**Facultad 4, Facultad CITEC**

**Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas**

# *"Sistema para realizar de forma automática el proceso de pruebas a la jugabilidad en los videojuegos del género plataformas 2D"*


**Autores**

Viviani Tamayo Sierra

Reynier López Argüello

**Tutor**

MSc. Yanetsi Millet Lombida


La Habana, noviembre del 2022

Año 64 de la Revolución

*"Grandes descubrimientos y mejoras implican invariablemente la cooperación de muchas mentes."*

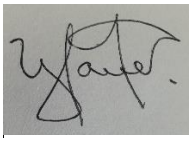
*Alexander Graham Bell*

# DECLARACIÓN DE AUTORÍA

Declaramos ser los autores del trabajo de diploma “Sistema para realizar de forma automática el proceso de pruebas a la jugabilidad en los videojuegos del género plataformas 2D” y reconocemos a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma con carácter exclusivo.

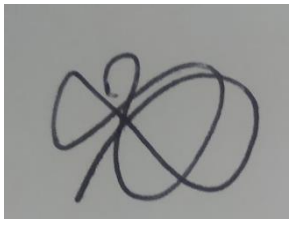
Para que así conste se firma la presente a los 30 días del mes de noviembre de 2022.

Yanetsi Millet Lombida



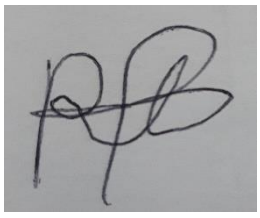
Firma del Tutor

Viviani Tamayo Sierra



Firma del autor

Reynier López Argüello



Firma del autor

# DATOS DE CONTACTO

Nombre y apellidos: Prof.Yanetsi Millet Lombida

Correo electrónico: ymillet@uci.cu

Años de graduado: 15 años

Especialidad de graduación: Ingeniero en Ciencias Informáticas

Institución a la que pertenece: Universidad de Ciencias Informáticas

Dirección: Carretera San Antonio de los Baños, Km 2 ½ Torrens, Boyeros, La Habana, Cuba, Código Postal 19370

# DEDICATORIA

**De Viviani Tamayo Sierra**

A mis padres Liuba y Nivaldo que han sido mi apoyo a lo largo de toda mi vida y en especial durante el transcurso de la carrera.

A mi abuela Teresa y mis tías Vivian y Leslie.

A toda mi familia por el impulso y el amor que me dan.

**De Reynier López Argüello**

Este trabajo y todos sus resultados se los dedico a mis padres, mi hermano, las personas más importantes en mi vida. Gracias por el amor, la entrega, la paciencia y la confianza que siempre me han dedicado. Gracias por creer en mí. Si hoy estoy aquí es por ustedes. Los amo.

# AGRADECIMIENTOS

Quisiéramos comenzar agradeciendo a todos esos profesores que nos ayudaron en la persecución de nuestro sueño a lo largo de la carrera.
A la tutora, la profesora Yanetsi por apoyarnos en esta etapa.

**De Viviani Tamayo Sierra**

A mis padres Liuba y Nivaldo por ser tan amorosos conmigo y apoyarme siempre, pero también por exigirme, gracias a ellos me convierto en ingeniera.

A mi abuela Teresa por ser un gran ejemplo de mujer para mí y ser tan cariñosa.

A mis tías queridas Vivian y Leslie por estar siempre para lo que he necesitado y ser unas mujeres de tan buen corazón.

A mi novio Franck por el amor que me ha dado y estar conmigo en todo momento apoyándome, ayudándome y dándome ánimo.

A mis amistades de la UCI: Yamisleidis, Adachelys, Leandro, Darles, Roxana, Solanch, Mardelis, Laura, Daimarilis y Rosalena que han estado conmigo en las mejores, pero en especial en las peores. No olvidaré todo lo que han hecho por mí.

A mi compañero de tesis Reynier por la ayuda y el esfuerzo brindado para realizar este trabajo de diploma.

A mi tutora Yanetsi por la atención prestada durante el desarrollo de la investigación.

Son muchos por mencionar, gracias a todos los que me ayudaron durante mis estudios.

# AGRADECIMIENTOS

**De Reynier López Argüello**

*Me gustaría agradecer a todas aquellas personas que de una forma u otra contribuyeron a la realización de este sueño.*

*A mis padres, que sin ellos no hubiese logrado nada, gracias por su apoyo, sacrificio, gracias por el coraje que me brindaron, por todo.*

*A mi rubia preferida, Susana Cremé, por siempre darme su apoyo y amistad, a Aroldo Leodán, mi compañero de cuarto por soportarme todos estos años, Marlon Vázquez (el mago), al pina Álvaro, Hassam, al calvo Brian por sus lecciones; Raikol quien es un oyente de primera, gracias por ser paciente conmigo.*

*A mis amistades vecinas: Javier, Ernesto, Rixon, quienes siempre me han ayudado en todo sin dudarlo.*

*Y de forma general a todas aquellas personas que creyeron en mí y siempre estuvieron a mi lado, ante todos ustedes me quito el sombrero.*

**RESUMEN**

El Centro de Tecnologías Interactivas, situado en la Facultad 4 de la Universidad de las Ciencias Informáticas, es pionero en el desarrollo de videojuegos de factura nacional y planifica el desarrollo de videojuegos del género plataformas. Sin embargo, el proceso de pruebas a los videojuegos se realiza manualmente y no se diseñan las pruebas tomando la jugabilidad como referente de calidad. La esencia de esta investigación es desarrollar un sistema que pruebe de forma automática la jugabilidad en los videojuegos del género plataformas.

El estudio del estado del arte determinó que no existe un sistema que de forma automática pruebe los videojuegos, pero sí de sistemas basados en Inteligencia Artificial que juegan videojuegos. El sistema que se implementó incluye entre sus principales funcionalidades, entrenar modelos visuales, crear redes neuronales, comprobar mecánicas de desplazamiento del personaje principal y obtener un reporte del cumplimiento de las mecánicas comprobadas.

La metodología por la que estuvo guiada el proceso de desarrollo de *software* fue el Proceso Unificado Ágil en la versión de la Universidad de las Ciencias Informáticas y se generó la documentación correspondiente a las etapas de Inicio, Ejecución y Cierre. Para la validación del sistema se aplicaron las pruebas unitarias y de aceptación.

**Palabras clave:** automática, jugabilidad, plataformas, pruebas, videojuegos.

**ABSTRACT**

The Center for Interactive Technologies, located in Faculty 4 of the University of Computer Sciences, is a pioneer in the development of national video games and plans the development of video games of the platform genre. However, the process of testing video games is done manually and the tests are not designed taking the playability as a quality reference. The essence of this research is to develop a system that automatically tests the playability of platform video games.

The study of the state of the art determined that there is no system that automatically tests video games, but there are systems based on Artificial Intelligence that play video games. The system that was implemented includes among its main functionalities, training visual models, testing displacement mechanics of the main character and obtaining a report of the compliance of the tested mechanics.

The methodology that guided the software development process was the Agile Unified Process in the version of the University of Informatics Sciences and the documentation corresponding to the stages of Startup, Execution and Closing was generated. For the validation of the system, unit and acceptance tests were applied.

**KEYWORDS**: automatic, gameplay, platforming, testing, video games.

## Índice:

**Índice de Figuras:**

**Índice de Tablas:**

# Introducción

Debido a la creciente evolución e informatización de todas las áreas de la sociedad, resulta cada vez más complejo garantizar la calidad de los sistemas de *software* que se desarrollan, por su complejidad y los requerimientos de los usuarios a quiénes van dirigidos. La NC ISO/IEC 25010:2016 sustenta la evaluación de la conformidad y la certificación de los productos y aplicaciones informáticas; las propiedades basadas en las características de la norma cubana se traducen en requisitos para Adecuación Funcional, Seguridad, Fiabilidad, Compatibilidad, Mantenibilidad, Portabilidad, Eficiencia de desempeño y Usabilidad (Institucional, 2020). Esta última característica a través de sus subcaracterísticas (inteligibilidad, aprendizaje, operabilidad, protección frente a errores de usuario, estética, accesibilidad) refleja la capacidad del producto de *software* para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones (ISO 25000, 2022).

Un *software* es fácil de usar si realiza la tarea para la que se está usando de forma cómoda, eficiente e intuitiva para el usuario. Ciertamente los factores que determinan si un *software* es usable o no, son sobre todo la facilidad de uso y facilidad de aprendizaje, pero es insuficiente ante un *software* basado en videojuego, debido a que no basta con estos factores para determinar si un videojuego es usable o no (Maestre, 2015).

Los videojuegos causan un gran impacto en la sociedad, formando parte de la vida cotidiana de la mayoría de las personas debido a su contenido llamativo y envolvente; pueden ayudar a desarrollar habilidades psicológicas, cognitivas, físicas y/o sociales y cualidades por lo que es aconsejable desarrollar productos para satisfacer a la mayor cantidad de usuarios posible (Alejo García-Naveira Vaamonde, 2018).

Al desarrollar un videojuego la misión principal es que las sensaciones que experimente el usuario al jugarlo sean las mejores, ya que el usuario es el verdadero protagonista. Para medir la experiencia del jugador ante un sistema de videojuego determinado se usa la propiedad llamada jugabilidad. La jugabilidad no sólo tiene en cuenta factores como la calidad técnica de los gráficos o el sonido; además se debe

prestar más atención a las mecánicas del juego y la experiencia del jugador (Regueiferos, 2012).

Continuamente se desarrollan numerosos estudios en torno a los videojuegos, ya sea para analizar sus ventajas y desventajas o para mejorar la calidad en el desarrollo de los mismos, por lo que una vez finalizada la etapa de implementación se lleva a cabo el proceso de pruebas. En esta etapa se corrigen los errores del proceso de programación y se mejora la jugabilidad a medida que se prueba el juego (Rodríguez L. P., 2021). Las pruebas en el desarrollo de *software* basados en videojuegos son fundamentales para garantizar la calidad del producto y la satisfacción del usuario, por tanto, es de vital importancia que las mismas puedan realizarse de forma automática y se centren en la experiencia del usuario, contemplando la jugabilidad como medida de calidad.

La tendencia a nivel mundial es desarrollar dos tipos de pruebas a la hora de medir los elementos que componen las mecánicas de los juegos, las pruebas exploratorias y las pruebas de *scripting*, las primeras se basan en la experiencia del probador para ir midiendo las diferentes mecánicas, además este debe observar detalladamente para detectar cualquier falla a la hora de las pruebas (retrasos de las animaciones, errores de físicas, entre otros) que influyen negativamente en la experiencia de usuario total. Las segundas se realizan por grupos de programadores de alta experiencia para desarrollar *Scripts* (fichero de código), los cuales miden el producto a nivel de código, la alta complejidad de estas pruebas radica en la extensa duración a la hora de diseñarlas (en el orden de meses) (Politowski y otros, 2022).

El Centro de Tecnologías Interactivas, ubicado en la Facultad 4 de la Universidad de las Ciencias Informáticas (UCI) incursiona en el desarrollo de videojuegos para el esparcimiento, entre los que sobresalen: La Súper Claria, Aventuras en la Manigua y Especies Invasoras. El centro continúa el desarrollo de nuevos productos específicamente del género de plataformas, donde el jugador debe avanzar a través de un mapa que contiene diferentes plataformas u objetos, colocados a diferente altura para llegar al objetivo; pero se ha detectado a través de fuentes de información (equipo de desarrollo) en el centro que:

Actualmente el proceso de pruebas a los videojuegos se realiza manualmente mediante usuarios especializados que tienen como función probar el producto en su versión beta; estos usuarios miden los diferentes elementos que influyen en la calidad del videojuego. Para el diseño de las pruebas los probadores se centran en las características de calidad de *software* que propone la NC ISO/IEC 25010: 2016; pero estas características son insuficientes para garantizar que la calidad del videojuego y la experiencia de un usuario ante el mismo sean satisfactorias, debido a que los videojuegos no constituyen sistemas interactivos tradicionales. La propiedad "jugabilidad", a través de sus elementos (calidad visual, fluidez de las animaciones, experiencia de juego, correcto funcionamiento de las mecánicas, entre otros) está dirigida a medir la calidad de experiencia de juego, pero en el centro durante el desarrollo del proceso de pruebas a un videojuego no se diseñan y realizan las pruebas basándose en esta propiedad.

Teniendo en cuenta la situación problemática expuesta anteriormente surge el siguiente **problema de investigación:** ¿Cómo realizar de forma automática el proceso de pruebas a la jugabilidad en los videojuegos del género plataformas 2D?

Por lo que se define como **objeto de estudio** de la investigación: procesos de pruebas en el desarrollo de videojuegos y la investigación se enmarca en el **campo de acción**: procesos de pruebas en el desarrollo de videojuegos del género plataformas.

Como **objetivo general** se persigue: desarrollar un sistema que realice de forma automática el proceso de pruebas a la jugabilidad en los videojuegos del género plataformas 2D.

A partir del objetivo general definido se proponen los siguientes **objetivos específicos:**

1- Elaborar el marco teórico-conceptual a partir del estudio de los referentes bibliográficos, tendencias y herramientas que se relacionen con el proceso de pruebas para videojuegos.

2- Diagnosticar el estado actual del proceso de pruebas para videojuegos en el Centro de Tecnologías Interactivas y analizar las propuestas de jugabilidad existentes en el entorno científico.
3- Definir una propuesta de jugabilidad con las propiedades y atributos que posibiliten analizarla en el género de videojuegos de plataformas y adaptarla aportando métricas que permitan medirla o cuantificarla.
4- Desarrollar un sistema que permita analizar y comprobar la jugabilidad a partir de las métricas propuestas.
5- Validar el sistema desarrollado.

Se combinan diferentes métodos científicos de investigación para la búsqueda y consulta de la información, estos son:

Teóricos:

Histórico-Lógico: permite analizar las tendencias actuales y herramientas existentes para la automatización de pruebas a videojuegos, además de términos y vocabulario, haciendo énfasis en el término jugabilidad.

Análisis y síntesis: permite realizar un estudio a nivel mundial de las pruebas a los videojuegos y las experiencias adquiridas en el Centro de Tecnologías Interactivas durante el despliegue de los videojuegos realizados en dicho centro.

Empíricos:

Observación: facilita la determinación de las deficiencias existentes y tomar las mejores prácticas de sistemas similares. Al aplicar el método se pudo determinar la inexistencia de sistemas para la automatización de pruebas a videojuegos, aunque existen mecanismos para la realización de Bots (robots), que son capaces de jugar determinados juegos para los que fueron diseñados.

Entrevista: se realizan entrevistas a integrantes del equipo de desarrollo del Centro de Tecnologías Interactivas para profundizar el proceso de pruebas a videojuegos que han desarrollado.

La investigación está estructurada por tres capítulos que se describen a continuación de forma breve:

•**Capítulo I. Fundamentación teórica**: se abordan los conceptos fundamentales relacionados con el proceso de pruebas a los videojuegos, herramientas similares, y los atributos de la jugabilidad a tener en cuenta para las pruebas, tanto a nivel mundial como en el ámbito en que se enmarca la investigación. Además, se describe las técnicas y algoritmos que se aplican para la creación de la propuesta de solución.

•**Capítulo II. Descripción de la propuesta de solución**: se realiza el análisis y diseño del sistema y se describe la estructura interna del sistema a partir de los artefactos generados por la metodología de desarrollo de *software* seleccionada. Además, se describe el proceso de entrenamiento del algoritmo de aprendizaje automático Fast R-CNN.

•**Capítulo III. Implementación y pruebas**: se muestran las pautas de programación seguidas y se evalúa la propuesta de solución mediante la realización de diferentes tipos de pruebas de *software*.

# Capítulo I: Fundamentación teórica

## 1.1. Conceptos y elementos relevantes para la investigación

En este capítulo se introducen los conceptos de automatización, calidad de *software*, videojuegos y jugabilidad tomando como referencia a diferentes autores. Se realiza un recorrido breve por los diferentes géneros de videojuegos, haciendo énfasis en los del género plataformas y se analiza la arquitectura básica de los videojuegos, así como el proceso de desarrollo de estos. Se definen los tipos de jugabilidad y la importancia de la misma a la hora de realizar pruebas a los videojuegos.

### 1.1.1 Calidad de software

La evolución de forma constante de los productos y herramientas de desarrollo de *software* ha provocado que se intensifique cada vez más la competencia a nivel mundial entre las empresas productoras de *software*. Los usuarios han adquirido un grado mayor de conocimiento y familiarización con las nuevas tecnologías al incluirlas en la ejecución de disímiles actividades de la vida diaria, en consecuencia, exigen productos con una calidad más elevada para satisfacer necesidades de mayor envergadura.

La calidad de *software* es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario, así lo define la IEEE, Std. 610-1990 (Aizprua y otros, 2019).

El concepto de calidad de *software*, según Pressman se asocia a la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo plenamente documentados y con las características implícitas que se espera de todo *software* desarrollado profesionalmente (Callejas-Cuervo y otros, 2016).

La Organización Internacional de Estándares (ISO) en su estándar ISO/ IEC 9126 define la calidad de *software* como: “la totalidad de características y propiedades de un producto de *software* que deben relacionarse con su idoneidad para cumplir con requisitos específicos y previamente definidos” (Stuber, 2021).

Para esta investigación se define como calidad de *software* la totalidad de características y propiedades de un producto de *software* que cumple con las necesidades del cliente.

### 1.1.2 Conceptualización de videojuegos

Los videojuegos se han convertido en el tipo de juego preferido del siglo XXI, estando presentes en multitud de ambientes sociales y culturales. Son el medio de entretenimiento favorito de personas de diferentes grupos etarios y han evolucionado de manera increíble en muy poco tiempo, siendo una de las industrias que más factura. El fin expreso de los videojuegos es entretener y divertir a los usuarios, pero se ha logrado también que además de diversión los usuarios tengan la posibilidad de aprender y desarrollar habilidades.

