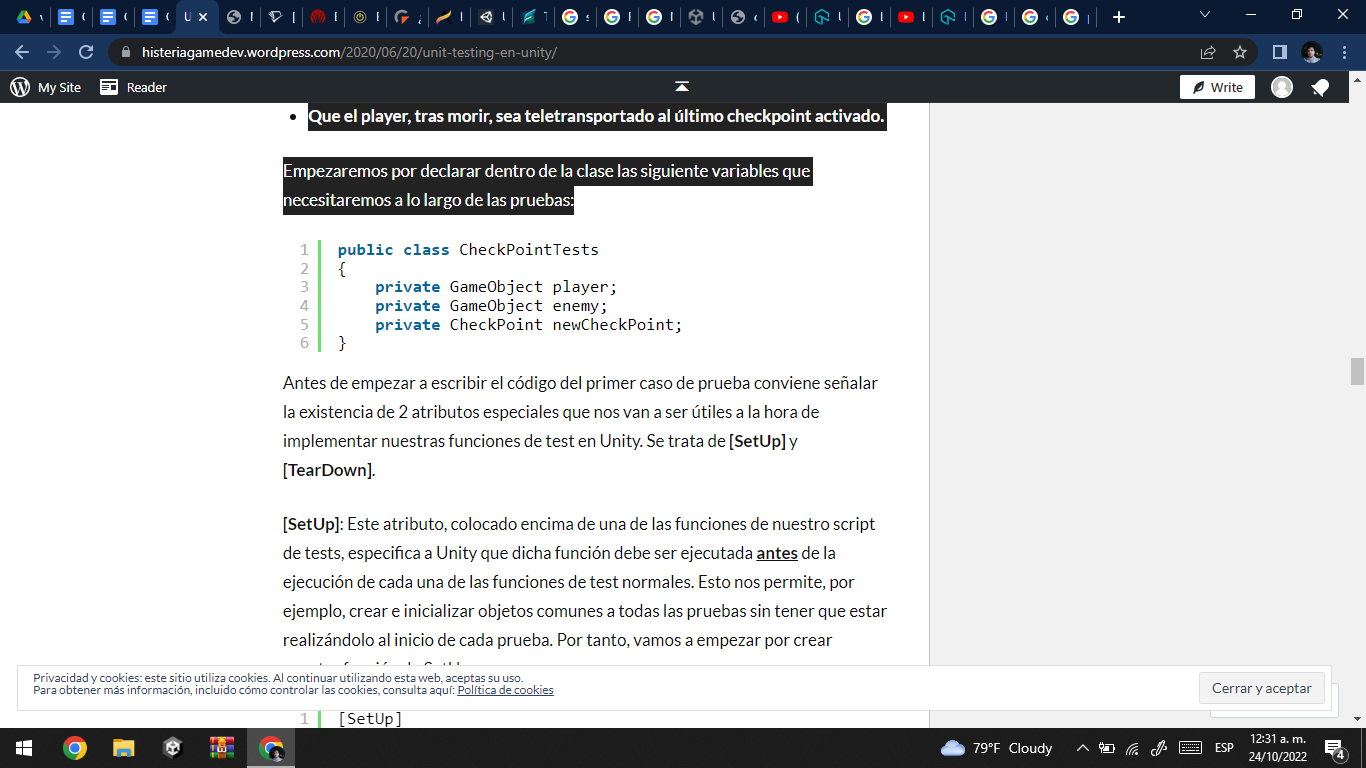
Es momento de abrir su propio script «*CheckPointTests*» y limpiarlo para partir de cero, así:



En este ejercicio, a modo de ejemplo, se abordará la implementación de los siguientes casos de prueba que, aunque por definición no puedan considerarse 100 % *tests* unitarios, ya que contemplan el testeo de *features* más generales, son perfectamente válidos y servirán para ilustrar el desarrollo de *tests* (queda a criterio propio el plantear todos los casos que estimes necesarios):

* Caso de prueba para que un *checkpoint* se cree correctamente.
* Caso de prueba para que un *checkpoint* sea activado por el *player* correctamente.
* Caso de prueba para que el *player*, tras morir, sea teletransportado al último *checkpoint* activado.

