Universidad de las Ciencias Informáticas | FACULTAD 4

Universidad de Ciencias Informáticas

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título: Componente para la generación de niveles del videojuego Coliseum.

Autor:

Rolando Martin Álvarez

Tutores:

MSc. Rubén Alcolea Núñez

Ing. Anaili Pérez Piedra

La Habana, Diciembre 2022

FRASE

"El único autógrafo digno de un hombre es el que deja escrito con sus obras."

José Martí

DECLARACIÓN DE AUTORÍA

Declaro por este medio que yo: Rolando Martin Álvarez, con carné de identidad 97010106340, soy el autor principal del trabajo final de tesis de pregrado que se titula: “Componente para la generación de niveles del videojuego Coliseum”. El cual ha sido desarrollado como parte del trabajo del Centro de Tecnologías Interactivas, en la línea de Videojuegos de la Facultad 4. Autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Y para que así conste, se firma la presente declaración jurada de autoría en La Habana, a los 7 días del mes de diciembre del año 2022.

_____

Autor
Rolando Martin Álvarez

_____

Tutor
MSc. Rubén Alcolea Núñez

_____

Tutora
Ing. Anaili Pérez Piedra

DEDICATORIA

Dedicado a mis padres con mucho cariño.

A mi pequeño hermano menor

A mi familia que tanto me ha apoyado

Los sueños se cumplen, solo se necesita determinación.

AGRADECIMIENTOS

De una forma o de otra siempre es difícil hablar de agradecimientos en un trabajo como este, donde queda demostrado el esfuerzo, dedicación y sacrificio de años en la universidad.

Primeramente, quisiera agradecer a mi hermano menor, mi mejor amigo, por su cariño y comprensión.

A mi abuela Arlenys y a mi tía Yanela, mis segundas madres, quienes no me dejaron caer nunca, aconsejándome y guiándome a la meta final.

A mi tío Gilberto por sus consejos de la vida.

A mi primo Tonito por sus enseñanzas.

A mi primo Oscar, que siempre estuvo a mi lado.

A mi tío Arle y mi tía Victorita por su ayuda constante y preocupación.

Agradezco a todos los amigos que han hecho que la convivencia en la UCI sea inolvidable, Roylan, Carlos, Pichy, Fuñi, Jorguito, Elier estando en los buenos momentos como en los no tan buenos.

A May, por su amor y dedicación.

A mis tutores por ayudarme a alcanzar la tan esperada meta.

A Dayanis, Rosario y Nathali, aunque ya no nos veamos mucho, siempre las tengo presente.

A los demás compañeros de la universidad, y familiares que no pude mencionar, porque la lista sinceramente es extensa, de gente bonita y de recuerdos bellos que me llevo de la UCI.

RESUMEN

El desarrollo de videojuegos experimenta cada año altas tasas de crecimiento debido a los avances alcanzados en la computación y las tecnologías. Se ha convertido en uno de los mercados más rentables de la industria del entretenimiento. Nuestro país recién comienza a dar los primeros pasos en este sector, teniendo producciones de gran aceptación en la población. Tal es el caso del videojuego Coliseum que, desde su lanzamiento, acumula miles de descargas y ha creado gran expectativa en los usuarios. Actualmente, los desarrolladores del videojuego se sienten limitados en la incorporación de nuevos contenidos para satisfacer a sus usuarios, debido al tamaño que ha ido adquiriendo el compilado. Debido a esto, en la presente investigación se realizó un estudio de las técnicas de generación de contenido que implementan otros videojuegos. Para ello fueron utilizados métodos teóricos y empíricos que, de manera general, permitieron conocer el estado actual del videojuego, para luego realizar una propuesta de solución que responda a esas necesidades. Luego de ser implementada la propuesta de solución se obtuvo un componente para la creación de niveles, utilizando generación procedural para reducir el tamaño del compilado del videojuego. Se muestran los resultados de un conjunto de pruebas realizadas al componente, con el fin de entregar al cliente una solución que redujo en un 3.88% el tamaño del compilado del videojuego tan solo para 20 niveles.

Palabras clave: videojuegos, Coliseum, generación procedural, creación de niveles.

ÍNDICE DE CONTENIDO

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La industria de los videojuegos es el sector económico involucrado en el desarrollo, la distribución, la mercadotecnia, la venta de videojuegos y del hardware asociado. Engloba a docenas de disciplinas de trabajo y emplea a miles de personas alrededor del mundo. En estos últimos años ha experimentado altas tasas de crecimiento, debido al desarrollo de la computación, capacidad de procesamiento y la estrecha relación entre películas de cine y los videojuegos, con lo cual los consumidores se sienten más identificados, debe ser comprendida como productora de mercancías para un público masivo y global, donde sus principales sustentos son el desarrollo e inversión en recursos de alta tecnología. (Requena Farinós, 2014)

La historia de los videojuegos tiene su origen a principio de los años 60 del siglo XX en plena guerra fría y durante el auge de la electrónica. Sin embargo, este fenómeno se sitúa mucho más atrás en el tiempo. Los acontecimientos sociales, culturales y económicos que a partir de la segunda mitad del siglo XIX dieron vida a las sociedades modernas y a las tecnologías de la comunicación e información, también son los antecedentes históricos y culturales de los videojuegos. (Trenta, 2012) Existe actualmente una era de progreso tecnológico dominada por la industria que promueve un modelo de consumo rápido, donde las nuevas superproducciones quedan obsoletas en pocos meses, el impacto que supone la industria se demuestra al proporcionar empleo a más de 12000 personas y generar beneficios multimillonarios que aumentan año tras año.

En Cuba, en los últimos años se ha comenzado a desarrollar y practicar la industria del videojuego. Entre los estudios de videojuegos existentes se encuentra ConWiro (Concepcion, 2020) y otras entidades como es el caso del ICAIC (Instituto Cubano Del Arte e Industria Cinematográficos) y la Universidad de Ciencias Informáticas. (Ramón, 2016)

La Universidad de Ciencias Informáticas (UCI) es considerada un pilar en el desarrollo de videojuegos en Cuba, complementando sus esfuerzos en la formación en el área de gráficos de computadoras, inteligencia artificial y simulación física. Con el empeño de fomentar la innovación y la experimentación en el campo del software recreativo, participa año tras año en la simultánea mundial de videojuegos, Global Game Jam (GGJ). Es una

iniciativa destinada a impulsar la creación de videojuegos entre las nuevas generaciones, y en la cual se reúnen equipos durante un fin de semana para, a partir de un tema, crear un prototipo de juego en 48 horas.

En el Centro de Tecnologías Interactivas de la UCI, específicamente en la línea de Videojuegos, se desarrollan productos y servicios informáticos asociados a esta industria, logrando resultados significativos y de gran aceptación. El videojuego Coliseum es uno de los productos más conocidos de este centro, pensado como una alianza entre los géneros MOBA (Multiplayer Online Battle Arena) y RPG (Role-Playing Game) de acción en tiempo real. El juego posee un modo campaña con 80 niveles, donde el jugador puede luchar contra hordas de enemigos y poderosos jefes de nivel, tanto en solitario como en compañía de amigos a través de una red de área local. (Domínguez, 2018)

Uno de los problemas que enfrentan los desarrolladores de este videojuego es el tamaño que ha ido adquiriendo la aplicación móvil a lo largo de los años con la integración de nuevos contenidos para satisfacer a sus usuarios. Cada nivel puede contener una o varias escenas, las cuales se almacenan en el proyecto del videojuego. Las escenas son un espacio tridimensional que contienen los objetos del videojuego, tales como mapas, enemigos y texturas, necesarios para crear los niveles y los menús. Esto provoca el crecimiento en megas de la aplicación debido a las escenas guardadas en cada nivel del videojuego. Como resultado, el tamaño de las escenas guardadas ocupa prácticamente la mitad del tamaño del videojuego.

El constante crecimiento del videojuego en el futuro traería dificultades para los usuarios a la hora de descargar la aplicación, demoras al instalar el juego y limitaciones para los desarrolladores en la creación de nuevos niveles para evitar cargar aún más el proyecto. Esto obstaculiza la evolución y desarrollo futuros de este exitoso videojuego.

Ante esta situación se plantea como **problema de investigación:** ¿Cómo reducir el tamaño del videojuego Coliseum al adicionar nuevos niveles?

Para dar solución a este problema, se tomará como **objeto de estudio**: la generación de niveles en videojuegos de tipo MOBA-RPG.

Y como **campo de acción**: el Videojuego Coliseum.

Teniendo en cuenta lo planteado anteriormente, **el objetivo general** de esta investigación es: Desarrollar un componente para el videojuego Coliseum que permita la generación de niveles y que reduzca el tamaño de las escenas guardadas en el videojuego.

Para cumplir con el objetivo planteado se establecen las siguientes **tareas de investigación**:

1. Elaboración del marco teórico a partir del estado del arte de la generación de niveles para videojuegos de tipo MOBA-RPG.
2. Caracterización del videojuego Coliseum y su arquitectura actual.
3. Identificación de requisitos para la generación de niveles de Coliseum.
4. Análisis de la propuesta de solución.
5. Desarrollo de los artefactos ingenieriles asociados al diseño de la propuesta de solución.
6. Implementación de un componente para Coliseum que incluya la generación de niveles.
7. Ejecución de las pruebas de software a la solución.
8. Validación de la propuesta de solución.

Los métodos de investigación utilizados para dar solución a la presente investigación son:

Teóricos:

Analítico-Sintético: Permitió llevar a cabo un análisis bibliográfico de los elementos fundamentales relacionados con el proceso de generación procedural, como es el caso de la generación de niveles y sus herramientas.

Histórico-Lógico: Posibilitó realizar un estudio cronológico sobre la evolución y desarrollo de las técnicas de generación procedural y características especiales desde su surgimiento hasta la actualidad.

Empíricos:

Observación: Permitió llevar un control de todos los procesos y entender la situación existente del proyecto, para adoptar las medidas necesarias en cada momento cumpliendo con la tarea de desarrollo.

Modelación: Facilitó la representación mediante diagramas de las características, procesos y componentes de la solución propuesta, así como la relación entre ellos.

Estructura del Documento:

La presente investigación está estructurada de la siguiente forma:

Capítulo 1: Fundamentación Teórica.

En este capítulo se describe el estudio de la teoría sobre la investigación, para identificar las principales tendencias en el desarrollo de videojuegos, relacionada con los procesos de investigación, desarrollo e innovación en centros de desarrollo de software. Se realiza una síntesis de los conceptos asociados al caso de estudio, la metodología a utilizar, el marco de trabajo, elementos de la arquitectura de software, así como las herramientas y tecnologías con las que se trabaja.

Capítulo 2: Planificación y diseño de la propuesta de solución.

En este capítulo se describen los principales fundamentos y características de la propuesta de solución. Se define la arquitectura, patrones de diseño, y se generan los artefactos definidos para el modelado, requisitos, análisis y diseño de la propuesta de solución.

Capítulo 3: Implementación y pruebas.

En este capítulo se muestra el desarrollo de la solución al problema planteado, cumpliendo con los requisitos especificados. Se definen los prototipos de interfaces y los artefactos generados en la implementación. Se muestran además los resultados obtenidos de las pruebas realizadas al software para comprobar su funcionamiento y tolerancia a errores.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Con el objetivo de lograr una mayor comprensión del alcance de la investigación y esclarecer su objeto de estudio, se exponen en el presente capítulo, los conceptos asociados al dominio de la investigación y se realiza un análisis del estado del arte de homólogos. Además, se presenta la metodología y el entorno de desarrollo a utilizar para dar solución al problema planteado.

1.1 Principales Conceptos

1.1.1 Videojuego Coliseum

Es un videojuego MOBA-RPG (Multiplayer Online Batlle Arena – Role Playing Game) de acción en tiempo real, desarrollado por especialistas de la línea de Videojuegos del centro de Tecnologías Interactivas de la UCI, en colaboración con los estudios animados del ICAIC. Es considerado una de las primeras incursiones en videojuegos con partidas multijugador en Cuba. El juego posee un modo campaña con más de 50 niveles, donde se podrá luchar contra hordas de enemigos y poderosos jefes de nivel, tanto en solitario como en compañía de amigos, mediante una red local. Tiene la característica de ser Cross-platform play, o sea, que permite poder conectarse en línea con otros jugadores de otras plataformas y poder guardar el progreso en esos diversos dispositivos (Clumsy, 2022). Esto se debe a que tiene una versión para PC y otra para Android, con compatibilidad para jugar entre las dos plataformas, solo deben estar conectados a la misma red.

Coliseum ha continuado su desarrollo desde su lanzamiento el 12 de septiembre de 2018. Debido a la gran aceptación que ha tenido por los usuarios cubanos, se convirtió en el videojuego más popular hasta la fecha, logrando miles de descargas en la plataforma, evidenciando su rentabilidad.

El modo multijugador adiciona el mapa MOBA, este modo de juego es muy popular entre los cubanos semejándose a juegos como DOTA o League of Legends. Cada partida comienza con dos equipos opuestos, de hasta 3 jugadores. Los jugadores cooperan entre sí, con el objetivo final de destruir la estructura principal del equipo enemigo conocida como base. En el camino a la base enemiga se debe luchar contra las torres defensivas, los guardianes y los héroes enemigos. A medida que se eliminan los enemigos se obtiene

puntos de experiencia, que permiten subir de nivel y obtener armas y aditamentos en la tienda. Si se muere durante la partida vuelve a renacer al lado de la base, donde también se puede regresar para regenerar vida. Este modo de juego requiere buena especialización de los integrantes del equipo, para desempeñar roles como Tanque, Apoyo o Matador, entre otros (Domínguez, 2019)

1.1.2 Inteligencia Artificial

La inteligencia Artificial (IA) es la habilidad de una máquina de presentar las mismas capacidades que los seres humanos, ejemplo el razonamiento, el aprendizaje, la creatividad y la capacidad de planear, Según la RAE (Real Academia Española) es el Desarrollo y utilización de ordenadores con los que se intenta reproducir los procesos de inteligencia humana. Es la capacidad de los sistemas de cómputo para usar algoritmos, aprender de los datos y utilizar lo aprendido en la toma de decisiones, tal y como lo haría un ser humano. Los sistemas basados en IA pueden analizar grandes volúmenes de información y en correspondencia la proporción de errores es mucho menor en comparación si la realizara un humano. Debido a las capacidades de aprender y tomar decisiones, hoy en día muchas tareas que antes estaban reservadas solo para los humanos las realizan los sistemas de IA (Rouhiainen, 2018).

La IA en los videojuegos se define como un conjunto de técnicas que se emplean para diseñar el comportamiento de los NPC (*No Placable Carácter*), sean estos enemigos o aliados del jugador. En la década de los 90´s se popularizó el uso de IA con los videojuegos de estrategia en tiempo real, como Age of *Empires* o *StarCraft,* demostrando los avances en este campo. En este tipo de videojuegos, los enemigos eran perfectamente capaces de diseñar tácticas y estrategias que se basaban en el movimiento del jugador. Utilizaban sus recursos de forma eficiente para ganar la partida y anticiparse a decisiones que pudiese tomar el jugador. En la industria de los videojuegos la IA se utiliza también para el diseño del mapeado, de niveles o incluso para la creación de videojuegos desde cero. Los sistemas procedurales emplean técnicas de IA capaces de generar terrenos y escenarios casi infinitos, utilizados en videojuegos de exploración, construcción o en *roguelikes* (sus mapas se crean de forma aleatoria) (Tokio, 2020).

1.1.3 Generación Procedural de Contenido en los videojuegos

Los videojuegos se han convertido en un pilar fundamental de la industria del entretenimiento, pues han sido objeto de estudio durante la investigación de nuevas técnicas de IA en el mundo académico. Entre los avances más destacados en la investigación se encuentran las técnicas de Generación Procedural de Contenido (PCG, por sus siglas en inglés), que permiten generar, mediante algoritmos, contenido específico para videojuego. El contenido, que puede ser indispensable para la construcción del videojuego o, por el contrario, opcional, varía desde los mapas y niveles en los que se desarrolla la partida hasta la música, pasando por las reglas, las misiones que el jugador tiene que cumplir, los modelos tridimensionales de los personajes y objetos y texturas, por poner ejemplos. El proceso de generación puede suceder durante la fase de desarrollo del juego, denominada como generación estática, o de forma dinámica al mismo tiempo que se está jugando. Según sea el proceso de generación, se puede seguir un esquema constructivo, donde el contenido se genera mediante unas reglas de transformación, las que son sometidas a validación y basan su funcionamiento en buscar el contenido más adecuado en todo el espacio de soluciones posibles (Julian Togelius, 2011).

Una de las ventajas inmediatas del uso de estas técnicas es el ahorro en coste que supone diseñar contenido de forma manual, también se puede contemplar un ahorro en el espacio que ocupa el juego, tanto en memoria como en almacenamiento físico, ya que es posible generar dinámicamente contenido y objetos del juego que de otra forma tendrían que ser cargados desde el disco a la memoria principal. Pero la ventaja más destacable es la posibilidad de adaptar el juego y al jugador de manera que la experiencia de juego sea única y personalizada, contribuyendo al objetivo de incrementar la satisfacción del jugador, principal intención de un videojuego. Con respecto a las desventajas más destacables, cabe mencionar la inversión inicial de tiempo requerida por el uso de la Generación Procedural de Contenido, para estudiar los requisitos del contenido a generar, modelar y crear la herramienta de generación, además de someter a validación de resultado obtenido. (Lara-Cabrera, 2015)

Propiedades deseadas en la generación de contenido:

Las soluciones basadas en la generación procedural de contenido tienen una serie de propiedades deseables o requeridas que pueden ser diferentes para cada aplicación, algunas de las propiedades comunes pueden ser:

- **Velocidad:** Los requerimientos de velocidad pueden variar notablemente según el rango de la aplicación, normalmente sería necesario que esta sea superior a como trabajaría un humano si está realizando un trabajo similar.
- **Confiabilidad:** El generador de contenido debe garantizar que este se cree dentro de unos criterios de calidad, cumplir unas reglas o parámetros que en mayor parte son establecidos por el propio algoritmo.
- **Control:** Sería deseable proporcionar cierto control, permitiendo especificar algunos aspectos iniciales del mismo contenido que se va a crear.
- **Expresividad y diversidad:** Medir la expresividad es difícil y generar contenido que sea diverso y con cierta calidad no es nada trivial, pero muy demandado.
- **Creatividad y Credibilidad:** Se requiere que el contenido generado parezca diseñado por una persona, evitando la sensación de que se trate de un proceso automático (Aíbar, 2021).

1.1.4 Técnicas de Generación Procedural

Existen muchos tipos de técnicas para la Generación Procedural de Contenido (PCG), todas con un mismo objetivo en común: generar los elementos necesarios para el desarrollo del videojuego. Entre ellas se encuentran las siguientes:

Basados en Restricciones

Los sistemas basados en restricciones definen restricciones para todos los objetos prediseñados que se utilizan en el sistema de generación. A partir del diseño de las reglas básicas, como el objetivo (por ejemplo, matar al monstruo final), punto de inicio, tamaño máximo del mapa, etc., se aplican reglas a las distintas partes que conforman el mapeado para que cada una solo sea utilizable de la manera correcta. Esto conlleva un gran número de condiciones a la hora de desarrollar el código, ya que se deben contemplar todas y cada una de las posibilidades previamente, e indicar paso por paso cómo debe procederse en esos casos en el propio código del programa. Como punto a favor tiene su facilidad de desarrollo, ya que básicamente consiste en codificar qué partes son utilizables y cómo se relacionan temporalmente unas con otras, qué objetos deben crearse previamente a otros. Como punto negativo tiene, obviamente, que cualquier variación importante que sea necesario incorporar en el entorno, acarrea el análisis de todas estas condiciones para cerciorarse que no afecte a otras, y que debido a esta necesidad de tener atadas todas las

posibilidades, también se delimita el número de variaciones que pueda alcanzar el entorno (Codes, 2015).

Autómata Celular

Una de las técnicas más simples de implementar para PCG es el autómata celular (AC), los resultados que da este sistema de PCG son mapas similares a cuevas, bosques o manadas de animales abiertas en función de las reglas y el número de iteraciones. El concepto de autómata celular fue presentado originalmente en los años 40 por Stanislaw Ulam y Jhon Von Neumann cuando intentaban modelar una máquina que fuera capaz de autorreplicarse. Inicialmente fueron interpretados como conjunto de células que crecían, se reproducían y morían a medida que pasaba el tiempo. Su nombre se debe a esta similitud con el crecimiento de las células.

Un autómata celular es un modelo matemático para un sistema dinámico que evoluciona de manera discreta. Consiste en una grilla de celdas, en donde cada una tiene número finito de estados, ejemplo: prendido y apagado. Las celdas contienen referencia a sus vecinos próximos y un estado inicial en t=0. Luego de varias iteraciones se genera un mapa en la cuadrícula el cual depende del tipo de reglas que se utilicen y el estado inicial de la cuadrícula. (Caparrini, 2016)

Los autómatas pueden tener importancia como ejemplos de sistemas dinámicos discretos o como un modelo de computación en paralelo; para un biólogo, el interés puede residir en que los autómatas celulares proporcionan una herramienta para modelar y comprender los fenómenos biológicos de una forma simple. Ver (Figura No. 1)

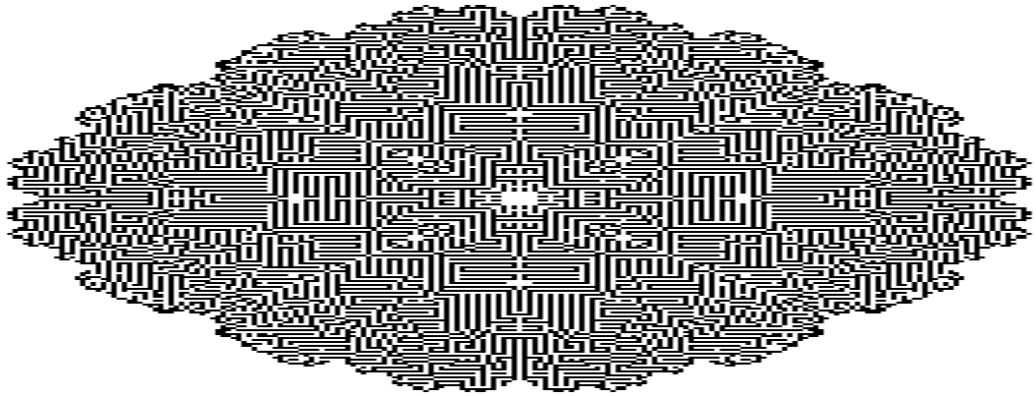

Figura 1: Autómata celular

Este método tiene algunas ventajas, entre las cuales se encuentran las siguientes:

- Posibilita la generación de una gran variedad de escenarios.
- Las reglas que se aplican para el cambio de estados.
- El aspecto desorganizado del resultado incrementa el nivel de realismo del escenario.

Algoritmo Genético

Los algoritmos genéticos (GA) son eficientes y robustos, y emplean técnicas de optimización y una representación genética para la solución de problemas (Soria, 2014). Esta representación genética es la encargada de codificar las soluciones, y cada solución se evalúa utilizando un valor tope que mide cuán buena es la solución codificada. Por ello, a medida que se realizan varias iteraciones, el algoritmo es capaz de mezclar los mejores valores para encontrar la solución más óptima en un rango de tiempo establecido. Procesan simultáneamente, no una solución al problema, sino todo un conjunto de ellas. Estos algoritmos trabajan con una codificación de soluciones potenciales al problema. El conjunto de todos ellos forma la población con la que trabaja el algoritmo. Ver (Figura No. 2)

Figura 2: Algoritmo genético

Para generar mapas, es posible diseñar un algoritmo genético que, tras crear una variedad de mapas completamente aleatorios y registrando los caminos entre ellos, realice una mezcla entre los mismos, aplicando a su vez mutaciones (añadir o quitar caminos), hasta alcanzar un mapa cuya función de ajuste (mínimo de celdas conectadas y enemigos, por ejemplo), satisficiese las necesidades del desarrollo. Aunque este tipo de algoritmo puede crear una variedad enorme de mapas hasta encontrar una solución correcta para el entorno del juego, esta solución puede implicar un alto costo computacional durante la generación de múltiples mapas y la ejecución del proceso evolutivo, el cual, si no se valida correctamente, puede ejecutarse indefinidamente.

Un algoritmo genético siempre ejecuta tres operaciones básicas:

- Reproducción: Proceso de selección para el posterior apareamiento e intercambio de genes, con base en el valor de su función de aptitud (Indica que tan apto es un individuo), se seleccionan con mayor probabilidad los individuos con una función de aptitud alta, logrando así una mayor probabilidad de contribuir con decencia a la próxima generación.
- Cruza: La cruza es un operador que lleva a cabo el intercambio de información genética de dos individuos de la población, a los que llamaremos padres, en el cual se combinan sus genes, que son bits de las cadenas binarias, para generar descendencia o hijos.
- Mutación: La mutación modifica bits de la cadena binaria de forma aleatoria con cierta probabilidad. El objetivo de la mutación es general nuevos individuos o hijos que formaran parte de la nueva población (J Arranz de la Peña, 2007).

1.2 Estado del arte

En la actualidad son muchísimos los videojuegos que emplean las técnicas de PCG para generar, de forma aleatoria o no, los objetos dentro de las escenas del videojuego. Con el objetivo de identificar ventajas y desventajas de estas, se realiza un análisis haciendo énfasis en las características y técnicas empleadas para generar el contenido dentro de los siguientes videojuegos:

Minecraft es un videojuego de construcción del tipo RPG, lanzado el 17 de mayo de 2009 y no tiene un fin claramente definido o misiones a cumplir (Lyons, 2019). Esto permite una

gran libertad en cuanto a la toma de decisiones a la hora de jugar. El juego posee un sistema que otorga logros por completar acciones definidas. Se desarrolla en primera persona, aunque existe la posibilidad de cambiarla a una perspectiva de tercera persona. El juego se centra en la colocación y destrucción de bloques, componiéndose de objetos tridimensionales cúbicos, colocados sobre un patrón de rejilla fija. Estos cubos representan principalmente elementos de la naturaleza, como piedra, minerales, troncos. Al iniciarse la partida se genera un mapa aleatorio en todos los casos donde los elementos fundamentales de la naturaleza se distribuyen por toda el área, los jugadores son libres de desplazarse por el entorno y modificarlo mediante la creación, recolección y transporte de bloques.

Hades es un juego de rol de acción roguelike estrenado el 17 de septiembre del 2020, el jugador controla a Zagreus el hijo del dios Hades, debe escapar del inframundo para llegar al Olimpo y reencontrase con su madre (Vaz, 2020). Para ello debe recorrer una serie de habitaciones aleatorias interconectadas, pobladas de enemigos y recompensas, donde tiene como restricción que no se puede abandonar la habitación hasta que todos los enemigos sean eliminados. En este caso, las habitaciones, y el orden en que aparecen, nunca son iguales, se generan aleatoriamente, lo mismo sucede con los enemigos, el tipo y la cantidad que el jugador debe enfrentar en cada caso y las recompensas en la habitación.

Don’t Starve es un videojuego de supervivencia 2D, del género acción y aventura, integra elementos de horror y mazmorras a su dinámica de juego. Desarrollado y lanzado por Klei Entertainment en el año 2013, para las plataformas Linux, OS y Microsoft Windows. El objetivo del videojuego es sobrevivir en un mundo lleno de criaturas y peligros, mantener saludable al personaje principal, recolectando recursos para fabricar objetos, alimentarse, defenderse y progresar en la aventura. Se utiliza PCG para generar el mundo, mezclando distintos ecosistemas, animales y construcciones. (Baños, 2016)

Tabla 1: Comparativa de los sistemas homólogos

Videojuego	**Tipo**	**Entorno**	**PCG**
Minecraft	RPG	3D	Utiliza PCG para la generación de

			mundos abiertos, con elementos de aleatoriedad cada vez que se inicia la partida.
Hades	Roguelike	3D	Utiliza PCG para la generación de mazmorras interconectadas como niveles.
Don' Starve	Survival	2D	Utiliza PCG para generación de terrenos y objetos con elementos de aleatoriedad cada vez que inicia la partida.

El estudio de los videojuegos mencionados anteriormente permite establecer criterios de similitud para identificar los principales elementos y características de la generación procedural en los videojuegos y adaptarlos a la propuesta de solución. En el caso del videojuego Coliseum los mapas ya están establecidos, uno para cada mundo (invierno, primavera, verano y kalaris), por tanto, no se hace necesario establecer elementos de aleatoriedad en las texturas o coliseos ya que cambiaría la esencia del videojuego. Se propone utilizar una única escena, donde se decida qué coliseo cargar en cada nivel, y los objetos que componen la escena, siguiendo reglas preestablecidas y empleando la técnica de PCG basado en restricciones, por lo que sería una generación semiautomática.

1.3 Entorno de desarrollo de la propuesta de solución

Metodología de desarrollo de software

Las metodologías de desarrollo de software constituyen procesos, en los que se definen técnicas y procedimientos a seguir en el proceso de desarrollo de un producto. El uso de

una metodología específica está asociado a características propias del proceso de desarrollo de software. Cada metodología tiene diferencias marcadas, sobre todo en la manera en la que se gestiona los procesos, artefactos, roles y actividades (Letelier, 2106). Existen numerosas propuestas metodológicas, entre ellas se encuentra las metodologías tradicionales y las metodologías ágiles. Las tradicionales se centran específicamente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Las ágiles brindan más importancia a la colaboración con el cliente y el desarrollo incremental del software con iteraciones muy cortas, mostrando efectividad en proyectos con requisitos cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo. El enfoque de las metodologías ágiles resulta ser el más adecuado en este caso.

Tabla 2: Comparación de metodologías agiles y tradicionales

Metodologías tradicionales	Metodologías ágiles
Predictivos	Adaptativos
Orientados a procesos	Orientado a personas
Proceso rígido	Proceso Flexible
Poca comunicación con el cliente	Comunicación Constante con el cliente
Entrega de software al finalizar el trabajo	Entregas constantes de software
Documentación extensa	Poca documentación

Metodologías ágiles

XP (Extreme Programming) es la metodología ágil más conocida, desarrollada por Kent Beck. Entre sus peculiaridades se encuentran las historias de usuarios, las cuales corresponden a una técnica de especificación de requisitos, donde el cliente describe las características y funcionalidades que el sistema debe poseer. Define la fecha de cumplimiento y alcance de una entrega funcional, con costos de implementación y número de interacciones para terminar. Se realizan además pequeñas entregas en ciclos cortos de desarrollo y retroalimentación con el cliente. En la etapa de pruebas destacan las pruebas de aceptación por parte del cliente, dando su aprobación del producto (Sánchez, 2020).

Características de la metodología XP:

- Comunicación constante entre el cliente y el equipo de desarrollo.
- Respuesta rápida a los cambios constantes.
- La planificación es abierta con un cronograma de actividades flexible.
- Los requisitos del cliente y el trabajo del equipo del proyecto son los principales factores de éxito de la metodología.

Fases de la metodología XP (Sánchez, 2020) **:**

Planificación: Identificación de las historias de usuario, donde el cliente establece sus prioridades y correspondientemente se realiza la estimación del tiempo requerido para cada una de ellas. Se realiza los acuerdos correspondientes de entrega de las versiones para su revisión y pruebas.

Diseño: Obtención del prototipo, se intentará trabajar con un código sencillo, haciendo lo mínimo posible para que funcione utilizando el diseño de software orientado a objetos, y las tarjetas CRC (Clase-Responsabilidad-Colaboración).

Desarrollo: Esta fase requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Prueba: La implantación de la solución es fundamental, para ello en esta etapa se realizan pruebas de aceptación con el cliente al producto final.

Figure 3: Metodología XP

SCRUM

Scrum es un marco de trabajo que logra la colaboración eficaz del equipo de trabajo, emplea un conjunto de reglas y se definen roles para generar una estructura de correcto funcionamiento. Define roles tales como Scrum Master quien lidera el equipo, asegurándose que se cumplan las reglas y procesos de metodología, el equipo de desarrollo es el grupo de profesionales encargados de convertir la lista de requerimientos en funcionalidades de software. (Montero, 2018). Esta metodología está especialmente indicada para proyectos con un cambio rápido de requisitos.

Características de la metodología SCRUM:

- El desarrollo de software se realiza mediante iteraciones, entregándose versiones del proyecto en desarrollo.
- Reuniones a lo largo del proyecto de manera sistemática.
- Participación activa con el cliente.
- Su foco principal está en las actividades de gestión de proyecto.

Fases de la metodología SCRUM (Sánchez, 2020)**:**

- **Inicio:** Fase encargada de estudiar y analizar el proyecto identificando las necesidades básicas de la versión a entregar por cada iteración.
- **Planificación:** Fase encargada de estimar y elaborar historias de usuarios, identificar y estimar tareas de desarrollo.
- **Implementación:** Fase encargada del cumplimiento del plan de los entregables, y la implementación de los requisitos descritos en las historias de usuarios.
- **Revisión y retrospectiva:** Fase encargada de hacer la revisión del proceso, a modo de evaluación interna del grupo respecto a su propio trabajo.
- **Lanzamiento:** Fase final del proyecto y entrega del producto al cliente.

Fundamentación de la decisión

Luego de los elementos planteados sobre las metodologías se decidió emplear XP como metodología de desarrollo de software debido a que se caracteriza por su cronograma flexible, respuesta rápida a cambios significativos del proyecto y por la comunicación, retroalimentación, simplicidad y robustez que conduce el desarrollo del trabajo.

Centrándose más en la ingeniería, con iteraciones cortas, un desarrollo basado en pruebas, juego de planificación, diseño simple, estándares de codificación y pruebas de aceptación, impulsado por el desarrollo del código.

Entorno de Desarrollo Integrado:

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft. Es software libre y multiplataforma, está disponible para Windows, GNU/Linux y macOS. Vs Code tiene una buena integración con git, cuenta con soporte para depuración de código, y dispone de un sin número de extensiones, que básicamente posibilita escribir y ejecutar código en cualquier lenguaje de programación. Incluye un terminar con todas las funciones, la cual se inicia fácilmente en el directorio de trabajo. VS Code puede instalar únicamente las herramientas de desarrollo requeridas, y personalizarlo de acuerdo a sus necesidades. En el desarrollo de la propuesta de solución se utilizó la versión 1.66.1, no debe utilizarse una versión más actual para que sea compatible con el proyecto del videojuego.

Lenguaje de Programación:

C# es un lenguaje de programación basado en objetos y con seguridad de tipos. Permite a los desarrolladores crear muchos tipos de aplicaciones seguras y solidas que se ejecutan en .NET. Tiene sus raíces en la familia de lenguajes C, y a los programadores de C, C++, Java y JavaScript les resulta familiar inmediatamente. Proporciona construcciones de lenguaje, agregando características para admitir nuevas cargas de trabajo y prácticas de diseño de software emergentes. Ofrece protección ante variables que no hacen referencia, proporciona un enfoque estructurado y extensible para la detección y recuperación de errores, admiten técnicas de programación funcional, crea un patrón común para trabajar con datos de cualquier origen y proporciona la sintaxis para crear sistemas distribuidos (Microsoft, 2022). La versión del lenguaje de programación seleccionado es la versión 4.0.

Lenguaje de Modelado UML:

El lenguaje de Unificado de Modelado (UML) es un lenguaje de modelado visual de propósito general, que se utiliza para especificar, visualizar, construir, y documentar los artefactos de un sistema de software. Captura decisiones y conocimiento sobre sistemas que deben ser construidos. Se usa para comprender, diseñar, ojear, configurar, mantener, y controlar la información sobre el sistema. Funciona con todos los métodos de desarrollo,

dominios de aplicación y medios. UML incluye conceptos semánticos, notación y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Pensado para ser apoyado por herramientas de modelado visuales e interactivas que dispongan de generadores de informes, pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos (Chaparro, 2020). Se utiliza este lenguaje de modelado en su versión 2.5.

Herramienta de Modelado:

Visual Paradigm para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción y despliegue. Es multiplataforma, utiliza UML como lenguaje de modelado, ayuda de manera rápida en la construcción de aplicaciones de mayor calidad y costo menor. Permite dibujar todo tipo de diagramas de clases, código inverso, general código desde diagramas y general documentación. Proporciona una mejor interfaz gráfica de usuario y una mayor base de datos de esquema de apoyo (Estrada, 2011). Para la presente investigación, se utiliza esta herramienta en su versión 8.0

Motor de Videojuego:

Unity3D es un motor de videojuegos multiplataforma creado por Unity Technologies. Está disponible como plataforma de desarrollo para Windows, Mac Os y Linux. Tiene soporte para mapeado de relieve, mapeado de reflejos, oclusión ambiental en espacio de pantalla, sombras dinámicas, texturas y efectos de post-procesamiento de pantalla completa. También incluye una solución de control de versiones para todos los assets de juego y scripts, utilizando PostgreSQL como backend, un sistema de audio con capacidad de reproducir audio comprimido, un motor de terreno y vegetación, funciones de iluminación, redes multijugador y funciones de búsqueda (Pemau, 2020). En el desarrollo de la propuesta de solución se utiliza esta herramienta en su versión 2018.1.0f2.

1.4 Conclusiones del capítulo

Durante el desarrollo de este capítulo se llevó a cabo un análisis profundo de los elementos teóricos-metodológicos que sirvieron de guía para la elaboración del presente trabajo, permitiendo llegar a las siguientes conclusiones:

1. La definición de los principales conceptos y documentación asociados al dominio de la investigación y las relaciones entre ellos, permitieron alcanzar una mayor comprensión de la propuesta de solución.

2. El estudio del Estado del arte de las técnicas de generación de contenido utilizadas en los videojuegos homólogos, evidenciaron en gran medida, que satisface las necesidades existentes hoy para reducir el tamaño del videojuego Coliseum.

3. Las herramientas y tecnologías seleccionadas permitieron llevar a cabo el desarrollo de la propuesta de solución e integrarlo al proyecto del videojuego existente.

CAPÍTULO 2: PLANIFICACIÓN Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN

En el análisis y diseño es importante establecer comunicación y acuerdos con el cliente y el equipo de desarrollo para elaborar la solución que más se ajuste a las necesidades del cliente. El objetivo del capítulo es presentar los principales componentes que conforman la propuesta de solución y explicar adecuadamente su funcionamiento. Se definen las historias de usuario, el uso de las tarjetas CRC, la obtención de los requisitos funcionales y se definen los artefactos que servirán de base para la fase de implementación.

2.1 Características de la propuesta de solución

Coliseum como videojuego tiene la forma del mapa ya establecida y diseñada, no es variable, sus coliseos se categorizan en: verano, invierno, kalaris y primavera, además de sus personajes que también están definidos y diseñados de acuerdo a roles: Tanque, Matador, Apoyo, etc. Sin embargo, el tipo de enemigos, las runas, las posiciones iniciales de la partida y las texturas sí son variables, y fueron los elementos principales de trabajo, ya que varían entre instancias en cada nivel. De los PCG investigados el que mejor se ajusta a las necesidades de la investigación y cumple con los objetivos de forma eficiente y rápida, cambiando la forma en que se organizan, es usando un sistema basado en restricciones. En el videojuego cada nivel está conectado a una escena específica, sin posibilidad de que las instancias varíen, como consecuencia cada vez que se adiciona un nuevo nivel aumenta el tamaño en memoria requerido.

Se propone almacenar la información necesaria para cada nivel en una carpeta de recursos con los elementos que varían entre escenas (coliseo, música, enemigos, posiciones iniciales). Luego se procede a crear una escena dinámica encargada de instanciar objetos en tiempos de ejecución. Una escena es el espacio tridimensional en el que aparecerán los componentes y objetos, que luego se visualizarán. En este caso, al crear una escena dinámica se cargan los objetos en tiempo de ejecución. Al escoger un personaje e iniciar la partida, se inicializa la escena dinámica, la cual contendrá dentro de sí misma los elementos comunes entre escenas (controladores, interfaces, cámara, eventos del sistema, apariciones de runas), necesarios para la jugabilidad. De igual manera la escena dinámica contendrá el método constructor, encargado de construir la escena según las restricciones especificadas para cada nivel, apoyándose de la carpeta de recursos e instanciando los objetos dentro de la escena. Entiéndase como objetos a todos los componentes que integra

una escena (enemigos, coliseo, música, runas, muros, controladores, cámara, eventos del sistema, interfaces …). Una vez que se construya la escena, se carga e inicia la partida. Este proceso se puede visualizar en la siguiente figura:

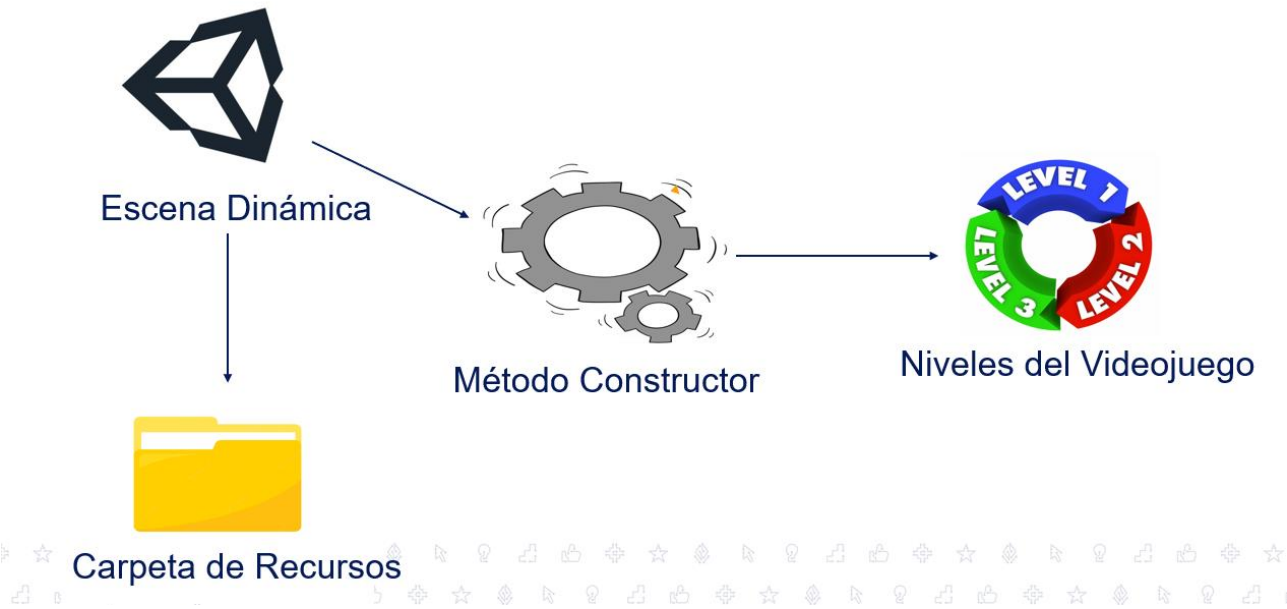

Figure 4: Propuesta de solución

2.2 Planificación

En esta sección se definen los principales aspectos, de acuerdo a las necesidades del cliente, y se elabora el plan de desarrollo, teniendo en cuenta los tiempos y costos de producción, así como los mecanismos y funcionalidades, que sirven como base en la fase de implementación.

2.2.1 Requisitos de la solución

Los requisitos son la base para el desarrollo de la solución de la problemática. Permite determinar cuáles son las funcionalidades que una aplicación debe poseer y bajo qué condiciones se debe instaurar. Describen las características de una solución que cumpla con las necesidades de las partes interesadas.

En los encuentros con el cliente se definieron los siguientes **requisitos funcionales** (RF):

RF1. El sistema debe permitir la inicialización de la escena.

RF2. El sistema debe construir la escena.

RF3. El sistema debe cargar la escena.

RF4. El sistema debe iniciar el nivel

2.2.1 Historias de Usuarios

Las historias de usuarios conforman una parte fundamental de la metodología de desarrollo de software XP. Del listado de requisitos definidos con el cliente se obtienen las siguientes historias de usuario:

Tabla 3: HU_1 Inicialización de la escena

Historia de Usuario	**Numero:1**
Nombre: Inicialización de la escena.	**Usuario:** Rolando Martin
Prioridad en Negocio: Media	**Riesgo en Desarrollo**: Baja
Puntos Estimados: 5 días	**Iteración**: 1
Descripción: El sistema debe inicializar la escena dinámica. Se accede a la clase encargada de cargar las escenas del videojuego y se realiza una modificación, solicitando que llame a la escena dinámica creada. Luego se borran las antiguas escenas del videojuego.	

Tabla 4: HU: Construir la escena

Historia de Usuario	**Numero:2**
Nombre: Construir la escena	**Usuario:** Rolando Martin
Prioridad en Negocio: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 15 días	**Iteración**: 1
Descripción: Primeramente, se debe cargar las configuraciones del nivel pasado por parámetro, accediendo a los objetos de la carpeta de recursos, que contiene el Coliseum, la música y los muros interiores, los enemigos, el tutorial del nivel si lo tiene y las posiciones iniciales. Luego se instancian en tiempo de ejecución estos objetos en la escena	

dinámica.

Tabla 5: HU: Cargar la escena

Historia de Usuario	**Numero:3**
Nombre: Cargar la escena	**Usuario:** Rolando Martin
Prioridad en Negocio: Alta	**Riesgo en Desarrollo**: Media
Puntos Estimados: 7 días	**Iteración**: 2
Descripción: Una vez que la escena está construida el sistema recibe la señal de que está preparada y se procede a cargar la escena en tiempo de ejecución.	

Tabla 6: HU: Iniciar el nivel

Historia de Usuario	**Numero:4**
Nombre: Iniciar el nivel	**Usuario:** Rolando Martin
Prioridad en Negocio: Alta	**Riesgo en Desarrollo**: Media
Puntos Estimados: 3 días	**Iteración**: 2
Descripción: Una vez que la escena está cargada emite la señal al scrpit para comenzar el juego.	

2.2.2 Estimación de esfuerzo por historias de usuarios

Las historias de usuario deben estar bien especificadas para que los desarrolladores puedan realizar una estimación precisa y poco riesgosa del tiempo que llevará su implementación. Es necesario tener claridad de la funcionalidad que se va a desarrollar y su complejidad para poder llevar a cabo una correcta estimación del esfuerzo para implementarla y los tiempos de entrega. Para el desarrollo del componente se utilizan los días como medida de estimación de esfuerzo, descrito a continuación:

Tabla 7: Estimación de esfuerzo

Historias de usuario	**Puntos de estimación (días)**
Inicialización de la escena dinámica	5
Construir la escena	15
Cargar la escena	7
Iniciar el nivel	3

Se realizó la estimación de esfuerzo por historias de usuario teniendo como resultados que la implementación del componente durará un tiempo total de 30 días o 6 semanas, teniendo en cuenta que una semana tiene 5 días laborables, de los cuales se trabaja 8 horas diarias.

2.2.3 Plan de iteraciones

Todo proyecto que emplea metodología XP debe dividir su desarrollo en iteraciones. Las iteraciones son fases o etapas de la implementación donde se consiguen los resultados en un tiempo estimado. En el plan de iteraciones se especifican detalladamente el orden de desarrollo de las Historias de usuario dentro de cada iteración, donde la duración total en días, corresponde con el esfuerzo estimados para implementar las HU.

- **Iteración 1:** en esta iteración se implementará las HU 1 y 2 que tiene una prioridad de desarrollo alta, en este caso la HU_1 (Inicialización de la escena dinámica) es considerada el punto de partida ya que conforma la base de la estructura del componente y la HU_2 (Construir la escena) es el método que controla la lógica del negocio.
- **Iteración 2:** en esta iteración se implementan las HU 3 y 4, las cuales tienen una prioridad de desarrollo media, siendo la HU_3 (Cargar la escena) la finalización del proceso de la carga dinámica de recursos, y la HU_4 (Iniciar el nivel) representa la integración del componente al proyecto del videojuego.

Tabla 8: Plan de iteraciones

Iteraciones	Historias de Usuarios	Duración	Prioridad
1	Inicialización de la escena Construir la escena	20 días	Alta
2	Cargar la escena Iniciar el nivel	10 días	Media

2.2.4 Plan de entregas

Luego de tener especificado el plan de iteraciones, se elabora un plan de entrega, con la intención de obtener una estimación real de cuánto tomaría la implementación para cada HU. En la siguiente tabla se muestra el plan de entrega.

Tabla 9: Plan de entrega

Producto	Iteración 1	Iteración 2
Componente para la generación de niveles del videojuego Coliseum.	Versión 1.0 del producto	Versión 1.0 del producto
Fecha de inicio	8/7/2022	5/8/2022
Fecha de entrega	4/8/2022	18/8/2022

2.3 Fase de Diseño

La metodología de desarrollo de software XP se basa en la implementación de un código sencillo, haciendo lo imprescindible para que el proyecto cumpla con el objetivo, buscando la solución más simple posible. Los principales elementos del diseño de software se

presentan a continuación.

2.3.1 Arquitectura de Software

La Arquitectura de Software se refiere a una planificación basada en modelos, patrones y abstracciones teóricas, que permite evaluar y analizar la efectividad del diseño para cumplir los objetivos propuestos, se considera un paso crítico antes de comenzar a programar, buscar alternativas arquitectónicas en una etapa temprana del proyecto facilita realizar cambios de diseño de forma fácil y sencilla, reduciendo riesgos asociados a la elaboración de software (Reynoso , 2004).

La arquitectura utilizada es la de Modelo-Vista-Controlador (MVC), es un paradigma que divide las partes que conforman el diseño de las aplicaciones, permitiendo la implementación por separado de cada elemento, garantizando así la actualización y mantenimiento de software de forma sencilla y en un reducido espacio de tiempo. Utilizando MVC se puede lograr una mejor organización del trabajo y mayor especificación (Reynoso , 2004).

El Modelo es un conjunto de clases que representan la información que el sistema debe procesar y contendrá los mecanismos para acceder a los datos. La Vista es la clase que se encarga de mostrar al usuario la información contenida en el modelo y la clase Controlador es un objeto que se encarga de dirigir el flujo de datos entre clases de la aplicación, teniendo acceso a las clases Modelo y Vista (Reynoso , 2004).

2.3.2 Patrones de diseño

Los patrones de diseño son una solución general, reutilizable y aplicable a diferentes problemas de diseño de software, se trata de plantillas que identifican problemas en el sistema y proporciona soluciones apropiadas a problemas generales. (F Cueto, 2015)Para el diseño de la solución se emplearon varios patrones de diseño los cuales se describen a continuación.

Patrones GRASP

Los patrones GRASP (General Responsibility Assignment Software Patterns) son guías o

principios que sirven para asignar responsabilidades a las clases. Para el diseño de la solución se emplearon varios patrones de diseño, los cuales se describen a continuación:

- **Experto:** este patrón permite asignar quién tiene la responsabilidad de la creación de un objeto, la responsabilidad recae en la clase que conoce toda la información necesaria para crearlo, el patrón está presente en la clase DynamicSceneLoader.cs que contiene todos los métodos e información necesaria para la carga dinámica de la escena del videojuego.
- **Creador:** este patrón identifica quién debe ser el responsable de la creación o instanciación de nuevos objetos o clases. El uso de este patrón de diseño se evidencia en la clase DynamicSceneLoader.cs, la cual se encarga de instanciar los GameObject necesarios en cada escenario.
- **Controlador:** este patrón identifica el encargado de coordinar el trabajo entre clases, se evidencia en la clase DynamicSceneHandler.cs, encargado del intercambio de datos entre la escena dinámica y la clase encargada de iniciar el juego.
- **Bajo acoplamiento:** este patrón se evidencia al tener las clases con la menor dependencia que se pueda, en caso de producirse una modificación en alguna de las clases que tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización y disminuyendo la dependencia entre clases.

Patrones GOF

Los patrones GOF (Gang of Four) se describen como una forma indispensable de enfrentarse a la programación, en los cuales se encuentran 3 grupos, Patrones de Creación, Patrones Estructurales y Patrones de Comportamiento, proponen soluciones concretas y técnicas, y se usan muy frecuentemente. Se utilizó el siguiente patrón para el diseño de la solución.

- **Singleton:** es un patrón de creación que garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

2.3.3 Descripción de las tarjetas CRC

Las tarjetas CRC (Clase, Responsabilidad y Colaboración) son una metodología para el diseño de software orientado a objetos creada por Kent Beck y Ward Cunningham, es una

técnica para la representación de sistemas OO (Orientado a Objetos), permite romper con el modo de procesamiento y de pensamiento para apreciar la tecnología de objetos (Reinaga, 2009). A continuación, se describen las tarjetas CRC utilizadas:

Tabla 10: Tarjeta CRC LoaderScena

Tarjeta CRC	
Clase: LoaderScena.cs	
Responsabilidades	**Colaboración**
Inicializar la escena dinámica, haciendo una modificación en la clase, cambiando la forma en la que se cargan las escenas.	Inicializar la escena dinámica.

Tabla 11: Tarjeta CRC DynamicSceneLoader

Tarjeta CRC	
Clase: DynamicSceneLoader.cs	
Responsabilidades	**Colaboración**
Instanciar los Objetos especificados para cada nivel dentro de la escena dinámica.	Preparar la escena auxiliándose de una carpeta de recursos.

Tabla 12: Tarjeta CRC DynamicSceneHandler

Tarjeta CRC	
Clase: DynamicSceneHandler.cs	
Responsabilidades	**Colaboración**
Controlar el flujo información entre la clase encargada de instanciar los objetos (DynamicSceneLoader.cs) y la clase encargada de cargar la escena (InitGamePlayers.cs).	Controlar el flujo de información.

Tabla 13: Tarjeta CRC InitGamePlayer

Tarjeta CRC	
Clase: InitGamePlayer.cs	
Responsabilidades	**Colaboración**
Una vez cargada la escena manda la señal para iniciar la partida.	Cargada la escena da comienzo al nivel.

2.3.4 Diagrama de clases

El diagrama de clases es una técnica central y difundida en los distintos métodos orientados a objetos. Describen los tipos de objetos del sistema, así como los distintos tipos de relaciones que puedan existir entre clases, convirtiéndose en la técnica más importante para el modelado conceptual de un sistema. El propósito de un diagrama de clases es describir las clases que conforman el modelo del sistema, creado y refinado durante la fase de análisis y diseño, siendo una guía en la implementación del sistema (F Cueto J. J., 2015).

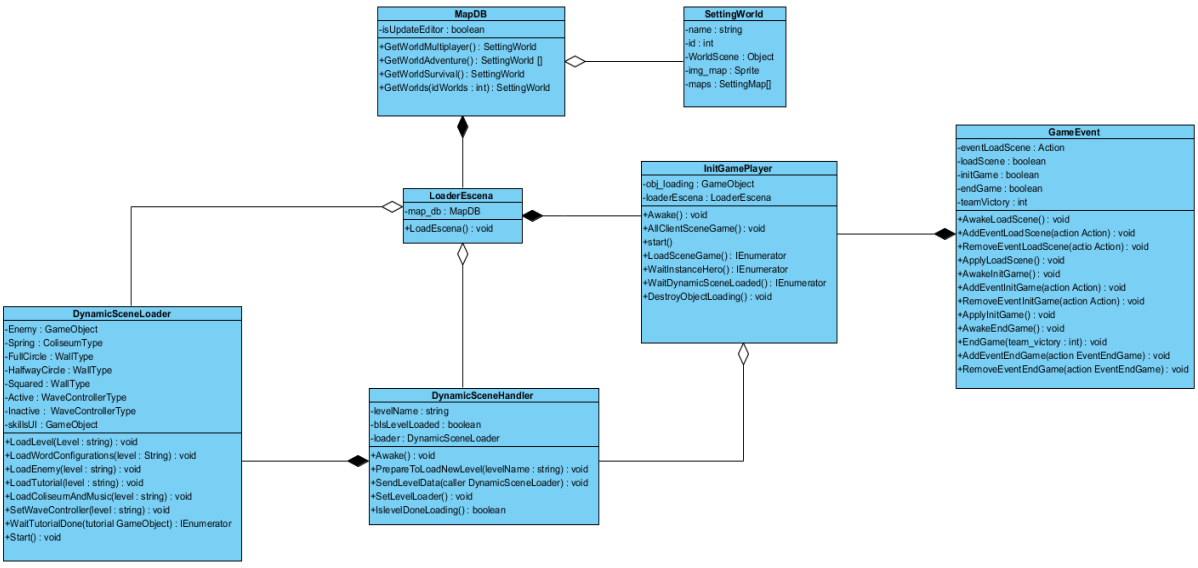

Figure 5(Diagrama de clases)

2.3.5 Estándar de Codificación

Los estándares de codificación son una serie de reglas que determinan cómo debe escribirse el código. El objetivo es lograr un código fácil de leer por los humanos, ayudan a

disminuir el esfuerzo necesario para la lectura del código escrito por un equipo de personas, haciendo más sencillo el mantenimiento del código a largo plazo.

- Se utiliza como estilo de escritura Pascal Case donde se combinan palabras directamente, sin usar ningún símbolo, estableciendo que la primera letra de cada palabra esté en mayúscula sin excepciones, estando el resto de letras en minúsculas.
- El idioma utilizado es el inglés y el lenguaje de programación C# para los nombres de métodos, estructuras, funciones y clases, las variables y constantes a excepción se escriben en minúsculas.
- Los métodos deberán ser verbos (en infinito), en mayúsculas cada primera letra inicial y las demás palabras que le continúen comenzarán con mayúscula también.

```
private void LoadEnemy(string level)
{
    string path = "DynamicSceneLoaderResources/Prefabs/Mutable/Enemy/" + level + "/Enemy";
    Enemy = Resources.Load(path) as GameObject;
    Enemy = Instantiate(Enemy);
    for (int i = 0; i < Enemy.transform.childCount; i++)
    {
        Enemy.transform.GetChild(i).gameObject.SetActive(false);
    }
}
```

Figure 6: Ejemplo de cómo se evidencian los estándares de codificación

2.4 Conclusiones del capítulo

Después de realizado el análisis y diseño de la propuesta de solución y haber generado los diferentes artefactos que dispone la metodología XP, se puede concluir lo siguiente:

1. El análisis de las características del videojuego permitió identificar las principales funcionalidades requeridas para la solución.
2. La identificación de los patrones de diseño y el estilo arquitectónico del componente, evidenciaron que la solución propuesta cuenta con un alto grado de resistencia ante posibles modificaciones.
3. La generación de las tarjetas CRC como artefactos requeridos por la metodología, documentaron la solución propuesta, facilitando su posterior implementación.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

En el presente capítulo se describe la fase de implementación y pruebas de la metodología usada. Una vez implementado el componente se le realizan las pruebas de funcionalidad y chequeos antes de ser presentada al cliente. Se muestra la validación de los resultados del componente en relación al cumplimiento del objetivo general de la investigación.

3.1 Interfaces principales del componente

Una vez desarrollado el componente de generación de contenidos para el videojuego Coliseum, es posible visualizar las pantallas del mismo, donde se observa el resultado obtenido durante la implementación de las historias de usuarios descritas en el subepígrafe 2.2.1.

Restricciones del videojuego

Las restricciones más significativas del componente para la generación de niveles del videojuego Coliseum son la secuencia y seguir el orden lógico de cómo está estructurado el juego. Los primeros 20 niveles corresponden al coliseo de primavera, los muros interiores del coliseo cuentan con 2 diseños (FullCircle y Squared) donde se pueden seleccionar estos tipos de muros para cada nivel, se decide entonces mantener los diseños tal y como venían establecidos en el juego original, con la posibilidad de modificar el terreno cuando se desee. La música y los enemigos se generan por nivel, teniendo en cuenta que los enemigos deben estar acorde a las estadísticas y habilidades del héroe y no deben ser ni muy fáciles ni imposibles de vencer. Las condiciones de ganar a todos los enemigos para poder pasar de nivel, así como terminar la partida si no hay enemigos en el mapa, están definidas en la configuración del mundo, así como las posiciones iniciales del héroe y las apariciones de las runas. En las imágenes siguientes se muestra el contenido de cada nivel a modo de ejemplificación.

Figure 7: Partida en el videojuego Coliseum

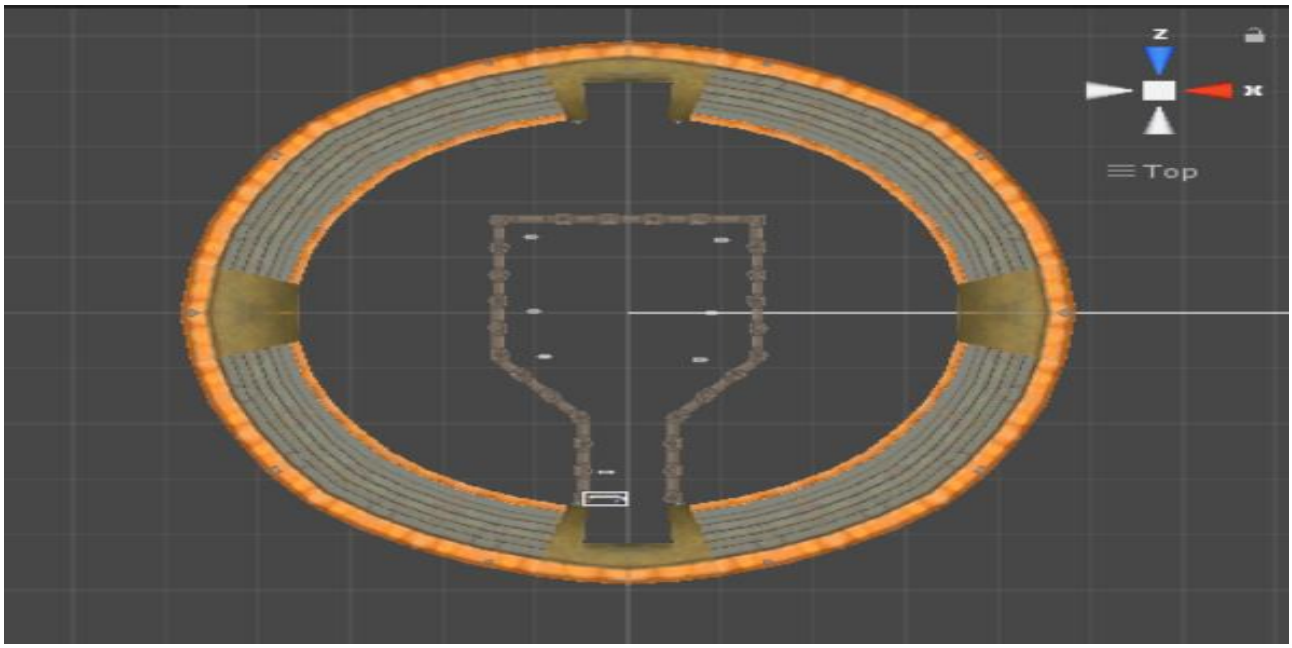

Figure 8: Diseño Squared y Coliseo de primavera

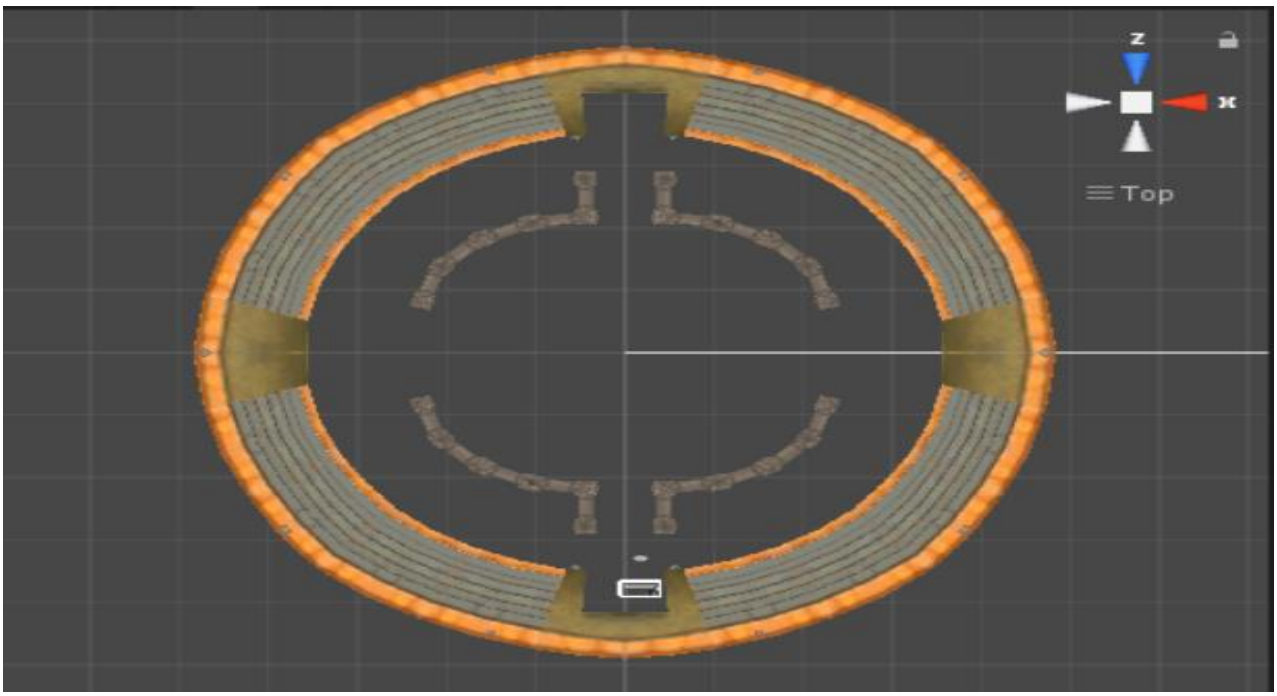

Figure 9: Diseño FullCircle y Coliseo primavera

Fase de Codificación

En esta fase se realiza la implementación de las historias de usuario definidas anteriormente. Para un desarrollo satisfactorio y mayor organización a la hora de implementar las HU se elaboran las tareas ingenieriles. Las tareas son especificadas por los desarrolladores, se realizan utilizando un lenguaje técnico y fácil de comprender. A continuación, se muestra cada iteración donde fueron implementadas las HU.

Primera iteración

En esta iteración se desarrollaron las HU 1 y 2, correspondiente con la documentación de los requisitos. Para ello se realizaron las siguientes tareas:

Tabla 14: Tarea de ingeniería Inicialización de la escena

Tarea	
Número de tarea: 1	Número de UH: 1
Nombre de la tarea: Inicialización de la escena	

Tipo de tarea: Implementación.	Estimación:5 días
Descripción: Se modificó la clase LoaderScena, para que por defecto inicialice la nueva escena dinámica creada Dynamic_Scene, cambiando la forma en la que se cargaban las escenas.	

Tabla 15: Tarea de ingeniería Construir escena

Tarea	
Número de tarea: 2	Número de UH: 2
Nombre de la tarea: Construir escena	
Tipo de tarea: Implementación.	Estimación:15 días
Descripción: Se implementó la clase DynamicSceneLoader.cs que contiene el método LoadLevel(), encargado de buscar los objetos ubicados en la carpeta de recursos e instanciarlos en la escena dinámica.	

Segunda Iteración

En esta iteración se desarrollaron las HU 3 y 4 identificadas como Cargar la escena e Iniciar el nivel. Para ello se realizaron las siguientes tareas:

Tabla 16: Tarea de ingeniería Cargar la escena

Tarea	
Número de tarea: 3	Número de UH: 3
Nombre de la tarea: Cargar la escena	
Tipo de tarea: Implementación.	Estimación:7 días
Descripción: Se implementó la clase DynamicSceneHandler.cs (Singleton) que contiene los métodos SetLevelLoaded() y IsLevelDoneLoading(), encargados de	

controlar cuándo la escena ha terminado de instanciar objetos y dar la señal de carga de la escena.

Tabla 17: Tarea de ingeniería Iniciar nivel

Tarea	
Número de tarea: 4	Número de UH: 4
Nombre de la tarea: Iniciar el nivel	
Tipo de tarea: Implementación.	Estimación:3 días
Descripción: Se modificó la clase InitGamePlayer.cs para que dé comienzo al nivel una vez que la escena esté cargada.	

3.2 Niveles de Pruebas

Para realizar las pruebas de software se debe tener en cuenta una serie de objetivos a nivel de trabajo. Éstas están agrupadas en distintas categorías o niveles en diferentes etapas del proceso de desarrollo.

1. Pruebas de integración: Las pruebas de integración se definen como un mecanismo de testeo de software, donde se realiza un análisis de los procesos relacionados con el ensamblaje o unión de los componentes, de modo que las pruebas de integración están a cargo del examen de las interfaces entre subsistemas o los grupos de componentes del programa o aplicación que se analiza, lo que contribuye a garantizar su funcionamiento correcto. Su importancia radica en que permite comprobar la interacción adecuada de los componentes del sistema, verifica que el sistema cumpla con cada una de las labores asignadas y se adapten a los requisitos (Saji, 2018).

2. Pruebas de Rendimiento: Las pruebas de rendimiento de software se definen como la comprobación del funcionamiento del sistema frente a múltiples escenarios, con el fin de examinar los componentes de la aplicación. Este tipo de pruebas se basan en garantizar la calidad y operatividad de un determinado sistema. Sirven para

comprobar si el sistema cumple o no con los parámetros de rendimiento establecidos, además de indicar que parte del sistema se debe mejorar (Medina, 2014).

3. Pruebas de aceptación: Las pruebas de aceptación representan aquella fase del ciclo de vida del desarrollo de software en la que el equipo de desarrollo tiene que garantizar que el sistema corresponde con los requerimientos definidos por el cliente y permitir al cliente que decida si acepta el producto, probándolo de conjunto con el desarrollador.

3.2.1 Integración del componente al videojuego

La integración de software es la práctica de conectar y unificar diferentes partes o subsistemas de software. Se realiza comúnmente a la hora de modificar o agregar nuevas funcionalidades. La integración permite que los sistemas que trabajan por separado, se combinen ahorrando tiempo y recursos. Favorecen el desarrollo de software, ya que se van adaptando tecnológicamente acordes a las necesidades que surgen, automatizando y favoreciendo su optimización (Saji, 2018).

Para la implementación de la propuesta de solución se creó una carpeta de recursos que contiene elementos clave como son el tipo de coliseo (primavera, verano, invierno, kalaris), la música correspondiente a cada mundo, los enemigos que corresponden a cada nivel y sus posiciones iniciales tanto del héroe como de los enemigos en el mapa. En la imagen siguiente se muestra cómo queda estructurada la carpeta de recursos en el proyecto.

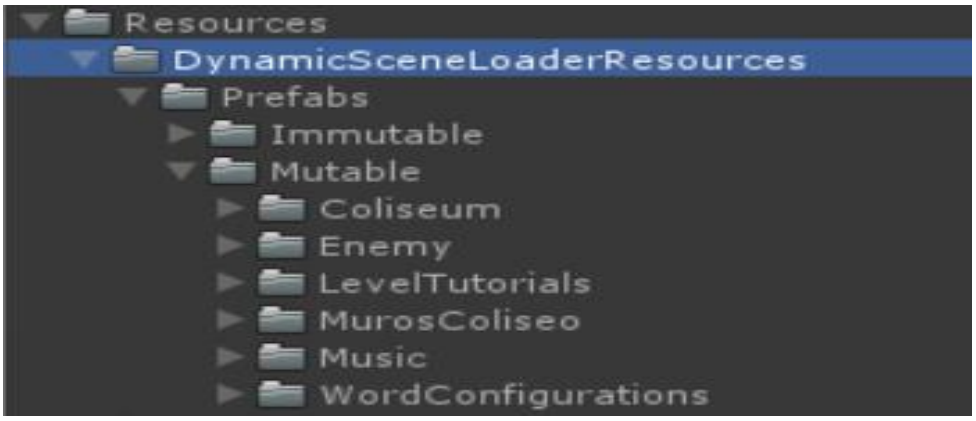

Figure 10(Carpeta de recursos)

Luego se creó una escena dinámica, encargada de generar los distintos niveles del videojuego. La escena dinámica tiene dentro de ella la clase encargada de construir el nivel (*DynamicSceneLoader.cs)*, que instancia los objetos en tiempo de ejecución apoyándose de la carpeta de recursos. De igual forma tiene los objetos comunes entre escenas (controladores, cámara, eventos del sistema, interfaces, aparición de runas). En la imagen que se muestra a continuación se evidencia cómo queda estructurada la escena.

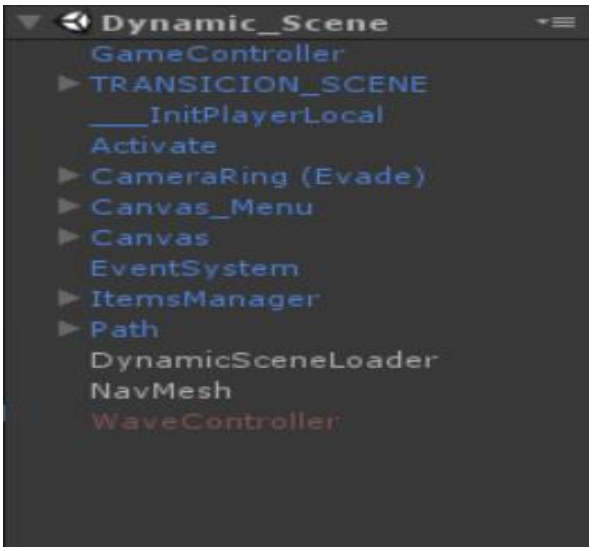

Figure 11(Escena dinámica

Una vez que se implementaron estos elementos se procedió a integrarlos al proyecto del videojuego, modificando el código de la clase con la responsabilidad de cargar la escena (LoaderScena), para que cargara por defecto la escena dinámica en vez de las antiguas escenas del proyecto. Luego se inició el juego y se escogió un nivel, como resultado se comprobó que el componente se integró satisfactoriamente al videojuego. En la siguiente imagen se muestra cómo se carga la escena *Dynamic_Scene* con el juego ejecutándose y que efectivamente instancia los objetos en la escena.

Figure 12: Evidencia de la integración del componente

Al escoger otros niveles el resultado fue el mismo, demostrando que una sola escena es capaz de generar todos los niveles del videojuego, apoyándose de una carpeta de recursos y una clase constructora. Con este resultado se precedió a eliminar las antiguas escenas del juego, 20 escenas en total que se tomaron de prueba, y de esta forma se redujo el tamaño del videojuego compilado.

3.2.2 Pruebas de rendimiento

Para ejecutar las pruebas de rendimiento se ejecutó el videojuego con el componente integrado en varios ordenadores con diferentes prestaciones, ya sea de mayor o menor procesamiento, con el objetivo de verificar el rendimiento del videojuego. Se tomó en cuenta la cantidad de fotogramas por segundos (FPS) a los que corría el videojuego para verificar la velocidad con que el sistema procesaba la información y las posibles causas de deficiencias.

A continuación, se muestran los resultados obtenidos en esta prueba:

Tabla 18: Prueba de rendimiento PC1

Prestaciones de la PC	**FPS promedio**	**FPS máximo**
I. Intel I5 2.40 GHz II. 6gb de RAM III. Windows 10 64x IV. 1920x1080	70	75
Observaciones		
• El sistema se mantuvo estable siendo ejecutado a niveles medios de gráficos y la interfaz se adaptó bien a la resolución de pantalla.		
Recomendaciones		
• Según las observaciones obtenidas se recomienda utilizar un nivel de gráficos altos.		

Tabla 19: Prueba de rendimiento PC2

Prestaciones de la PC	**FPS promedio**	**FPS máximo**
V. Intel I3 2.40 GHz VI. 4gb de RAM VII. Windows 10 64x VIII. 1366x768	60	65
Observaciones		
• El sistema se mantuvo estable siendo ejecutado a niveles medios de gráficos y la interfaz se adaptó bien a la resolución de pantalla.		
Recomendaciones		

- Según las observaciones obtenidas se recomienda utilizar un nivel de gráficos medios.

Al concluir las pruebas de rendimiento se evidencia que no es necesario corregir ningún defecto de rendimiento del videojuego, funcionado todo como es debido.

3.2.3 Pruebas de Aceptación

Este tipo de pruebas comprueba si el sistema funciona según lo esperado, siendo las últimas pruebas realizadas, el cliente verifica que el sistema cumple con sus expectativas, son pruebas funcionales y se basan en los requisitos definidos al comienzo del proyecto. Se confeccionó un caso de prueba por cada requisito funcional.

Tabla 20: Prueba de aceptación Inicializar escena

Caso de prueba de Aceptación	
Código: P1HU	**Numero de UH**: 1
Nombre: Inicializar escena	
Descripción: Se verifica el correcto funcionamiento del requisito funcional, esperando que la escena sea inicializada.	
Condiciones de ejecución: Se debe crear la escena y realizar las modificaciones para que se integre al proyecto a la hora de cargar las escenas.	
Resultado esperado: La escena se inicializa y el proyecto del videojuego la carga.	
Evaluación de la prueba: Satisfactorio	

Tabla 21: Prueba de aceptación Construir la escena

Caso de prueba de Aceptación	
Código: P2HU	**Numero de UH**: 2
Nombre: Construir la escena	
Descripción: Se verifican el correcto funcionamiento del método de construcción, siendo	

este capaz de instanciar objetos y mostrarlos en la escena creada.
Condiciones de ejecución: Se debe tener una carpeta de recursos
Resultado esperado: La herramienta instancia objetos y los agrega a la escena
Evaluación de la prueba: Satisfactorio

Tabla 22: Prueba de aceptación Cargar la escena

Caso de prueba de Aceptación	
Código: P3HU	**Numero de UH**: 3
Nombre: Cargar la escena	
Descripción: Se verifica el correcto funcionamiento del requisito, siendo capaz el sistema de cargar la escena recién construida	
Condiciones de ejecución: La escena debe enviar una señal de que está lista para cargarse.	
Resultado esperado: La escena se carga.	
Evaluación de la prueba: Satisfactorio	

Tabla 23: Prueba de aceptación Iniciar el nivel

Caso de prueba de Aceptación	
Código: P4HU	**Numero de UH**: 4
Nombre: Iniciar el nivel	
Descripción: Se verifica el correcto funcionamiento del requisito, siendo capaz de poder iniciar el nivel sin errores.	
Condiciones de ejecución: La escena debe estar cargada.	
Resultado esperado: El videojuego funciona como un todo y sin presentarse errores, permitiendo jugar el nivel.	

Evaluación de la prueba: Satisfactorio

3.3 Validación de la propuesta de solución

Para realizar la validación de la propuesta de solución se tomaron como casos de prueba una muestra de 20 niveles del total de 80 que presenta el proyecto del videojuego, para demostrar la lógica y la fundamentación teórica descrita en los sub-epígrafes anteriores. De esta forma se cumple con el objetivo de desarrollar un componente capaz de generar los niveles del videojuego Coliseum y reducir el tamaño del compilado. Mientras que una sola escena dinámica sea capaz de generar todos los elementos necesarios para cada nivel del videojuego, apoyándose de una carpeta de recursos, no se hace necesario almacenar las antiguas escenas que contenía el videojuego, por tanto, se pueden borrar del proyecto y aun así poder jugar los niveles del videojuego. Al guardar el ejecutable del videojuego en la versión original y en la modificada se puede notar que efectivamente se reduce el tamaño del videojuego.

Tabla 24: Tamaño de los compilados del videojuego

Versión Original del Proyecto	Versión Modificada del Proyecto
387 MB	372 MB

Para 20 niveles del videojuego se ha reducido un 3.88% del tamaño del compilado, se estima que los resultados sean mucho más satisfactorios si se llega a implementar la solución al videojuego completo. Esta técnica empleada es poco invasiva, pues no afecta los tiempos de carga de cada nivel y además el tiempo de compilación del proyecto se redujo en gran medida, ya que solo se tiene que construir una escena.

3.4 Conclusiones del capítulo

1. El correcto uso de los estándares de codificación, facilitó la comprensión de la estructura general del sistema a través de sus componentes.
2. La implementación del componente y su posterior validación, evidenció el cumplimiento del objetivo propuesto, logrando reducir en un 3.88% el compilado del videojuego para 20 niveles.
3. La ejecución de la estrategia de pruebas especificadas permitió garantizar un componente funcional que reconoce las especificaciones del cliente.

CONCLUSIONES GENERALES

El desarrollo de la presente investigación permitió arribar a las siguientes conclusiones generales:

- El empleo de la técnica de Generación Procedural de Contenido “Basado en restricciones” contribuyó a la generación de niveles del videojuego Coliseum.
- La creación de una única escena dinámica construida a partir del almacén de recursos comunes definidos para cada nivel del videojuego, permitió reducir en un 3.88% el tamaño del compilado para 20 niveles.
- El componente desarrollado contribuyó a la investigación de la generación procedural de contenido en videojuegos y proyectos de la universidad.

RECOMENDACIONES

Para el desarrollo de futuras investigaciones relacionadas con la presente, se propone:

- Implementar la solución propuesta para los 80 niveles del videojuego Coliseum.
- Actualizar el proyecto a nuevas versiones de Unity para poder implementar diferentes métodos que reduzcan el tamaño del juego compilado como son: Unity Advanced Asset Managment, Unity Addressables, Async Asset Managment.
- Investigar otras técnicas y algoritmos para la generación automática de los niveles.

REFERENCIAS BIBLIOGRÁFICAS

Aíbar, V. (2021). *Diseño e implementación de un servicio generador de mapas procedurales para videojuegos modelados como sistemas distribuidos*. Obtenido de https://riunet.upv.es/bitstream/handle/10251/174105/Vilanova%20-%20Diseno%20e%20implementacion%20de%20un%20servicio%20generador%20de%20mapas%20procedurales%20para%20videojuego....pdf?sequence=1&isAllowed=y

Azkue, J. d. (1 de enero de 2022). *Generación procedural*. Obtenido de IDIS: https://proyectoidis.org/generacion-procedural/

Baños, G. (2016). *desarrollo del diseño y grafismo de un videojuego*. Obtenido de https://upcommons.upc.edu/handle/2117/89226

Caparrini, F. S. (30 de 10 de 2016). *Autómatas Celulares*. Obtenido de http://www.cs.us.es/~fsancho/?e=66

Cardozzo, D. (2016). *Desarrollo de Software*. Obtenido de Books: https://books.google.es/books?hl=es&lr=&id=tBaYCwAAQBAJ&oi=fnd&pg=PA1&dq=requisitos+de+software+pressman&ots=uGwRrf8JL-&sig=iOY-kTelGTkLby69aoHptZ7Uxhs#v=onepage&q=requisitos%20de%20software%20pressman&f=false

Chaparro, R. (2020). *Diagramas esenciales del lenguaje unificado de modelado*. Obtenido de https://repository.unad.edu.co/jspui/bitstream/10596/38052/1/cvramirezc.pdf

Clumsy. (2022). *Cross Play – Qué es, Significado*. Obtenido de https://clumsygames.com/diccionario-gamer/cross-play/

Codes, P. R. (2015). *Generador procedimental de terrenos basado en restricciones*. Obtenido de https://oa.upm.es/43527/1/TFG_PABLO_RODRIGUEZ_CODES.pdf

Concepcion, J. R. (23 de abril de 2020). *ConWiro, el estudio independiente de videojuegos en Cuba*. Obtenido de Fonoma Blog: https://blog.fonoma.com/conwiro-videojuegos-cuba-ffdb59e91cf3

Darkcrizt. (2019). *Unity 3D, el motor de juegos* . Obtenido de https://blog.desdelinux.net/unity-3d-el-motor-de-juegos-se-ha-actualizado-a-su-nueva-version-2019-2/

Domínguez. (2019). Obtenido de http://www.cubadebate.cu/especiales/2019/12/20/la-sacerdotisa-irianys-llega-con-el-moba-de-coliseum-3-0/

Estapé, J. A. (24 de 08 de 2019). *Inteligencia artificial: qué es, cómo funciona y para qué se utiliza en la actualidad* . Obtenido de ComputerHoy:

https://computerhoy.com/reportajes/tecnologia/inteligencia-artificial-469917

Estrada, E. (2011). *Trabajo de diploma*. Obtenido de https://dspace.uclv.edu.cu/bitstream/handle/123456789/7732/Trabajo%20de%20Diploma%20final.pdf?sequence=1&isAllowed=y

F Cueto, J. J. (2015). *Diagrama de clases*. Obtenido de https://d1wqtxts1xzle7.cloudfront.net/37472390/31096724-Diagrama-de-Clases-en-UML-with-cover-page-v2.pdf?Expires=1670244905&Signature=ST4-ulN5B9Co-P3KTySgVy9zia0glU2B7ebTRY5UgZPVxag3aZTqPt~ImU~uxsT6ImuGHXjuMwB~GCuNbDJbZqyvLiln6XFmbgPXNe9KvaYuBZOOjv8JZeRX-

Farinós, R. (2014). *Analisis de la industria de los videojuegos*. Obtenido de riunet: https://riunet.upv.es/handle/10251/45702

J Arranz de la Peña, A. P. (2007). *Algoritmos genéticos*. Obtenido de https://d1wqtxts1xzle7.cloudfront.net/35889597/Algoritmos_Geneticos_app-with-cover-page-v2.pdf?Expires=1670245881&Signature=dL~ndd38lH-GS4QqcuYV6tkAXEGUbdCEqcytUEyyEECU1Tow6NG2Hn5L7bqZUoNfg5~VuGu0ftHPrpYJ5fWaPyt0LxKh33orrlcG~wN8ys5Zi7Bkp7WYLLaay30Mj32RyZ~

Julian Togelius, G. N. (2011). *Search-based Procedural Content*. Obtenido de http://julian.togelius.com/Togelius2011Searchbased.pdf

Lara-Cabrera. (2015). *Generacion Automatica de Contenido para videojuegos*. Obtenido de Riuma: https://riuma.uma.es/xmlui/bitstream/handle/10630/11713/TD_LARA_CABRERA_Raul.pdf?sequence=1

Letelier, P. (2106). *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)*. Obtenido de http://www.cyta.com.ar/ta0502/b_v5n2a1.htm

Lyons, B. (25 de 05 de 2019). *La historia completa de Minecraft, bloque a bloque*. Obtenido de GAMEREACTOR: https://www.gamereactor.es/la-historia-completa-de-minecraft-bloque-a-bloque/

Manobanda Tuapanta, E. R. (2020). *Análisis de metodologías Scrum y XP en la implementación de un sistema multiplataforma de gestión en el Banco de Germoplasma de la Universidad Técnica de ….* Obtenido de http://repositorio.utc.edu.ec/bitstream/27000/6865/1/UTC-PIM-000223.pdf

Medina, J. (2014). *Pruebas de rendimiento*. Obtenido de https://books.google.es/books?hl=es&lr=&id=kTvbBgAAQBAJ&oi=fnd&pg=PA9&dq

=prueba+de+rendimiento&ots=wPoqoqYbQU&sig=yAH_GkikOa5fyqhRSzi-jfxRthM#v=onepage&q=prueba%20de%20rendimiento&f=false

Microsoft. (22 de 9 de 2022). *Informacion general C#*. Obtenido de https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/

Montero, B. M. (7 de 2018). *Metodologías ágiles frente a las tradicionales en*. Obtenido de https://www.researchgate.net/profile/Harry-Vite-Cevallos/publication/327537074_Metodologias_agiles_frente_a_las_tradicionales_en_el_proceso_de_desarrollo_de_software/links/5b942061a6fdccfd542a2b13/Metodologias-agiles-frente-a-las-tradicionales-en-el-proce

Pemau, M. C. (2020). *Desarrollo de un Videojuego en UNITY* . Obtenido de https://oa.upm.es/58120/1/TFG_MARIO_DE_LA_CALLE_PEMAU.pdf

Poyatos, F. (2017). *Evolución del sector del videojuego desde una perspectiva estratégica*. Obtenido de https://tauja.ujaen.es/handle/10953.1/6595

Ramón, M. d. (7 de 10 de 2016). *¿Existe una industria de videojuegos en Cuba?* Obtenido de CUBADEBATE: http://www.cubadebate.cu/noticias/2016/10/07/existe-una-industria-de-videojuegos-en-cuba/

Reinaga, H. (2009). *Aspectos Tempranos: un enfoque basado en Tarjetas CRC*. Obtenido de https://revistas.unal.edu.co/index.php/avances/article/view/14448

Requena Farinós, C. (2014). *Industria de los videojuegos en España.* Obtenido de riunet: https://riunet.upv.es/handle/10251/45702

Reynoso , C. (2004). *Introducción a la Arquitectura de Software*. Obtenido de https://d1wqtxts1xzle7.cloudfront.net/64699769/Arquitectura_software-with-cover-page-v2.pdf?Expires=1670244736&Signature=V8KhqR3k1o-4vP4c122gIAMbQT9Kqj0-PgTAtm24ZBsurYZTAmiAIjlR80jmmTgeDRZXvM-S20vKH2Fab8cc5t75deh3Kh8bbDJZ~cxZ0R5gsNh0ylW7ZSmqfhZYPR1ytgRg4-

Rouhiainen, L. (2018). *Inteligencia Artificial.*

Ruiz-Moyano. (2017). *Memoria Generación Automática de Mecánicas de Juego*. Obtenido de Riuma.

Saji, G. (2018). *Modelo de Definición, Pruebas e integración de herramientas tecnológicas al Proceso de evaluación del desarrollo de las competencias*. Obtenido de http://190.119.145.154/handle/UNSA/7363

Sánchez, R. (2020). *Análisis de metodologías Scrum y XP*. Obtenido de http://repositorio.utc.edu.ec/bitstream/27000/6865/1/UTC-PIM-000223.pdf

Soria, C. (2014). *ALGORITMOS GENÉTICOS* . Obtenido de Dialnet.

Tokio. (28 de 12 de 2020). *La inteilgencia artificial en los videojuegos*. Obtenido de https://www.tokioschool.com/noticias/inteligencia-artificial-videojuegos/

Trenta, M. (2012). *Orígenes del videojuego: conexiones históricas y sociales de un producto cultural*. Obtenido de https://www.revistalatinacs.org/12slcs/2012_actas/189_Trenta.pdf

Vaz, B. (2020). *'Hades', la maldición de Sísifo* . Obtenido de ELESPAÑOL: https://www.elespanol.com/el-cultural/blogs/homo_ludens/20201029/hades-maldicion-sisifo/532066795_12.html