La Real Academia Española (RAE) define a un videojuego como un dispositivo electrónico que permite, mediante mandos apropiados, simular juegos en las pantallas de un televisor, una computadora u otro dispositivo electrónico (Real Academia Española, 2021).

Los videojuegos son juegos electrónicos en los que están involucradas una o más personas interactuando entre sí, es decir, es todo tipo de juego digital interactivo que su base principal es entretener y divertir por un lapso de tiempo prolongado, utilizando soportes de interfaces como los ordenadores, las videoconsolas, las consolas portátiles o máquinas recreativas (Rodriguez, 2022).

Un videojuego es una actividad interactiva que se juega por medio de un dispositivo audiovisual. Al ser juegos, los videojuegos están compuestos por diversos elementos y reglas específicos, por una narrativa, e implican la participación activa de uno o más jugadores” (Solano, 2022).

En esta investigación se define videojuego como un juego digital interactivo que simula juegos en un dispositivo electrónico con el fin de divertir y está compuesto por reglas específicas.

### 1.1.3 Géneros de los videojuegos

Desde la aparición de los videojuegos hasta la actualidad los géneros en los que se clasifican se han diversificado, estos géneros se conforman atendiendo a factores como: ambientación, tipo de interacción entre el jugador y el dispositivo y representación gráfica y su sistema de juego, siendo este último el criterio más habitual para su clasificación (Sevilla, 2022). Clasificar un videojuego tomando como criterio su género es difícil en estos días, ya que con el fin de complacer a los usuarios se desarrollan videojuegos en los que se fusionan diferentes géneros.

Se pueden clasificar los videojuegos en los siguientes grandes grupos (Rodriguez, 2022):

Acción: son juegos dónde la acción y el movimiento, combates y batallas son los factores predominantes. Requieren mucha rapidez en la interacción del usuario con el juego, un ejemplo de este tipo de videojuego puede ser God of War.

Aventuras: se pueden asemejar a una película interactiva donde el jugador se sumerge en el mundo del juego asumiendo un rol protagónico. Este tipo de videojuegos describen experiencias largas, ejemplo de ello es Hotel Dusk.

RPG: el jugador desempeña un rol determinado o personalidad concreta, debe mejorar sus habilidades al interactuar con el entorno mientras sigue el argumento de la historia, ejemplo de este tipo de videojuego es Final Fantasy.

Estrategia: el jugador debe planificar, crear esquemas y dirigir operaciones a largo plazo para alcanzar los objetivos, un ejemplo lo encontramos en el videojuego Civilization.

Simulación: se basan en la reproducción de forma realista del funcionamiento de actividades, ejemplo de ello son Los Sims (simulador social).

Educativos y de entrenamiento mental: tienen como objetivo transmitir conocimientos al jugador o mejorar habilidades mentales. Aprende con Pipo es un ejemplo de este tipo de videojuegos.

Disparo o Shoot´em up: conocidos también con los nombres de Shooters o FPS, en este tipo de videojuegos el jugador debe abrirse camino a base de disparos. El juego Doom es un ejemplo.

Deportes: simulan deportes como el fútbol, baloncesto, entre otros; el FIFA es uno de los más conocidos.

Puzzles: consisten en resolver diferentes tipos de puzzles, ya sean espaciales o lógicos, prueban la destreza y habilidad del jugador. Uno de los más famosos de este tipo de videojuegos es el Tetris.

Musicales: se centran en la música, tanto karaoke, bailar o tocar algún instrumento. Un perfecto ejemplo es Guitar Hero.

Carreras: el reto es que el jugador muestre habilidades al volante, se desarrollan carreras en vehículos (motos, carros), se destacan juegos como Need For Speed.

MMOG: juegos masivos de multijugador online, los jugadores pueden interactuar con otros de forma cooperativa o competitiva, los jugadores se agrupan en clanes para vencer a otros. World of Warcraft constituye un ejemplo de este género de videojuego.

Luchas: el jugador debe ir ganando combates para conseguir un premio o campeonato. Cada jugador dentro del juego realiza movimientos específicos y resulta compleja la combinación de teclas para lograrlo; un ejemplo es Mortal Kombat.

Mixtos: poseen elementos de diferentes géneros, pueden mezclar la aventura con la acción, plataformas y RPG, ejemplos son The Legend Of Zelda, GTA y Castelvania.

Plataformas: el protagonista debe avanzar a través de un mapa que contiene diferentes plataformas u objetos, colocados a diferente altura para llegar al objetivo, un ejemplo clásico es Super Mario Bros.

Esta investigación atendiendo a las necesidades del Centro de Tecnologías Interactivas está centrada en los videojuegos del género plataformas.

### 1.1.4 Videojuegos del género plataformas

Los videojuegos del género de plataformas son juegos que se caracterizan por tener que caminar, correr, saltar, o escalar sobre una serie de acantilados, con enemigos, mientras que se recogen objetos para poder completar el juego (Cruz, 2017). Se originaron a finales de los setenta y principios de los ochenta con un estilo en 2D. Es uno de los géneros fundamentales en la historia de los videojuegos que alcanza su máxima popularidad desde mediados de los 80 hasta mediados de los 90. En esta época uno de cada tres videojuegos era de plataformas (Miguel, 2019).

El arcade Space Panic de Universal en 1980, se considera el primer juego de plataformas, aunque es algo constantemente debatido. Este videojuego no satisface las definiciones más modernas del género, ya que no existía la habilidad de saltar. Donkey Kong, creado por Nintendo y lanzado en julio de 1981, fue el primer videojuego que permitió a los jugadores saltar por encima de los obstáculos y los agujeros, lo que lo convirtió en el primer verdadero videojuego de plataformas (Miguel, 2019).

Existen varios estilos y tipos de videojuegos de plataformas:

- Plataforma 2D (Super Mario Bros, Sonic the Hedgehog, Donkey Kong)
- Plataforma 2.5D (Limbo, Inside, Deadlight)
- Plataforma 3D (Prince of Persia the Sands of Time, Crash Bandicoot, Spyro)
- Isométrico (Fairlight, Death)
- Exploración (Castelvania, Tomb Raider, Metroid)

La investigación se centra en los videojuegos de plataformas 2D, estos utilizan gráficos planos, llamados sprites, y no tienen geometría tridimensional. Se dibujan en la pantalla como imágenes planas, y la cámara no tiene perspectiva. Los videojuegos de plataformas en 2D se caracterizan por utilizar gráficos con apariencia plana y scroll de pantalla, que puede ser horizontal (Super Mario Bross o Sonic the Hedgehog) o de forma vertical (Bubble Bobble, Donkey Kong o Super Cow) (Unity, 2019).

### 1.1.5 Arquitectura básica de un videojuego

Se presenta a continuación una arquitectura en capas que resume los ejemplos más destacados de la literatura especializada y una breve explicación de cada capa. Ver Figura 1.


*Figura 2 Arquitectura Básica de un Videojuego (Sánchez J. L., 2010).*

Game Interface o interfaz del juego, es la parte que interactúa directamente con el jugador y mantiene la comunicación entre este y el videojuego. Presenta todos los contenidos, escenas, opciones y los controles necesarios para poder interactuar dentro del videojuego, así como mostrar el look and feel (sensaciones y emociones emitidas por los diferentes elementos de la interfaz, así como la parte sonora del juego).

Game Engine o motor del juego hace referencia a una serie de rutinas que permiten la ejecución de todos los elementos del juego. Se puede decir que el Motor del Juego equivale a una conjunción del Motor Gráfico, Motor de Inteligencia Artificial, Motor de Sonido y Motor Físico. Es donde se gestiona la Inteligencia Artificial, personalidad, comportamiento y habilidad de los elementos del juego, sonidos asociados a cada elemento del juego y los aspectos gráficos, incluyendo la cinemática.

Game Mechanic o mecánica del juego es la parte fundamental de un videojuego, ya que encierra las características que diferencian a un juego de otro. Se concretiza el género del videojuego, los objetivos, reglas y la forma de interactuar para lograrlo.

### 1.1.6 Proceso de desarrollo de un videojuego

La realización de un videojuego requiere metodologías de desarrollo de *software* y mecanismos propios. El proceso de desarrollo de un videojuego cuenta con las etapas de preproducción donde se elabora el concepto del videojuego y la etapa de producción donde se incluyen el Diseño de juego, Diseño técnico, Implementación, Pruebas Alpha, Pruebas Beta y Gold Master (Monteagudo, 2018). En la Figura 2 se muestra una combinación de las etapas de desarrollo de un videojuego.

Diseño de Juego: se detallan los elementos que componen el videojuego y se finaliza el Documento de Diseño del Videojuego (GDD).

Diseño técnico: en esta etapa se describe como se implementará el juego y se generan los diagramas que describan la interacción con el usuario y los estados que atravesará el videojuego como *software*. Es la etapa en que se genera el documento de diseño. Incluye la planificación del juego y se identifican las tareas necesarias para su desarrollo, también se fijan plazos para la ejecución de dichas tareas.

Implementación: se finalizan todos los contenidos del juego: misiones, scripts, IA y efectos.

Pruebas Alpha: son también conocidas como Code Complete**,** durante este tipo de pruebas ya se tiene el producto terminado, y el mismo es probado por un equipo que ha estado involucrado en el diseño y desarrollo del juego, en busca de errores para su refinamiento, uno de los aspectos a probar es la jugabilidad.

Pruebas Beta: son también conocidas como Content Complete, en estas pruebas se terminan todas las variaciones del contenido (textos en diferentes idiomas, doblajes, etc.); se realizan por un equipo externo al equipo de desarrollo, que intentan que el juego vea la luz con la menor cantidad de defectos posibles. Se hacen comprobaciones legales con el fin de que los juegos se adapten a las leyes del país

dónde se van a publicar.

Gold Master: el juego que se publica y envia a la fábrica para su producción con todo el contenido del arte (diseño de portada, caja, etc) y los manuales de usuario (Monteagudo, 2018).

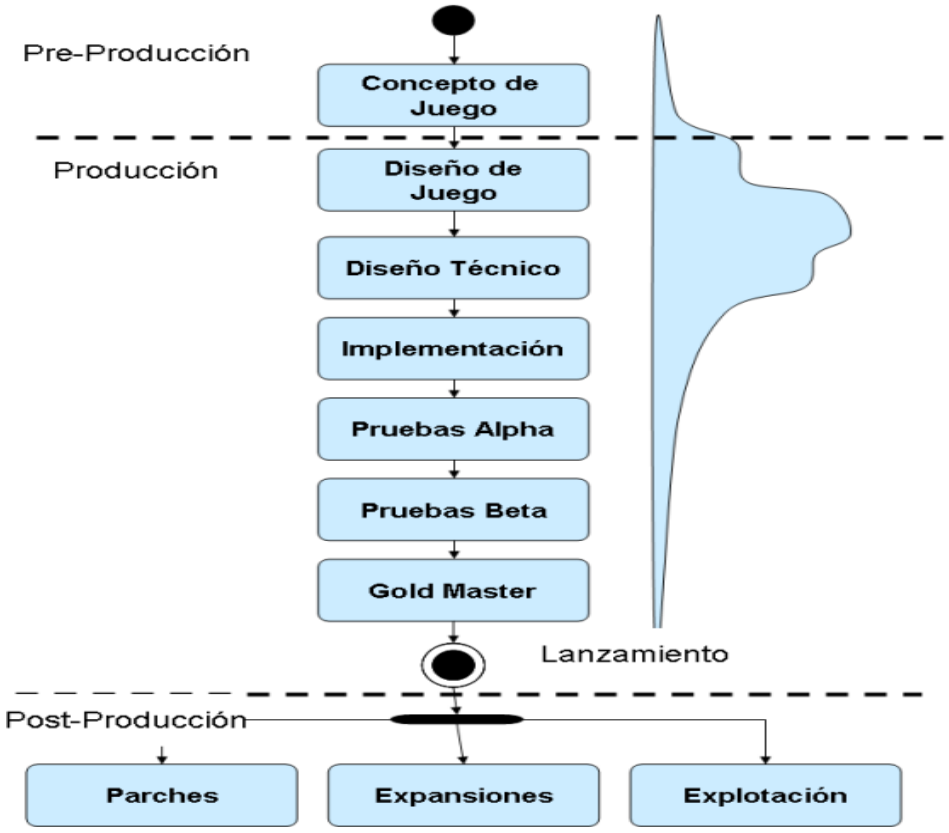



*Figura 3 Etapas en la Producción de un videojuego (Monteagudo, 2018)*

### 1.1.7 Proceso de pruebas

Cada característica y mecánica en el juego necesita ser probada para garantizar la calidad del mismo. Un juego que no ha sido probado de forma exhaustiva no está listo para el lanzamiento; incluso hay diferentes tipos de jugadores de prueba (Tecnoempleo, 2020).

En la tabla que se presenta a continuación, Tabla.1, se muestran los aspectos a tener en cuenta para realizar las pruebas al producto final (Tovar & Rodríguez, 2018).

*Tabla 1 Elementos para el desarrollo de pruebas para videojuegos*

| **Medidas de Calidad** | | | |
|---|---|---|---|
| **Objetivos del Juego** | Calidad | Experiencias de usuario | |
| **Análisis** | Jugabilidad | Diseño de Pruebas | Casos de Prueba |
| **Implementación** | Funcionalidad | Observación | Aplicación |
| **Finalidad** | Seguridad | Análisis | Resultados |
| | Integración | Cuestionario | |

## 1.2 De la usabilidad a la jugabilidad

Los videojuegos no constituyen sistemas interactivos tradicionales, están diseñados específicamente para divertir al usuario; por lo que no es suficiente con la característica de usabilidad para decidir si un videojuego satisface o no las necesidades y expectativas del usuario. Para caracterizar la experiencia del usuario ante un videojuego es necesario analizar una serie de atributos diferentes a los que se toman en cuenta en el caso de un sistema interactivo tradicional, es imprescindible analizar la jugabilidad.

### 1.2.1 Definición de la jugabilidad y sus atributos

La RAE define la jugabilidad como la facilidad de uso que un juego, especialmente un videojuego, ofrece a sus usuarios (Real Academia Española, 2022).

La jugabilidad es el conjunto de propiedades que describen la experiencia del jugador ante un sistema de juego determinado, cuyo principal objetivo es divertir y entretener de forma satisfactoria y creíble ya sea solo o en compañía (Sánchez J. L., 2022).

Se define jugabilidad en esta investigación como el grado en el que un videojuego satisface las expectativas del usuario.

La jugabilidad tiene partes de usabilidad, pero en los videojuegos no puede ser

considerada solamente como usabilidad. Aunque presenten atributos en común, en los videojuegos dichos atributos adquieren un matiz diferente. Se proponen los siguientes atributos para caracterizar a la jugabilidad (Regueiferos, 2012):

- Satisfacción: agrado o complacencia del jugador ante el juego o parte de este.
- Aprendizaje: facilidad para comprender el sistema y mecánica del videojuego.
- Efectividad: tiempo y recursos necesarios para lograr los objetivos propuestos en el videojuego.
- Inmersión: capacidad para creerse lo que se juega e integrarse en el mundo virtual mostrado en el juego.
- Motivación: característica del juego que mueve a la persona a realizar determinadas acciones y persistir en ellas para su culminación.
- Emoción: impulso involuntario originado como respuesta a los estímulos del videojuego, que induce sentimientos y desencadena conductas de reacción automática.
- Socialización: atributos que hacen apreciar el videojuego de distinta manera al jugarlo en compañía (multijugador).

### 1.2.2 Facetas de la jugabilidad

Las facetas de la jugabilidad permiten la identificación de forma más fácil de sus atributos y su relación con los diferentes elementos de un videojuego, las facetas de la jugabilidad son las siguientes (Sánchez J. L., 2022):

- Jugabilidad intrínseca se mide en la propia naturaleza del juego y cómo se proyecta al jugador. Está ligada al diseño del *GamePlay* y a la implementación del *Game Mechanic*, analizando cómo se representan las reglas, ritmos, objetivos y mecánica del videojuego.
- Jugabilidad interactiva faceta asociada a todo lo relacionado con la interacción con el usuario, diseño del I.U, mecanismos de diálogo y sistemas de control.

Fuertemente unida al *Game Interface*.

- Jugabilidad artística asociada a la calidad artística y estética de todos los elementos del videojuego a la naturaleza de éste. Entre ellos están la banda sonora, melodías, historia y su forma de narración, efectos sonoros, ambientación, calidad visual y gráfica.

- Jugabilidad intrapersonal también conocida como jugabilidad personal o perceptiva, tiene como objetivo la percepción que tiene el usuario del videojuego y los sentimientos que a éste le produce.

- Jugabilidad interpersonal, también tiene el nombre de jugabilidad en grupo, muestra las sensaciones o percepciones de los usuarios, y la conciencia de grupo, que aparecen cuando se juega en compañía ya sea de forma competitiva, colaborativa o cooperativamente.

- Jugabilidad mecánica está asociada a la calidad del videojuego como sistema de *software*. Está ligada al *Game Engine*, se centra en características como la fluidez de las escenas cinemáticas, la correcta iluminación, sonido, comportamiento de los personajes del juego y del entorno y movimientos gráficos.

### 1.3 Indicadores a evaluar y análisis de mecánicas de videojuegos

Los videojuegos son el pasatiempo preferido de gran parte de las personas a nivel mundial y a menudo se crean comunidades de videojuegos para debatir sobre el tema y jugar. Se destaca 3D juegos que es una comunidad de críticos donde realizan evaluaciones a los videojuegos, dónde para la evaluación toman la jugabilidad, el sonido, los gráficos e innovación como los principales indicadores, los que son representados por estrellas. Por otra parte, la comunidad *Meristation* realiza un análisis completo de los aspectos positivos y negativos del videojuego, pero a diferencia de 3D juegos, mide el sonido, los gráficos y la jugabilidad en una escala del 1 al 10 (Atiés, 2017).

Es notable que la jugabilidad es un aspecto que se mide a nivel mundial en las comunidades de críticos de videojuegos más destacadas, se debe tener en cuenta para la producción de videojuegos capaces de competir en el panorama internacional y satisfacer al público.