Debe iniciar entonces por declarar dentro de la clase, las siguientes variables que son necesarias a lo largo de las pruebas:

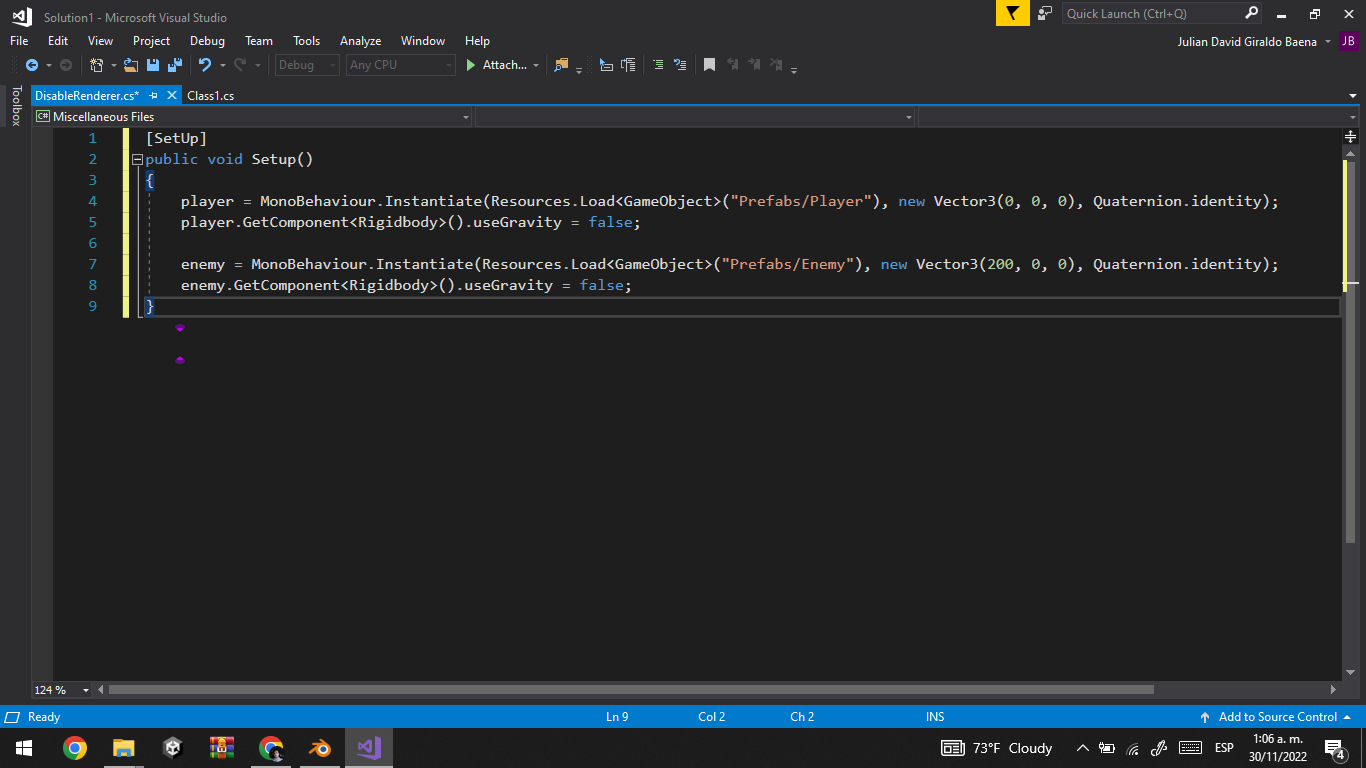


Antes de iniciar a escribir el código del primer caso de prueba conviene señalar la existencia de 2 atributos especiales que van a ser útiles a la hora de implementar sus funciones de test en *Unity*.

Se trata de **[*SetUp*]** y **[*TearDown*]**.

**[*SetUp*]: e**ste atributo, colocado encima de una de las funciones de su script de *tests*, específica a *Unity* que dicha función debe ser ejecutada antes de la ejecución de cada una de las funciones de test normales.

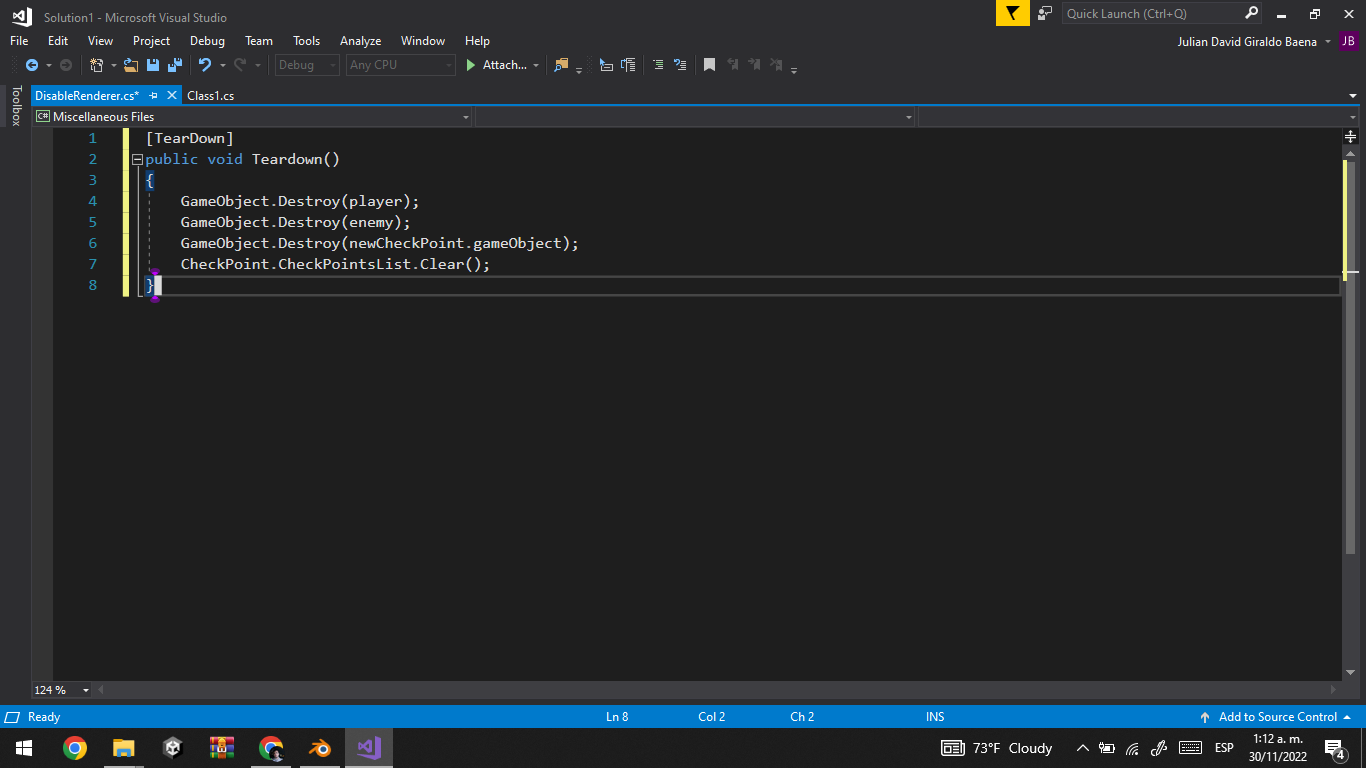
Esto permite, por ejemplo, crear e inicializar objetos comunes a todas las pruebas sin tener que estar realizándose al inicio de cada prueba. Por tanto, empiece por crear la función de *SetUp*:



En este caso se hace lo siguiente:

1. Instanciar el *prefab* del *player* en el punto (0, 0, 0) de la escena.
2. Desactivar la gravedad (esto se hace para facilitar las pruebas, ya que en esta escena de prueba no se cuenta con un suelo).
3. Instanciar el *prefab* del enemigo en un punto alejado al *player* (200, 0, 0).
4. Desactivar la gravedad (por el mismo motivo de antes).