Evaluar las mecánicas fundamentales de los videojuegos como la locomoción y acciones, inciden directamente en la jugabilidad del videojuego en cuestión al estar contempladas en las facetas de la jugabilidad, que a su vez están ligadas a la arquitectura del videojuego. Son los aspectos fundamentales que el jugador sin percibirlo va evaluando a medida que juega el videojuego.

En los videojuegos se llama **Locomoción** a la mecánica que engloba todos los mecanismos encargados de los comportamientos de traslación de unidades en el juego (Toro D. S., 2016).

Existen acciones que no dependen de recursos disponibles como saltar, y otras donde los recursos pueden agotarse o no, como disparar con pistolas de balas infinitas o pistolas de balas limitadas. Esta mecánica recibe el nombre de **Acciones** y controla los mecanismos encargados de la activación, visualización y ejecución de acciones de las unidades (Toro D. S., 2016).

### 1.3.1 Análisis de las mecánicas en los videojuegos Super Cow y *Super Mario Bros*

Para determinar las mecánicas más características y genéricas de los videojuegos del género plataformas se analizan a continuación dos videojuegos de dicho género: *Super Cow* y *Super Mario Bros*.

Estrenado el 28 de septiembre de 2007 *“Super Cow”* es un divertido juego de plataformas orientado a los más pequeños; el jugador se sumerge en la piel de una vaca cuyo objetivo es proteger a todos los animales de la granja de los planes de clonación del malvado *Professor Duriarti*. *Super Cow* debe recorrer un sinfín de bosques y granjas recogiendo todas las monedas y demás objetos que encuentre por su camino y acabando con todos los animales creados por el *Professor Duriarti*. A continuación, se realiza un análisis del comportamiento de las mecánicas en el

videojuego *Super Cow* (Toledo, 2022).

Locomoción: *Super Cow* salta a los pozos, se mueve en el suelo, en los techos de los edificios, a lo largo de las ramas de los árboles y sobre nubes, a la derecha o izquierda para recoger las monedas a la vez que debe evadir los obstáculos que se presentan de forma imprevista y pueden acabar con su vida (Toledo, 2022).

Acciones: *Super Cow* para neutralizar al enemigo salta sobre él, en el caso de que el enemigo es especialmente grande debe saltar de dos a tres veces sobre el mismo (Toledo, 2022).

Reconocimiento de enemigos: *Super Cow* debe ser capaz de identificar a sus enemigos para poder neutralizarlos y que no acaben con su vida. Entre los enemigos de *Super Cow* se encuentran el caracol con ojos grandes, perro enojado, topo, cuervo negro y cuervo blanco (Toledo, 2022).

*"Super Mario Bros"* es un popular videojuego realizado por el diseñador y productor de videojuegos de origen japonés Shigeru Miyamoto y lanzado al mundo el 13 de septiembre de 1985 para la consola Nintendo Entertainment System. Describe las aventuras de dos fontaneros Mario y Luigi, quienes deben enfrascarse en el rescate de la Princesa Peach, del Reino Champiñón que fue secuestrada por el rey de los koopas, Bowser (Remedios, 2019). Los jugadores deben atravesar 8 mundos distintos, cada uno de los cuales ofrece 4 niveles. Es considerado el primer videojuego de plataformas de desplazamiento lateral de Nintendo y se ha convertido en un hito debido a la trascendencia de su diseño y papel en la industria de los videojuegos (Mesa, 2019).

Locomoción: Super Mario atraviesa cuevas y castillos, baja por tuberías y recorre plataformas corriendo y saltando hasta enfrentarse con Bowser para rescatar a la adorable princesa Peach (Marthis, 2022).

Acciones: *Super Mario* con el uso de potenciadores es capaz de lanzar bolas de fuego y cambiar a tamaños gigantes o miniatura (Marthis, 2022).

Reconocimiento de enemigos: *Super Mario* debe ser capaz de identificar a sus enemigos para defenderse de ellos y lograr su objetivo. El principal enemigo de *Super Mario* es *Bowser*, pero se enfrenta también a las tropas *koopas* o *goombas* (Marthis, 2022).

Luego del análisis de los videojuegos *"Super Cow"* y *"Super Mario Bros"* se aprecia que en los videojuegos de plataforma resultan indispensables las mecánicas de locomoción y acciones y el reconocimiento de los enemigos, ya que son los factores que posibilitan logar el objetivo del videojuego. El personaje principal debe moverse y evadir obstáculos, pero la forma en que lo hace varía en dependencia del videojuego y las circunstancias. Resulta también muy importante el reconocimiento del personaje principal, los enemigos y otros personajes secundarios. Por tanto, se decide tomar estas mecánicas como indicadores teniendo en cuenta las variaciones de locomoción y acciones que pueden presentarse.

### 1.4 Sistemas homólogos

Se investigaron los sistemas que realizan el proceso de pruebas a videojuegos de forma automática y no existe un sistema como este a nivel mundial. Los sistemas que tienen características que pueden ser útiles para dar solución al problema de investigación son los sistemas de Inteligencia Artificial (IA) que juegan videojuegos. Se describen a continuación dos de los sistemas de IA que juegan videojuegos más destacados: *Alpha* Go y *OpenAI Five*.

*Alpha* Go

*Alpha* Go es un algoritmo de Inteligencia Artificial que juega Go (juego de estrategia tradicional chino) desarrollado por el equipo de *DeepMind* y basado en redes neuronales profundas y aprendizaje por refuerzo. Con este algoritmo se intentó capturar el aspecto intuitivo del juego que deciden usar los grandes maestros del Go mediante aprendizaje por refuerzo, que se basa en tomar acciones dentro de cierto entorno. El programa aprendió a jugar Go gracias a miles de partidas con amateurs y profesionales. Posteriormente fue entrenado jugando contra diferentes versiones

de sí mismo, aprendiendo de los errores que cometía y mejorando hasta superar al campeón mundial de Go (Montesino, Instituto de Ingeniería del Conocimiento, 2022).

El algoritmo *Alpha* Go usa dos redes neuronales artificiales de tipo convolucional, una de evaluación (*value network*), que estima la probabilidad de que la posición actual permita ganar, y otra de estrategia (*police network*) que evalúa todos y cada uno de los movimientos en la posición actual (Silver, 2016).

*OpenAI Five*

*OpenAI Five* es un equipo de cinco redes neuronales que juega Dota 2, aprendió jugando más de 10 000 años de juego contra sí mismo. Demostró la capacidad de lograr un desempeño de nivel experto, aprender la cooperación humano-IA y operar a escala de Internet (Montesino, Instituto de Ingeniería del Conocimiento, 2022).

## 1.5 Metodología, herramientas y tecnologías a utilizar en la propuesta de solución

El proceso de desarrollo de *software* se apoya en el uso de diferentes herramientas y tecnologías, las cuales, en unión con la metodología seleccionada, conforman el ambiente de desarrollo de un sistema. En el siguiente epígrafe se selecciona la metodología de desarrollo de *software* y se describen las herramientas y tecnologías de las cuales se hace uso para desarrollar el sistema. Para la elección se toman como parámetros que sean herramientas de software libre y de código abierto, la documentación disponible, además de las prestaciones y ventajas a partir de las características del sistema.

### 1.5.1 Metodología de desarrollo de software

Una metodología de desarrollo de *software* se refiere al entorno que se usa para estructurar, planificar y controlar el proceso de desarrollo de un sistema de informático. Suele estar documentada y promovida por algún tipo de organización ya sea pública o privada. Tienen como principal objetivo aumentar la calidad del *software* que se produce en todas y cada una de sus fases de desarrollo. Las metodologías para el desarrollo del *software* imponen un proceso disciplinado sobre el desarrollo de *software* con el fin de hacerlo más predecible y eficiente. No existe una

metodología de *software* universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (Maida & Pacienzia, 2015).

### 1.5.2 Metodología de desarrollo de software AUP versión UCI

La UCI desarrolló una versión de la metodología de desarrollo de *software* AUP (Proceso Ágil Unificado), con el fin de crear una metodología que se adapte al ciclo de vida definido por la actividad productiva de la universidad. En esta versión se mantiene la fase de Inicio para el ciclo de vida de los proyectos, pero se modifican los objetivos de la misma y se unifican en la fase de Ejecución las fases restantes de la metodología AUP. Además, se agrega la fase de Cierre (Atiés, 2017).

Fases de AUP-UCI:

Inicio: durante esta fase se llevan a cabo las actividades relacionadas con la planeación del proyecto. Se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzos y costos, y decidir si ejecutar o no el proyecto.

Ejecución: durante esta fase se ejecutan las actividades requeridas para desarrollar el *software*, incluyendo el ajuste de planes del proyecto considerando los requisitos y la arquitectura.

Cierre: durante esta fase se analizan tanto los resultados del proyecto como la ejecución y se realizan las actividades formales de cierre del proyecto.

La metodología AUP-UCI consta de cuatro escenarios, entre ellos el #4 que no modela negocio y se modela el sistema con Historias de Usuario. Atendiendo a las características del sistema se decide optar por el escenario #4 pero para modelar el sistema se sustituyen las Historias de Usuario por la Descripción de los requisitos funcionales y no funcionales y Descripción de Casos de Uso del Sistema. Se toma esta decisión con el fin de definir lo que el sistema debe realizar para satisfacer las necesidades del cliente y presentar el flujo de trabajo básico del sistema y sus respuestas ante los eventos.

### 1.5.3 Herramienta de Ingeniería de Software Asistida por Computadora

Son un conjunto de programas y ayudas a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un *software*, permitiendo la creación de todo tipo de diagramas (Morales y otros, 2019). En la metodología AUP, se hace necesario el uso de una herramienta de Ingeniería de *Software* Asistida por Computadora en las fases de Análisis, Diseño e Implementación, donde se generan la mayor cantidad de artefactos.

**Visual Paradigm for UML v8.0**

*Visual Paradigm for UML* es una plataforma de modelado diseñada para apoyar a los arquitectos de sistemas, desarrolladores y diseñadores de *software*. *Visual Paradigm* es una herramienta que soporta todo el ciclo de vida de desarrollo de un *software*: Análisis y Diseño, Construcción, Pruebas y Despliegue (IONOS, 2021).

Se decide usar porque es una herramienta de software libre, fácil de instalar y actualizar y es de la que se tiene mayor dominio. Ofrece múltiples ventajas para la elaboración de los artefactos ingenieriles que genera la Metodología de Desarrollo de *Software* AUP-UCI.

### 1.5.4 Lenguaje de desarrollo

El desarrollo de sistemas de automatización en el plano mundial tiene gran variedad de lenguajes, siendo complejo decidirse por uno de estos.

**Python 3.8**

Python es un lenguaje de programación de alto nivel, multiplataforma y de código abierto. Posee un intérprete o consola que permite probar ciertas capacidades del lenguaje sin tener que crear un módulo de éste. Cuenta con una librería estándar que cubre las necesidades básicas de un programador, dando cobertura a tópicos como cadenas, estructuras de datos, funciones numéricas y matemáticas, servicios de los sistemas operativos, criptografía, entre otros (Becas Santander, 2021).

No posee facilidades para el desarrollo de interfaces de usuario para plataforma *desktop* por lo que se recurre al uso de librerías especializadas como *TKinter* para el diseño de las interfaces del sistema (Foundation, 2022).

Python es el lenguaje que se decide usar en la implementación porque es de código abierto y ofrece ventajas para programar IA, *Machine Learning,* y *Deep Learning.*

### 1.5.5 Editor de texto para la creación de código fuente

**Visual Studio Code**

Es un editor de texto desarrollado por *Microsoft* que ofrece muchas prestaciones para la programación, no es un Entorno de Desarrollo Integrado (IDE) en sí, pero es extensible para cubrir tales prestaciones mediante la instalación de extensiones y otros elementos tales como SDK para el desarrollo en diferentes lenguajes como *C#* (Velazco, 2021).

Se utiliza *Visual Studio Code* porque es gratuito y de código abierto. Para la confección de código *Python* se configura la herramienta con el intérprete *MiniConda* con un entorno para *Python* versión 3.8.

### 1.5.6 Anotador de imágenes

Con el fin de entrenar el modelo *Fast R-CNN***,** es necesario crear archivos especiales que contienen información de las anotaciones realizadas a las imágenes en la fase de preparación del set de datos de entrenamiento y de validación.

Es una herramienta basada en la web que permite anotar imágenes fácilmente y compartirlas al instante. Se utiliza principalmente para realizar el proceso de anotación de imágenes para tareas de reconocimiento de objetos tales como: *Image Segmentation* (segmentación de imágenes), *Semantic Segmentation* (segmentación semántica de imágenes), *Key Point Detection* (detección de puntos clave para la estimación de poses), y, *Panoptic Segmentation* (combinación entre segmentación semántica y segmentación de instancias) (Russell, 2018).

Se utiliza la herramienta para crear las anotaciones necesarias en el entrenamiento del modelo *Fast R-CNN* en la tarea de segmentación de instancias.

### 1.5.7 Entorno de desarrollo Python

#### Intérprete *Python*

*Anaconda Distribution* es una distribución de *Python* que funciona como un gestor de entorno, un gestor de paquetes. Es una *Suite* de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con *Python* (Toro L. , 2017).

Se utiliza la distribución *MiniConda* con *Python 3.8.*

### 1.5.8 Librerías utilizadas

Una librería es un conjunto de archivos que se utilizan para desarrollar un *software*. Suele estar compuesta de códigos y datos, y su fin es ser utilizada por otros programas de forma totalmente autónoma (devCamp, 2020).

#### OPENCV

Es una biblioteca de código abierto que contiene implementaciones que abarcan más 2500 algoritmos, está especializada en Visión Artificial y *Machine Learning*. Su utilización principal es para la detección de objetos y rostros, reconocer escenarios y extraer modelos 3D. Sirve de proveedor de infraestructura para las aplicaciones relacionadas con la visión artificial; es multiplataforma, capaz de ejecutarse en los principales sistemas operativos (*Windows, Mac OS X, Linux, Android o iOS*). Aunque está escrito en C++, se pueden utilizar en otros lenguajes de programación como *Java*, *C#* y *Python* (Rodríguez H. , 2021).

Se utiliza para la representación visual de las segmentaciones realizadas por el modelo de Aprendizaje automático, así como para la precarga de la salida de video del sistema operativo que es lo que se va a utilizar para que el modelo de visión pueda realizar sus predicciones.

***PyWin32 (win32gui, win32ui, win32con)***

Es una librería que da acceso a los principales elementos del sistema operativo *Windows* (API de *Windows*), en el sistema se emplea para la obtención de las imágenes que son resultado de la salida de video de la pantalla; así como para otras prestaciones referentes al manejo de elementos del sistema operativo (Naidu, 2018).

***Pynput (pynput. keyboard, pynput. mouse)***

Esta biblioteca permite controlar y supervisar los dispositivos de entrada.

Contiene sub paquetes para cada tipo de dispositivo de entrada soportado (Palmer, 2015).

**Detectron2**

*Detectron2* es la biblioteca de próxima generación de *Facebook AI Research* que proporciona algoritmos de detección y segmentación de última generación. Es la sucesora de *Detectron* y *maskrcnn-benchmark*. Es compatible con varios proyectos de investigación de visión por ordenador y aplicaciones de producción en *Facebook* (Wu y otros, 2019).

## 1.6 Redes Neuronales Convolucionales basadas en Regiones

*Fast R-CNN* (*Region-based Convolutional Neural Networks*) se basa en trabajos anteriores para clasificar eficazmente las propuestas de objetos mediante redes convolucionales profundas. En comparación con los trabajos anteriores, *Fast R-CNN* emplea varias innovaciones para mejorar la velocidad de entrenamiento y prueba, al tiempo que aumenta la precisión de la detección (Girshick, 2015). En la Figura.3 se muestra el proceso que sigue *Fast R-CNN* para la detección de objetos.

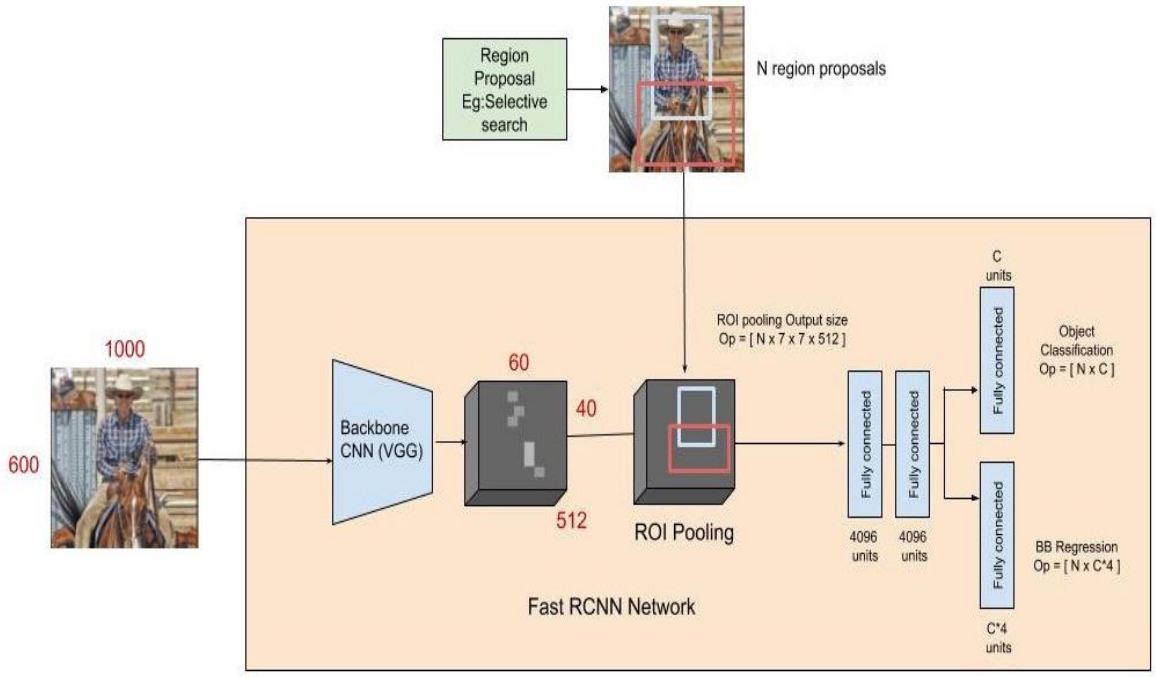



*Figura 4 Diagrama de la estructura de Fast R-CNN fuente (Girshick, 2015)*

El algoritmo utiliza como entrada una imagen y un set de proposiciones de objetos. La red primero procesa la imagen en su totalidad mediante el uso de varias capas convolucionales (*convolutional layer*) y capas de agrupamiento máximo (*max pooling layer*) para producir un mapa de características convolucionales. A continuación, para cada propuesta de objeto, una capa de agrupación de regiones de interés (RoI) extrae un vector de características de longitud fija del mapa de características. Cada vector de características se introduce en una secuencia de capas totalmente conectadas (fc) que finalmente se ramifican en dos capas de salida hermanas: una que produce estimaciones de probabilidad *softmax* sobre K clases de objetos más una clase de "fondo" y otra capa que produce cuatro números de valor real para cada una de las K clases de objetos. Cada conjunto de 4 valores codifica las posiciones refinadas de los recintos para una de las K clases (Girshick, 2015).