**[*TearDown*]: e**ste atributo, colocado encima de una de las funciones del script de *tests*, específica a *Unity* que dicha función debe ser ejecutada después de la ejecución de cada una de las funciones de test normales. Esto permite, por ejemplo, destruir o limpiar objetos comunes a todas las pruebas sin tener que estar realizándose al final de cada prueba. Por tanto, debe crear la función de *TearDown*:

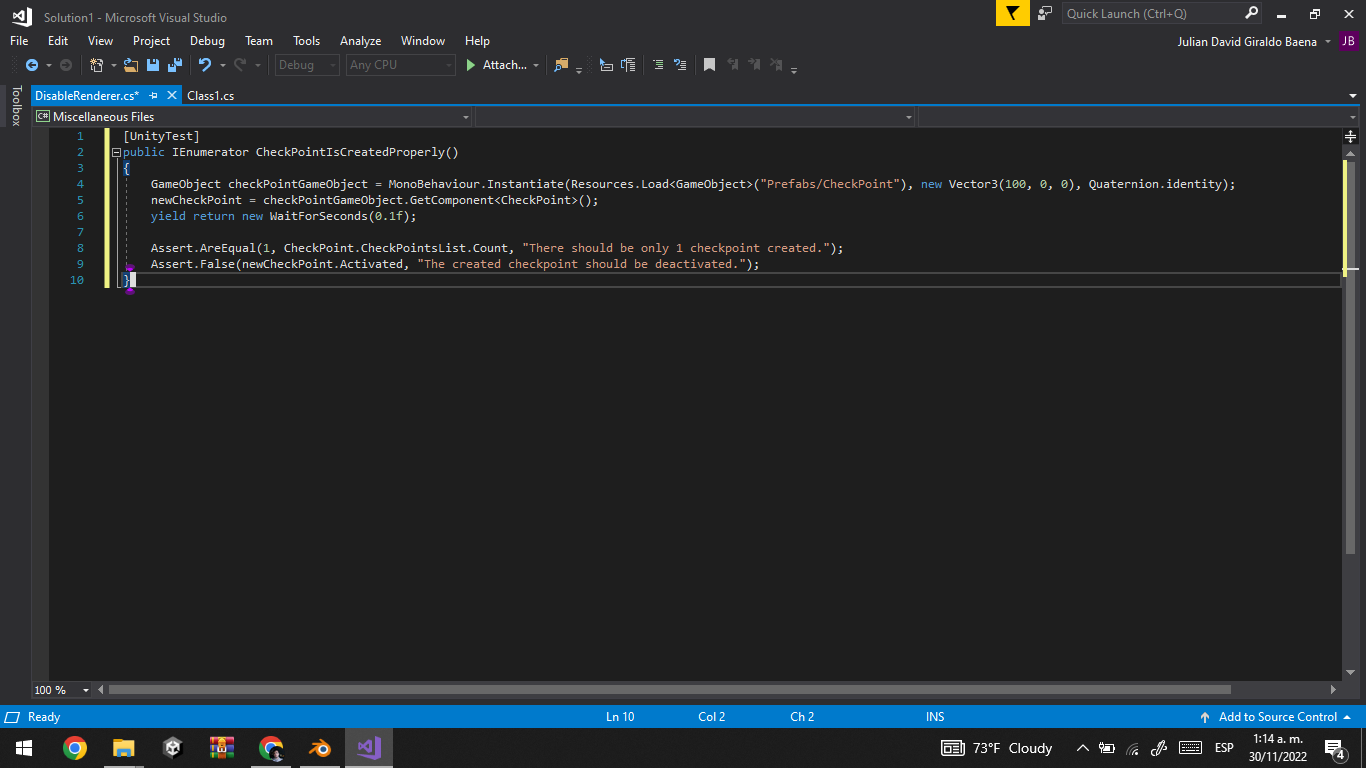


En este caso se hace lo siguiente:

1. Destruir el objeto instanciado del *player*.
2. Destruir el objeto instanciado del enemigo.
3. Destruir el objeto instanciado del *checkpoint*.
4. Limpiar la lista de *checkpoints* en escena.

Bien, ya puede empezar a escribir sus 3 casos de prueba.

Empecemos con el primero: la creación correcta de un *checkpoint* en escena.



En este caso se hace lo siguiente:

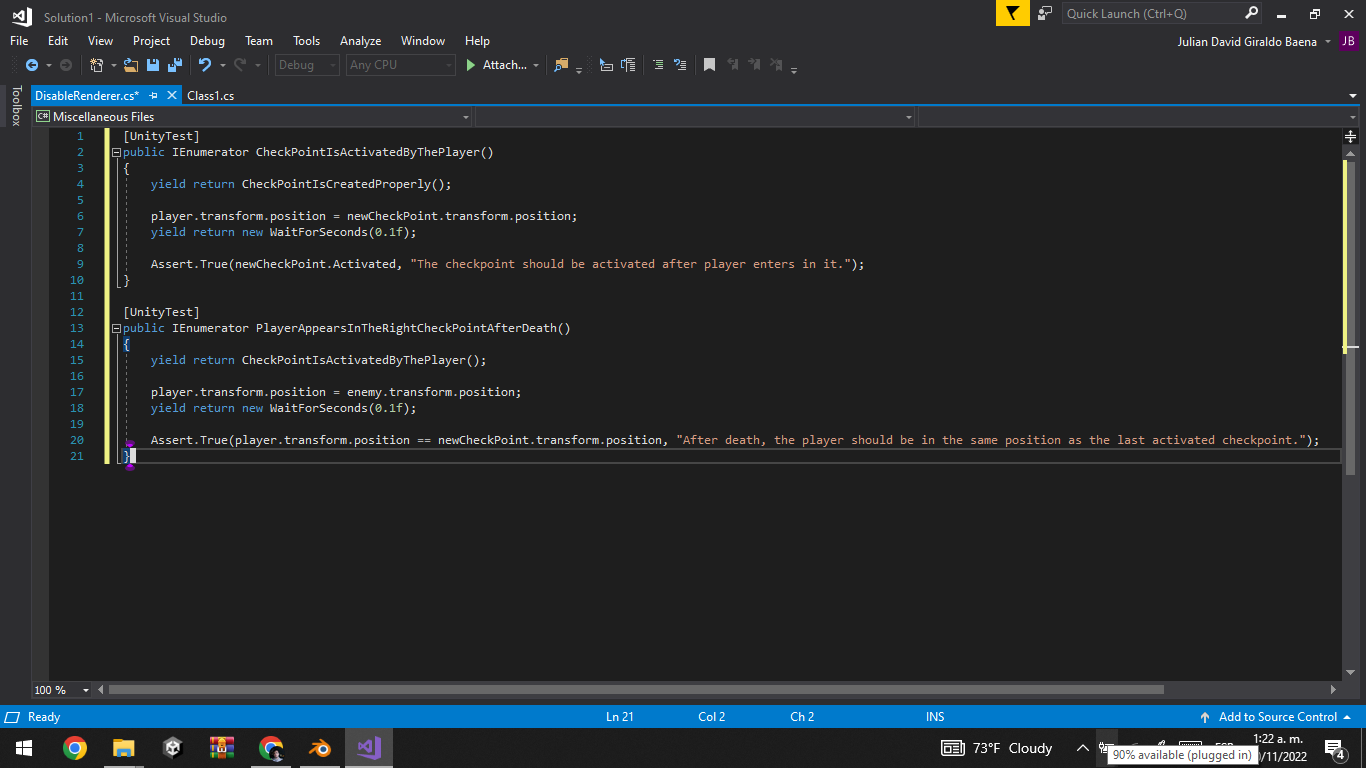
1. Instanciar el *prefab* del *checkpoint* en el punto (100, 0, 0) de la escena, para que no esté tocando al *player*.
2. Obtener su componente *CheckPoint*, que es donde está el código que se quiere probar del juego (producción).
3. Esperar 0.1 segundos (esto es bastante común hacerlo en los *tests* cuando se quiere asegurar de darle tiempo a *Unity* para que refleje ciertas operaciones en la escena, en este caso la instancia del *checkpoint*).
4. Una vez instanciado el *checkpoint*, realizar los siguientes chequeos:
   * El número de objetos en la lista de *checkpoints* en escena debe ser 1.
   * El nuevo *checkpoint* debe estar desactivado.

Es importante validar el ejemplo que acaba de realizar, por lo que le invitamos a comprobar si su ejercicio quedo correcto haciendo lo que se indica en el video que se expone a continuación:

|  |
| --- |
| **Infografía puntos calientes**  CF011\_3\_Analisis\_Codigo |

A partir de ahora ya maneja lo básico y necesario para realizar cualquier tipo de *test*. Así que acabemos de ver los siguientes casos de prueba:

«La activación correcta de un *checkpoint*» y «El posicionamiento del *player* al último *checkpoint* después de morir».