## Conclusiones parciales

- El estudio del arte de los principales conceptos asociados a la investigación posibilitó determinar la relevancia que tiene la jugabilidad en el momento de realizar pruebas a los videojuegos.
- El estudio de los videojuegos del género plataformas “Super Mario” y “Super Cow”, permitió determinar las mecánicas más frecuentes en este tipo de videojuegos y tomarlas como referencia para medir la jugabilidad.
- El análisis de los sistemas de IA que juegan videojuegos brindó como dato esencial que estos sistemas usan dos o más redes neuronales y aprendizaje reforzado, sirviendo como referencia para el desarrollo del sistema.
- Tras el análisis de las habilidades técnicas del equipo de desarrollo y las necesidades funcionales y no funcionales del cliente para la implementación de la propuesta de solución se determinó que la metodología AUPvUCI escenario #4 se ajusta perfectamente a las características del sistema que se desea desarrollar.

# Capítulo II. Descripción de la propuesta de solución

Definir las características de un sistema es clave para desarrollar los artefactos ingenieriles que brindan una visión estática del sistema, el modelado de las mismas ofrece una panorámica completa para realizar el análisis y diseño y posteriormente transformarlas en código fuente. Al plantear las características del sistema se construyen las bases de la futura solución.

En este capítulo se presenta el análisis y diseño de la solución propuesta, se describe el sistema y se realiza la captura de los requisitos funcionales y no funcionales. Para obtener una descripción de las funcionalidades que debe cumplir el sistema se describen los casos de uso del negocio, se realiza el diagrama de casos de uso del sistema y se especifican los casos de uso del sistema. Además, se exponen las pautas de programación empleadas en el desarrollo del sistema para garantizar el entendimiento y la legibilidad del código y se modela el *hardware* utilizado en la implementación a través del diagrama de despliegue.

## 2.1 Descripción del sistema propuesto

El sistema propuesto tiene como objetivo medir la jugabilidad a través de las mecánicas de los videojuegos del género plataformas en 2D. Los usuarios del sistema son los miembros del equipo de desarrollo de videojuegos del Centro de Tecnologías Interactivas y no es necesaria la autenticación para acceder a las funcionalidades.

El sistema debe brindar la posibilidad de entrenar el módulo visual y probarlo, así como crear un agente de IA para ejecutar las acciones en el entorno (videojuego). Los usuarios del sistema pueden comprobar las mecánicas básicas de los videojuegos de plataformas como son: caminar y correr hacia la derecha o izquierda, saltar, agacharse, escalar y otras que son genéricas en otros géneros de videojuegos tales como: interactuar, atacar, o esquivar. Al finalizar la comprobación de las mecánicas se obtiene un reporte con el nivel de cumplimiento de las mecánicas que han sido comprobadas.

Para la ejecución del sistema primeramente es necesario ejecutar el videojuego al que se le desea realizar las pruebas. El sistema debe contar con cuatro ventanas permitiendo la configuración del modelo visual, el agente de IA, las mecánicas a comprobar y para el reporte de las mecánicas comprobadas. No debe ser posible ir a la configuración siguiente sin realizado la actual.

Para la comprobación de las mecánicas, y que esta sea de forma automática se hace uso de un "agente" de IA, específicamente mediante el uso de algoritmos de aprendizaje reforzado del tipo *DQN* (*Deep Q Network*) con una variación del algoritmo.

### 2.1.1 DQN clásico

En las redes profundas *Q* se definen:

1- Acción (*a*): acción que es capaz de realizar el agente sobre el estado actual y que pertenece al conjunto finito de acciones *A.*
2- A: conjunto finito de todas las acciones posibles que puede realizar el agente sobre todos los estados *s* que pertenecen al universo finito de estados *S*.
3- Estado (s): es una representación del entorno donde se desenvuelve el agente.
4- Agente: es el actor que va a realizar acciones sobre el entorno.
5- t: instante de tiempo.

Luego haciendo uso de los procesos de decisión de *Markov* (*MDP's*) por sus siglas en inglés, que son procesos de decisión secuencial de carácter estocásticos[1] donde el costo y la función de transición solamente dependen del estado actual, se logra agentes que son capaces de aprender sobre la marcha además de que su comportamiento resulta no determinista (Markov, 1990).

El modelo *MDP's* combinado con *DQN*, propone un sistema de acciones secuenciales que siguen el siguiente patrón (Markov, 1990):

$(S_0, A_0, R_1, \ldots. S_n, A_n, R_{n+1})$

[1] Estocástico: refiérase a los sistemas donde su comportamiento intrínseco es no determinista

Donde $S_0$ se refiere al estado inicial, $A_0$ es la acción inicial, $R_1$ es la recompensa obtenida después de realizar la acción inicial sobre el estado inicial, sucesivamente hasta el instante de tiempo n, ver Figura 4.

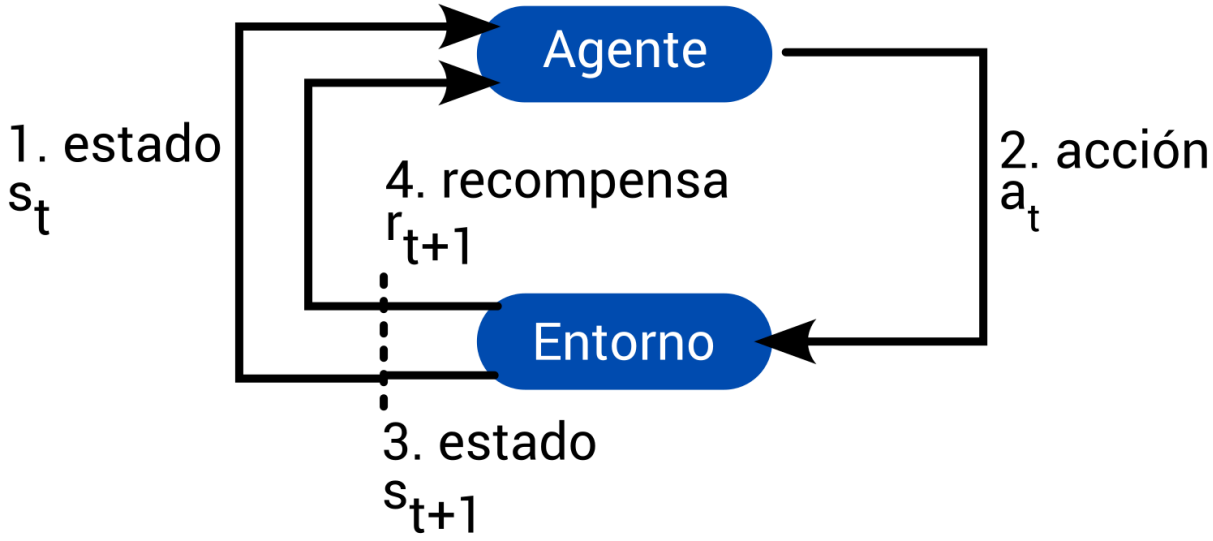



*Figura 5 Diagrama de acciones del agente en el entorno. Fuente < (Reinforcement Learning-Developing Intelligent Agents , 2022) >*

1- El agente recibe una representación de entorno en la forma *st*
2- Luego selecciona una acción *at* del conjunto *A* y la ejecuta en un emulador del entorno
3- El entorno cambia hacia la forma *st+1* y se genera la recompensa
4- Luego para el próximo instante de tiempo el agente recibe el estado actual y la recompensa del instante de tiempo anterior
5- Las líneas discontinuas representan un punto de partida donde después de realizada la acción inicial el agente recibe además del estado actual, la recompensa de la acción anterior sobre el estado anterior.

La función principal del agente es maximizar la cantidad de recompensas futuras para todos los instantes de tiempo en tareas continuas, para tareas discontinuas el agente trata de maximizar la obtención de descontada de recompensas (Reinforcement Learning-Developing Intelligent Agents , 2022).

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Donde *γ* es un factor de descuento perteneciente al rango [0,1], *k* es un factor de peso que provoca que el agente priorice las recompensas más cercanas por sobre las futuras y *R* es la recompensa (Reinforcement Learning-Developing Intelligent Agents , 2022).

Las funciones de valor son las que van a determinar las acciones que tome el agente, se definen dos tipos, *State Value Function* (función estado valor *vπ*) y *Action Value Function* (función acción valor *Qπ*); esta última representa cuan bueno es para el agente tomar alguna acción sobre un estado siguiendo la política *π* (Reinforcement Learning-Developing Intelligent Agents , 2022).

$$Q_\pi(s,a) = E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|S_t = s, A_t = a\right]$$

La mejor acción que pueda tomar el agente sobre un estado ***s*** siguiendo una política *π* se define como:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

La cual otorga la mayor recompensa esperada que se pueda obtener por cada política *π* para cada par acción-estado; esta debe satisfacer entonces la función de optimalidad de Bellman:

$$Q_*(s,a) = \mathrm{E}\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

Por lo que el objetivo de *Q-learning* es encontrar la política más óptima mediante el aprendizaje del valor *Q* para cada par acción-estado (Reinforcement Learning-Developing Intelligent Agents , 2022). Una forma de lograr ese objetivo es el de la incorporación de redes neuronales profundas para la aproximación de los valores *Q* al valor *Q-optimo*. Estas redes neuronales obtienen como entrada, en el caso de agentes que *"juegan"* videojuegos, varias capturas de imagen del juego de forma consecutivas que son alimentadas a la red de la misma forma que cualquier red

neuronal convolucional. Como salida se obtiene las estimaciones de la función *Q* correspondientes a cada acción posible para el estado de entrada, esta red se denomina como *Policy Network* (red de política *P*). En la Figura 5, luego de ejecutada la mejor acción posible en un emulador se procede a realizar un paso del estado obtenido por una segunda red neuronal clon denominada *Target Network* (red objetivo *T*) para ir aproximando el valor *Q* al valor *Q-óptimo* según la ecuación de Bellman, con estos valores se calcula el error mediante (Hardik, 2021):

$$loss = \mathrm{E}\left[R_{t+1} + \gamma \max_{a'} q_*(s', a')\right] - E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$$

Se actualizan todos los pesos de las neuronas mediante un algoritmo de propagación hacia atrás (*Back Propagation*) en la red *P.* Luego de varios instantes de tiempo *x*, se actualiza la red *T* con los pesos de *P*.

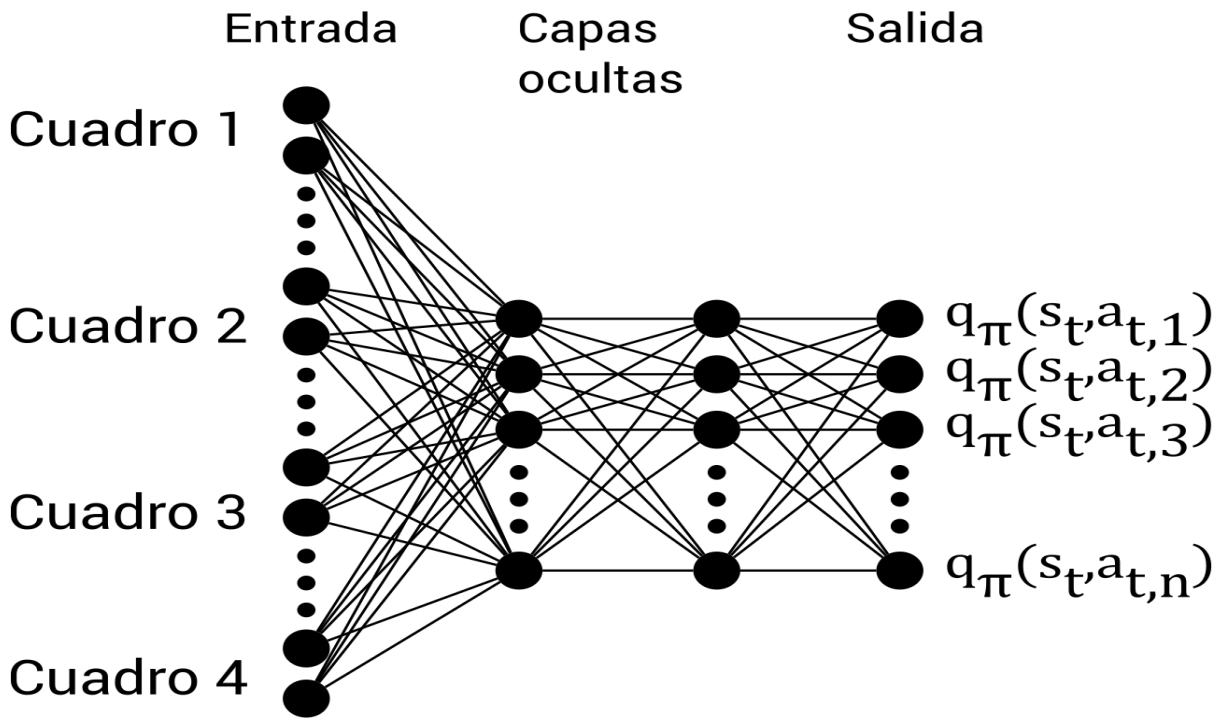



*Figura 6 diagrama para una red neuronal convolucional DQN.Fuente: elaboración propia*

El proceso de aprendizaje se realiza mediante el uso iterativo de experiencias pasadas en lo que se conoce como *Replay Memory* (memoria de juego), donde el conocimiento se almacena en forma de tupla:

$$e_i = (s_t, a_t, R_{t+1}, s'_{t+1})$$

Considerando al estado actual $s_t$, la acción actual $a_t$, la recompensa obtenida $R_{t+1}$ y el estado en el instante de tiempo siguiente $s'_{t+1}$. Luego se realiza una segmentación del *dataset* para entrenamiento y validación para el proceso de entrenamiento (Hardik, 2021).

### 2.1.2 Variación del algoritmo DQN mediante simplificación del estado de entrada

Como parte del desarrollo del modelo se realiza una variación del algoritmo mediante una simplificación de la red al simplificar el estado que sirve como entrada.

En la construcción del estado suministrado a las redes se hace uso de un mecanismo que mediante un componente de segmentación de imágenes (*FAST R-CNN*). Se construye una versión simplificada del estado, reduciendo significativamente la cantidad de conexiones entre la capa de entrada y la primera capa oculta. Además, se elimina el uso de capas convolucionales al no ser necesario el tratamiento de las imágenes. Esta nueva versión simplificada del estado contiene solo la información relevante para el agente (posición actual, dirección de desplazamiento, posición de diferentes elementos en la vecindad del agente etc.), lo que conlleva a una reducción del poder de procesamiento necesario para realizar ambos procesos de alimentación de la red (*FORWARD PROPAGATION*), así como el proceso de propagación del error (*BACK PROPAGATION*). Además, permite la incorporación de más neuronas en las capas ocultas o más capas, lo que hace posible la realización de mejores aproximaciones de los valores *Q* estimados.

El proceso de entrenamiento del componente de segmentación de imágenes, se realiza de forma más acelerada. Permite entrenar al modelo con nueva información referente a los juegos que se prueben. Posibilita simplificar las tuplas de experiencia significativamente incrementando la capacidad de la memoria de juego.

## 2.2 Modelo de dominio

El modelo del dominio es un modelo conceptual en el que se identifican y explican los conceptos significativos en un dominio de problema; es considerado un diccionario

visual de abstracciones, conceptos, vocabulario e información del dominio del problema (Rubio, 2015).

A continuación, se realiza la definición de las clases del Modelo de dominio:

Sistema: el sistema mide la jugabilidad mecánica en los videojuegos del género plataformas.

Videojuego: videojuego que se prueba en el sistema.

Jugabilidad mecánica: propiedad que mide el sistema en un videojuego.

Mecánica: mecánicas a comprobar del videojuego.

Género: plataformas es el género de los videojuegos que el sistema prueba.

En la Figura 6 se muestra el Diagrama de dominio elaborado para plasmar los conceptos más importantes de la investigación.

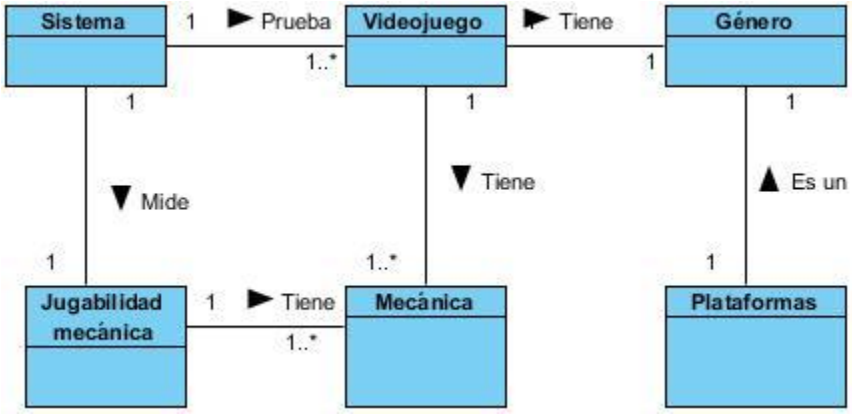



*Figura 6 Diagrama de dominio. Fuente: elaboración propia*

El sistema mide la jugabilidad mecánica, la cual tiene mecánicas presentes en los videojuegos del género plataformas, como son: caminar, correr, saltar, atacar, recibir daño, agachar, esquivar y renacer.

## 2.3 Requisitos funcionales y requisitos no funcionales

### 2.3.1 Requisitos Funcionales

Los requisitos funcionales son declaraciones de los servicios que presta el sistema, de manera en que este reacciona en situaciones particulares (Sommerville, 2015). Se definen los siguientes requisitos funcionales para la propuesta de solución:

*Tabla 2 Descripción de los requisitos funcionales.*

| No. | Nombre | Descripción | Complejidad | Prioridad |
|---|---|---|---|---|
| 1 | RF-1:<br>Entrenar nuevo modelo visual | El sistema debe permitir al usuario entrenar un nuevo modelo visual para lo cual se deberán tener los siguientes datos:<br>• Ruta hacia la carpeta de imágenes de entrenamiento,<br>• Ruta hacia el archivo de anotaciones de entrenamiento,<br>• Nombre del set de datos de entrenamiento,<br>• Ruta del archivo de anotaciones de prueba, | Alta | Alta |

| | | | | |
|---|---|---|---|---|
| | | • Nombre del set de datos de prueba,<br>• Ruta del archivo de configuración del modelo visual,<br>• Ruta del directorio de salida del modelo visual,<br>• Dispositivo de aceleración,<br>• Iteraciones de entrenamiento. | | |
| 2 | RF-2:<br>Usar modelo existente | El sistema debe permitir el uso de un modelo visual existente para lo cual se debe activar el campo de búsqueda:<br>• Buscar modelo existente. | Alta | Alta |
| 3 | RF-3:<br>Restablecer | El sistema debe permitir que se | Media | Media |

| | | | | |
|---|---|---|---|---|
| | valores del modelo visual | restablezcan los valores anteriores. | | |
| 4 | RF-5:<br>Entrenar modelo visual | El sistema debe permitir el entrenamiento de un modelo visual nuevo o ya existente. | Alta | Alta |
| 6 | RF-6:<br>Probar modelo visual | El sistema debe mostrar las imágenes de entrenamiento con el porciento de predicción en cada uno de los objetos. | Media | Alta |
| 7 | RF-7:<br>Pasar a la siguiente configuración | El sistema debe permitir pasar a la siguiente configuración una vez se haya configurado el módulo actual. | Media | Alta |
| 8 | RF-8:<br>Crear nuevo agente | El sistema debe permitir la creación de un nuevo agente para lo cual se deberán insertar los siguientes datos: | Alta | Alta |

| | | | | |
|---|---|---|---|---|
| | | • Factor de aprendizaje,<br>• Factor de descuento,<br>• Inicializador de pesos de la red. | | |
| 9 | RF-9:<br>Restablecer valores del nuevo agente | El sistema debe permitir restablecer los valores del agente a los valores anteriores. | Media | Media |
| 10 | RF-10:<br>Cargar agente desde configuración existente. | El sistema debe permitir cargar un agente desde una configuración ya existente para lo cual se debe insertar:<br>• Ruta hacia el archivo de configuración. | Media | Alta |
| 11 | RF-11:<br>Listar mecánicas | El sistema debe mostrar un listado de todas las mecánicas que se pueden comprobar. | Media | Alta |

| | | | | |
|---|---|---|---|---|
| 12 | RF-12:<br><br>Realizar desplazamiento a la derecha | El sistema debe simular la entrada de la tecla asignada para el desplazamiento a la derecha (mantener presionada la tecla) en iteraciones de 5-10 segundos de retraso, y medir por cada iteración si hubo un desplazamiento del personaje hacia la derecha. Para ello se debe insertar:<br><br>• *Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | Alta | Alta |
| 13 | RF-13:<br><br>Realizar desplazamiento a la izquierda | El sistema debe simular la entrada de la tecla asignada para el desplazamiento a la izquierda (mantener presionada la tecla) | Alta | Alta |

| | | | | |
|---|---|---|---|---|
| | | en iteraciones de 5-10 segundos de retraso, y medir por cada iteración si hubo un desplazamiento del personaje hacia la derecha. Para ello se debe insertar:<br><br>*Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | | |
| 14 | RF-14:<br><br>Realizar desplazamiento hacia arriba (escalar hacia arriba) | El sistema debe simular la entrada de la tecla asignada para el desplazamiento hacia arriba (escalar hacia arriba) (mantener presionada la tecla) en iteraciones de 5-10 segundos de retraso, y medir por cada iteración si hubo un desplazamiento del personaje hacia arriba sobre un | Media | Alta |

| | | | | |
|---|---|---|---|---|
| | | objeto identificado como escalera. Para ello se debe insertar:<br><br>*Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | | |
| 15 | RF-15:<br><br>Realizar desplazamiento hacia abajo (escalar hacia abajo) | El sistema debe simular la entrada de la tecla asignada para el desplazamiento hacia arriba (escalar hacia abajo) (mantener presionada la tecla) en iteraciones de 5-10 segundos de retraso, y medir por cada iteración si hubo un desplazamiento del personaje hacia abajo sobre un objeto identificado como escalera. Para ello se debe insertar: | Media | Alta |

| | | | | |
|---|---|---|---|---|
| | | *Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | | |
| 16 | RF-16:<br>Realizar salto simple | El sistema debe simular la entrada de la tecla asignada para salto en iteraciones de 5-10 segundos de retraso, y medir por cada iteración si hubo un desplazamiento del personaje hacia arriba. Para ello se debe insertar:<br>*Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | Media | Alta |
| 17 | RF-17:<br>Realizar salto doble (salto de larga duración) | El sistema debe simular la entrada de la tecla asignada para salto dos veces seguidas en iteraciones de 5-10 segundos de retraso, y medir por cada iteración si | Media | Alta |

| | | | | |
|---|---|---|---|---|
| | | hubo un desplazamiento del personaje hacia arriba mayor que en RF-18. Para lo cual se debe insertar: *Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | | |
| 18 | RF-18:<br>Realizar salto en diagonal derecha | El sistema debe simular la entrada de la tecla asignada para salto en diagonal derecha en iteraciones de 5-10 segundos de retraso, y medir por cada iteración si hubo un desplazamiento del personaje hacia arriba y en diagonal hacia la derecha. Para ello se debe insertar: *Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | Media | Alta |

| | | | | |
|---|---|---|---|---|
| 19 | RF-19:<br>Realizar salto en diagonal izquierda | El sistema debe simular la entrada de la tecla asignada para salto en diagonal izquierda en iteraciones de 5-10 segundos de retraso, y medir por cada iteración si hubo un desplazamiento del personaje hacia arriba y en diagonal hacia la izquierda. Para ello se debe insertar: *Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | Media | Alta |
| 20 | RF-20:<br>Comprobar la mecánica interactuar | El sistema debe comprobar la interacción del personaje con el entorno. | Media | Alta |
| 21 | RF-21:<br>Atacar | El sistema debe simular la entrada de la tecla asignada para atacar en iteraciones de 5-10 | Media | Alta |

| | | | | |
|---|---|---|---|---|
| | | segundos de retraso, y medir por cada iteración si el personaje cambia al estado de atacar. Para ello se debe insertar: *Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | | |
| 22 | RF-22: Atacar usando el mouse | El sistema debe simular la entrada de la tecla del mouse asignada para atacar en iteraciones de 5-10 segundos de retraso, y medir por cada iteración si el personaje cambia al estado de atacar. Para ello se debe seleccionar:<br>• Lado derecho o<br>• Lado izquierdo | Media | Alta |
| 23 | RF-23: Esquivar | El sistema debe simular la interrupción de *hardware* | Media | Alta |

| | | | | |
|---|---|---|---|---|
| | | correspondiente a la acción esquivar, en iteraciones de 5s de retraso, y medir por cada iteración si el personaje esquivó a los personajes que se acercan a él. | | |
| 24 | RF-24: Comprobar la mecánica agacharse | El sistema debe simular la entrada de la tecla asignada para agacharse estando encima de un elemento identificado como suelo u obstáculo en iteraciones de 5s de retraso, y medir por cada iteración si hubo desplazamiento hacia abajo. Para ello se debe insertar: *Press shortcut* (tecla que debe ser presionada para ejecutar la acción). | Media | Alta |

| | | | | |
|---|---|---|---|---|
| 25 | RF-25: Comprobar la mecánica morir | El sistema debe detectar cuando la barra de salud del personaje esté vacía o cuando el personaje caiga al vacío. | Media | Alta |
| 26 | RF-26: Obtener reporte | El sistema debe emitir un reporte con el nivel de cumplimiento de las mecánicas al finalizar la comprobación de las mismas. | Alta | Alta |
| TÉCNICOS | | | | |
| 27 | RF-27: Acceder a la salida de video en tiempo real | El sistema para su funcionamiento de recoger en tiempo real la salida de video. | Alta | Alta |
| 28 | RF-28: Reconocimiento de objetos (detección) | El sistema debe aplicar mecanismos de aprendizaje automático para la detección y clasificación de objetos. | Muy alta | Crítica |

| | | | | |
|---|---|---|---|---|
| 29 | RF-29:<br>Reconocimiento de personajes (jugador) | El sistema debe aplicar mecanismos de aprendizaje automático para la detección y clasificación de objetos para reconocer y fijar al personaje jugador. | Muy alta | Crítica |
| 30 | RF-30:<br>Reconocimiento de personajes (neutrales) | El sistema debe aplicar mecanismos de aprendizaje automático para la detección y clasificación de personajes neutrales. | Muy alta | Crítica |
| 31 | RF-31:<br>Reconocimiento de personajes (amistosos) | El sistema debe aplicar mecanismos de aprendizaje automático para la detección y clasificación de personajes amistosos. | Muy alta | Crítica |

| | | | | |
|---|---|---|---|---|
| 32 | RF-32:<br>Reconocimiento de personajes (enemigos) | El sistema debe aplicar mecanismos de aprendizaje automático para la detección y clasificación de personajes enemigos. | Muy alta | Crítica |
| 33 | RF-33:<br>Clasificación de objetos | El sistema debe aplicar mecanismos de aprendizaje automático para la clasificación de objetos sin etiqueta (aprendizaje). | Muy alta | Crítica |
| 34 | RF-34:<br>Aprendizaje de la red neuronal | Ante cada clasificación sin etiqueta, la red debe aprender a clasificar los diferentes elementos que detecte. | Muy alta | Crítica |
| 35 | RF-35:<br>Generar interrupciones de teclado | El sistema debe ser capaz de generar interrupciones de teclado aun | Media | Alta |

| | | | | |
|---|---|---|---|---|
| | | estando en segundo plano. | | |
| 36 | RF-36:<br>Generar interrupciones de puntero | El sistema debe ser capaz de generar interrupciones de puntero aun estando en segundo plano. | Media | Alta |
| 37 | RF-37: Personalización de las mecánicas a probar | El sistema debe poseer un listado de las mecánicas que se pueden probar y permitir crear una lista personalizada o utilizar la que posee por defecto. | Media | Alta |

### 2.3.2 Requisitos no funcionales

Los requisitos no funcionales (RNF) se refieren a las cualidades o características del *software*, estos son especialmente importantes pues si el usuario no se siente atraído por estos puede no sentir satisfacción y desistir del sistema. Según Sommerville, los requisitos no funcionales son los que actúan para obligar la solución y se conocen a veces como apremios o requisitos de calidad. Se refieren a todos los requerimientos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento que debe poseer el sistema (Sommerville, 2015).

Calisoft evalúa la calidad de las aplicaciones informáticas y productos de software basado en la verificación del cumplimiento de los requisitos seguidamente especificados. Los requisitos están agrupados y clasificados según características y

sub características establecidas en el modelo de calidad de la norma ISO/IEC 25010:2011 (Calisoft, 2017).

Requisitos no funcionales de usabilidad:

RNF1- Ofrecer una navegación de pasos secuenciales.

RNF-2 Brindar una interfaz con alto contraste de color entre los elementos de fondo y el texto.

RNF-3 Proveer el idioma español en todas las interfaces del sistema.

Requisitos de Eficiencia de Desempeño:

RNF-4 Poseer tarjeta gráfica con tecnología CUDA y de 2GB o superior.

RNF-5 Garantizar que el equipo o terminal tenga capacidad de memoria RAM de 8 GB.

RNF-6 Asegurar que la versión de *Python* instalada sea 3.8.

RNF-7 Garantizar un sistema operativo *Windows* 10 o superior.

RNF-8 Disponer de capacidades de almacenamiento superiores a 10 GB.

Restricciones de implementación y diseño:

RNF-9 Utilizar el IDE *Visual Studio Code* en su versión 1.73.1.

RNF-10 Emplear las librerías *OPENCV* v-4.6.0.66, *Pynput* v-1.7.6, *Detectron2* v-0.6, *Pywin32* v-304.

Requisitos de hardware:

RNF-11 Poseer los dispositivos de entrada salida de datos siguientes: monitor o pantalla, teclado, puntero (mouse).

## 2.4 Casos de Uso del Sistema

Un caso de uso especifica una secuencia de acciones, incluyendo variantes que el sistema puede llevar a cabo y producen un resultado observable para un actor concreto (Mora, 2017).

### 2.4.1 Diagrama de Casos de Uso del Sistema

El diagrama de casos de uso del sistema representa gráficamente los procesos y la interacción con los actores del sistema (Atiés, 2017). Se muestra a continuación el diagrama de casos de uso del sistema de la propuesta de solución.

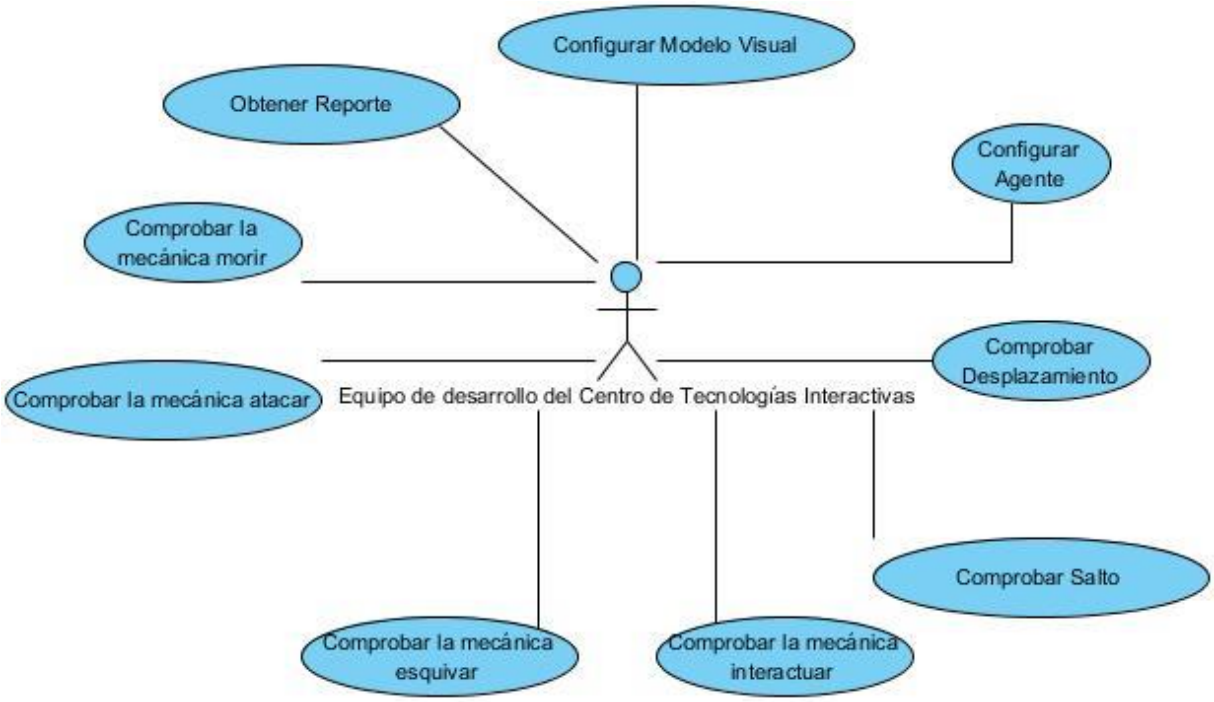



*Figura 7 Diagrama de caso de uso de sistema. Fuente: elaboración propia*

### 2.4.2 Descripción de Casos de Uso del Sistema

Se presenta en esta sección la descripción del Caso de Uso del Sistema “Comprobar desplazamiento”, las restantes se encuentran en el Anexo II**.**

### CU3. Comprobar desplazamiento

*Tabla 3 Descripción del Caso de Uso “Comprobar desplazamiento”*

| | |
|---|---|
| **Objetivo** | Permitir comprobar las mecánicas: realizar desplazamiento a la derecha, realizar desplazamiento a la izquierda, realizar |

| | |
|---|---|
| | desplazamiento hacia arriba (escalar hacia arriba), realizar desplazamiento hacia abajo (escalar hacia abajo). |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. |
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea comprobar las mecánicas: realizar desplazamiento a la derecha, realizar desplazamiento a la izquierda, realizar desplazamiento hacia arriba (escalar hacia arriba), realizar desplazamiento hacia abajo (escalar hacia abajo) o una combinación de las mismas. |
| **Complejidad** | Alta |
| **Prioridad** | Alta |
| **Precondiciones** | El videojuego al que se le desea comprobar las mecánicas debe estar en ejecución al igual que el sistema. |
| **Postcondiciones** | Se comprobaron las mecánicas: realizar desplazamiento a la derecha, realizar desplazamiento a la izquierda, realizar desplazamiento hacia arriba(escalar hacia arriba), realizar desplazamiento hacia abajo(escalar hacia abajo) o una combinación de las mismas. |

| **Flujo de eventos** | | |
|---|---|---|
| **Flujo básico “Comprobar desplazamiento”** | | |
| **Actor** | | **Sistema** |
| 1. | Desea comprobar las mecánicas: realizar desplazamiento a la derecha, realizar desplazamiento a la izquierda, realizar desplazamiento hacia arriba (escalar hacia arriba), realizar desplazamiento hacia abajo (escalar hacia abajo) o una combinación de las mismas. | |

| | | |
|---|---|---|
| 2. | | Muestra un listado con todas las mecánicas. |
| 3. | Selecciona las mecánicas: realizar desplazamiento a la derecha, realizar desplazamiento a la izquierda, realizar desplazamiento hacia arriba (escalar hacia arriba), realizar desplazamiento hacia abajo (escalar hacia abajo) o una combinación de las mismas. Selecciona la recompensa a obtenerse por la ejecución de cada mecánica que ha sido seleccionada.Introduce “Nombre del juego”, “Capacidad” y “Cantidad de episodios a jugar”. Presiona el botón “Guardar”. | |
| 4. | | Guarda la configuración de las mecánicas. |
| 5. | Presiona el botón “Empezar a jugar”. | |
| 6. | | Comprueba la(s) mecánicas seleccionada(s). |
| **Flujos alternos** | | |
| **2a. Listado de mecánicas vacío** | | |
| **Actor** | | **Sistema** |
| 2. | | Carga una pantalla en blanco sin mecánicas. |

## 2.5 Modelo de despliegue

Los diagramas de despliegue se utilizan para visualizar los procesadores/nodos/dispositivos de *hardware* de un sistema, los enlaces de

comunicación entre ellos y la colocación de los archivos de *software* en ese *hardware* (Creately, 2022). A continuación, se muestra el modelo de despliegue de la propuesta de solución:


*Figura 8 Diagrama de despliegue. Fuente elaboración propia*

**Descripción del diagrama de despliegue**

- **PC_Cliente:** nodo que representa el equipo o terminal dónde se ejecuta el sistema.

**Conclusiones parciales**

- El análisis del algoritmo clásico DQN determinó que el mismo no resulta suficiente para desarrollar el sistema del agente, lo que conllevó al desarrollo de la variación DDQN con modificaciones en el mecanismo que suministra estados.
- El Modelo de dominio brindó una visión general de los conceptos más importantes del dominio del problema.
- La definición de los requisitos funcionales que luego fueron agrupados en Casos de Uso del Sistema posibilitó obtener la descripción detallada de las funcionalidades que el sistema debe cumplir.
- La recopilación de los requisitos no funcionales proporcionó una visión de las cualidades y características que debe poseer el sistema.

- Con la descripción de los Casos de Uso del Sistema fue posible conocer el flujo de trabajo que debe seguir el sistema, conociendo su funcionamiento general y las respuestas al usuario.
- Se elaboró el modelo de despliegue que posibilitó describir la distribución física del sistema para conocer la distribución de las funcionalidades entre los nodos de cómputo.

# Capítulo III. Implementación y pruebas

Las reglas que comprenden lo relacionado al código fuente de un *software* son importantes para establecer su utilización y asegurar que durante el desarrollo del mismo cualquier programador pueda entender el código y trabajar de forma coordinada (Atiés, 2017). En este capítulo se exponen los estándares de codificación que se utilizan en la implementación del sistema, además se realizan las pruebas al sistema para que cumpla con los requisitos funcionales establecidos y obtener un producto de *software* de calidad.

## 3.1 Estándar de codificación

Con el objetivo de mantener el código legible y facilitar cambios en caso de ser necesario se adoptaron diferentes estándares de codificación; seguidamente se muestra un listado con los estándares empleados para el desarrollo del sistema.

- Minúscula y guión bajo para el nombre de variables y métodos.
- Primera letra con mayúscula y el resto minúscula para el nombre de las clases.
- Mayúscula para la declaración de constantes.
- Espacio después de cada estructura de control (*if, foreach, while, for*, etc).
- Máximo de 79 caracteres en cada línea.
- Importaciones en líneas separadas.
- Indentación con tabulador establecido a 4 espacios.

## 3.2 Pruebas de software

La metodología AUP-UCI define ocho disciplinas, refiriéndose las tres últimas a pruebas de *software*, estas son: Pruebas Internas, de Liberación y Aceptación. Las pruebas al producto de *software* se realizan con el objetivo de verificar el proceso de desarrollo y que el sistema cumpla con los requerimientos previamente definidos.

### 3.2.1 Pruebas unitarias

Las pruebas unitarias de *software* pueden definirse como un mecanismo de comprobación del funcionamiento de las unidades de menor tamaño de un programa o aplicación específico. Este tipo de pruebas son de vital importancia para la

detección de errores, permiten garantizar que cada una de las unidades de *software* analizadas se encuentran funcionando de la forma que deberían e independientemente.

Para la realización de las pruebas unitarias se implementa en el editor de texto Visual Studio Code la clase Test y en la misma los métodos de prueba. En la Figura 9 se muestra un ejemplo del código implementado para comprobar el correcto funcionamiento de las redes neuronales y que la salida de la red corresponda con los valores esperados. En la Figura 10 se muestra el resultado de la prueba realizada que fue favorable porque se obtiene en todas las iteraciones un valor esperado.

Valor esperado consiste en un único identificador de la acción que debe realizar el agente de todas las acciones posibles.

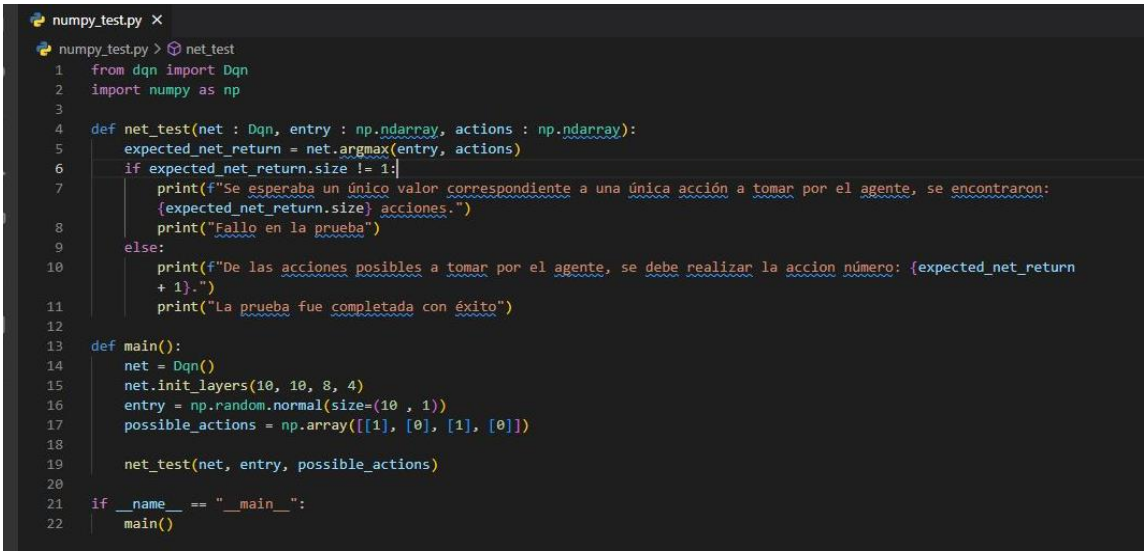



*Figura 9 Ejemplo del código de las pruebas unitarias.*

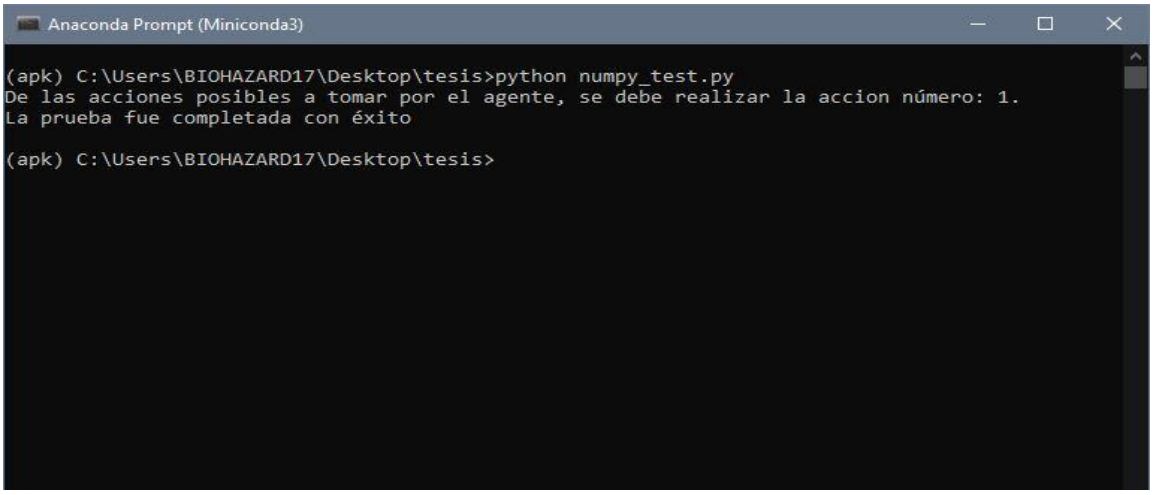



*Figura 10 Resultado de pruebas unitarias*

En la Figura 11 se evidencia el código implementado para comprobar el funcionamiento del modelo visual.

```
def test_visual_model(self):
    file_names = self.get_file_names()
    self.predictor_threshold= self.threshold_dsb.value()  # type:
    ignore
    predictor = Predictor(
        self.cfg_save_path,
        self.output_dir,
        self.predictor_threshold
    )

    predictor.plot_examples(file_names)
```

```
30
31  def plot_examples(self, images):
32
33      for file in images:
34          im = cv.imread(file)
35          predictions = self.predictor(im)
36          v= Visualizer(im[:,:,::-1], metadata={},
            instance_mode=ColorMode.SEGMENTATION)
37          output = v.draw_instance_predictions
            (predictions["instances"].to("cpu"))
38          cv.imshow("Result", output.get_image()[:,:,
            ::-1])
39          key = cv.waitKey(3) & 0xFF
40          if key == ord('q'):
41              break
```

*Figura 11 Código para comprobar modelo visual. Fuente: elaboración propia.*

Como resultado de la comprobación del funcionamiento del modelo visual se obtiene una imagen donde se visualiza los resultados del componente de detección de objetos en el formato clase del objeto y nivel de precisión. Ver Figura 12.


*Figura 12 Resultados de la prueba al componente de detección de objetos. Fuente: elaboración propia.*

### 3.2.2 Pruebas de aceptación

Las pruebas de aceptación se realizan para verificar que el sistema funciona y tiene un rendimiento conforme a las especificaciones, permiten obtener los primeros comentarios de un número limitado de usuarios finales y clientes; se crean a partir de los Casos de Uso (IBM, 2021). Para validar la solución implementada se definen Casos de Prueba; se muestra a continuación la descripción del Caso de Prueba "Evaluar la comprobación del desplazamiento", el resto se encuentra en el Anexo III:

**Caso de prueba: Evaluar la comprobación del desplazamiento**

**Caso de Uso asociado: Comprobar desplazamiento**

*Tabla 4 Caso de prueba "Evaluar la comprobación del desplazamiento"*

| ID Escenario | Escenario | Nombre del juego | Capacidad | Cantidad de episodios a jugar | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|---|---|
| 1.1 | Entrada de datos válidos | V<br>Super Mario | V<br>400 | V<br>1 | El sistema permite al usuario comprobar el | Resultado satisfactorio ya que el sistema posibilita al |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | desplazamiento. | usuario comprobar el desplazamiento. |
| 1.2 | Ausencia de campos llenos de datos válidos | I<br>Nulo | V<br>400 | V<br>1 | El sistema no permite comprobar el desplazamiento. | Resultado satisfactorio ya que el Sistema Operativo bloquea la aplicación por errores internos del mismo y no permite la comprobación del salto. |
| 1.3 | Entrada de datos incorrectos | I<br>123 | V<br>400 | V<br>1 | El sistema no permite comprobar el desplazamiento. | Resultado satisfactorio ya que el Sistema Operativo bloquea la aplicación por errores internos del mismo y no permite la comprobación del salto. |

Las no conformidades (NC) no significativas detectadas fueron principalmente errores ortográficos. La no conformidad más significativa detectada es que el reporte se genera con imprecisiones en los resultados. El equipo de desarrollo analiza las NC detectadas y las soluciona para que el sistema cumpla con totalidad su objetivo.

Estos escenarios de prueba fueron verificados por el cliente en una prueba al sistema donde este comprobó que los principales requisitos funcionales fueron cumplidos y que el sistema cumple con las necesidades del cliente, reflejado en el aval del cliente, ver Anexo I.

## 3.3 Conclusiones parciales

- La utilización de estándares de codificación garantizó el entendimiento y la legibilidad del código y organización en la implementación.
- Con la realización de las pruebas unitarias se comprobó el correcto funcionamiento de cada unidad de código y para fomentar la conformidad del cliente respecto al producto se efectuaron las pruebas de aceptación.
- Las pruebas que se aplicaron arrojaron no conformidades que fueron corregidas para ofrecer un sistema con completitud en cada una de sus funciones.
- La realización de las pruebas permitió comprobar demoras en el mecanismo de toma de decisiones lo que provoca que la ejecución de las pruebas sobre el videojuego sea más prolongada.
- La demora de las predicciones señaló la necesidad de la utilización de más capacidad de procesamiento a nivel de hardware, específicamente a nivel de CPU y tarjeta gráfica dedicada.

# CONCLUSIONES GENERALES

- A partir del estudio del estado del arte realizado se detectó que no existe a nivel mundial una herramienta que realice de forma automática las pruebas a los videojuegos y se pudo apreciar la necesidad de una herramienta de este tipo en el Centro de Tecnologías Interactivas.
- El análisis de los conceptos más destacados de la jugabilidad y de sus facetas y atributos posibilitó cuantificarla y medirla a través de las mecánicas contempladas en la faceta de jugabilidad mecánica; tomando en cuenta las mecánicas más significativas en los videojuegos del género plataformas.
- La selección de la metodología AUP-UCI y las herramientas seleccionadas afianzaron el éxito en el desarrollo del sistema y se generaron los artefactos necesarios para ser consultados en caso de realizar una extensión del sistema para otros géneros de videojuegos.
- Se obtuvo como resultado de la investigación un sistema que de forma automática prueba la jugabilidad en videojuegos del género plataformas midiendo las mecánicas con el uso de un agente de Inteligencia Artificial.
- La realización de las pruebas unitarias y de aceptación permitió la obtención de un sistema con calidad que cumple con las expectativas del cliente, pero también determinó que se necesita de mayor procesamiento para realizar el proceso de forma más rápida.

# RECOMENDACIONES

- Continuar el desarrollo del sistema y hacerlo adaptable a otros géneros de videojuegos.
- Extender el uso del sistema a otros centros de desarrollo de videojuegos en el país.
- Comprobar otras facetas de la jugabilidad.

**Referencias bibliográficas**


1. Aizprua, S., Ortega, A., & Chong, L. V. (05 de mayo de 2019). Calidad del software,una perspectiva continua. *CENTROS:Revista Científica Universitaria, 8*(2). http://portal.amelica.org/ameli/jatsRepo/228/228986011/index.html
2. Alejo García-Naveira Vaamonde, M. J. (30 de octubre de 2018). Beneficios cognitivos, psicológicos y personales del uso de los videojuegos y esports: una revisión. *Revista de Psicología Aplicada al Deporte y el Ejercicio Físico, 3*(2), págs. 1-14. https://doi.org/https://doi.org/105093/rpadef2018a15
3. Atiés, P. P. (2017). *Herramienta para la evaluación de videojuegos.* Universidad de las Ciencias Informáticas. La Habana: Universidad de las Ciencias Informáticas.Facultad 4. https://repositorio.uci.cu/jspui/handle/123456789/8130
4. *Becas Santander*. (09 de abril de 2021). https://www.becas-santander.com/es/blog/python-que-es.html
5. Calisoft. (2017). *Requisitos de la calidad para sistemas informáticos y productos de software.*
6. Callejas-Cuervo, M., Alarcón-Aldana, A. C., & Álvarez-Carreño, A. M. (08 de octubre de 2016). Modelos de calidad del software, un estado del arte. (U. L. Cali, Ed.) *Entramado, 13*, págs. 236-250. https://doi.org/https://doi.org/10.18041/entramado.2017v13n1.25125
7. Creately. (18 de octubre de 2022). *Creately*. https://creately.com/blog/es/diagramas/tutorial-de-diagrama-de-despliegue/
8. Cruz, A. A. (2017). *Arquitectura de software para videojuegos desarrollados sobre el motor de juego Unity 3D.* Trabajo de Diploma, Universidad de las Ciencias Informáticas, La Habana. https://repositorio.uci.cu/jspui/handle/123456789/9382

9. devCamp. (2020). *devCamp*. https://devcamp.es/que-es-libreria-programacion/
10. Foundation, P. S. (2022). *Python*. https://docs.python.org/es/3/library/tkinter.html
11. Girshick, R. (2015). Fast r-cnn. *Proceedings of the IEEE international conference on computer vision*, (págs. 1440-1448).
12. Hardik, D. (13 de febrero de 2021). *Analitics Vidhya*. https://www.analyticsvidhya.com/blog/2021/02/understending-the-bellman-optimality-equation-in-reinforcement-learning
13. IBM. (2021). *IBM*. https://www.ibm.com/es-es/topics/software-testing
14. Institucional, D. d. (01 de diciembre de 2020). *Desoft*. https://www.desoft.cu/es/noticias/261
15. IONOS, D. G. (01 de junio de 2021). *Digital Guide IONOS*. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/las-mejores-herramientas-uml/
16. *ISO 25000*. (2022). https://iso25000.com/index.php/normas-iso-25000/iso-25010
17. *ISO25000 Calidad del producto software*. (2017). Retrieved 20 de 5 de 2022, from http://iso25000.com/index.php/normas-iso-25000/iso-25010?limit=3&limitstart=0.
18. Maestre, F. L. (2015). *DocPlayer*. https://docplayer.es/4837503-De-la-usabilidad-a-la-jugabilidad-diseno-de-videojuegos-centrado-en-el-jugador.html
19. Maida, E. G., & Pacienzia, J. (2015). *Metodologías de desarrollo de software.* Pontificia Universidad Católica Argentina Santa Maria de los Buenos Aires. Buenos Aires: Biblioteca Central ¨San Benito Abad¨. https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf
20. Markov, M. L. (1990). *Hhandbooks in operations research and managment science* (Vol. 2).

21. Marthis. (18 de mayo de 2022). Diario de friki: https://diariodefriki.com/2022/05/18/resena-super-mario-bros/
22. Mesa, J. (17 de septiembre de 2019). *La Voz Escolar*. https://lavozescolar.opennemas.com/articulo/humor/super-mario-bross/20190917053903000959.html
23. Miguel, P. D. (25 de enero de 2019). *33 bits*. https://portal.33bits.net/los-plataformas-bidimensionales/
24. Monteagudo, Y. I. (2018). *Proceso de pruebas en el desarrollo de videojuegos.* Universidad de las Ciencias Informáticas. La Habana: Universidad de las Ciencias Informáticas. https://repositorio.uci.cu/jspui/handle/123456789/7929
25. Montesino, J. A. (2022). *Instituto de Ingeniería del Conocimiento*. www.icc.uam.es/innovacion/alphazero-y-el-go/
26. Montesino, J. A. (2022). *Instituto de Ingeniería del Conocimiento*. www.icc.uam.es/innovacion/openia-five-juego-dota2/
27. Mora, J. L. (2017). *Modelamiento de sistemas de software Escuela de Ingeniería Informática Pontificia Universidad Católica de Valparaíso.* Pontificia Universidad Católica de Valparaíso.
28. Morales, D. B., Borrell, J. B., & Armas, L. J. (junio de 2019). Aplicación móvil para el análisis de la información captada en SIGEv3.0. (U. d. Informáticas, Ed.) *Serie Científica de la Universidad de las Ciencias Informáticas, 12*(6), págs. 55-71. http://publicaciones.uci.cu
29. Naidu, V. (2018). G-PY: A Game Playing AI to Simulate Real Time Close-Quarter Firefights Using 3D First-Person-Shooter Games. *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA*, (págs. 734-739). https://ieeexplore.ieee.org/xpl/conhome/8466240/proceeding
30. Palmer, M. (2015). *Read the docs*. https://pynput.readthedocs.io/en/latest/

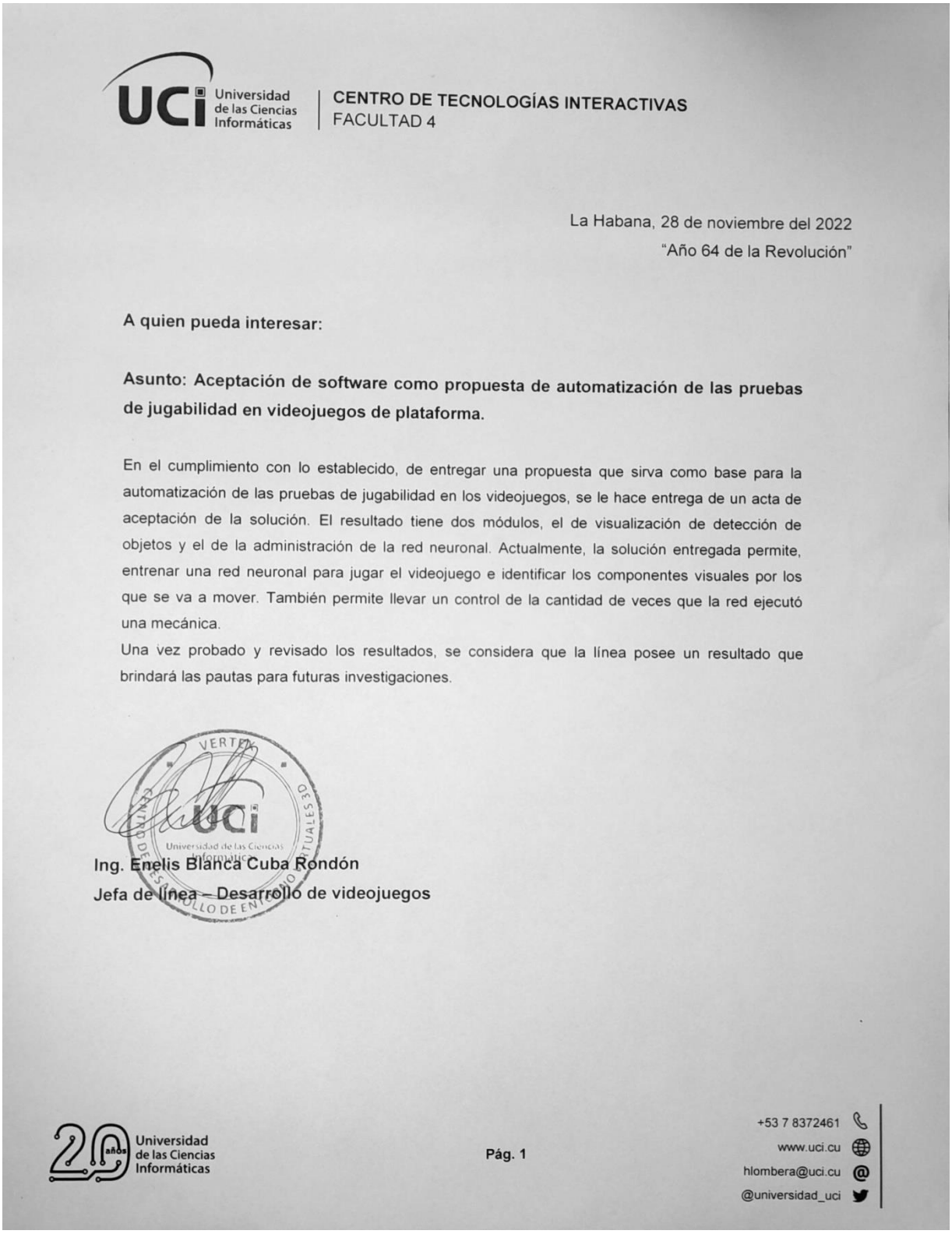
31. Politowski, C., Guéhéneuc, Y.-G., & Petrillo, F. (25 de febrero de 2022). *Towards Automated Video Game Testing:Still a Long Way to Go.* Pittsburgh. https://arxiv.org/abs/2202.1777
32. *Real Academia Española*. (2021). https://dle.rae.es/videojuego
33. *Real Academia Española*. (2022). htttps://dle.rae.es/jugabilidad
34. *Real Academia Española*. (2022). https://dle.rae.es/locomoción
35. Regueiferos, H. L. (11 de octubre de 2012). Jugabilidad, importancia en videojuegos. *Revista Tino*(30). https://revista.jovenclub.cu/jugabilidad-importancia-en-videojuegos/
36. *Reinforcement Learning-Developing Intelligent Agents*. (16 de agosto de 2022).
37. Remedios, Y. M. (04 de junio de 2019). Super Mario Bros en la historia. *Joven Club*. https://revista.jovenclub.cu/super-mario-bros-en-la-historia/
38. Rodriguez, D. (09 de febrero de 2022). *ConceptoDefinición*. https://conceptodefinicion.de/videojuegos/.
39. Rodríguez, H. (27 de abril de 2021). *Crehana*. https://www.crehana.com/blog/transformacion-digital/que-es-opencv/
40. Rodríguez, L. P. (3 de septiembre de 2021). ¿Cómo se hace un videojuego? *Somos Jóvenes*. https://medium.com/somos-j%C3%B3venes/c%C3%B3mo-se-hace-un-videojuego-ef244a7d7e4d
41. Rubio, P. V. (2015). *DocPlayer*. https://docplayer.es/7716422-Ingenieria-del-software-i.html
42. Russell, B. C. (2018). LabelMe: a database and web-based tool for image annotation. *International journal of computer vision, 77*(1), págs. 157-173.
43. Sánchez, J. L. (Mayo de 2010). Caracterización de la experiencia del jugador en videojuegos.

44. Sánchez, J. L. (05 de julio de 2022). *ResearchGate*. https://www.researchgate.net/publication/47441039_Jugabilidad_Caracterizacion_de_la_experiencia_del_jugador_en_videojuegos
45. Sevilla, B. d. (2022). *Expobus*. (U. d. Sevilla, Editor) https://expobus.us.es/s/level/page/generos
46. Silver, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 484-489.
47. Solano, E. A. (2022). *Panda Cinemático*. https://pandacinematico.com/videojuego-definicion-caracteristicas/
48. Sommerville, I. (2015). https://www.uv.mx/personal/fcastaneda/files/2015/08/F_Capitulo_5_Requerimientos_del_software.pdf
49. Stuber, C. (17 de marzo de 2021). *Tecnología Android*. https://tecnologiandroid.com/que-es-la-calidad-del-software/
50. Tecnoempleo. (5 de mayo de 2020). *Blog Tecnoempleo*. https://www.google.com/amp/s/blog.tecnoempleo.com/candidatos/2020/15420/las-7-etapas-del-desarrollo-de-un-videojuego/%3famp=1
51. Toledo, Á. (30 de enero de 2022). *Uptodown*. htttps://super-cow.uptodown.com/windows
52. Toro, D. S. (2016). *Paquete de mecanicas para el desarrollo de videojuegos de tipo estrategia tactica sobre UNITY 3D.* Universidad de las Ciencias Informáticas. La Habana: Universidad de las Ciencias Informáticas.Facultad-5. https://repositorio.uci.cu/jspui/handle/123456789/7864
53. Toro, L. (15 de septiembre de 2017). *Desde Linux* . https://blog.desdelinux.net/ciencia-de-datos-con-python/

54. Tovar, L. A., & Rodríguez, A. V. (2 de noviembre de 2018). Propuesta de Diseño de Pruebas para Videojuegos Aplicados En la Salud. *Revista Daena*.
55. Unity. (17 de diciembre de 2019). *Unity*. https://unity.com/difference-between-2D-and-3D-games
56. Velazco, R. (26 de mayo de 2021). *SoftZone*. https://www.softzone.es/programas/utilidades/visual-studio-code/
57. Wu, Y., Massa, A. K., & Girshick, W.-Y. L. (2019). *GitHub*. https://github.com/facebookresearch/detectron2

# Anexos

## Anexo I: Carta de aceptación del cliente.



Universidad de las Ciencias Informáticas | CENTRO DE TECNOLOGÍAS INTERACTIVAS
FACULTAD 4

La Habana, 28 de noviembre del 2022
"Año 64 de la Revolución"

**A quien pueda interesar:**

**Asunto: Aceptación de software como propuesta de automatización de las pruebas de jugabilidad en videojuegos de plataforma.**

En el cumplimiento con lo establecido, de entregar una propuesta que sirva como base para la automatización de las pruebas de jugabilidad en los videojuegos, se le hace entrega de un acta de aceptación de la solución. El resultado tiene dos módulos, el de visualización de detección de objetos y el de la administración de la red neuronal. Actualmente, la solución entregada permite, entrenar una red neuronal para jugar el videojuego e identificar los componentes visuales por los que se va a mover. También permite llevar un control de la cantidad de veces que la red ejecutó una mecánica.

Una vez probado y revisado los resultados, se considera que la línea posee un resultado que brindará las pautas para futuras investigaciones.

**Ing. Enelis Blanca Cuba Rondón**
**Jefa de línea – Desarrollo de videojuegos**

Universidad de las Ciencias Informáticas

**Pág. 1**

+53 7 8372461
www.uci.cu
hlombera@uci.cu
@universidad_uci

## ANEXO II: Descripción de los Casos de Uso del sistema.

### CU1. Configurar modelo visual.

| | |
|---|---|
| **Objetivo** | Permitir entrenar un modelo visual, usar modelo existente, restablecer valores del modelo visual y probar modelo visual. |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. |
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea: entrenar un modelo visual, usar modelo existente, restablecer valores del modelo visual y probar modelo visual. |
| **Complejidad** | Media |
| **Prioridad** | Alta |
| **Precondiciones** | El sistema debe estar en ejecución. |
| **Postcondiciones** | Se entrenó un nuevo modelo visual, se usó modelo existente, se restablecieron los valores del modelo visual y/o fue probado el modelo visual. |

| **Flujo de eventos** | | |
|---|---|---|
| **Flujo básico “Configurar modelo visual”** | | |
| **Actor** | | **Sistema** |
| 1. | Desea entrenar un nuevo modelo visual usar un modelo existente, restablecer los valores de modelo visual y probar modelo visual o una combinación de estas opciones. | |
| 2. | | Muestra una pantalla con las opciones de entrenar un nuevo modelo visual o usar un modelo existente. |
| **Sección 1: “Entrenar nuevo modelo visual”** | | |

| **Flujo básico “Entrenar nuevo modelo visual”** | | |
|---|---|---|
| **Actor** | | **Sistema** |
| 1. | Selecciona opción “Entrenar nuevo modelo” | |
| 2. | | Habilita los siguientes campos a introducir:<br>• Ruta hacia la carpeta de imágenes de entrenamiento,<br>• Ruta hacia el archivo de anotaciones de entrenamiento,<br>• Nombre del set de datos de entrenamiento,<br>• Ruta del archivo de anotaciones de prueba,<br>• Nombre del set de datos de prueba,<br>• Ruta del archivo de configuración del modelo visual,<br>• Ruta del directorio de salida del modelo visual,<br>• Dispositivo de aceleración,<br>• Iteraciones de entrenamiento. |

| | | |
|---|---|---|
| | | Y el botón “Entrenar”. |
| 3. | Introduce los datos y presiona el botón “Entrenar”. | |
| 4. | | Verifica que todos los campos estén llenos y que los datos introducidos estén correctos. Realiza el entrenamiento del modelo visual. |
| **Sección 2: “Usar modelo existente”** | | |
| **Flujo básico “Usar modelo existente”** | | |
| **Actor** | | **Sistema** |
| 1. | Selecciona la opción “Usar modelo existente”. | |
| 2. | | Habilita el campo de búsqueda:<br>• Buscar modelo existente.<br>Y el botón “Entrenar”. |
| 3. | Presiona el botón “Entrenar”. | |
| 4. | | Verifica que todos los campos estén llenos y que los datos introducidos estén correctos. Realiza el entrenamiento del modelo visual. |
| **Sección 3: “Restablecer valores del modelo visual”** | | |

| **Flujo básico “Restablecer valores del modelo visual”** | | |
|---|---|---|
| **Actor** | | **Sistema** |
| 1. | Presiona el botón “Restablecer valores”. | |
| 2. | | Restablece los valores a los que tenía el modelo anteriormente. |
| **Sección 4: “Probar modelo visual”** | | |
| **Flujo básico “Probar modelo visual”** | | |
| **Actor** | | **Sistema** |
| 1. | Presiona el botón “Probar”. | |
| 2. | | Muestra las imágenes de entrenamiento con el porcentaje de predicción en cada uno de los objetos. |

**CU2. Configurar agente.**

| | |
|---|---|
| **Objetivo** | Permitir crear un nuevo agente, restablecer valores del nuevo agente, cargar agente desde configuración existente. |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. |
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea: crear un nuevo agente, restablecer valores del nuevo agente, cargar agente desde configuración existente. |
| **Complejidad** | Media |
| **Prioridad** | Alta |
| **Precondiciones** | El sistema debe estar en ejecución. |

<table>
<tr><td colspan="2">Postcondiciones</td><td colspan="2">Se creó un nuevo agente, se restablecieron los valores del agente valores, se cargó un agente desde configuración existente.</td></tr>
<tr><td colspan="4">Flujo de eventos</td></tr>
<tr><td colspan="4">Flujo básico “Configurar agente”</td></tr>
<tr><td colspan="2">Actor</td><td colspan="2">Sistema</td></tr>
<tr><td>1.</td><td>Desea crear un nuevo agente, restablecer valores del nuevo agente, cargar agente desde configuración existente.</td><td colspan="2"></td></tr>
<tr><td>2.</td><td></td><td colspan="2">Muestra una pantalla con las opciones: crear nuevo agente y cargar desde configuración existente.</td></tr>
<tr><td colspan="4">Sección 1: “Cargar nuevo agente”</td></tr>
<tr><td colspan="4">Flujo básico “Cargar nuevo agente”</td></tr>
<tr><td colspan="2">Actor</td><td colspan="2">Sistema</td></tr>
<tr><td>1.</td><td>Selecciona opción “Cargar nuevo agente”</td><td colspan="2"></td></tr>
<tr><td>2.</td><td></td><td colspan="2">Habilita los siguientes campos a introducir:<br>• Factor de aprendizaje,<br>• Factor de descuento,<br>• Inicializador de pesos de la red.<br>Y el botón “Entrenar”.</td></tr>
<tr><td>3.</td><td>Introduce los datos y presiona el botón “Entrenar”.</td><td colspan="2"></td></tr>
</table>

| | | |
|---|---|---|
| 4. | | Verifica que todos los campos estén llenos y que los datos introducidos estén correctos. Crea nuevo agente. |
| **Sección 2: “Restablecer valores del nuevo agente”** | | |
| **Flujo básico “Restablecer valores del nuevo agente”** | | |
| **Actor** | | **Sistema** |
| 1. | Presiona el botón “Restablecer”. | |
| 2. | | Restablece los valores del agente a los que tenía anteriormente. |
| **Sección 3: “Cargar agente desde configuración existente”** | | |
| **Flujo básico “Cargar agente desde configuración existente”** | | |
| **Actor** | | **Sistema** |
| 1. | Selecciona la opción “Cargar desde configuración existente”. | |
| 2. | | Habilita el campo de búsqueda:<br>• Ruta hacia archivo de configuración. |

**CU4. Comprobar salto.**

| | | |
|---|---|---|
| **Objetivo** | Permitir comprobar las mecánicas: realizar salto simple, realizar salto doble (salto de larga duración), realizar salto en diagonal derecha, realizar salto en diagonal izquierda. | |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. | |
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea comprobar las mecánicas: realizar salto simple, realizar salto doble (salto de larga duración), realizar salto en diagonal derecha, realizar salto en diagonal izquierda o una combinación de las mismas. | |
| **Complejidad** | Alta | |
| **Prioridad** | Alta | |
| **Precondiciones** | El videojuego al que se le desea comprobar las mecánicas debe estar en ejecución al igual que el sistema. | |
| **Postcondiciones** | Se comprobaron las mecánicas: realizar salto simple, realizar salto doble (salto de larga duración), realizar salto en diagonal derecha, realizar salto en diagonal izquierda o una combinación de las mismas. | |
| **Flujo de eventos** | | |
| **Flujo básico “Comprobar salto”** | | |
| **Actor** | | **Sistema** |
| 1. | Desea comprobar las mecánicas: realizar salto simple, realizar salto doble (salto de larga duración), realizar salto en diagonal derecha, realizar salto en diagonal izquierda o una combinación de las mismas. | |
| 2. | | Muestra un listado con todas las mecánicas. |
| 3. | Selecciona las mecánicas: realizar salto simple, realizar salto doble (salto de | |

| | | |
|---|---|---|
| | larga duración), realizar salto en diagonal derecha, realizar salto en diagonal izquierda o una combinación de las mismas. Selecciona la recompensa a obtenerse por la ejecución de cada mecánica que ha sido seleccionada. Introduce “Nombre del juego”, “Capacidad” y “Cantidad de episodios a jugar”. Presiona el botón “Guardar”. | |
| 5. | | Guarda la configuración de las mecánicas. |
| 6. | Presiona el botón “Empezar a jugar”. | |
| 7. | | Comprueba la(s) mecánicas seleccionada(s). |
| **Flujos alternos** | | |
| **2a. Listado de mecánicas vacío** | | |
| **Actor** | | **Sistema** |
| 1. | | Carga una pantalla en blanco sin mecánicas. |

**CU5. Comprobar la mecánica interactuar.**

| | |
|---|---|
| **Objetivo** | Permitir comprobar la interacción del personaje principal con el entorno. |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. |
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea comprobar la mecánica: interactuar. |
| **Complejidad** | Media |
| **Prioridad** | Alta |

| | | |
|---|---|---|
| **Precondiciones** | El videojuego al que se le desea comprobar las mecánicas debe estar en ejecución al igual que el sistema. | |
| **Postcondiciones** | Se comprobó la mecánica: interactuar. | |
| **Flujo de eventos** | | |
| **Flujo básico “Comprobar la mecánica interactuar”** | | |
| **Actor** | | **Sistema** |
| 1. | Desea comprobar la mecánica: interactuar. | |
| 2. | | Muestra un listado con todas las mecánicas. |
| 3. | Selecciona la mecánica: interactuar y la recompensa a obtener por ejecutar la acción. Introduce “Nombre del juego”, “Capacidad” y “Cantidad de episodios a jugar”. Presiona el botón “Guardar”. | |
| 5. | | Guarda la configuración. |
| 6. | Presiona el botón “Empezar a jugar”. | |
| 7. | | Comprueba la mecánica seleccionada. |
| **Flujos alternos** | | |
| **2a. Listado de mecánicas vacío** | | |
| **Actor** | | **Sistema** |
| 1. | | Carga una pantalla en blanco sin mecánicas. |

**CU6. Comprobar la mecánica atacar.**

| | |
|---|---|
| **Objetivo** | Permitir comprobar las mecánicas :atacar y atacar con mouse. |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. |

| | | |
|---|---|---|
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea comprobar las mecánicas: atacar y atacar con mouse. | |
| **Complejidad** | Alta | |
| **Prioridad** | Alta | |
| **Precondiciones** | El videojuego al que se le desea comprobar las mecánicas debe estar en ejecución al igual que el sistema. | |
| **Postcondiciones** | Se comprobaron las mecánicas: atacar, atacar con mouse. | |
| **Flujo de eventos** | | |
| **Flujo básico “Comprobar la mecánica atacar”** | | |
| **Actor** | | **Sistema** |
| 1. | Desea comprobar las mecánicas: atacar y/o atacar con mouse. | |
| 2. | | Muestra un listado con todas las mecánicas. |
| 3. | Selecciona las mecánicas: atacar y/o atacar con mouse y la recompensa a obtener por ejecutar la acción. Introduce “Nombre del juego”, “Capacidad” y “Cantidad de episodios a jugar”. Presiona el botón “Guardar”. | |
| 5. | | Guarda la configuración. |
| 6. | Presiona el botón “Empezar a jugar”. | |
| 7. | | Comprueba la mecánica seleccionada. |
| **Flujos alternos** | | |
| **2a. Listado de mecánicas vacío** | | |
| **Actor** | | **Sistema** |

| | | |
|---|---|---|
| 1. | | Carga una pantalla en blanco sin mecánicas. |

**CU7. Comprobar la mecánica esquivar.**

| | | |
|---|---|---|
| **Objetivo** | Permitir comprobar la mecánica: esquivar. | |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. | |
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea comprobar la mecánica: esquivar. | |
| **Complejidad** | Media | |
| **Prioridad** | Alta | |
| **Precondiciones** | El videojuego al que se le desea comprobar las mecánicas debe estar en ejecución al igual que el sistema. | |
| **Postcondiciones** | Se comprobó la mecánica: esquivar. | |
| **Flujo de eventos** | | |
| **Flujo básico “Comprobar la mecánica esquivar”** | | |
| **Actor** | | **Sistema** |
| 1. | Desea comprobar la mecánica: esquivar. | |
| 2. | | Muestra un listado con todas las mecánicas. |
| 3. | Selecciona la mecánica: esquivar y la recompensa a obtener por la ejecución de la acción. Introduce “Nombre del juego”, “Capacidad” y “Cantidad de episodios a jugar”. Presiona el botón “Guardar”. | |
| 5. | | Guarda la configuración. |

| | | |
|---|---|---|
| 6. | | Comprueba la mecánica seleccionada. |
| **Flujos alternos** | | |
| **2a. Listado de mecánicas vacío** | | |
| **Actor** | | **Sistema** |
| 1 | | Carga una pantalla en blanco sin mecánicas. |

**CU8. Comprobar la mecánica agacharse.**

| | | |
|---|---|---|
| **Objetivo** | Permitir comprobar la mecánica: agacharse. | |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. | |
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea comprobar la mecánica: agacharse. | |
| **Complejidad** | Media | |
| **Prioridad** | Alta | |
| **Precondiciones** | El videojuego al que se le desea comprobar las mecánicas debe estar en ejecución al igual que el sistema. | |
| **Postcondiciones** | Se comprobó la mecánica: agacharse. | |
| **Flujo de eventos** | | |
| **Flujo básico “Comprobar la mecánica agacharse”** | | |
| **Actor** | | **Sistema** |
| 1. | Desea comprobar la mecánica: agacharse. | |
| 2. | | Muestra un listado con todas las mecánicas. |
| 3. | Selecciona la mecánica: agacharse y la recompensa a obtener por la ejecución de la acción. Introduce “Nombre del | |

| | | |
|---|---|---|
| | juego", "Capacidad" y "Cantidad de episodios a jugar". Presiona el botón "Guardar". | |
| 5. | | Guarda la configuración. |
| 6. | | Comprueba la mecánica seleccionada. |
| **Flujos alternos** | | |
| **2a. Listado de mecánicas vacío** | | |
| **Actor** | | **Sistema** |
| 1 | | Carga una pantalla en blanco sin mecánicas. |

**CU9. Comprobar la mecánica morir.**

| | | |
|---|---|---|
| **Objetivo** | Permitir comprobar la mecánica: morir. | |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. | |
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea comprobar la mecánica: morir. | |
| **Complejidad** | Media | |
| **Prioridad** | Alta | |
| **Precondiciones** | El videojuego al que se le desea comprobar las mecánicas debe estar en ejecución al igual que el sistema. | |
| **Postcondiciones** | Se comprobó la mecánica: morir. | |
| **Flujo de eventos** | | |
| **Flujo básico "Comprobar la mecánica morir"** | | |
| **Actor** | | **Sistema** |
| 1. | Desea comprobar la mecánica: morir. | |

| | | |
|---|---|---|
| 2. | | Muestra un listado con todas las mecánicas. |
| 3. | Selecciona la mecánica: morir y la recompensa a obtener por la ejecución de la acción. Introduce “Nombre del juego”, “Capacidad” y “Cantidad de episodios a jugar”. Presiona el botón “Guardar”. | |
| 5. | | Guarda la configuración. Comprueba la mecánica seleccionada. |
| **Flujos alternos** | | |
| **2a. Listado de mecánicas vacío** | | |
| **Actor** | | **Sistema** |
| 1 | | Carga una pantalla en blanco sin mecánicas. |

**CU9. Obtener reporte.**

| | |
|---|---|
| **Objetivo** | Permitir obtener el reporte con el nivel de cumplimiento de las mecánicas comprobadas. |
| **Actores** | Equipo de desarrollo del Centro de Tecnologías Interactivas. |
| **Resumen** | El caso de uso se inicia cuando un miembro del equipo de desarrollo desea: obtener reporte. |
| **Complejidad** | Media |
| **Prioridad** | Alta |
| **Precondiciones** | El videojuego al que se le han comprobado las mecánicas debe estar en ejecución al igual que el sistema. |
| **Postcondiciones** | Se obtuvo un reporte. |

| **Flujo de eventos** | | |
|---|---|---|
| **Flujo básico “Obtener reporte”** | | |
| **Actor** | | **Sistema** |
| 1. | Desea obtener el reporte con el nivel de cumplimiento de las mecánicas comprobadas. | |
| 2. | | Muestra una pantalla con el botón “Obtener reporte”. |
| 3. | Presiona el botón “Obtener reporte”. | |
| 4. | | Emite un reporte con el nivel de cumplimiento de las mecánicas comprobadas. |
| 5. | Presiona el botón “Aceptar”. | |

**ANEXO III: Casos de Prueba.**

**Caso de prueba: Evaluar configuración del agente.**

**Caso de Uso asociado: Configurar agente.**

| ID EC | Escenario | Factor de aprendizaje | Factor de descuento | Inicializador de pesos de la red | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|---|---|
| 1.1 | Entrada de datos válidos | V<br>0.025 | V<br>0.001 | V<br>RELU | El sistema permite configurar el agente. | Resultado satisfactorio ya que el sistema posibilita la configuración del agente. |
| 1.2 | Ausencia de campos llenos de | I<br>Nulo | V<br>0.001 | V<br>1 | El sistema permite configurar el agente. | Resultado satisfactorio ya que el sistema posibilita |

| | | | | | | |
|---|---|---|---|---|---|---|
| | datos válidos | | | | | configurar el agente usando el factor de aprendizaje que tiene por defecto. |

**Caso de prueba: Evaluar configuración del modelo visual.**

**Caso de Uso asociado: Configurar modelo visual.**

Se definen las siguientes siglas para los campos:

ID Escenario (ID EC).

Escenario (EC).

Ruta hacia la carpeta de imágenes de entrenamiento (RCIE).

Ruta hacia el archivo de anotaciones de entrenamiento (RAAE).

Nombre del set de datos de entrenamiento (NSDE).

Ruta hacia la carpeta de imágenes de prueba (RCIP).

Ruta hacia el archivo de anotaciones de prueba (RAAP).

Nombre del set de datos de prueba (NSDP).

Ruta del archivo de configuración del modelo visual (RACMV).

Ruta del directorio de salida del modelo (RDSM).

Dispositivo de aceleración (DA).

Iteraciones de entrenamiento (IE).

| ID EC | Escenario | RCIE | RAAE | NSDE | RCIP | RAAP | NSDP | RACMV | RDSM | DA | IE | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1.1 | Entrada de datos válidos | V | V | V | V | V | V | V | V | V | V | El sistema permite entrenar el modelo visual. | Resultado satisfactorio. |
| 1.2 | Ausencia de campos llenos de datos válidos | I (nulo) | V | V | V | V | V | V | V | V | V | El sistema no permite entrenar el modelo visual. | Resultado satisfactorio. |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

**Caso de prueba: Evaluar la comprobación del salto.**

**Caso de Uso asociado: Comprobar salto.**

| ID EC | Escenario | Nombre del juego | Capacidad | Cantidad de episodios a jugar | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|---|---|
| 1.1 | Entrada de datos válidos | V<br>Super Mario | V | V<br>1 | El sistema permite al usuario comprobar el salto. | Resultado satisfactorio ya que el sistema posibilita al usuario comprobar el salto. |
| 1.2 | Ausencia de campos llenos de datos válidos | I<br>Nulo | V | V<br>1 | El sistema no permite comprobar el salto. | Resultado satisfactorio ya que el sistema no posibilita comprobar el salto. |
| 1.3 | Entrada de datos incorrectos | I<br>123 | V | V<br>1 | El sistema no permite comprobar el salto. | Resultado satisfactorio ya que el sistema no posibilita comprobar el salto. |

**Caso de prueba: Evaluar la comprobación de la mecánica interactuar.**

**Caso de Uso asociado: Comprobar la mecánica interactuar.**

| ID EC | Escenario | Nombre del juego | Capacidad | Cantidad de episodios a jugar | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|---|---|
| 1.1 | Entrada de datos válidos | V<br>Super Mario | V | V<br>1 | El sistema permite al usuario comprobar el | Resultado satisfactorio ya que el sistema posibilita al usuario |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | desplazamiento. | comprobar la mecánica interactuar. |
| 1.2 | Ausencia de campos llenos de datos válidos | I<br>Nulo | V | V<br>1 | El sistema no permite comprobar la mecánica interactuar. | Resultado satisfactorio ya que el sistema no posibilita comprobar la mecánica interactuar. |
| 1.3 | Entrada de datos incorrectos | I<br>123 | V | V<br>1 | El sistema no permite comprobar la mecánica interactuar. | Resultado satisfactorio ya que el sistema no posibilita comprobar la mecánica interactuar. |

**Caso de prueba: Evaluar la comprobación de la mecánica atacar.**

**Caso de Uso asociado: Comprobar la mecánica atacar.**

| ID EC | Escenario | Nombre del juego | Capacidad | Cantidad de episodios a jugar | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|---|---|
| 1.1 | Entrada de datos válidos | V<br>Super Mario | V | V<br>1 | El sistema permite al usuario comprobar la mecánica atacar. | Resultado satisfactorio ya que el sistema posibilita al usuario comprobar la mecánica atacar. |
| 1.2 | Ausencia de campos llenos de datos válidos | I<br>Nulo | V | V<br>1 | El sistema no permite comprobar la mecánica atacar. | Resultado satisfactorio ya que el sistema no posibilita comprobar la |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | mecánica atacar. |
| 1.3 | Entrada de datos incorrectos | I<br>123 | V | V<br>1 | El sistema no permite comprobar la mecánica atacar. | Resultado satisfactorio ya que el sistema no posibilita comprobar la mecánica atacar. |

**Caso de prueba: Evaluar la comprobación de la mecánica esquivar.**

**Caso de Uso asociado: Comprobar la mecánica esquivar.**

| ID EC | Escenario | Nombre del juego | Capacidad | Cantidad de episodios a jugar | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|---|---|
| 1.1 | Entrada de datos válidos | V<br>Super Mario | V | V<br>1 | El sistema permite al usuario comprobar la mecánica esquivar. | Resultado satisfactorio ya que el sistema posibilita al usuario comprobar la mecánica esquivar. |
| 1.2 | Ausencia de campos llenos de datos válidos | I<br>Nulo | V | V<br>1 | El sistema no permite comprobar la mecánica esquivar. | Resultado satisfactorio ya que el sistema no posibilita comprobar la mecánica esquivar. |
| 1.3 | Entrada de datos incorrectos | I<br>123 | V | V<br>1 | El sistema no permite comprobar la mecánica esquivar. | Resultado satisfactorio ya que el sistema no posibilita comprobar la |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | mecánica esquivar. |

**Caso de prueba: Evaluar la comprobación de la mecánica morir.**

**Caso de Uso asociado: Comprobar la mecánica morir.**

| ID EC | Escenario | Nombre del juego | Capacidad | Cantidad de episodios a jugar | Respuesta del sistema | Resultado de la prueba |
|---|---|---|---|---|---|---|
| 1.1 | Entrada de datos válidos | V<br>Super Mario | V | V<br>1 | El sistema permite al usuario comprobar la mecánica morir. | Resultado satisfactorio ya que el sistema posibilita al usuario comprobar la mecánica morir. |
| 1.2 | Ausencia de campos llenos de datos válidos | I<br>Nulo | V | V<br>1 | El sistema no permite comprobar la mecánica morir. | Resultado satisfactorio ya que el sistema no posibilita comprobar la mecánica morir. |
| 1.3 | Entrada de datos incorrectos | I<br>123 | V | V<br>1 | El sistema no permite comprobar la mecánica morir. | Resultado satisfactorio ya que el sistema no posibilita comprobar la mecánica morir. |

**Caso de prueba: Evaluar la obtención del reporte.**

**Caso de Uso asociado: Obtener reporte.**

Como en este caso no se tienen datos de entrada, se decide describir el Caso de Prueba en otro formato.

| | |
|---|---|
| Condiciones de ejecución | Ya se han comprobado las mecánicas seleccionadas. |
| Pasos de ejecución | Presionar el botón “Obtener reporte”. |
| Resultado esperado | Se emite un reporte con el nivel en que se han cumplido las mecánicas probadas. |