**Nota:** como vemos, dentro de una función de *test*, podemos perfectamente hacer llamadas a otras funciones de *test* siempre y cuando tenga sentido para nuestras pruebas.

**Ejecutar *tests***

En este punto, lo único que nos queda es realizar la ejecución de las pruebas. Para ello, tenga en cuenta lo que se indica en el siguiente video:

**Video tutorial Script Test Runner**

Se debe validar siempre el funcionamiento de los *scripts* que se desarrollen, por lo que le invitamos a que revise el siguiente video tutorial, se recomienda su descarga:

¡Disfruta del *testing*!

**4. Pruebas del sistema**

Se dividieron las pruebas de sistema en tres bloques diferenciados atendiendo a los requisitos no funcionalidades establecidos previamente en este documento, como son:

**Pruebas de portabilidad**

En cuanto a la portabilidad se había pedido que el juego pudiera ser ejecutado de manera multiplataforma. Se seleccionaron entre las plataformas disponibles las más usadas en el mundo de la informática hoy en día. Estas plataformas son los ordenadores y los dispositivos móviles como teléfonos o tabletas. De entre esta gran variedad de plataformas se decidió que, en cuanto al ordenador, el juego pudiera ser ejecutado en entornos Windows y que en móviles pudiera ser ejecutado en entornos Android, a pesar de que hay otras alternativas como MacOS para sobremesa o IOS en móviles.

Para garantizar el cumplimiento de estos requisitos no funcionales se estableció que la prueba consistiría en que se pudiera ejecutar correctamente el juego en la plataforma deseada. Después de poder ejecutar el juego en las distintas plataformas sin ningún problema se concluyó que la prueba estaba superada.

**Pruebas de compatibilidad**

En cuanto a la compatibilidad lo que se buscaba era que, al tratarse de un juego multijugador, éste permitiera que se jugará desde distintos dispositivos la misma partida. A esto se le denomina, en el *argot* de los videojuegos, juego cruzado. Para realizar estas pruebas se procedió a crear una partida con dos jugadores.

Uno accedió al juego vía ordenador y el segundo desde su dispositivo móvil Android. Se jugó la partida completa y se concluyó entonces que la partida podía ser jugada sin ningún problema, puesto que no se produjo ningún error. Para el juego es indistinto que una persona se conecte desde un dispositivo u otro.

**Pruebas de rendimiento**

Para realizar y analizar las pruebas de rendimiento se hizo uso de la herramienta *Profiler* de *Unity*. El *Profiler* ayuda a optimizar el juego. Se trata de un reporte en ejecución que calcula el tiempo empleado en la ejecución distintas áreas del juego o los recursos consumidos, esto lo puede revisar de manera más detallada en el video que se expone a continuación:

**Video tutorial Pruebas de rendimiento**

Con el fin de contextualizarlo de una mejor manera sobre este tema, le invitamos a que revise el siguiente video tutorial, se recomienda su descarga:

**Pruebas de aceptación**

Una vez finalizado el juego se deben realizar, también, las pruebas de aceptación. Para realizar dichas pruebas se debe acudir mínimo a dos fuentes, con el objetivo de que prueben el juego y den sus impresiones sobre la apariencia física del mismo. Esto proporcionará nuevos reportes y añadirá una nueva dimensión a las pruebas de aceptación. Una nueva dimensión que será tenida en cuenta para trabajos próximos.

1. **SÍNTESIS**

En el siguiente esquema se resumen los pasos y temas claves para aplicar técnicas de testeo y pruebas para los videojuegos:



1. **ACTIVIDADES DIDÁCTICAS**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA 1 | |
| Nombre de la Actividad | Los videojuegos |
| Objetivo de la actividad | Identificar los tipos y técnicas de testeo para la eficacia de un videojuego, de acuerdo a funcionalidades. |
| Tipo de actividad sugerida |  |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | Ver archivo de la actividad en carpeta de Anexos: CF011\_228108\_Actividad\_didactica\_1\_DI |

1. **MATERIAL COMPLEMENTARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| 2. Cómo iniciar las pruebas o *testing* en un videojuego | Unity Manual. (s.f.). *Unity Test Framework overview.* | [otro] | <https://docs.unity3d.com/Packages/com.unity.test-framework@1.3/manual/index.html> |
| 4. Pruebas del sistema | Kodeco. (2019). *Introduction To Unity Unit Testing.* | [otro] | <https://www.kodeco.com/9454-introduction-to-unity-unit-testing> |
| 4. Pruebas del sistema | Unity Learn. (s,f.). *Unit 10 - Testing.* | [otro] | <https://learn.unity.com/tutorial/unit-10-testing> |

1. **GLOSARIO**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| Ad Hoc: | que es apropiado, adecuado o especialmente dispuesto para un determinado fin. |
| CPU: | una unidad central de procesamiento o CPU, es una pieza de hardware que permite que tu computadora interactúe con todas las aplicaciones y programas instalados. |
| *Framework*: | es un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de *software*. |
| QA: | un QA (*Quality Assurance*) o analista QA es el profesional responsable de asegurar la calidad del *software* y de prevenir fallos en él. |
| *SetUp*: | es una herramienta de los sistemas operativos y los programas informáticos que permite configurar diversas opciones de acuerdo a las necesidades del usuario. |
| Teastear / *testing:* | testear es obtener la información necesaria para mejorar el sistema que se está probando, pero también para mejorar los propios procesos de desarrollo y de pruebas. Hay diferentes objetivos en el *testing*: Buscar los defectos. Ganar confianza respecto al nivel de calidad. |
| *Test Runner:* | un *test* *runner* es una utilidad que nos permite escribir y correr *tests* para una aplicación. Existen gran variedad de *tests runner* como por ejemplo mocha. |
| UX: | el diseño UX se refiere al término «diseño de experiencia de usuario», mientras que UI significa «diseño de interfaz de usuario». |

1. **REFERENCIAS BIBLIOGRÁFICAS:**

Awad, W. (2021). *Game Testing Automation Guidance*. <https://www.theseus.fi/bitstream/handle/10024/505977/Game%20Testing%20Automation%20Guidance.pdf?sequence=2&isAllowed=y>

Da Silva Lima, G. et al. (2021). *Devops methodology in game development with Unity 3D*. <https://www.ihci-conf.org/wp-content/uploads/2021/07/04_202105C029_Lima.pdf>

Koepke, B, Pelletier, B., Adair, D., Jhawar, R., Macaulay, I. & Bielecki, T. (2013). *Agile Game Development.* <http://kremer.cpsc.ucalgary.ca/courses/seng403/W2013/papers/05GameDevelopment.pdf>

Paduraru, C., Paduraru, C. & Stefanescu, A. (2021). *Automated game testing using computer vision methods*. [https://ieeexplore.ieee.org/abstract/document/9680292/authors#authors](https://ieeexplore.ieee.org/abstract/document/9680292/authors" \l "authors)

Scandinaro, D. (2021). *On-Demand QA Testing with Unity Automated QA.* <https://blog.unity.com/games/on-demand-qa-testing-with-unity-automated-qa>

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Autor (es) | Nombre | Cargo | Dependencia | Fecha |
| Julián David Giraldo Baena | Experto  Temático | Regional Risaralda - Centro de Diseño e Innovación Tecnológica Industrial | Noviembre de 2022 |
| Luz Aída Quintero Velásquez | Diseñadora Instruccional | Regional Distrito Capital, Centro de Gestión Industrial | Diciembre de 2022 |
|  | Gloria Amparo López Escudero | Diseñadora Instruccional | Regional Norte de Santander - Centro de la Industria, la Empresa y Los Servicios | Diciembre de 2022 |
| Andrés Felipe Velandia Espitia | Asesor Metodológico | Regional Distrito Capital – Centro de Diseño y Metrología | Diciembre de 2022 |
| Rafael Neftalí Lizcano Reyes | Responsable Equipo Desarrollo Curricular | Regional Santander - Centro industrial del diseño y la manufactura | Diciembre de 2022 |

1. **CONTROL DE CAMBIOS**

**(Diligenciar únicamente si realiza ajustes a la Unidad Temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |