Universidad Autónoma del Estado de Hidalgo

Instituto de Ciencias Básicas e Ingeniería

Licenciatura en Ciencias Computacionales

TESIS

Para obtener el grado de Licenciado en Ciencias Computacionales

Propuesta de metodología para la ingeniería de software y proceso creativo para el desarrollo de un videojuego en 2D

Presenta

José Armando Rodríguez Cortez

Director:

M.C.C. Gonzalo Alberto Torres Samperio

Comité tutorial:

M.C.C. Gonzalo Alberto Torres Samperio

Dr. Juan Carlos González Islas

Dr. Edgar Olguín Guzmán

Mtro. Arturo Curiel Anaya

Universidad Autónoma del Estado de Hidalgo
Instituto de Ciencias Básicas e Ingeniería
School of Engineering and Basic Sciences

Mineral de la Reforma, Hgo., a 02 de diciembre de 2025

Número de control: ICBI-D/3057/2025
Asunto: Autorización de impresión.

MTRA. OJUKY DEL ROCÍO ISLAS MALDONADO
DIRECTORA DE ADMINISTRACIÓN ESCOLAR DE LA UAEH

Con Título Quinto, Capítulo II, Capítulo V, Artículo 51 Fracción IX del Estatuto General de nuestra Institución, por este medio, le comunico que el Jurado asignado al egresado de la Licenciatura en Ciencias Computacionales **José Armando Rodríguez Cortez,** quien presenta el trabajo de titulación **"Propuesta de metodología para la ingeniería de software y proceso creativo para el desarrollo de un videojuego en 2D",** ha decidido, después de revisar fundamento en lo dispuesto en el Título Tercero, Capítulo I, Artículo 18 Fracción IV; dicho trabajo en la reunión de sinodales, **autorizar la impresión del mismo**, una vez realizadas las correcciones acordadas.

A continuación, firman de conformidad los integrantes del Jurado:

Presidente: Dr. Edgar Olguín Guzmán

Secretario: Dr. Juan Carlos González Islas

Vocal: M.C.C. Gonzalo Alberto Torres Samperio

Suplente: M.C.C. Arturo Curiel Anaya

Sin otro particular por el momento, reciba un cordial saludo.

Atentamente
"Amor, Orden y Progreso"

Mtro. Gabriel Vergara Rodríguez
Director del ICBI

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO ICBI DIRECCIÓN

GVR/YCC

Ciudad del Conocimiento, Carretera Pachuca-Tulancingo Km. 4.5 Colonia Carboneras, Mineral de la Reforma, Hidalgo, México. C.P. 42184
Teléfono: 771 71 720 00 Ext. 40001
direccion_icbi@uaeh.edu.mx, vergarar@uaeh.edu.mx

Universidad Autónoma del Estado de Hidalgo
Instituto de Ciencias Básicas e Ingeniería
School of Engineering and Basic Sciences
Área Académica de Computación y Electrónica
Department of Electronics and Computer Science

Mineral de la Reforma, Hgo., a 05 de diciembre del 2025

Número de control: ICBI-AACyE/2746/2025
Asunto: Integración en el repositorio institucional.

MTRO. JORGE EDUARDO PEÑA ZEPEDA
DIRECTOR DE BIBLIOTECAS Y CENTROS DE INFORMACIÓN.

Por medio del presente, hago constar que la tesis en formato digital titulado: **"Propuesta de metodología para la ingeniería de software y proceso creativo para el desarrollo de un videojuego en 2D"**, que presenta el alumno **José Armando Rodríguez Cortez** con número de cuenta **419062**, es la versión final validada por el Comité Tutorial y cumple con el oficio de autorización de impresión, por lo que solicito su integración en el repositorio institucional de tesis.

Sin otro particular, me despido de usted.

Atentamente
"Amor, Orden y Progreso"

Dra. Anilú Franco Árcega
Coordinadora de la Licenciatura en Ciencias Computacionales

José Armando Rodríguez Cortez

AFA / KGB

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO · INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA · UAEH · ÁREA ACADÉMICA DE COMPUTACIÓN Y ELECTRÓNICA

Ciudad del Conocimiento, Carretera Pachuca-Tulancingo Km. 4.5 Colonia Carboneras, Mineral de la Reforma, Hidalgo, México. C.P. 42184
Teléfono: 52 (771) 71 720 00 Ext. 40052, 40053
aacye_icbi@uaeh.edu.mx, jesus_ordaz@uaeh.edu.mx

Contenido

Índice de figuras 6
Agradecimientos y Dedicatoria 8
Resumen 9
Abstract 10
Introducción 11
Problemática 12
Propuesta de solución 14
1.- Adopción de una Metodología Ágil Adaptada 14
2.- Implementación de Herramientas Colaborativas Integradas 14
3.- Establecimiento de Protocolos de Comunicación y Documentación 15
4.- Prototipado Iterativo y Pruebas de Usuario Tempranas 15
5.- Evaluación Continua y Mejora del Proceso 15
Justificación 16
Antecedentes 18
Objetivos 19
Objetivo General 19
Objetivos Específicos 19
Alcances y limitaciones 20
Alcances 20
Limitaciones 20
Capítulo 1 Marco teórico y Conceptual 22
1.1 Ingeniería de Software en el Desarrollo de Videojuegos 22
1.1.1 Metodologías de Desarrollo de Software 22
1.1.2 Arquitectura de Software en Videojuegos 2D 22
1.1.3. Proceso Creativo en el Desarrollo de Videojuegos en 2D 23
1.1.3.1 Etapas del Proceso Creativo 23
1.1.4. Intersección entre Ingeniería de Software y Proceso Creativo 23
1.1.4.1 Tensiones entre lo Técnico y lo Creativo 23
1.1.4.2 Metodologías Híbridas y Colaboración Efectiva 24
1.1.5. Herramientas y Tecnologías en el Desarrollo de Videojuegos en 2D 24
1.1.6.- Ingeniería de software 25

1.1.6.1.- Ciclo de vida del Desarrollo de Software 25
1.1.6.2.- Principios de la ingeniera de software 25
1.1.6.3.- Herramientas de la ingeniería de software 25
1.1.7.- Procesos Creativos 26
1.1.7.1.- Etapas del Proceso Creativo 26
1.1.7.2.- Técnicas y Estrategias Creativas 26
1.1.7.3.- Herramientas para el Proceso Creativo 26
1.1.8.- Optimización 27
1.1.8.1.- Tipos de Optimización 27
1.1.8.2.- Etapas del Proceso de Optimización 27
1.1.8.3.- Técnicas y Métodos de Optimización 28
1.1.8.4.- Aplicaciones de la Optimización 28
1.1.8.5.- Importancia de la Optimización 29
1.1.9.- Optimización de Software 29
1.1.9.1.- Tipos de Optimización 29
1.1.9.2.- Etapas del Proceso de Optimización en Software 30
1.1.9.3.- Herramientas de Optimización en Software 31
1.1.9.4.- Aplicaciones de la Optimización de Software 31
1.1.10.- Desarrollo de Videojuegos 31
1.1.10.1.- Fases, elementos clave y técnicas de desarrollo 31
1.2.- Estado del Arte 33
Capítulo 2 Metodología 35
1.- Modelo en Cascada 35
1.1 Fases del Modelo en Cascada 35
2.- Programación Extrema (XP) 37
2.1.- Fases de la Programación Extrema 37
3.- Desarrollo Ágil 39
3.1.- Fases del Desarrollo Ágil 39
Cuadro Comparativo 41
Elección de Metodología 42
Capítulo 3 Diseño y Desarrollo 43
1.-Planificación 43
2.- Desarrollo e Implementación 44

Creación de Código 50
Capítulo 4 Resultados 68
4.1- Encuestas Realizadas 73
4.2- Estadísticas 74
4.3.- Retrospectiva y Pruebas 81
Capítulo 5 Conclusiones y trabajo futuro 82
Glosario 84
Referencias 88

Índice de figuras

Ilustración 1 Diseño de Nivel 1 44
Ilustración 2 Enemigo Slime 45
Ilustración 3 Enemigo Slime de Fuego 45
Ilustración 4 Obstáculo Pinchos 46
Ilustración 5 Obstáculo Plataformas Elevada 46
Ilustración 6 Objeto Amatista 47
Ilustración 7 Objeto Corazón 47
Ilustración 8 Punto de transporte 1 48
Ilustración 9 Personaje del Juego 49
Ilustración 10 Punto de Transporte 2 49
Ilustración 11 Código Player 1 Parte 1 51
Ilustración 12 Código Player Parte 2 52
Ilustración 13 Código Player 3 Parte 3 53
Ilustración 14 Código Player 4 Parte 4 54
Ilustración 15 Código enemigo parte 1 55
Ilustración 16 Código Enemigo Parte 2 55
Ilustración 17 Código Enemigo Parte 3 56
Ilustración 18 Código Parámetros del Juego 58
Ilustración 19 Código enemigo flama Parte 1 59
Ilustración 20 Código enemigo Flama Parte 2 60
Ilustración 21 Código Enemigo Flama Parte 3 60
Ilustración 22 Código Obstáculo con Picos 61
Ilustración 23 Código Objeto Monedas 61
Ilustración 24 Código Objeto Vidas 62
Ilustración 25 Código Objeto Amatista 62
Ilustración 26 Código Cambio de Nivel 63
Ilustración 27 Interfaz de FL Studio 64
Ilustración 28 interfaz BandLab 64
Ilustración 29 Nivel 2 65
Ilustración 30 Enemigo Camión 65
Ilustración 31 Enemigo Murciélago 66
Ilustración 32 Nivel 3 66
Ilustración 33 Enemigo Esqueleto 67
Ilustración 34 Modelo UML 68
Ilustración 35 Inicio del prototipo de juego 69

Ilustración 36 Tutorial del prototipo 70
Ilustración 37 Prototipo nivel 1 70
Ilustración 38 Primer punto de transporte 71
Ilustración 39 Prototipo Nivel 2 71
Ilustración 40 Punto de transporte del segundo nivel 72
Ilustración 41 Prototipo Tercer nivel 72
Ilustración 42 Primera pregunta 74
Ilustración 43 Segunda pregunta 75
Ilustración 44 Tercera pregunta 76
Ilustración 45 Cuarta pregunta 76
Ilustración 46 Quinta pregunta 77
Ilustración 47 Sexta pregunta 78
Ilustración 48 Séptima pregunta 78
Ilustración 49 Octava pregunta 79
Ilustración 50 Novena pregunta 80
Ilustración 51 Decima pregunta 80

Agradecimientos y Dedicatoria

Quiero agradecer esto a todas las personas que me ayudaron en todo este proceso, desde aquel agosto de 2019, hasta hoy día que estoy terminando este trabajo.

A los amigos que de alguna u otra manera me dieron apoyo en seguir y no rendirme, aunque creyera que no podría terminarlo.

A mi asesor, que estuvo pendiente en cada revisión, en cada corrección y siempre me guio con paciencia y dirección.

A mi familia por siempre confiar en mí y motivarme a no defraudarlos, a mi tía por haber confiado en mí, dándome todo tipo de apoyo que estuvo a su alcance.

Y, por último, siendo la persona más importante en todo este proceso, a mi madre, que es la razón por la cual hoy puedo realizar este trabajo, porque gracias a ella, llegue a este punto de mi vida, porque las veces que tenía que levantarme a las 5 de la mañana para poder llegar puntual a la secundaria ella ya estaba despierta desde las 4, porque a pesar de que tuve muchos fallos y tropiezos por todo este camino que tuve desde primaria hasta el nivel universidad, ella jamás me dio la espalda y me quito su apoyo, aunque muchas veces no lo haya aprovechado como hubiera debido.

Te dedico esta tesis a ti má, porque este logro más que mío, es tuyo porque sin ti, nada de esto hubiera sido posible, gracias por todo.
Te amo mama.

Resumen

Este trabajo se enfocó en explicar las fases del proceso creativo, el cómo se crea la programación para un prototipo de videojuego en 2D, cómo funciona la optimización, explicar varios tipos de metodologías, y por sobre de todo, la implementación de una metodología hibrida que combina las mejoras practicas del proceso creativo, y de la ingeniería de software, ya que gracias a las investigaciones que se hicieron para la creación de este trabajo, se pudo notar que no hay una metodología orientada a la versatilidad y cambios repentinos que puede tener el proceso creativo sin tener que modificar, retrasar o afectar a la ingeniería de software, ya que esta no esta tan orientada a esos cambios repentinos.

Lo que se buscó en primera instancia fue estudiar las diferentes metodologías y que se plasmaran en este trabajo para tener un punto de salida, y de ahí partir hacia lo que la metodología buscaba comprobar, si era una buena propuesta de metodología para crear un prototipo de videojuego en 2D.

El siguiente paso fue estudiar sobre optimizaciones, herramientas, y procesos de desarrollos sobre videojuegos, que fue en la parte donde se puso especial atención para que el prototipo pudiera lograrse dentro del tiempo especificado, ya que al haber combinado las mejores prácticas del proceso creativo y de la ingeniería de software, se buscaba que los tiempos de cada sprint, tanto de la ingeniera de software como del proceso creativo se hicieran en menor tiempo, buscando también aplicar un poco sobre lo aprendido de optimización.

Una vez habiendo logrado lo mencionado en el párrafo anterior, se inició con el desarrollo del prototipo de videojuego en 2D, que si bien, se encontraron con ciertos problemas a la hora de su elaboración, tanto del lado de la ingeniería de software como del proceso creativo, se manejó el calendario establecido de una manera brillante por parte de ambos equipos, ya que los sprints se respetaron en su totalidad, haciendo que ambas partes pudieran trabajar de la mano sin retrasarse mutuamente.

El proceso de creación demoro 10 semanas aproximadamente, haciendo una labor de 6 horas diarias, se tuvo un presupuesto de 500 pesos mexicanos con lo cual, para el proyecto, se compró un curso de Domestika (Véase referencias) para poder tener de apoyo adicional al solucionar algunos problemas de programación que se tuvieron, el equipo de desarrollo estuvo conformado por una sola persona, al igual que el equipo creativo, siendo en ambos casos, la misma persona.

Al final, se pudo obtener un prototipo funcional, jugable, escalable y actualizable a largo plazo, aplicando todo lo estudiado de optimización, programación y, sobre todo, aplicando la metodología que se propuso en este trabajo.

Abstract

This work will focus on explaining the phases of the creative process, how the programming for a 2D video game prototype is created, how optimization works, explaining various types of methodologies, and above all, the implementation of a hybrid methodology that combines the best practices of the creative process and software engineering, thanks to the research that was done for the creation of this work, it was noted that there is no methodology oriented to versatility and sudden changes that the creative process can have, without having to modify, delay or affect software engineering, since it is not so oriented to those sudden changes.

The initial objective was to study the different methodologies and have them reflected in this work to provide a starting point, and from there, move on to what the methodology sought to verify: whether it was a good methodology for creating a 2D video game prototype.

The next step was to study optimizations, tools, and development processes for video games, which was the part where special attention was paid to ensuring that the prototype could be achieved within the specified time, since by combining the best practices of the creative process and software engineering, the goal was to achieve a shorter time for each sprint, both for software engineering and the creative process, while also seeking to apply some of what was learned about optimization.

Once having achieved what was mentioned in the previous paragraph, the development of the 2D video game prototype began, which although they encountered certain problems at the time of its elaboration, both on the side of software engineering as well as the creative process, the established calendar was handled in a brilliant way by both teams, since the sprints were fully respected, allowing both parties to work hand in hand without delaying each other.

The creation process took approximately 10 weeks, with 6 hours of work per day. The project budget was 500 Mexican pesos, which is why a Domestika course (see references) was purchased for the project to provide additional support in troubleshooting some programming issues. The development team consisted of a single person, as did the creative team, who were the same person in both cases.

In the end, a functional, playable, scalable, and long-term updatable prototype was achieved by applying everything studied in optimization, programming, and, above all, by applying the methodology proposed in this work.

Introducción

Según Product Hackers (2022), la industria de los videojuegos ha experimentado un crecimiento exponencial en las últimas décadas y se encuentra entre las formas de entretenimiento más influyentes y rentables del mundo. En este amplio ámbito, los juegos arcade en 2D no solo evocan recuerdos de generaciones de jugadores anteriores, sino también brindan un legado creativo enriquecedor para los desarrolladores que buscan innovar en un formato accesible y ampliamente comprendido, puesto que sigue siendo relevante aún en la actualidad.

El diseño de videojuegos 2D requiere la colaboración de la ingeniería de software y los procesos creativos, aunque estos campos pueden desempeñarse de manera distinta. El proceso creativo se enfoca en la narración, el diseño visual y otros elementos fundamentales que impulsan el éxito de un videojuego e impactan la experiencia del usuario, mientras que la ingeniería de software proporciona lo necesario para llevar a cabo juegos de manera eficiente, segura y escalable.

El diseño y desarrollo de videojuegos 2D requiere una comprensión profunda de los principios y métodos de ingeniería de software como el control de calidad, la gestión de versiones y el ciclo de vida. Mientras tanto, debemos establecer un entorno en el que el concepto se prototipe, evalúe y refine hasta que se desarrolle una visión coherente que resuene con el público objetivo.

Esta investigación analiza la intersección de la ingeniería de software y el proceso creativo ya que busca cubrir las etapas clave del desarrollo de videojuegos 2D y analiza las formas más efectivas de gestionar dichos proyectos.

También analiza herramientas y métodos para convertir lo conceptual en productos viables, además, para garantizar que el producto final no solo cumpla con los estándares técnicos, sino que también proporcione una experiencia atractiva para los jugadores, se analizan los desafíos comunes que surgen durante el desarrollo de videojuegos y se proponen métodos para superarlos.

En última instancia, el objetivo de este proyecto es proponer un marco teórico y práctico para aquellos interesados en el desarrollo de videojuegos 2D desde una perspectiva técnica, creativa o de gestión. Este proyecto busca enfatizar los aspectos técnicos y creativos que se combinan para crear videojuegos que no solo funcionan bien, sino que también capturan la imaginación y el interés de los jugadores.

Problemática

El desarrollo de videojuegos 2D, aunque simplificado en comparación con los proyectos 3D, plantea desafíos importantes que requieren una cuidadosa integración de la ingeniería de software y los procesos creativos. Hablando sobre la ingeniería de software, los desarrolladores enfrentan la complejidad de crear código eficiente, estándar y escalable intentando admitir el diseño de juegos sin sacrificar el rendimiento ni la estabilidad. Por otra parte, el proceso creativo requiere una flexibilidad constante, lo que permite cambios rápidos en el diseño visual, la narración y la mecánica del juego para lograr una experiencia final coherente e innovadora.

Uno de los principales problemas que surge durante esta integración es la falta de métodos y herramientas capaces de equilibrar los requisitos técnicos y creativos, ya que tradicionalmente, los proyectos de ingeniería de software se gestionan mediante métodos estrictos y estructurados, como el desarrollo ágil o en cascada, que priorizan la planificación detallada y la gestión de riesgos, pero por otro lado, el proceso creativo suele ser más complicado y exploratorio, con ideas que evolucionan y cambian rápidamente, respondiendo con nuevas inspiraciones o comentarios de pruebas de usuarios.

Lo comentado por Isidro Ros en su nota en muycomputer (2021), los equipos de desarrollo a menudo enfrentan decisiones difíciles, como elegir entre mantener el código estable o incorporar cambios innovadores que podrían cambiar la estructura del software, esto puede conducir a un ciclo de desarrollo improductivo en el que la creatividad se ve limitada por ciertas limitaciones técnicas o el código se vuelve inflado debido a constantes cambios no planificados.

También existe el desafío adicional de la comunicación entre los equipos técnico y creativo ya que la falta de un lenguaje común y de herramientas de colaboración eficaces puede provocar malentendidos y desalineación de los objetivos, lo que repercute negativamente en la calidad y la cohesión del producto final.

Estos problemas pueden llevar a videojuegos que, aunque técnicamente funcionales, no logran realizar la visión creativa original ni ofrecer una experiencia de usuario satisfactoria, o por otro lado, puede llevar a videojuegos que logran realizar la visión creativa original, pero no funcionales del todo técnicamente, por lo tanto, existe la necesidad de investigar y desarrollar métodos y herramientas que permitan una integración perfecta entre la ingeniería de software y los procesos creativos en el desarrollo de videojuegos 2D.

Es importante encontrar un equilibrio que permita a los equipos técnicos y creativos trabajar juntos, aumentando la eficiencia y la calidad del producto final sin comprometer la creatividad ni la estabilidad.

Resolver este problema no sólo mejorará el desarrollo de videojuegos 2D, sino que también proporcionará un marco replicable para otros proyectos de la industria de los videojuegos.

Propuesta de solución

Para abordar la problemática identificada, se propone la inclusión de un marco metodológico híbrido que combine las mejores prácticas de la ingeniería de software con enfoques que promuevan la flexibilidad creativa. Este marco, que denominaremos **"Desarrollo Creativo Ágil" (DCA)**, buscara crear un equilibrio dinámico entre la estabilidad técnica y la libertad artística, buscando optimizaciones en el flujo de trabajo y garantizando la calidad del producto final.

1.- Adopción de una Metodología Ágil Adaptada

El desarrollo ágil ha demostrado ser eficaz en proyectos que requieren iteraciones rápidas y adaptación a cambios. Sin embargo, para que sea aplicable en el contexto del desarrollo de videojuegos 2D, será necesario ajustar sus principios para permitir un mayor enfoque en la creatividad. Esto se puede lograr mediante la integración de sprints, donde se prioricen las exploraciones artísticas y de diseño antes de pasar a una implementación técnica.

En este marco, cada ciclo de desarrollo incluiría:

- **Sprints Técnicos:** Centrados en la implementación, optimización y pruebas de características del juego.
- **Sprints Creativos:** Dedicados a la ideación, prototipado y evaluación de aspectos visuales, narrativos y de jugabilidad.

Ambos tipos de sprints se alternarían y estarían interconectados mediante revisiones conjuntas, asegurando que las decisiones creativas se realicen con una comprensión clara de las limitaciones técnicas, y viceversa.

2.- Implementación de Herramientas Colaborativas Integradas

Para facilitar la comunicación y el trabajo conjunto entre los equipos técnicos y creativos, se propone la adopción de herramientas de desarrollo que integren la gestión del código con la gestión de recursos creativos. Plataformas como Github o GitLab pueden ser expandidas para incluir plugins o integraciones que permitan a los diseñadores visualizar y modificar directamente los elementos de arte, sonido y diseño narrativo dentro del entorno de desarrollo.

Estas herramientas deberían soportar:

- **Versiones Creativas:** Permitiendo al apartado creativo iterar en sus diseños sin interrumpir el flujo de trabajo de los desarrolladores.
- **Comentarios Cruzados:** Facilitando la comunicación donde los equipos puedan comentar cambios y ajustes, asegurando una alineación continua entre la visión creativa y la implementación técnica.

3.- Establecimiento de Protocolos de Comunicación y Documentación

Para reducir los malentendidos y la desalineación de objetivos, se sugiere la creación de protocolos de comunicación claros y una documentación accesible para todos los miembros del equipo. Esto incluiría la definición de un lenguaje común que unifique términos técnicos y creativos, y la utilización de herramientas de documentación colaborativa como Confluence o Notion.

La documentación continua y la comunicación frecuente se estructurarían de la siguiente manera:

- **Reuniones Semanales:** Donde se revisen los avances de los sprints técnicos y creativos, permitiendo ajustes en tiempo real.
- **Documentación Accesible:** Esta documentación se deberá actualizar conforme a los cambios en el diseño y desarrollo, ofreciendo una referencia constante para todos los miembros del equipo.

4.- Prototipado Iterativo y Pruebas de Usuario Tempranas

Finalmente, se propone un enfoque centrado en el usuario final, mediante la creación de prototipos jugables tempranos y la realización de testeos de usuarios frecuentes, estos prototipos permitirán evaluar la coherencia entre la visión creativa y la experiencia de juego real, junto con posibles problemas antes de que se integren en la versión final del juego.

Los ciclos de pruebas incluirían:

- **Pruebas Internas Rápidas:** Realizadas al final de cada sprint creativo para ajustar la dirección antes de la implementación técnica.
- **Pruebas Externas con Usuarios:** En las fases medias del desarrollo, para recoger comentarios y criticas reales y ajustar las mecánicas de juego y el diseño basado en las expectativas y preferencias del público objetivo.

5.- Evaluación Continua y Mejora del Proceso

El marco DCA debe ser flexible y evolucionar con el proyecto. Por lo tanto, se recomienda la evaluación continua de su efectividad a través de retrospecciones después de cada ciclo completo de sprints técnicos y creativos. Esto permitirá ajustes y mejoras en la metodología para adaptarse a las necesidades específicas del proyecto y del equipo.

Justificación

El desarrollo de videojuegos 2D ha resurgido en la última década, impulsado por la nostalgia y el surgimiento de desarrolladores independientes que ven este formato como una alternativa viable, ya que es una herramienta poderosa y accesible para ellos.

En la nota de 33bits de Pedro Diaz San Miguel (2019) en la industria de los videojuegos, los videojuegos 2D siguen siendo una parte importante del ecosistema, sobresaliendo en géneros como los de plataformas, aventuras, juegos de rol, entre muchos otros, sin embargo, este proceso de creación de videojuegos plantea desafíos específicos que requieren una profunda integración entre la ingeniería de software y el proceso creativo, ya que la ingeniería de software proporciona la base técnica necesaria para el desarrollo de videojuegos garantizando que el código sea sólido, escalable y eficiente.

Sin embargo, el éxito de un videojuego no solo depende de su desempeño técnico sino también de la calidad de la experiencia que brinda al jugador, aquí es donde entra en juego el proceso creativo, desde el diseño visual hasta la narración y la jugabilidad, que son vitales para captar y mantener la atención del jugador.

Esto crea la necesidad de un marco metodológico que pueda integrar eficazmente estos dos campos, ya que los desarrolladores enfrentan el desafío de equilibrar el rigor técnico con la flexibilidad creativa, un desafío que, si no se aborda adecuadamente puede resultar en que el producto final sea inconsistente o no cumpla con la visión creativa original.

Además, la falta de herramientas de comunicación entre los equipos técnicos y creativos puede provocar malentendidos, doble trabajo y en últimas instancias, mayores costos y tiempo de desarrollo.

El desarrollo de videojuegos 2D, aunque más accesible en términos de recursos que los videojuegos 3D, sigue siendo un proceso complejo que requiere una gestión cuidadosa de cada fase del proyecto, es importante que los métodos de ingeniería de software se adapten para respaldar el proceso creativo y que el proceso creativo se organice de manera que no afecte la estabilidad técnica del producto final.

Esta propuesta busca contribuir al conocimiento y la práctica en el campo del desarrollo de videojuegos, proponiendo soluciones metodológicas que faciliten la creación de productos de alta calidad al hacer investigaciones y proponer un marco híbrido que permita una colaboración perfecta entre los aspectos técnicos y creativos del desarrollo de videojuegos 2D, con el objetivo e no sólo mejorar el proceso de desarrollo sino también mejorar el nivel de resultados del producto, beneficiando tanto a los desarrolladores como a los jugadores.

Además, la creciente accesibilidad de las herramientas de desarrollo y la popularidad de las plataformas de distribución digital han permitido que más

desarrolladores independientes ingresen al mercado, haciendo que esto sea aún más importante a la hora de crear un marco que simplifique y optimice el proceso de desarrollo, permitiendo a los creadores centrarse en la innovación y la calidad sin toparse con los cuellos de botella técnicos o creativos que suelen surgir en estos proyectos.

Antecedentes

Tomando información de la nota anteriormente mencionada de 33bits, durante las décadas de 1970 y 1980, con la proliferación de consolas de videojuegos y los arcades, los videojuegos 2D dominaron la industria, juegos como Pong (1972), Super Mario Bros (1985) y The Legend of Zelda (1986) no sólo establecieron estándares para el diseño de videojuegos, sino que también enfatizaron la importancia de una arquitectura de software robusta para garantizar la jugabilidad y la estabilidad del juego.

Aunque estos juegos son visualmente simples en comparación con los gráficos 3D de las últimas décadas, fue esta simplicidad la que sentó las bases para el desarrollo de experiencias interactivas profundas donde la narración, las imágenes y la mecánica de juego son esenciales para el éxito.

Con el tiempo, el desarrollo de videojuegos 2D ha seguido siendo importante, incluso con la llegada de los gráficos 3D y las experiencias de realidad virtual, de hecho, la simplicidad y accesibilidad de los juegos 2D ha permitido que florezcan los estudios independientes o también conocidos como “indies”, creando obras que priorizan la innovación en el diseño, narración y mecánicas de juego como "Celeste" (2018) y "Hollow Knight". (2017), que demuestran que los juegos 2D pueden ofrecer experiencias únicas y complejas como cualquier juego 3D, destacando la importancia de un enfoque equilibrado entre la ingeniería de software y el proceso creativo.

En el caso de estudio de Raquel Echeandía en el portal Dialnet menciona que la ingeniería de software y el desarrollo de videojuegos ha adoptado y adaptado diferentes enfoques para gestionar la creciente complejidad de los proyectos, la adopción de metodologías ágiles como la Scrum y Kanban, ha permitido a los desarrolladores responder rápidamente a los cambios, iterar ideas y mejorar la colaboración entre equipos, sin embargo, estos enfoques a menudo se centran en el dominio técnico lo que hace que se entre en conflicto con la naturaleza exploratoria del proceso creativo en el diseño de videojuegos.

Por otro lado, el proceso creativo en el desarrollo de videojuegos involucra una variedad de actividades, desde conceptualizar historias y personajes hasta crear arte visual y componer la música, este proceso es de naturaleza iterativa, ya que las ideas iniciales a menudo se refinan o cambian por completo a medida que avanza el desarrollo y las pruebas de usuario.

Esta naturaleza flexible y exploratoria del proceso creativo puede chocar con la rigidez de la metodología del desarrollo de software, creando tensión entre los equipos técnicos y creativos, aunado a la creciente complejidad de los videojuegos y la necesidad de mantener coherencia entre la visión creativa y la ejecución artística ha llevado a la búsqueda de nuevas formas de integrar estos dos campos.

Objetivos

Objetivo General

- Desarrollar un marco metodológico híbrido que integre la ingeniería de software y el proceso creativo, optimizando el desarrollo de videojuegos 2D, para mejorar la calidad técnica y la coherencia creativa del producto final.

Objetivos Específicos

1. Analizar las metodologías actuales de ingeniería de software aplicadas al desarrollo de videojuegos 2D, identificando sus fortalezas y sus limitaciones en relación con el proceso creativo.
2. Investigar las prácticas y enfoques creativos utilizados en la conceptualización y diseño de videojuegos 2D, con el fin de identificar los puntos de roce y conflicto con los procesos técnicos.
3. Desarrollar y proponer un marco metodológico híbrido que combine aspectos técnicos y creativos, permitiendo iteraciones rápidas y ajustes flexibles sin comprometer la estabilidad y escalabilidad del software.
4. Implementar un prototipo de videojuego 2D utilizando el marco metodológico propuesto en la propuesta de solución, para evaluar su efectividad en la integración de la ingeniería de software y el proceso creativo.
5. Realizar pruebas y validaciones del prototipo con usuarios y desarrolladores, recopilando comentarios para refinar el marco metodológico y asegurar su aplicabilidad en proyectos futuros.
6. Documentar las lecciones aprendidas y las mejores prácticas derivadas de la aplicación del marco metodológico en el desarrollo del prototipo, proporcionando una guía para futuros proyectos de videojuegos 2D.

Alcances y limitaciones

Alcances

1. **Desarrollo de un Marco Metodológico:** El proyecto enfocará en desarrollar un marco metodológico híbrido que integre la ingeniería de software y el proceso creativo, diseñado específicamente para el desarrollo de videojuegos 2D.
2. Este marco incluirá principios y prácticas que puedan ser aplicados por equipos de desarrollo de diferentes tamaños y niveles de experiencia.
3. **Implementación de un Prototipo de Videojuego en 2D:** Se diseñará y desarrollará un prototipo de videojuego en 2D utilizando el marco metodológico propuesto. Este prototipo servirá como caso de estudio para validar la efectividad del marco en un entorno real de desarrollo.
4. **Evaluación y Validación del Marco:** Se realizarán pruebas con usuarios y revisiones por parte de desarrolladores expertos para evaluar la calidad técnica y creativa del prototipo, así como la eficacia del marco metodológico en facilitar la colaboración entre equipos técnicos y creativos.
5. **Documentación de Mejores Prácticas:** La investigación resultará en la documentación de mejores prácticas y lecciones aprendidas, proporcionando una guía que pueda ser utilizada por otros desarrolladores interesados en aplicar este marco metodológico en proyectos futuros.
6. **Enfoque en Juegos en 2D:** La investigación se limitará al desarrollo de videojuegos en 2D, abordando tanto las especificidades técnicas como creativas que son únicas para este formato en comparación con los juegos en 3D.

Limitaciones

1. **Generalización del Marco Metodológico:** Aunque el marco metodológico desarrollado estará diseñado para ser aplicable a una variedad de proyectos de videojuegos en 2D, su efectividad puede variar dependiendo del tamaño del equipo, los recursos disponibles y la complejidad del proyecto. La validación del marco se realizará a través de un prototipo específico, por lo que su generalización a otros tipos de proyectos o formatos de videojuegos (como 3D o VR) puede requerir ajustes adicionales.
2. **Recursos Limitados para el Desarrollo del Prototipo:** El prototipo de videojuego en 2D desarrollado como parte de este proyecto estará limitado por los recursos disponibles, incluyendo tiempo, personal y herramientas. Esto puede afectar la complejidad y la escala del juego final, lo que a su vez podría influir en la evaluación del marco metodológico.

3. **Enfoque en el Proceso de Desarrollo:** El proyecto se centrará en el proceso de desarrollo y la integración de ingeniería de software y creatividad. Aspectos relacionados con la comercialización, distribución o recepción en el mercado del videojuego desarrollado no serán abordados en profundidad.
4. **Pruebas y Feedback Limitados:** Aunque se planea realizar pruebas de usuario y revisiones por parte de expertos, el número de participantes y la variedad de perfiles pueden estar limitados por la disponibilidad y el alcance del proyecto. Esto podría restringir la cantidad de feedback obtenida, lo que podría afectar la validez de las conclusiones.
5. **Restricciones Tecnológicas:** Las herramientas y tecnologías utilizadas para desarrollar el prototipo estarán limitadas a aquellas accesibles dentro del marco temporal y de recursos de la investigación. Esto puede influir en las decisiones de diseño y desarrollo, así como en la aplicabilidad del marco a entornos con diferentes capacidades tecnológicas.

Capítulo 1 Marco teórico y Conceptual

1.1 Ingeniería de Software en el Desarrollo de Videojuegos

La ingeniería de software es la disciplina que se ocupa del diseño, desarrollo, mantenimiento y gestión de software de alta calidad. ("¿Qué es la Ingeniería de Software? | COHETE.digital")

En el contexto del desarrollo de videojuegos, la ingeniería de software proporciona las bases técnicas necesarias para garantizar que un juego funcione de manera eficiente, sea escalable y esté libre de errores graves.

1.1.1 Metodologías de Desarrollo de Software

Las metodologías de desarrollo de software como el modelo en cascada, el desarrollo ágil y la programación extrema (XP) han sido ampliamente utilizadas en la industria del software. Sin embargo, en el contexto del desarrollo de videojuegos, especialmente en 2D, se han adaptado ciertas metodologías para acomodar la naturaleza iterativa y creativa del proceso de diseño.

- **Desarrollo Ágil:** Se ha convertido en una de las metodologías preferidas en el desarrollo de videojuegos debido a su flexibilidad y capacidad de adaptación. Metodologías ágiles como Scrum y Kanban permiten a los equipos de desarrollo iterar rápidamente sobre ideas, integrar el feedback del usuario y ajustar el enfoque según sea necesario.
- **Programación Extrema (XP):** Aunque XP es menos común en la industria del videojuego, algunos de sus principios, como la entrega frecuente de versiones funcionales del software y la participación continua del cliente, pueden ser beneficiosos en proyectos donde la retroalimentación temprana y constante es crítica para el éxito del juego.

1.1.2 Arquitectura de Software en Videojuegos 2D

Según arcus-global (2018), la arquitectura de software en los videojuegos en 2D es fundamental para garantizar que el juego pueda soportar los requisitos de diseño, como la carga gráfica, la física del juego y la inteligencia artificial sin tener que comprometer el rendimiento.

Para esto, se utilizan patrones de diseño como el Modelo-Vista-Controlador (MVC), que permite una separación clara entre la lógica del juego, la interfaz de usuario y la manipulación de datos, facilitando así la gestión de cambios y la implementación de nuevas funcionalidades.

1.1.3. Proceso Creativo en el Desarrollo de Videojuegos en 2D

El proceso creativo en el desarrollo de videojuegos involucra la concepción, diseño y refinamiento de ideas que eventualmente se transforman en el producto final. Este proceso es crucial para definir la estética del juego, su narrativa, y las mecánicas de juego que lo diferencian en el mercado.

1.1.3.1 Etapas del Proceso Creativo

- **Conceptualización:** En esta fase, se generan las ideas principales del juego, incluyendo la historia, los personajes, el estilo artístico y las mecánicas de juego.
- La conceptualización es un proceso iterativo que a menudo involucra brainstorming, prototipado rápido y discusión entre los miembros del equipo creativo.
- **Diseño y Prototipado:** Después de la conceptualización, el equipo creativo desarrolla prototipos que permiten experimentar con las ideas propuestas. En el caso de los videojuegos en 2D, el prototipado puede incluir el diseño de niveles, la creación de arte en píxeles y la implementación básica de mecánicas de juego.
- El prototipado es esencial para visualizar cómo se integrarán las ideas en el producto final y para identificar posibles problemas antes de entrar en la fase de desarrollo completo.
- **Iteración y Refinamiento:** A medida que el proyecto avanza, el proceso creativo involucra la constante iteración y refinamiento de los elementos del juego. Esto puede incluir ajustes en el diseño visual, la narrativa o las mecánicas de juego, basados en pruebas internas y feedback de usuarios.

1.1.4. Intersección entre Ingeniería de Software y Proceso Creativo

La integración de la ingeniería de software con el proceso creativo es uno de los mayores desafíos en el desarrollo de videojuegos. Mientras que la ingeniería de software se enfoca en la estabilidad, eficiencia y escalabilidad del código, el proceso creativo requiere flexibilidad y capacidad de adaptación rápida a nuevas ideas y cambios de dirección.

1.1.4.1 Tensiones entre lo Técnico y lo Creativo

La principal tensión entre la ingeniería de software y el proceso creativo radica en sus enfoques inherentemente diferentes. Mientras que la ingeniería de software busca minimizar los riesgos y controlar los cambios para garantizar la calidad del

producto final, el proceso creativo a menudo prospera en un entorno donde las ideas pueden evolucionar y cambiar sin restricciones.

Esto puede generar fricciones en el equipo, retrasos en el proyecto y, en algunos casos, compromisos que afectan la calidad final del juego.

1.1.4.2 Metodologías Híbridas y Colaboración Efectiva

Para abordar estas tensiones, se han propuesto diversas metodologías híbridas que buscan integrar lo mejor de ambos mundos. Estas metodologías combinan la estructura y las buenas prácticas de la ingeniería de software con la flexibilidad y el enfoque exploratorio del proceso creativo. Un ejemplo de esto es la metodología “Desarrollo Ágil Creativo” (DAC), que alterna sprints técnicos con sprints creativos, permitiendo a ambos equipos trabajar de manera sinérgica hacia un objetivo común.

Además, la colaboración efectiva entre los equipos técnicos y creativos es esencial. Herramientas colaborativas que permiten la integración de código con recursos creativos, junto con una comunicación clara y continua, son claves para garantizar que la visión del juego se mantenga coherente y que el proceso de desarrollo sea lo más fluido posible.

1.1.5. Herramientas y Tecnologías en el Desarrollo de Videojuegos en 2D

El desarrollo de videojuegos en 2D se apoya en un conjunto de herramientas y tecnologías que facilitan tanto el proceso creativo como la implementación técnica.

- **Motores de Juego:** Herramientas como Unity, Godot, y GameMaker Studio son ampliamente utilizadas en el desarrollo de videojuegos en 2D. Estos motores de juego proporcionan un entorno integrado donde los desarrolladores pueden crear, probar y depurar juegos, a la vez que ofrecen capacidades avanzadas para el manejo de gráficos, física y sonido.
- **Software de Arte y Animación:** Herramientas como Adobe Photoshop, Aseprite, y Spine son esenciales para la creación de arte y animaciones en 2D. Estas herramientas permiten a los diseñadores trabajar en píxeles, crear sprites, y animar personajes y escenarios de manera eficiente.
- **Sistemas de Control de Versiones:** Git y GitHub son herramientas fundamentales para la gestión de código y recursos creativos en proyectos de desarrollo de videojuegos. Estas herramientas permiten a los equipos trabajar de manera colaborativa, gestionar cambios en el código y mantener un historial de versiones, lo que es crucial para proyectos en constante evolución.

1.1.6.- Ingeniería de software

Es la disciplina que se ocupa del diseño, el desarrollo, mantenimiento, pruebas y evaluación del software, a diferencia de la programación pura, esta disciplina se enfoca en aplicar principios de ingeniería para crear software robusto, eficiente y mantenible.

1.1.6.1.- Ciclo de vida del Desarrollo de Software

- **Recolección de Requisitos**: Identifica y documenta las necesidades del usuario final y los objetivos del sistema.
- **Análisis de requisito**: Descompone los requisitos para comprenderlos en detalle y especificar lo que debe hacer el software.
- **Diseño**: Creación de una arquitectura o estructura del software.
- **Implementación (Codificación):** Desarrollo del software propiamente dicho, aquí se escribe código en el lenguaje de programación seleccionado.
- **Pruebas**: Evaluación del software para identificar y corregir errores, procurando que cumpla con los objetivos.
- **Despliegue**: Instalación y puesta en funcionamiento del software en el entorno del usuario.
- **Mantenimiento**: Actualizaciones, mejoras y correcciones de errores que puedan surgir una vez que el software se encuentra en uso.

1.1.6.2.- Principios de la ingeniera de software

Se basa en varios principios fundamentales para asegurar que el software resultante sea de alta calidad:

- **Modularidad**: Dividir el sistema en componentes o módulos independientes para facilitar su desarrollo, mantenimiento y prueba.
- **Reutilización**: Uso de componentes de software previamente desarrollados para reducir costos y tiempo.
- **Mantenibilidad**: El software debe diseñarse para facilitar su mantenimiento y actualización a futuro.
- **Escalabilidad**: Capacidad del software para manejar un aumento en la carga o volumen de datos sin perdida significativa de rendimiento.
- **Confiabilidad**: El software debe funcionar de manera correcta bajo diversas condiciones y manejar errores de manera adecuada.

1.1.6.3.- Herramientas de la ingeniería de software

- **Sistemas de Control de Versiones**: Para gestionar cambios en el código fuente (Como github).
- **Entornos de desarrollo (IDE):** Ofrecen un entorno unificado para escribir, probar y depurar código (Visual Studio, Eclipse).

1.1.7.- Procesos Creativos

Es la metodología que guía el desarrollo de una idea desde la concepción inicial hasta la ejecución final, se caracteriza por ser no lineal, iterativo y a menudo, involucra tanto la inspiración momentánea como el análisis racional, a diferencia de otros procesos este valora tanto la intuición como la lógica.

1.1.7.1.- Etapas del Proceso Creativo

- **Preparación:** Recolectar información, investigar y adquirir el conocimiento necesario sobre el problema o tema en cuestión, se analizan referencias, se estudian casos previos y se busca entender a fondo el contexto.
- **Incubación:** Deja que la mente procese la información de manera subconsciente, durante este periodo las ideas no se trabajan activamente, pero la mente sigue buscando conexiones.
- **Iluminación (Insight):** Es cuando surge una idea o una solución, esta fase a menudo es descrita como la parte más emocionante y reveladora del proceso creativo.
- **Evaluación:** Análisis crítico de la idea generada, aquí se evalúa si la idea es factible, si cumple con los objetivos planteados y si es realmente innovadora.
- **Elaboración:** Es el desarrollo completo de la idea, aquí se pasa de la fase conceptual a la práctica, detallando, refinando y concretando la solución o la obra creativa.

1.1.7.2.- Técnicas y Estrategias Creativas

- **Tormenta de ideas (BrainStorming):** Reunir un grupo para generarla mayor cantidad posible de ideas sin juzgarlas, aquí la cantidad se prioriza sobre la calidad.
- **Mapas mentales:** Visualizar ideas y conceptos de manera gráfica, conectándolos de forma jerárquica y encontrando relaciones que no son obvias a primera vista,
- **Pensamiento Lateral:** Proponer soluciones fuera de lo convencional, a diferencia del pensamiento lógico, el pensamiento lateral fomenta la creatividad al abordar problemas desde ángulos inesperados.
- **SCAMPER:** Técnica que invita a modificar una idea a través de preguntas clave: Sustituir, Combinar, Adaptar, Modificar, Poner en otro uso, Eliminar, Reorganizar.
- **Analogías y Metáforas:** Utilizar conceptos o situaciones de otros contextos para encontrar inspiración y generar nuevas ideas aplicables al problema en cuestión.

1.1.7.3.- Herramientas para el Proceso Creativo

- **Software de Mapas Mentales**: Permiten estructurar visualmente las ideas (MindMeister o Xmind).

- **Plataformas de Colaboración**: Para que equipos creativos puedan trabajar juntos, compartir recursos y desarrollar conceptos (Miro o Notion).
- **Diarios Creativos o Cuadernos de Bocetos**: Son espacios para capturar ideas espontaneas, dibujar, escribir o plasmar conceptos sin restricciones.
- **Generadores de ideas**: Aplicaciones o juegos que presentan desafíos creativos o sugerencias para inspirar nuevas ideas.

1.1.8.- Optimización

Es un proceso que mejora la eficiencia, rendimiento o efectividad de un sistema, proceso o solución, implica encontrar la mejor opción posible dentro de un conjunto de alternativas bajo ciertos criterios específicos como minimizar costos, maximizar ganancias, reducir tiempo o mejorar la calidad.

1.1.8.1.- Tipos de Optimización

- **Matemática**: Utiliza métodos matemáticos y algoritmos para encontrar la mejor solución a problemas formulados como modelos matemáticos, y subdivide en optimización lineal, que es donde las relaciones entre las variables son lineales, adecuada para problemas que pueden ser descritos mediante ecuaciones o desigualdades lineales, la optimización no lineal, que es cuando las relaciones entre variables no son lineales, es más complejo y se aplica a problemas con relaciones no proporcionales, Optimización entera, aquí las variables de decisión solo pueden tomar valores enteros, se utiliza en problemas donde las soluciones no pueden ser fraccionadas, como la planificación de recursos o el diseño de rutas, y por ultimo Organización Dinámica, que se aplica a problemas en los que la decisión optima depende de una secuencia de eventos a lo largo del tiempo.
- **Optimización Heurística**: Emplea métodos aproximados que buscan soluciones venas, pero no necesariamente optimas, es útil cuando el espacio de búsqueda es demasiado grande para explorar exhaustivamente.
- **Optimización Multiobjetivo**: Se utiliza cuando un problema tiene varios objetivos que pueden ser conflictivos entre sí, como minimizar costos mientras se maximiza la calidad, busca un equilibrio optimo mediante técnicas como el frente de Pareto, donde se consideran soluciones que no son superadas simultáneamente en todos los objetivos.

1.1.8.2.- Etapas del Proceso de Optimización

- **Formulación del Problema**: Definición claro del problema, objetivos, variables de decisión y las restricciones, es fundamental identificar que es lo que se quiere optimizar y en qué condiciones.
- **Modelo Matemático**: Crear un modelo matemático que represente el problema, incluyendo una función objetivo que debe ser optimizado, así como las restricciones que limitan las posibles soluciones.

- **Selección del método de optimización**: Elegir la técnica o algoritmo adecuado según el tipo de problema.
- **Resolución del Problema**: Aplicar el método seleccionado para encontrar a solución óptima, en esta fase se implementan los cálculos o simulaciones necesarios.
- **Validación y análisis de resultados**: Se comprueba que la solución obtenida cumple con los objetivos planteados y as restricciones definidas, se analiza su la solución es razonable y se ajustan los parámetros si es necesario.
- **Implementación y Monitoreo**: Implementar la solución en el contexto real y monitorear su efectividad, si surgen cambios o problemas en el entorno, podría ser necesario ajustar la optimización.

1.1.8.3.- Técnicas y Métodos de Optimización

- **Métodos Exactos**: Se encuentra la solución exacta del problema (Método Simplex y Método de Branch and Bound, por mencionar algunos)
- **Métodos Aproximados o Heurísticos**: Ofrecen soluciones satisfactorias cuando los métodos exactos son computacionalmente inviables (Algoritmos genéticos y recocido simulado).
- **Métodos de optimización Convexa**: Se aplican a problemas donde la función objetivo es convexa, lo que asegura que cualquier mínimo local es también el mínimo global.
- **Métodos de Gradiente**: Utilizan derivadas para moverse hacia la dirección de mejora en el espacio de soluciones, como el método del gradiente descendente.
- **Programación lineal y no lineal**: Son técnicas clásicas que utilizan restricciones lineales o no lineales para definir el espacio de búsqueda.

1.1.8.4.- Aplicaciones de la Optimización

- **Ingeniería y Diseño**: Optimiza estructuras, circuitos electrónicos y procesos industriales.
- **Logística y transporte**: Optimización de rutas, planificación de la cadena de suministro y asignación de recursos para reducir tiempos y costos.
- **Finanzas**: Maximización de beneficios en inversiones, minimización de riesgos financieros y gestión optima de carteras de inversión.
- **Computación**: Optimización de algoritmos para mejorar la eficiencia computacional, reducir el consumo de recursos y acelerar tiempos de ejecución.

1.1.8.5.- Importancia de la Optimización

- **Reducción de Costos**: Minimizar el uso de recursos y los costos asociados sin comprometer la calidad.
- **Mejorar la Productividad**: Aumentar la eficiencia en procesos y tareas, reduciendo tiempos y aumentando la calidad del producto final.
- **Mejor toma de Decisiones**: Ofrecer alternativas claras y precisas para elegir la mejor opción bajo las circunstancias dadas.
- **Sostenibilidad**: En el contexto ambiental, la optimización permite el uso eficiente de recursos naturales, reduciendo el impacto ambiental.

1.1.9.- Optimización de Software

Busca mejorar el rendimiento del código, reducir el consumo de recursos y aumentar la velocidad de ejecución, manteniendo o mejorando la funcionalidad del programa.

Abarca una serie de técnicas que van desde la optimización a nivel de código fuente hasta la configuración de hardware en el que se ejecuta la aplicación.

1.1.9.1.- Tipos de Optimización

A) Optimización del Rendimiento

- **Optimización de Algoritmos**: Mejora la eficiencia de los algoritmos utilizados, implicando la elección de algoritmos con la menor complejidad computacional posible (O(n), O (log n), etc.), reduciendo el tiempo de ejecución y el uso de recursos.
- **Paralelización**: Dividir tarea en múltiples subprocesos o hilos que se pueden ejecutar en paralelo, utilizando mejor los recursos del hardware.
- **Optimización de Consultas a Bases de Datos**: Rediseñar consultas SQL, optimizar índices y reducir la cantidad de acceso a la base de datos para mejorar el tiempo de respuesta.
- **Compilación y Código Maquina**: Ajustar las opciones del compilador para generar código más eficiente y utilizar optimizaciones específicas de hardware.

B) Optimización del uso de Memoria

- **Manejo Eficiente de la Memoria**: Minimizar el uso de la memoria al evitar fugar de memoria (memory leaks) y utilizar estructuras de datos que ocupen menos espacio.
- **Caching**: Guardar temporalmente resultados calculados para evitar cálculos repetidos, mejorando el rendimiento a costa el uso de memoria.
- **Optimización de la Recolección de Basura (Garbage Collection):** Ajustar las configuraciones del recolector de basura en lenguajes con manejo automático de memoria (Java, C#) para minimizar pausas innecesarias.

C) Optimización de Código

- **Eliminación de Código Muerto**: Quitar código que no se utiliza o que es redundante, haciendo que el programa sea más limpio y eficiente.
- **Refactorización**: Mejorar la estructura del código sin cambiar su comportamiento, esto incluye modularización, simplificación de funciones complejas y mejoras de la legibilidad.
- **Inlining**: Integrar funciones pequeñas directamente en el flujo principal del programa, evitándolo la sobrecarga de llamadas a funciones.
- **Optimización de Ciclos**: Minimizar la cantidad de iteraciones en bucles o simplificar la lógica interna para mejorar la eficiencia.

D) Optimización de la Experiencia de Usuario (UX)

- **Optimización del Tiempo de Carga:** Minimizar el tiempo que tarda el software en iniciar o cargar elementos esenciales, optimizando la entrada de recursos.
- **Interacción Fluida:** Reducir el tiempo de espera para el usuario al cargar contenido en segundo plano, usar animaciones suaves y proporcionar retroalimentación rápida.
- **Compresión de Recursos:** Reducir el tamaño de los archivos (imágenes, scripts, etc.) que el software necesita cargar, mejorando la velocidad de carga.

1.1.9.2.- Etapas del Proceso de Optimización en Software

- **Identificación del Problema**: Detectar que parte del software necesita la optimización, esto puede involucrar el análisis del rendimiento, la evaluación del uso de memoria o la identificación de cuellos de botella específicos.
- **Medición y Análisis**: Recopilar datos precisos sobre el rendimiento del software, usando herramientas como profilers, monitores de memoria y análisis de tiempos de ejecución.
- **Modelado y Ajustes**: Aplicar técnicas de optimización específicas para solucionar los problemas identificados, esto puede incluir cambios en el código, justes en la configuración del sistema o la implementación de nuevas estructuras de datos.
- **Pruebas**: Validar que las optimizaciones realizadas funciones correctamente sin afectar negativamente el comportamiento del software, siendo fundamental realizar pruebas de regresión para asegurar que no se introduzcan nuevos errores.
- **Monitoreo Continuo**: Después de la optimización es importante seguir monitoreando el rendimiento para detectar posibles problemas futuros o identificar nuevas oportunidades de mejora.

1.1.9.3.- Herramientas de Optimización en Software

- **Profilers**: Permiten identificar cuellos de botella en el código y áreas que sonsumen mucho tiempo o recursos (VisualVM, Perf, gprof, JProfiler).
- **Analizadores de Memoria**: Para identificar fugas de memoria o uso ineficiente (Valgrind, Memory Analyzer y HeapDump).
- **Pruebas de Carga y Rendimiento**: Para evaluar cómo responde el software bajo diferentes cargas de trabajo (JMeter, Gatling o LoadRunner).
- **Sistemas de Caching**: Permiten implementar caching efectivo para mejorarel rendimiento (Redis, Memcached, Varnish).
- **Minificadores y Compresores**: Reducen el tamaño del código JavaScript o imágenes para mejorar la velocidad de carga (UglifyJS o ImageOptim).

1.1.9.4.- Aplicaciones de la Optimización de Software

- **Aplicaciones Web**: Mejora del tiempo de carga, optimización de peticiones HTTP, reducción del tamaño de archivos y optimización del manejo del DOM.
- **Aplicaciones Móviles**: Minimización del uso de batería, reducción del tamaño de la aplicación, optimización del rendimiento gráfico y mejora de la capacidad de respuesta.
- **Software Empresarial**: Optimización de consultas a bases de datos, gestión eficiente de la memoria en servidores y mejora de la escalabilidad para manejar grandes volúmenes de datos.
- **Sistemas Embebidos**: Uso eficiente de recursos limitados, como memoria y CPU, optimización del consumo energético y maximización del rendimiento en dispositivos con hardware restringido.
- **Videojuegos**: Optimización del rendimiento gráfico, mejora de la física, y simulaciones en tiempo real, minimización de latencia y optimización del uso de la GPU.

1.1.10.- Desarrollo de Videojuegos

Es un proceso complejo y multidisciplinario que abarca desde la conceptualización de una idea hasta la creación de un producto interactivo final que puede ser jugado en diferentes plataformas.

Involucra a profesionales de diversas áreas, como la programación, el diseño gráfico, la música, la narrativa y la gestión de proyectos, quienes colaboran para crear una experiencia de juego envolvente y entretenida.

1.1.10.1.- Fases, elementos clave y técnicas de desarrollo

A) Conceptualización y Preproducción

- **Idea inicial**: La creación de un videojuego inicia con una idea o concepto en general, este concepto incluye una revisión básica el tipo de juego, ya sea acción, aventura, RPG, la historia, ambientación y la estética.

- **Documento de Concepto**: Se elabora un documento que recoge la idea principal, incluyendo detalles sobre la narrativa, los personajes, el diseño visual, las mecánicas de juego y los objetivos, este documento sirve como guía inicial para todo el equipo.
- **Prototipado Rápido**: Se desarrollan prototipos básicos del juego para probar las mecánicas fundamentales y verificar si la idea es viable, en esta fase se experimenta y se ajustan aspectos esenciales del diseño antes de comprometerse a una producción completa.
- **Planificación del Proyecto**: Se establece un plan detallado que incluye el cronograma de desarrollo, la asignación de tareas, los recursos necesarios y los hitos importantes, esto implica identificar el motor de juego que se utilizara, plataformas destino, el presupuesto y el tiempo estimados.

B) Diseño del Juego

- **Mecánicas de Juego.**
- **Diseño de Niveles.**
- **Narrativa y Diseño de Personajes.**
- **Diseño de Arte y Estilo Visual.**

C) Producción

- **Programación y Desarrollo de Software:** Los programadore implementan las mecánicas de juego y la lógica que define el comportamiento del mundo virtual.
- **Gráficos y Animación:** Los artistas gráficos crean los modelos 3D o los sprites 2D, texturas y elementos visuales que conformaran el aspecto del juego.
- **Audio y Música:** Los diseñadores de sonido crean efectos de audio, diálogos y ambientaciones sonoras que enriquecen la experiencia del juego.

D) Pruebas y Depuración

- **Pruebas de Jugabilidad:** Se hace una evaluación de la jugabilidad, buscando problemas mecánicos.
- **Pruebas de Rendimiento:** Se realizan pruebas para medir el rendimiento del juego en diferentes dispositivos.
- **Depuración y Corrección de Errores:** Se corrigen errores detectados durante las pruebas, se ajustan las mecánicas de juego y solucionan problemas de estabilidad.

1.2.- Estado del Arte

El proyecto de titulación de Carrasco Juan, Ramírez Luis, Duarte Andrés y Barrera Jesús tiene como objetivo atraer a potenciales aspirantes a la carrera de ingeniería de sistemas y computación mediante la creación de un videojuego, ya que se percataron que la cantidad de aspirantes a dicha carrera ha ido cayendo con el pasar de los años, mencionando un estudio realizado por MinTic en 2015. (Rodríguez et al., 2019)

Se menciona que en ese estudio una de las principales causas de que la carrera no tenga tantos aspirantes como en otros años es por la forma en cómo se está mostrando la carrera, dando a los potenciales aspirantes una idea totalmente equivocada sobre esta misma, dando paso a que dichos aspirantes busquen otra carrera que se les haga más atractiva con respecto a la anteriormente mencionada.

Con eso en mente, desarrollaron un videojuego para demostrar que la carrera de sistemas y computación no es como se ha llegado a dar entender, incluso demostrando que puede ser divertida buscando así llegar a personas que no tengan un total entendimiento de cómo es esa carrera, y demostrar a las que si lo tengan que no es como imaginaban.

Este trabajo es especial, por la capacidad que tiene de demostrar con la creación y la implementación de un videojuego que ciertas cosas no son como se pueden llegar a imaginar o como incluso, se pueden llegar a platicar, además de que demuestra que los videojuegos no solo funcionan como un medio de entretenimiento o distracción, si no que también como un medio educativo y de divulgación informativa.

Félix Etxeberria Balerdi (2009)

El trabajo de Félix Etxeberria Balerdi (2009) llama la atención, ya que a diferencia de muchos otros trabajos consultados para el proceso de creación de este proyecto, no habla sobre procesos de desarrollo, ni de metodologías, si no sobre lo que en realidad es un videojuego, explicando sus inicios, historia, como han ido evolucionando, ofreciendo tablas de porcentajes desde que tipos de videojuegos son los enfoques preferidos de las personas, como violencia fantástica, deportivos, educativos, estando estos últimos en último lugar con un pobre 2%.

También proporciona una tabla de los mejores juegos de 1997 junto con una tabla que proporciona el gusto tanto de niñas como de niños, aunando una tabla que proporciona el tiempo de juego de cada uno, de la fuente de Funk (1993).

Pero no solo se queda ahí la investigación de Félix Etxeberria Balerdi, si no que profundiza más mencionando la psicología del aprendizaje y videojuegos, mencionando que lo fundamental es que una tarea tenga el suficiente atractivo o motivación para promover el aprendizaje.

Y se considera que esto es cierto, ya que hoy en día muchas personas tildan a los videojuegos de que no aportan nada bueno dentro del aprendizaje, dejando de lado que los videojuegos pueden proporcionar muchas facilidades al aprendizaje, y a actividades motrices, con consolas como la Wii, o los sistemas de realidad virtual.

Aparte de todo esto, también hace una mención sobre influencias negativas que pueden proporcionar los videojuegos, como la violencia, sexismo, sociabilidad, creatividad entre varios otros, haciendo que esto en particular sea algo destacable, ya que no muchos trabajos mencionan estas partes negativas que un videojuego puede tener si no se sabe manejar de manera correcta, haciendo que se le deba poner especial atención a estos detalles.

Ana Ma Manrubia Pereira (2015)

En el trabajo de Ana Ma Manrubia Pereira llamado “El proceso productivo del videojuego: fases de producción” (2015), menciona la importancia de los videojuegos en los últimos años, haciendo que su proceso de desarrollo sea una parte fundamental en su creación.

Explica la proceso de producción de los videojuegos, iniciando por la preproducción iniciado en el concepto de juego, que es una de las partes más importantes a la hora de crear un videojuego, ya que aquí es cuando se decide el género, se crea la historia y se hacen los bocetos de los niveles, o espacios donde el personaje interactuara así como también el gameplay, que es la parte más importante de un videojuego, ya que de esta depende si será un videojuego innovador, que aporte cosas nuevas a los videojuegos, y también de esta dependerá si el juego es un éxito, o no.

Una vez terminada esta parte es cuando se puede pasar al apartado de producción, iniciando con el diseño de juego, que es donde se especifican los elementos que compondrán al juego, y después se inicia el diseño artístico para implementar la historia, creación de sonidos, música, efectos, etc. Las interfaces que se tendrán en el producto, y los gráficos.

El trabajo es interesante porque ayudo a ser una guía en el proceso que este proyecto estaba teniendo, si bien los sprints ayudaron demasiado, tener una cronología de creación como la de este trabajo, ayudo para poder saber como guiarse a la hora de ir terminando esos sprints.

Capítulo 2 Metodología

1.- Modelo en Cascada

Esta es una de las que fueron las primeras metodologías formales de desarrollo de software, ocupa un enfoque secuencial que sigue una estructura rígida y organizada, donde cada fase del proyecto debe de completarse antes de pasar a la siguiente, se le llama “cascada” por qué tiene la similitud con el agua de una cascada, ya que el flujo del trabajo avanza en una sola dirección, hacia adelante.

1.1 Fases del Modelo en Cascada

Requisitos

Se recopilan y documentan todos los requisitos del proyecto o producto a desarrollar.

Los requisitos deben ser totalmente entendidos y ser definidos claramente antes de continuar.

Diseño del Sistema

A partir del punto anterior, se diseña la arquitectura del proyecto, incluyendo las especificaciones de hardware y software, así como el diseño de la base de la estructura de la aplicación.

Implementación

En esta fase, los programadores traducen el diseño en código, creando el sistema o producto.

Aquí se desarrolla el código fuente y se integran los módulos desarrollados para formar el sistema completo

Pruebas

Después de que el sistema haya sido implementado se realiza una fase de pruebas exhaustivas para asegurarse de que el software funcione según los requisitos y esté libre de errores.

Despliegue

Después de que el sistema ha sido probado y verificado, se implementa en el entorno real donde será utilizado por los usuarios finales.

Mantenimiento

Después del despliegue, el sistema entra en la fase de mantenimiento, donde se realizan correcciones de errores y mejoras según sea necesario.

El modelo en Cascada presenta varias ventajas y desventajas que deben ser consideradas. Entre los aspectos positivos, destaca su claridad y estructura; gracias a su enfoque secuencial, cada fase cuenta con un inicio y un final claramente definidos, lo que facilita la planificación y el seguimiento del progreso. Además, fomenta una documentación extensa en cada etapa, lo que mejora la comprensión del proyecto y facilita la transferencia de conocimiento, sirviendo como referencia para el futuro.

La naturaleza predecible y lineal del modelo también simplifica la gestión de proyectos, permitiendo a los integrantes del equipo evaluar hitos específicos y realizar un seguimiento efectivo del avance y el presupuesto. Este enfoque resulta especialmente adecuado para proyectos pequeños donde los requisitos son bien conocidos, claros y poco propensos a cambios.

Sin embargo, el modelo en Cascada también presenta inconvenientes significativos. Uno de los más destacados es su falta de flexibilidad; si surgen cambios en los requisitos durante fases posteriores, adaptarse a ellos puede resultar costoso y complicado, ya que el modelo no contempla la retroalimentación continua ni la modificación de los procesos en curso. Además, la poca retroalimentación del cliente puede ser un problema, ya que este solo tiene acceso al producto final una vez que se ha completado la implementación, lo que puede generar sorpresas o insatisfacción si los resultados no cumplen con sus expectativas.

Asimismo, dado que las pruebas se realizan solo después de la implementación, los errores o fallos pueden no detectarse hasta etapas avanzadas del proyecto, complicando su corrección. Por último, este modelo no es adecuado para proyectos complejos o con requisitos cambiantes, ya que su estructura secuencial dificulta la adaptación a cambios imprevistos.

2.- Programación Extrema (XP)

Esta es una metodología ágil enfocada en mejorar la calidad del software y la capacidad de respuesta a los cambios de requisitos, desarrollada en la década de 1990 por Kent Beck y se basa en una serie de buenas prácticas que buscan mejorar la eficiencia el desarrollo de software a través de ciclos rápidos y comunicación constante.

2.1.- Fases de la Programación Extrema

Planificación

Se define el alcance del proyecto mediante las llamadas “historias de usuario” que describen las funcionalidades que el cliente desea, y a partir de esto, el equipo estima el tiempo necesario para cada tarea y se organizan las iteraciones.

Diseño

En diseño siempre se mantiene lo más simple posible, lo que significa que el equipo solo construye lo que es necesario para satisfacer al usuario, también se usan las técnicas llamadas “diseño guiado por pruebas”.

Codificación

El equipo trabaja en pares, lo que significa que dos desarrolladores colaboran en una misma estación de trabajo, y mientras uno está escribiendo el código, el otro lo revisa en tiempo real, promoviendo la calidad de dicho código desde un inicio.

Pruebas

Esta metodología tiene un fuerte enfoque en las pruebas, ya que se realizan pruebas automatizadas y unitarias desde el principio del ciclo de desarrollo, y cada funcionalidad tiene que pasar dichas pruebas para ser considerada completa.

Entrega

Al final de cada iteración, se entrega una versión funcional del software que puede ser aprobada y revisada por el cliente, asegurando que el software ese siempre en un estado utilizable y que se pueda mejorar en función de feedback recibido.

La programación extrema (XP) presenta una variedad de ventajas y desventajas que vale la pena considerar. Entre sus aspectos positivos, destaca su capacidad de respuesta rápida a cambios, ya que su flexibilidad permite adaptarse rápidamente a nuevos requisitos o a cambios en las prioridades del cliente. Asimismo, la mejora continua del proyecto es fundamental; las entregas frecuentes y la retroalimentación constante facilitan la realización de ajustes, asegurando que el producto evolucione conforme a las expectativas del cliente.

Además, XP promueve una alta calidad del código. Gracias a la programación en pareja y al enfoque en pruebas automatizadas, se reducen los errores desde las primeras etapas del desarrollo. Otro beneficio es la simplicidad en el diseño, que evita la creación de código innecesariamente complejo, facilitando el mantenimiento y reduciendo la probabilidad de errores a largo plazo. Finalmente, esta metodología fomenta un ambiente de trabajo dinámico y participativo, lo que puede incrementar la motivación y el compromiso del equipo.

No obstante, la programación extrema también presenta desventajas. Una de las más significativas es que puede ser difícil de implementar en equipos grandes; esta metodología funciona mejor en grupos pequeños y altamente colaborativos, mientras que, en equipos más grandes, la coordinación y la comunicación pueden volverse complicadas.

Además, requiere una alta participación del cliente en el proceso de desarrollo, lo que puede ser problemático si el cliente no dispone del tiempo o de los recursos necesarios para colaborar de manera continua.

Otro riesgo es la posible falta de planificación a largo plazo; el enfoque en iteraciones cortas y entregas rápidas puede llevar a descuidar la planificación futura, lo que podría ocasionar problemas de escalabilidad o integración en etapas posteriores del proyecto. Aunque la programación en pareja tiene sus beneficios, no todos los desarrolladores se sienten cómodos trabajando de esta forma, y en ciertas circunstancias, puede ralentizar al equipo. Por último, el enfoque intensivo en pruebas automatizadas y de unidad puede incrementar los costos de desarrollo, especialmente si se requiere tiempo adicional para crear y mantener los entornos de prueba.

3.- Desarrollo Ágil

Esta metodología se centra en la flexibilidad, colaboración y la entrega incremental de software funcional, creada debido a las limitaciones de los enfoques más tradicionales como el modelo en cascada anteriormente mencionado en el documento, y basado en los principios del manifiesto ágil, el principal objetivo es entregar valor al cliente de manera rápida y efectiva, ajustándose a los cambios en los requisitos a lo largo del proyecto.

3.1.- Fases del Desarrollo Ágil

Planificación del Sprint

Esta metodología se organiza en iteraciones cortas llamadas sprints, generalmente de 1 a 4 semanas, durante la planeación el equipo selecciona las historias de usuario que se van a desarrollar durante el sprint.

Desarrollo e Implementación

Durante el sprint, el equipo trabaja en la implementación de las historias elegidas, el equipo tiene autonomía para resolver problemas y ajustar el código en función de los cambios que se vayan encontrando.

Revisión del Sprint

Al final de cada sprint, el equipo revisa lo que ha completado y presenta un software funcional al cliente. Esta entrega de software permite que el cliente pruebe el sistema pueda dar feedback sobre él.

Retrospectiva

Después de la revisión, el equipo se reúne para discutir lo que funciona bien, lo que no funciona bien y como se puede mejorar en el próximo sprint, con el objetivo de aprender y mejorar continuamente.

Pruebas

A lo largo del proceso se desarrollan pruebas continuas, a menudo automatizadas para asegurar la calidad del software, estas pruebas garantizan que cada funcionalidad agregada no rompa el sistema y cumpla con los requisitos.

El desarrollo ágil ofrece múltiples ventajas y desventajas que es importante considerar. Entre sus pros, la flexibilidad ante el cambio se destaca como una de las mayores fortalezas, ya que permite adaptarse a modificaciones en los requisitos en cualquier momento del proyecto. Además, la entrega continua es un aspecto fundamental; cada sprint genera un software funcional, lo que significa que el cliente puede recibir avances incrementales a lo largo del desarrollo, permitiendo realizar ajustes basados en una retroalimentación rápida.

La mejora continua es otra ventaja, ya que las reuniones de retrospectiva permiten identificar problemas y optimizar procesos, aumentando la eficiencia del equipo. Además, la alta colaboración y transparencia son características esenciales; el cliente está involucrado activamente en cada fase del proyecto, lo que genera una comunicación fluida y proporciona visibilidad sobre el estado de este.

Finalmente, esta metodología tiende a aumentar la satisfacción del cliente, ya que las entregas frecuentes y la participación en el desarrollo le otorgan un mayor control sobre el producto final, reduciendo el riesgo de que el resultado no cumpla con sus expectativas.

Sin embargo, el desarrollo ágil también presenta desventajas. Una de ellas es la falta de previsión a largo plazo; al centrarse en sprints cortos y entregas rápidas, puede haber una carencia de planificación estratégica, lo que podría generar problemas en la integración o escalabilidad del sistema. Además, requiere un compromiso activo del cliente, ya que el éxito de la metodología depende de su participación frecuente. Si el cliente no tiene tiempo para colaborar o proporcionar retroalimentación, el proyecto puede desviarse de su rumbo.

Otro inconveniente es la dificultad para estimar el tiempo y el costo total del proyecto. La naturaleza iterativa del proceso complica la predicción precisa de cuánto tiempo y presupuesto serán necesarios para completarlo. Asimismo, la priorización de la creación de software funcional sobre la documentación puede resultar en una falta de información detallada, lo que puede dificultar el mantenimiento a largo plazo del sistema. Por último, si no se gestiona adecuadamente, el enfoque ágil puede dar lugar a una sensación de caos o desorganización, especialmente en equipos que no están familiarizados con esta metodología.

Cuadro Comparativo

Cuadro Comparativo: Metodologías de Desarrollo de Software en Videojuegos en 2D

Aspecto	Desarrollo Ágil	Programación Extrema (XP)	Modelo en Cascada
Enfoque Principal	Flexibilidad y adaptabilidad mediante ciclos iterativos.	Calidad del código y colaboración continua.	Estructura secuencial y planificación detallada.
Ciclo de Desarrollo	Iteraciones rápidas (sprints), con entregas incrementales.	Iteraciones rápidas con integración continua y pruebas frecuentes.	Fases secuenciales (análisis, diseño, implementación, pruebas).
Documentación	Documentación mínima necesaria, enfocado en el trabajo en equipo.	Documentación ligera, enfocado en la codificación y pruebas continuas.	Documentación extensa y detallada en cada fase.
Flexibilidad	Alta flexibilidad para adaptarse a cambios y nuevos requisitos.	Alta flexibilidad en la integración de cambios durante el desarrollo.	Baja flexibilidad, cambios en fases avanzadas son difíciles.
Gestión de Proyectos	Gestión mediante reuniones diarias y revisión de sprints.	Gestión basada en la retroalimentación constante y ajustes técnicos.	Gestión estricta con un enfoque en la planificación inicial.
Colaboración	Comunicación continua entre los miembros del equipo y clientes.	Colaboración cercana entre desarrolladores y clientes para ajustes constantes.	Comunicación estructurada y formal entre los equipos.
Enfoque en la Calidad	Mejora continua del producto a través de revisiones y feedback.	Enfoque en la calidad del código y la funcionalidad mediante pruebas frecuentes.	Control de calidad al final de cada fase del proyecto.
Adecuación para Videojuegos en 2D	Adecuado para proyectos con cambios frecuentes en el diseño y mecánicas.	Beneficioso para proyectos que requieren alta calidad técnica y cambios rápidos.	Menos adecuado para proyectos que necesitan flexibilidad creativa.
Ejemplos de Herramientas	Jira, Trello, herramientas de integración continua (CI).	Herramientas de integración continua, sistemas de control de versiones.	Herramientas de gestión de proyectos tradicionales.

Elección de Metodología

La metodología elegida para este proyecto fue la metodología ágil, ya que se adaptaba bien a los objetivos y resultados que se buscaban alcanzar. Esta metodología se caracteriza por su enfoque en la documentación no excesivamente detallada y la realización de sprints iterativos para la producción de cada componente del proyecto. Sin embargo, se realizaron algunas modificaciones para alinearla con el proceso creativo mencionado anteriormente, evitando así posibles fricciones y permitiendo un trabajo continúo, llamándola “Proceso Creativo Ágil” o DCA para abreviar dentro de este documento. Aunque se presentaron ligeras fallas, el enfoque se mantuvo funcional y no impactó negativamente en los tiempos ni en la producción del proyecto.

Capítulo 3 Diseño y Desarrollo

La metodología que se escogió fue la metodología ágil modificada para el DCA, hay que tener en cuenta que el proyecto fue hecho por una sola persona, así que cada vez que nos refiramos al equipo creativo, el equipo de desarrollo o el compositor de música, será la misma persona.

También resaltar que algunos de los assets se tomaron de un curso que se compró en Domestika, ya que por falta de conocimiento en el píxel art, y tiempo de desarrollo se optó por hacerlo así, pero no sin antes dar gracias a Domestika por la gran ayuda que esto proporciono al prototipo de videojuego en 2D que se planteó.

1.-Planificación del sprint

Durante 2 semanas se estuvo analizando que es lo que se quería lograr con la metodología anteriormente mencionada, haciendo esas modificaciones para que el proceso creativo pudiera ir de la mano junto con la ingeniería de software sin tener que esperar a saber que se había hecho o que no, lo primero fue ajustar fechas para cada parte del proyecto, incluyendo diseños de nivel, escenarios, jugabilidad, mecánicas de juego, diseño de enemigos, diseño de plataformas y diseño de fondos, por decir lo más importante.

El equipo creativo tuvo como meta planificar como seria la mecánica de juego, ya que analizo que sin eso no tendrían un punto de salida para el desarrollo total del juego, el prototipo de juego se basó en una mecánica de mundo libre, no lineal, queriendo decir que podrías ir de un punto A, a un punto C, o de un punto B a un punto A, no que forzosamente se tuviera que seguir un orden lineal, para así, poder hacer más dinámica la jugabilidad de exploración.

También se optó por diseñar una mecánica plataformera como los clásicos juegos en 2D, tales como Kirby, Mario Bros, o Hollow Night, por mencionar algunos, aparte de que se buscó que fuera por niveles, como se mencionó antes, así que se tenía que diseñar cada nivel, y lo más importante, como se pudiera avanzar o regresar a otro nivel.

El equipo de desarrollo, o de ingeniería de software, se dedicó a hacer la programación en base a las mecánicas de jugabilidad que el equipo creativo planteo, si bien se encontraron con varios problemas que más adelante se van a abordar, el trabajar mediante sprints iterativos hizo que esto no significara una demora que afectara los tiempos propuestos para el calendario que se planteó.

2.- Desarrollo e Implementación

Ilustración 1 Diseño de Nivel 1

El primer nivel conceptualizado fue un escenario ambientado en un bosque (véase Ilustración 1), ya que, según la historia desarrollada para el protagonista, este debía atravesar un bosque antes de llegar a los siguientes niveles. Para este nivel se diseñó una mecánica de plataformas sencilla, con saltos y enemigos básicos, con el objetivo de que el jugador pudiera familiarizarse con las mecánicas del juego. Los enemigos fueron seleccionados cuidadosamente para armonizar con la ambientación del bosque.

El primer enemigo diseñado fue una criatura tipo Slime (Ilustración 2), que podía ser eliminada al saltar sobre ella. Se introdujo una variación en la que algunos Slimes requerían un segundo salto para ser derrotados. Como es habitual en los videojuegos, eliminar a estos enemigos otorgaba puntos al jugador. El daño que infligen ocurre únicamente cuando el jugador se acerca demasiado, quitándole un corazón de vida.

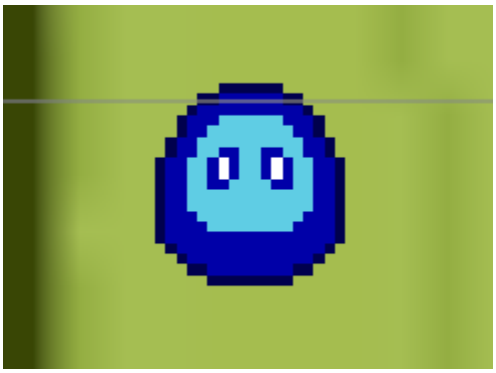


Ilustración 2 Enemigo Slime

El siguiente enemigo diseñado para este nivel fue una flama (Ilustración 3), una versión mutada del Slime que, de ser inofensiva, se convirtió en una amenaza peligrosa.

La mecánica propuesta para este enemigo se centraba en evitarlo a toda costa, ya que no podía ser eliminado. Al contrario, cualquier contacto con la flama resultaría en la eliminación instantánea del jugador.

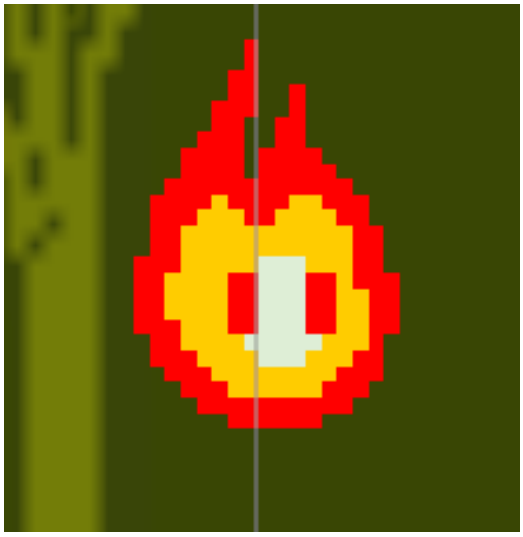


Ilustración 3 Enemigo Slime de Fuego

Aunque los enemigos principales eran solo dos, el equipo sintió que el prototipo necesitaba un mayor desafío. Para lograrlo, decidieron agregar obstáculos, como los pinchos (Ilustración 4). Al igual que la flama, estos eliminaban al jugador con solo tocarlos, pero tenían la ventaja de ser estáticos, lo que permitía al jugador esquivarlos con mayor facilidad y continuar explorando el nivel sin complicaciones.

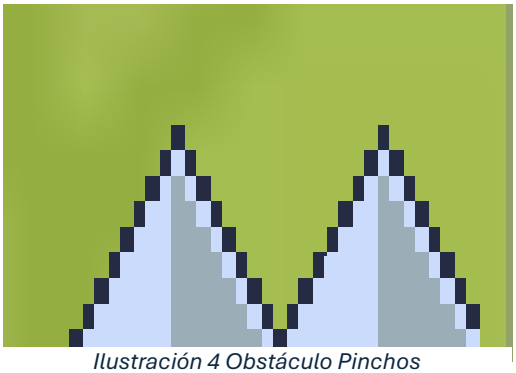


Ilustración 4 Obstáculo Pinchos

El siguiente obstáculo diseñado fueron plataformas que requerían saltos, como es común en este tipo de juegos (Ilustración 5). Sin embargo, al implementarlas, surgió un error que llevó al equipo creativo a buscar una solución. Como resultado, se introdujo una nueva mecánica en el juego: el doble salto, activado solo cuando el jugador estuviera cerca de una pared. Además, se estableció un límite de un salto extra por plataforma, permitiendo repetirlo solo después de volver a tocar el suelo.

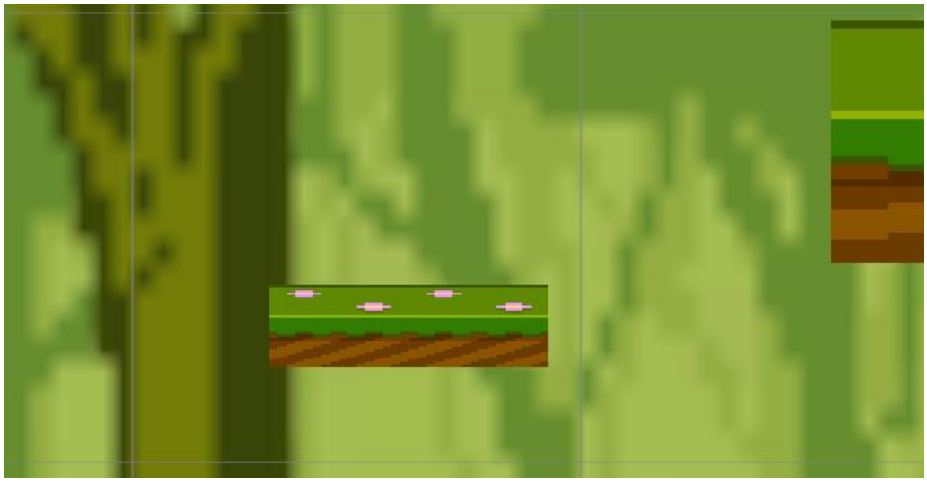


Ilustración 5 Obstáculo Plataformas Elevada

El equipo creativo también consideró que explorar un nivel sin una motivación adicional resultaría monótono. Por ello, decidieron incorporar elementos que incentivaran la exploración completa del escenario. se añadieron monedas de oro, que los jugadores podrían recolectar, así como corazones que restaurarían 1 punto

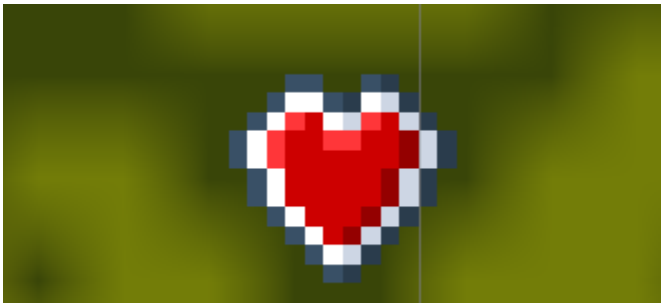


Ilustración 7 Objeto Corazón

Ilustración 6 Objeto Amatista

de vida, además, se incluyeron objetos más valiosos, las amatistas, que otorgarían una mayor cantidad de puntos y vidas adicionales (Ilustraciones 6 y 7).

Inicialmente, para los cambios de nivel, se había pensado en un "botón" que, al saltar sobre él, transportara al jugador al siguiente nivel vinculado. Sin embargo, se consideró que esto rompía con la estética de mundo abierto del juego. En su lugar, se optó por implementar una zona que, al tocarla, teletransportara al jugador al nivel programado (Imagen 8). Esta solución aumentaba la inmersión, permitiendo que el

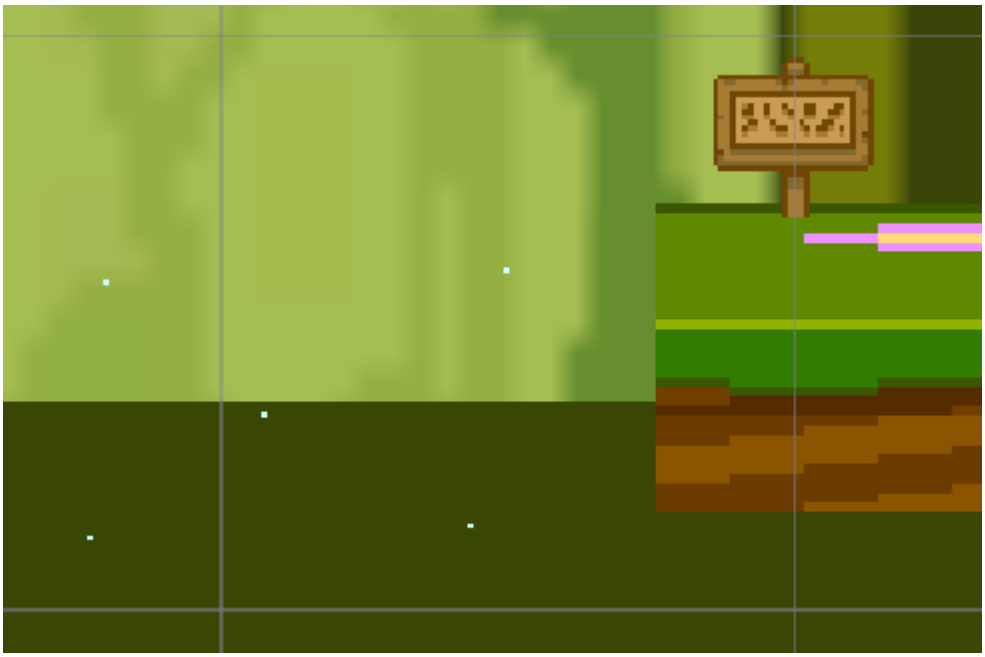


Ilustración 8 Punto de transporte 1

jugador accediera a los nuevos niveles ya sea caminando o cayendo, haciendo la mecánica más dinámica.

Para señalizar estos puntos de transporte, se añadió un cartel (Ilustración 9), de manera que el jugador pudiera identificar estas áreas sin ser tomado por sorpresa.

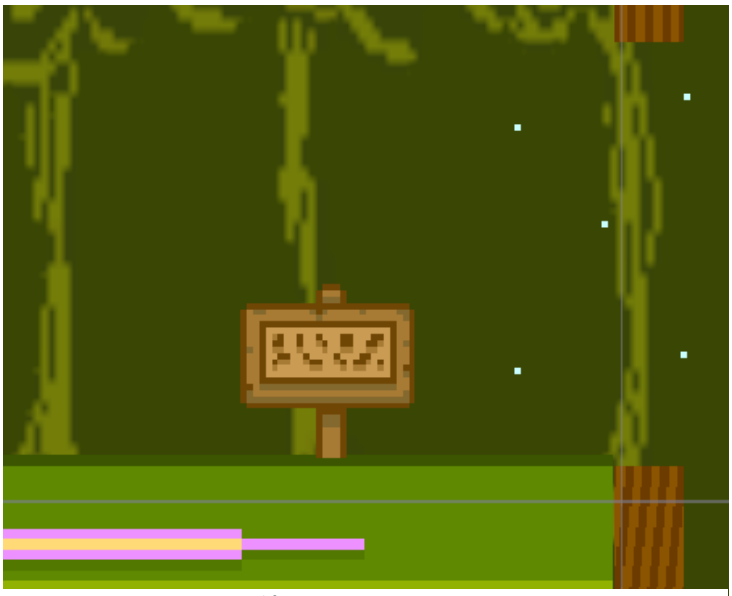


Ilustración 10 Punto de Transporte 2

Y finalmente, pero no menos importante, el personaje principal (Ilustración 10). Para su diseño, el equipo creativo decidió utilizar un asset adquirido en un curso de Domestika (véase en referencias), ya que encajaba con la visión que tenían para el juego. Aunque originalmente se consideró crear un personaje desde cero, debido a

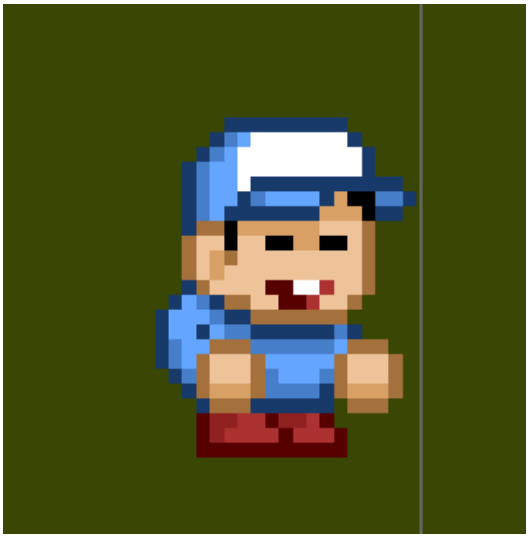


Ilustración 9 Personaje del Juego

las complicaciones mencionadas al inicio de este apartado, se optó por usar este recurso, agradeciendo nuevamente a Domestika por el material proporcionado.

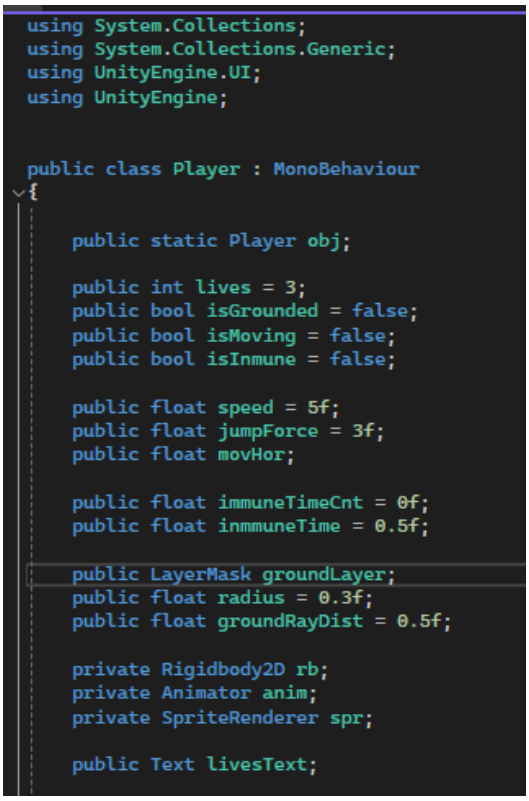
El personaje principal contaría con diversas mecánicas: la habilidad de saltar, moverse tanto hacia la derecha como hacia la izquierda, realizar un doble salto cuando estuviera cerca de una pared para alcanzar zonas más altas del nivel, y una hitbox de recolección. Esta hitbox permitiría al personaje tanto dañar a los enemigos susceptibles al ataque, como recoger monedas de oro, corazones y amatistas distribuidas por los diferentes niveles.

Creación de Código

Una vez definido todo lo anterior, fue el turno del equipo de desarrollo para convertir las ideas del equipo creativo en algo funcional. Para ello, eligieron Unity como la plataforma en la que comenzarían a crear el entorno y desarrollar las mecánicas planteadas. Unity, al utilizar C# como lenguaje de programación por defecto, facilitó que el equipo continuara con este lenguaje, aprovechando sus beneficios para el desarrollo de juegos. El código fue editado y creado en Visual Studio, seleccionado por su simplicidad y practicidad, lo que permitía un mejor control sobre el proceso de programación.

El primer paso fue programar al personaje, ya que este constituiría la base sobre la cual se implementaría el resto de las funcionalidades. El equipo se aseguró de que todo lo demás interactuara correctamente con el código del personaje, realizando ajustes para corregir errores o afinar ciertos detalles, con el objetivo de mantener un código limpio y eficiente.

El código denominado "Player" (Ilustración 11) fue el más extenso de todos los que se programaron, ya que centralizaba la mayor parte de las interacciones del jugador.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;


public class Player : MonoBehaviour
{

    public static Player obj;

    public int lives = 3;
    public bool isGrounded = false;
    public bool isMoving = false;
    public bool isInmune = false;

    public float speed = 5f;
    public float jumpForce = 3f;
    public float movHor;

    public float immuneTimeCnt = 0f;
    public float inmmuneTime = 0.5f;

    public LayerMask groundLayer;
    public float radius = 0.3f;
    public float groundRayDist = 0.5f;

    private Rigidbody2D rb;
    private Animator anim;
    private SpriteRenderer spr;

    public Text livesText;
```

Ilustración 11 Código Player 1 Parte 1

La primera parte son todas las librerías que Unity necesita para poder compilar y hacer funcionar lo que se llegue a programar, se pueden añadir más librerías para diferentes casos, como lo veremos más adelante.

En la segunda parte de más abajo se programaron todas las variables que el personaje iba a tener, como las vidas, variables booleanas para saber si se estaban en ciertos estados, fuerza de salto, el radio del hitbox que se mencionó anteriormente, animaciones para poder hacer que el personaje tenga un dinamismo y no solo este estático como una imagen, y textos de vida y puntaje.

En la función Start (Ilustración 12) se cargan todos los parámetros iniciales, que serían las animaciones, el rigidbody para las físicas, tipo gravedad del personaje, colisiones, etc.

En la función update, es para el apartado que va a ir actualizando el comportamiento del personaje, como el movimiento, ya que la mayor parte del

```
private void Awake()
{
    obj = this;
}

// Start is called before the first frame update
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
    spr = GetComponent<SpriteRenderer>();

    UpdateLivesText();
}

// Update is called once per frame
void Update()
{
    //int puntajeActual = DatosPersistentes.Instance.GetPuntaje();
    //int vidasActuales = DatosPersistentes.Instance.GetVidas();

    movHor = Input.GetAxisRaw("Horizontal");

    isMoving = (movHor != 0f);

    isGrounded = Physics2D.CircleCast(transform.position, radius, Vector3.down, groundRayDist, groundLayer);

    if (Input.GetKeyDown(KeyCode.Space))
        jump();


    anim.SetBool("isMoving", isMoving);
    anim.SetBool("isGrounded", isGrounded);

    flip(movHor);

}
```

Ilustración 12 Código Player Parte 2

código que aquí podemos ver es del movimiento, y que pasa cuando hacemos click sobre la tecla asignada para que el personaje salte o se mueva de manera horizontal

o verticalmente click sobre la tecla asignada para que el personaje salte o se mueva ya sea de manera horizontal o verticalmente.

La función FixedUpdate (Ilustración 13) funciona para corregir algunos errores de movimiento que puedan llegar a ocurrir al momento que la función Update hace el compilado dentro del prototipo de videojuego.

```
void FixedUpdate()
{
    rb.velocity = new Vector2(movHor * speed, rb.velocity.y);
}

void jump()
{
    if (!isGrounded) return;

    rb.velocity = Vector2.up * jumpForce;
}

void flip(float _xValue)
{
    Vector3 theScale = transform.localScale;

    if (_xValue < 0)
        theScale.x = Mathf.Abs(theScale.x) * -1;
    else
        if (_xValue > 0)
        theScale.x = Mathf.Abs(theScale.x);

    transform.localScale = theScale;
}

public void getDamage()
{
    lives--;
    UpdateLivesText();

    if (lives <= 0){
        //this.gameObject.SetActive(false);
        Game.obj.gameOver();

    }

}
```

Ilustración 13 Código Player 3 Parte 3

La función Jump controla la velocidad de salto, tanto de inicio como de caída, la función flip hace que el personaje detecte cuando se quiere ir a la izquierda y se “voltee” hacia la izquierda y viceversa, si voltea hacia la derecha, girara hacia la derecha.

La función getDamage funciona para que el contador de vidas baje si el personaje recibe daño, al igual que tiene un condicional que si el personaje llega a 0 vidas el juego se reinicie.

La función get vidas (Ilustración 14) funciona para que se pueda agregar en la función UpdateLivesText para poder poner un texto con un contador de vidas visible en la interfaz de la pantalla que se actualiza al momento de recibir daño con la

```
//Nueva Linea
public int GetVidas()
{
    return lives;
}

public void addLive()
{
    lives++;
    UpdateLivesText();

    if(lives > Game.obj.maxLives)
    lives = Game.obj.maxLives;

}

void UpdateLivesText()
{
    livesText.text = "Vidas: " + lives.ToString();
}

void OnDestroy()
{
    obj = null;
}
```

Ilustración 14 Código Player 4 Parte 4

función antes mencionada, o que se actualice también con la función addLives que sería cuando el personaje recoge una vida adicional, también teniendo un condicional que para que no llegue a más de una cantidad establecida dentro de los parámetros del personaje que se declararon mucho más arriba

El siguiente código que se programo fue el de los enemigos (Ilustración 15), como se puede observar tiene las mismas librerías que la de Player, pero en algunos parámetros cambian, ya que aquí se coloca que tenga movimiento por sí solo,

respetando de izquierda a derecha, que el enemigo haga un check si aún hay plataforma delante de él, y si no, que cambie de dirección.

Se le asigno un RaycastHit para que pueda ir analizando si no hay un precipicio delante de él y no se caiga alterando así la experiencia de juego que se buscaba conseguir.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Enemy : MonoBehaviour
{

    private Rigidbody2D rb;

    public float movHor = 0f;
    public float speed = 3f;

    public bool isGroundFloor = true;
    public bool isGroundFront = false;

    public LayerMask groundLayer;
    public float frontGrndRayDist = 0.25f;
    public float floorCheckY = 0.52f;
    public float frontCheck = 0.51f;
    public float frontDist = 0.001f;

    public int scoreGive = 50;

    public RaycastHit2D hit;
```

Ilustración 15 Código enemigo parte 1

Este código fue uno de los más complejos de programar por las mecánicas que se le buscaron dar al enemigo, que sería el hecho de que cheque si aún queda plataforma delante de él, que pueda revisar si hay un precipicio delante de él y la más importante que el equipo de desarrollo pensó, que cambie de dirección cada

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();
}

// Update is called once per frame
void Update()
{
    //No caer en precipicio
    isGroundFloor = (Physics2D.Raycast(new Vector3(transform.position.x, transform.position.y - floorCheckY, transform.position.z), new Vector3(movHor, 0, 0), frontGrndRayDist, groundLayer));

    if (!isGroundFloor)
        movHor = movHor * -1;

    //Cambio en pared
    if (Physics2D.Raycast(transform.position, new Vector3(movHor, 0, 0), frontCheck, groundLayer))
        movHor = movHor * -1;

    //Cambio Enemigo
    hit = Physics2D.Raycast(new Vector3(transform.position.x + movHor * frontCheck, transform.position.y, transform.position.z), new Vector3(movHor, 0, 0), frontDist);

    if (hit != null)
        if(hit.transform != null)
            if (hit.transform.CompareTag("Enemy"))
            movHor = movHor * -1;


}

void FixedUpdate()
{
    rb.velocity = new Vector2(movHor * speed,rb.velocity.y);
}
```

Ilustración 16 Código Enemigo Parte 2

vez que se encuentre con otro enemigo, para que así no se queden atascados a la mitad de una plataforma (Ilustración 16).

En este apartado (Ilustración 17) ya se trabajan los comportamientos que se tendrán al interactuar con el enemigo, desde que te haga daño en la función OnCollisionEnter2D, que el enemigo sea eliminado con la función OnTriggerEnter2D y dentro de esta función se tiene un condicional para que el

```
void FixedUpdate()
{
    rb.velocity = new Vector2(movHor * speed,rb.velocity.y);
}

void OnCollisionEnter2D(Collision2D collision)
{
    //danio
    if (collision.gameObject.CompareTag("Player"))
    {
        //danio player
        Player.obj.getDamage();
    }

}

void OnTriggerEnter2D(Collider2D collision)
{
    //danio enemigo
    if (collision.gameObject.gameObject.CompareTag("Player"))
    {
        //danio enemigo
        getKilled();
    }

    if (collision.gameObject.CompareTag("Player"))
    {
        Game.obj.addScore(scoreGive);
    }

}

void getKilled()
{
    gameObject.SetActive(false);
}
```

Ilustración 17 Código Enemigo Parte 3

enemigo sea eliminado y te dé el puntaje que se haya establecido en la parte de más arriba.

Y, por último, la función getKilled funcionaria para “desactivar” al enemigo y que no siga apareciendo.

Este código (Ilustración 18) se hizo para poder controlar parámetros dentro del juego, como que se actualice la puntación cada vez que se elimine un enemigo y que se vaya mostrando en pantalla en tiempo real.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class Game : MonoBehaviour
{

    public static Game obj;

    public int maxLives = 5;

    public bool gamePaused = false;
    public int score = 0;

    public Text scoreText;



    void Awake()
    {
        obj = this;
    }


    // Start is called before the first frame update
    void Start()
    {
        gamePaused = false;
        UpdateScoreText();

    }


    public void addScore(int scoreGive)
    {
        score += scoreGive;
        UpdateScoreText();
    }

    //Nueva linea
    public int GetPuntaje()
    {
        return score;
    }

    public void gameOver()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }

    void UpdateScoreText()
    {
        scoreText.text = "Score: " + score.ToString();
    }

    void OnDestroy()
    {
```

Ilustración 18 Código Parámetros del Juego

La función gameOver se hizo para que reiniciara el nivel y se pudiera volver a intentar, también dentro del código se coloca la cantidad de vidas máximas que el personaje va a poder tener, cada vez que el juego se llegase a pausar, entre varias otras cosas que se complementan con otros códigos que se van a ir mencionando o que ya se mencionaron.

El siguiente código que se programo fue el del enemigo flama (Ilustración 19), ya que como se mencionó en el apartado de proceso creativo se buscaba que su comportamiento fuera diferente al del enemigo común, así que en algunas cosas se va a parecer al enemigo común, como el hecho de que haga un check para saber si hay enemigos delante de él, o si hay un precipicio y cambie de dirección, la velocidad de movimiento que va a tener, etc.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Flama : MonoBehaviour
{

    private Rigidbody2D rb;

    public float movHor = 0f;
    public float speed = 3f;

    public bool isGroundFloor = true;
    public bool isGroundFront = false;

    public LayerMask groundLayer;
    public float frontGrndRayDist = 0.25f;
    public float floorCheckY = 0.52f;
    public float frontCheck = 0.51f;
    public float frontDist = 0.001f;

    public int scoreGive = 50;

    public RaycastHit2D hit;
```

Ilustración 19 Código enemigo flama Parte 1

En esta parte es muy parecido a lo que tiene el enemigo común (Ilustración 20), con algunas modificaciones para que no sea parecido, ya que sigue teniendo el efecto de que no caiga en un precipicio, o cambie de dirección en una pared, y también cambie de dirección cuando choque con un enemigo, pero tiene la función agregada de flip, ya que por el diseño que se obtuvo del enemigo flama, cada vez que cambiaba de dirección solo quedaba viendo hacia un lado, así que se le puso la mecánica de que rotara el sprite como lo hace el personaje, pero se tuvo el reto de hacerlo de cierta manera automático, ya que el enemigo flama al igual que el común,

se mueve por sí solo, y a diferencia con el personaje Player, no tenía una tecla que se pudiera vincular con ese cambio de posición.

```
// Update is called once per frame
void Update()
{
    //No caer en precipicio
    isGroundFloor = (Physics2D.Raycast(new Vector3(transform.position.x, transform.position.y - floorCheckY, transform.position.z), new Vector3(movHor, 0, 0), frontGrndRayDist, groundLayer));

    if (!isGroundFloor)
        movHor = movHor * -1;

    //Cambio en pared
    if (Physics2D.Raycast(transform.position, new Vector3(movHor, 0, 0), frontCheck, groundLayer))
        movHor = movHor * -1;

    //Cambio Enemigo
    hit = Physics2D.Raycast(new Vector3(transform.position.x + movHor * frontCheck, transform.position.y, transform.position.z), new Vector3(movHor, 0, 0), frontDist);

    if (hit != null)
        if(hit.transform != null)
            if (hit.transform.CompareTag("Enemy"))
            movHor = movHor * -1;


    flip(movHor);


}

void FixedUpdate()
```

Ilustración 20 Código enemigo Flama Parte 2

```
void FixedUpdate()
{
     rb.velocity = new Vector2(movHor * speed,rb.velocity.y);
}

void OnCollisionEnter2D(Collision2D collision)
{
    //danio
    if (collision.gameObject.CompareTag("Player"))
    {
        //danio player
        Player.obj.getDamage();
    }

}

void flip(float _xValue)
{
    Vector3 theScale = transform.localScale;

    if (_xValue < 0)
        theScale.x = Mathf.Abs(theScale.x) * -1;
    else
        if (_xValue > 0)
        theScale.x = Mathf.Abs(theScale.x);

    transform.localScale = theScale;
}

void OnTriggerEnter2D(Collider2D collision)
{
    //danio enemigo
    if (collision.gameObject.gameObject.CompareTag("Player"))
    {
        //danio enemigo
        getKilled();
    }

    if (collision.gameObject.CompareTag("Player"))
    {
        Game.obj.addScore(scoreGive);
    }

}

void getKilled()
```

Ilustración 21 Código Enemigo Flama Parte 3

Todo el trabajo que se hizo para poder lograr ese cambio de dirección en el enemigo se logró en la función flip (Ilustración 21), que funciona de manera similar a la del personaje player, pero se modificó para que detectara cada vez que el enemigo cambiaba de dirección para que también pudiera trabajar el cambio de vista del sprite, rotándolo 180 grados dando la sensación de que el personaje gira y tiene un uso de razón pequeño, aparte de que como se mencionó, se buscaba que el enemigo eliminara el personaje con solo tocarlo, así que dentro de la función es OnTriggerEnter2D, se le coloco que haga una comparación de la Tag del Personaje para así poderlo eliminar con solo tocarlo que es lo que se quería.

El siguiente código que se programo fue en de los obstáculos con picos que se mencionaron más arriba (Ilustración 22), el código por sí solo solo fue un copy paste

de la función que se programó más arriba llamada getKilled, solamente haciéndole unas modificaciones para que funcione de manera correcta.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Killer : MonoBehaviour
{
    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            Game.obj.gameOver();
        }
    }
}
```

Ilustración 22 Código Obstáculo con Picos

Los siguientes códigos que se programaron fueron los de las vidas y de las monedas que se podían ir recogiendo a lo largo del nivel

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour
{
    public int scoreGive = 100;

    void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            Game.obj.addScore(scoreGive);

            gameObject.SetActive(false);
        }
    }
}
```

Ilustración 23 Código Objeto Monedas

Este código es para las monedas (Ilustración 23), y como se puede observar solo se les agrego una función OnTriggerEnter2D para que reaccionen cuando el Player pase por sobre ellas y así puedan dar el puntaje que se le haya colocado,

añadiéndolo al texto que se tiene en la interfaz y desactivando dicho objeto para que no vuelva a aparecer en el juego.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Vida : MonoBehaviour
{

    public int scoreGive = 30;

    void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            Game.obj.addScore(scoreGive);
            Player.obj.addLive();

            gameObject.SetActive(false);
        }
    }
}
```

Ilustración 24 Código Objeto Vidas

Y este otro código es para las vidas (Ilustración 24), como se puede apreciar, da puntaje, pero también agregar una vida al contador de vidas dentro de la interfaz

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Gema : MonoBehaviour
{

    public int scoreGive = 500;

    void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            Game.obj.addScore(scoreGive);
            Player.obj.addLive();

            gameObject.SetActive(false);
        }
    }
```

Ilustración 25 Código Objeto Amatista

con un condicional cada vez que se toque, pero, así como con la moneda, se eliminara del juego para que no se pueda volver a tomar.

El código de la amatista es el mismo que el de la vida (Ilustración 25), pero con el parámetro scoreGive modificado para que dé más puntaje, pero en general no cambia en nada más que no sea eso.

Y por último se programó el cambio de nivel (Ilustración 26), para poder viajar de cualquier nivel que se quisiese a otro nivel, siempre y cuando se encontrase el punto de ese cambio.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class CambioDeEscena : MonoBehaviour
{

    public string nombreDeLaEscena;

    public GameObject datosPersistentes;

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {

            //DatosPersistentes datos = datosPersistentes

            //datos.puntaje = Game.obj.score;
            //datos.vidas = Player.obj.lives;

            SceneManager.LoadScene(nombreDeLaEscena);
        }
    }

}
```

Ilustración 26 Código Cambio de Nivel

En este código se añadió la librería SceneManagement, que ayudaría con ese cambio de nivel que se buscaba poder lograr, y al igual que en otros códigos se usó una función OnTrigerEnter2D para poder activarla y poder ir al nivel que tenga ingresado en el parámetro nombreDeLaEscena.

Para la música se usó la plataforma online BandLab (Ilustración 28), porque fue la que más se prestaba al uso de plugins y dispositivos midi que se necesitaban para la creación de audio, y para la masterización, mezcla de audios y edición se usó el software FL Studio (Ilustración 27), con la música lo que se buscó conseguir era que

sintiera adecuada a lo que estaba pasando en los diferentes niveles también añadiendo un poco de la imaginación del compositor de cada canción que se usó.

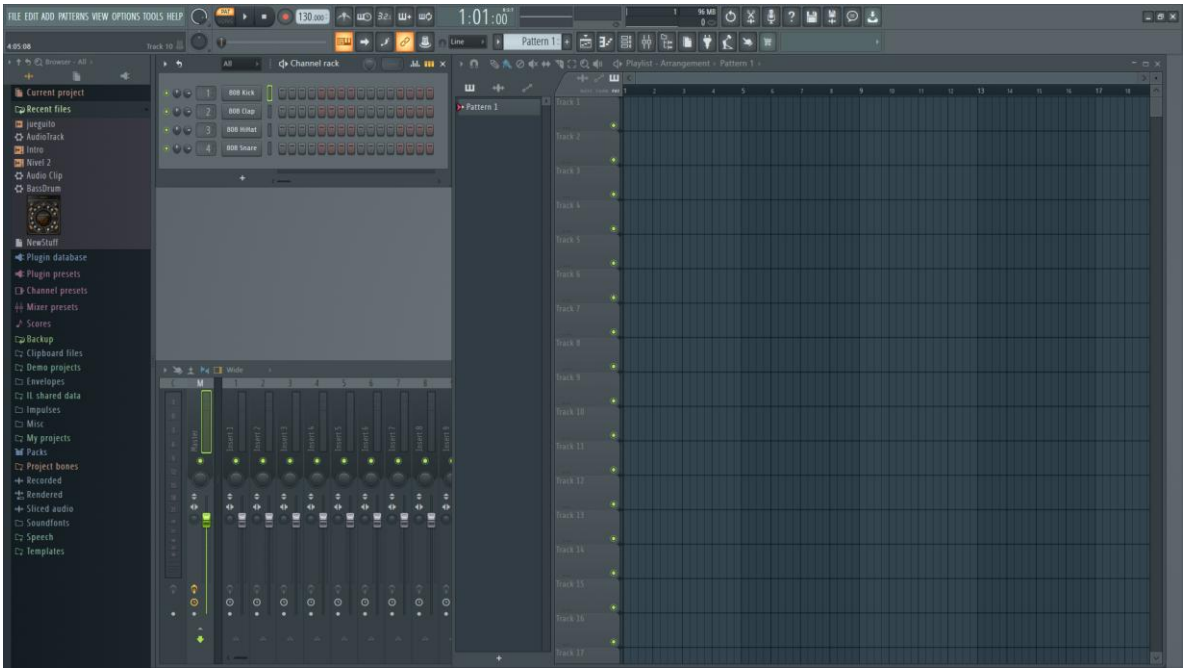


Ilustración 27 Interfaz de FL Studio

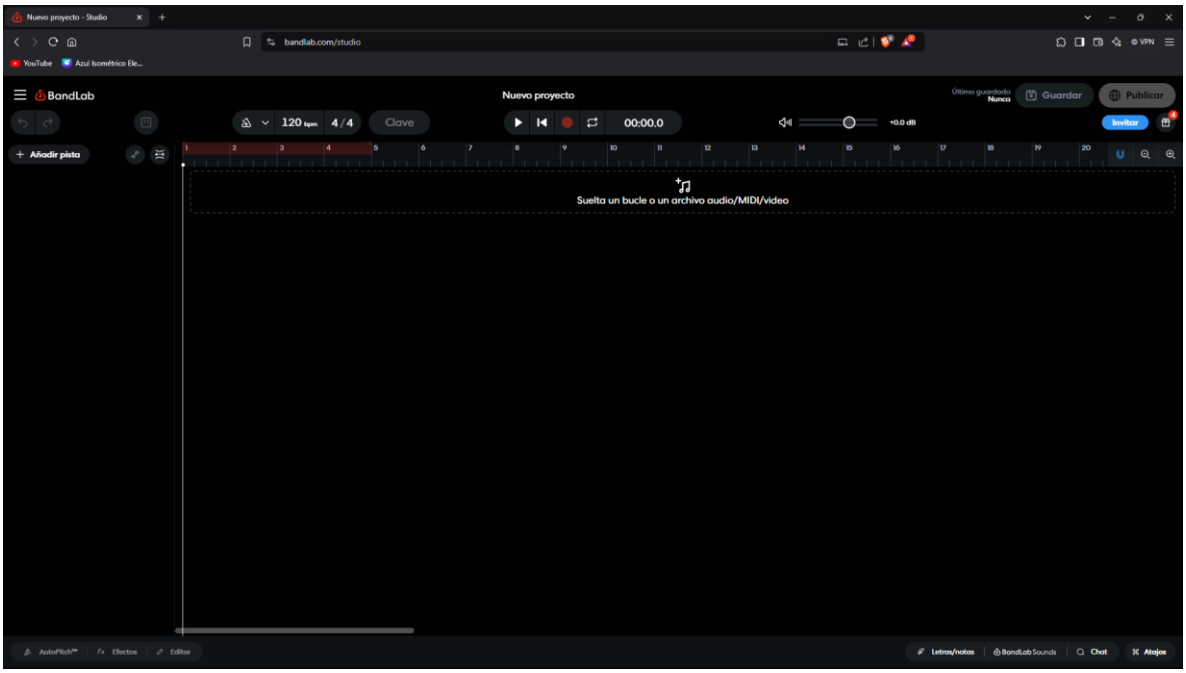


Ilustración 28 interfaz BandLab

Para el segundo nivel se pensó en la idea de un nivel de destreza (Ilustración 29), que se tuviera que saltar en el tiempo correcto, y se terminó de idealizar en un nivel de ciudad intentando esquivar camiones que estuvieran sobre las plataformas, para

así poder ir avanzando, y a diferencia de los enemigos del nivel 1, en el nivel 2 no desaparecerían, haciendo un poco más complejo el hecho de que se tenga que investigar todo el nivel.

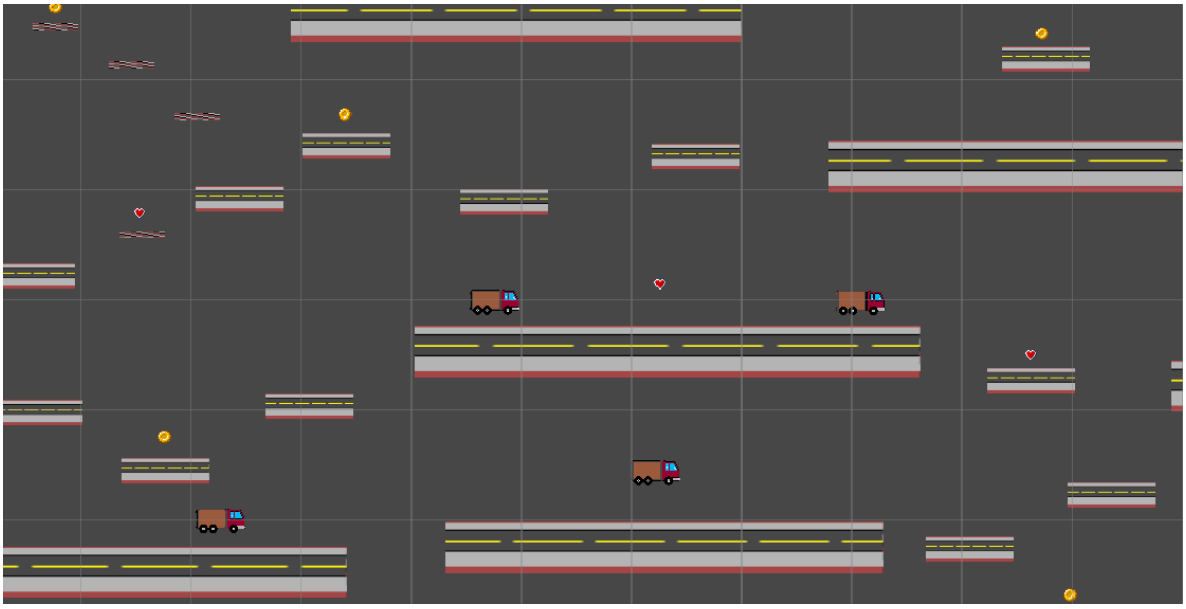


Ilustración 29 Nivel 2

Se siguieron usando las vidas, monedas y amatistas que más arriba se mencionaron, únicamente se diseñó a manera de que las plataformas significaran un desafío para el jugador, ya que en este nivel no habría enemigos, pero si habrá obstáculos, que en este caso serian camiones, que no se eliminan del juego, solo se pueden evitar saltando a tiempo para evitar ser golpeados por ellos.

El diseño el camión se pensó simple (Ilustración 30), pero pensado para que encajase en lo que se buscaba transmitir al jugador, una sensación de que pudiera o no pudiera llegar a esquivarlo si no saltaba a tiempo, para su código de

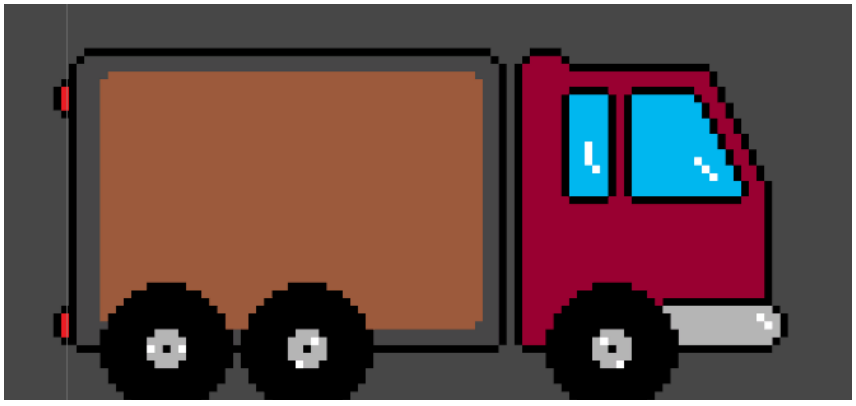


Ilustración 30 Enemigo Camión

funcionamiento se reutilizo el del enemigo flama, únicamente desactivando los parámetros que quitara todas las vidas al tocarlo.

Para el tercer nivel y siguiendo la historia que se planteó (Ilustración 31), se diseñó el nivel en base a un castillo que, los enemigos serian murciélagos y esqueletos, y las plataformas estarían un poco acomodadas dando la alusión a las torres de vigilancia que tenían los castillos

Los enemigos tienen las mismas mecánicas que el nivel 1, solo se pueden eliminar a los esqueletos cayendo sobre ellos (Ilustración 33), y los murciélagos no se pueden eliminar (Ilustración 32) y también te quitarían todos tus corazones con solo

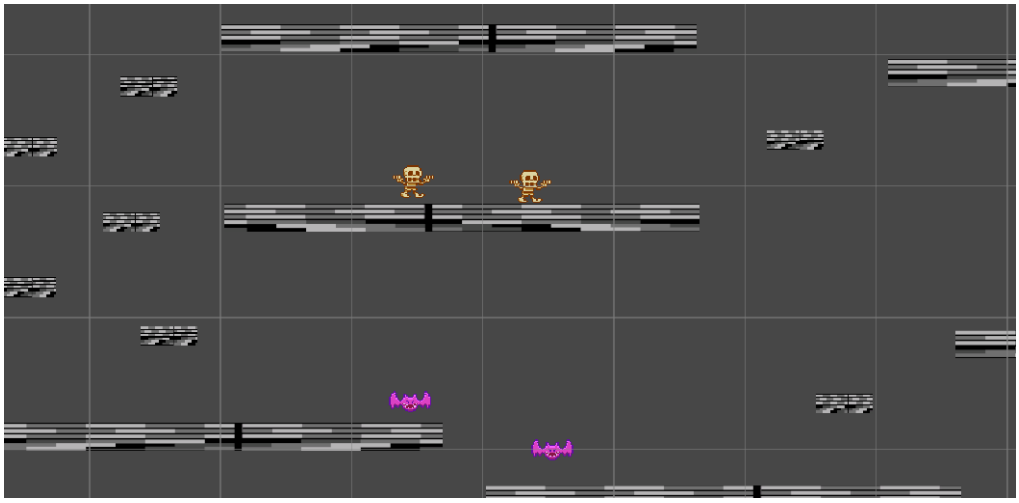


Ilustración 32 Nivel 3

Ilustración 31 Enemigo Murciélago

tocarte, así que igual que, con el enemigo flama, sería mejor solo correr y tratar de esquivarlos

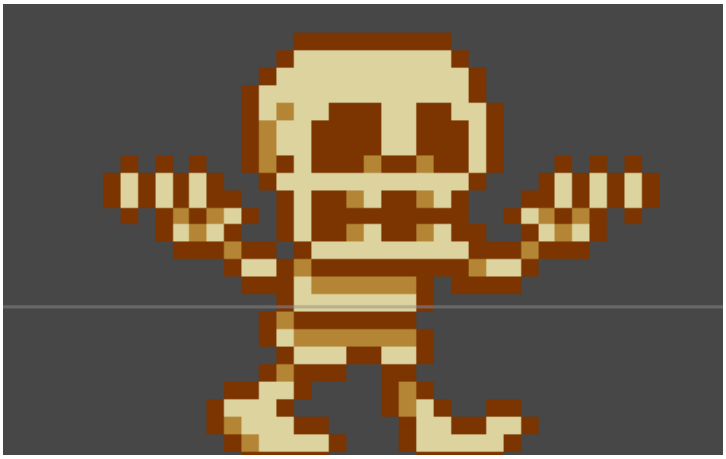


Ilustración 33 Enemigo Esqueleto

Capítulo 4 Resultados

Como se mencionó, se buscó que la ingeniera de software pudiera trabajar de la mano con el proceso creativo creando una metodología hibrida tomando las buenas prácticas de cada apartado, así que como se mostrara en la Ilustración 34, se creó un modelo UML a seguir para poder trabajar y respetar esto mismo que se quería conseguir.

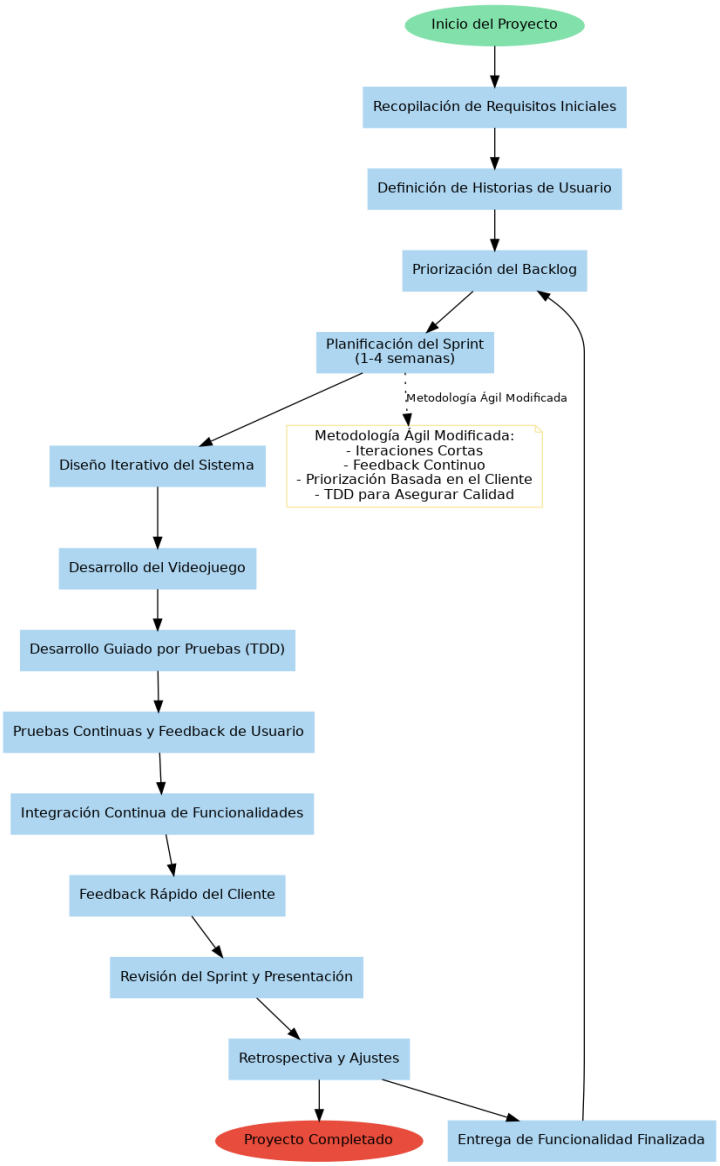


Ilustración 34 Modelo UML

El prototipo se terminó en un tiempo aproximado de 9 a 10 semanas, haciendo reuniones cada 4 semanas para informar avances, depurar errores y compartir nuevas ideas, o hacer observaciones.

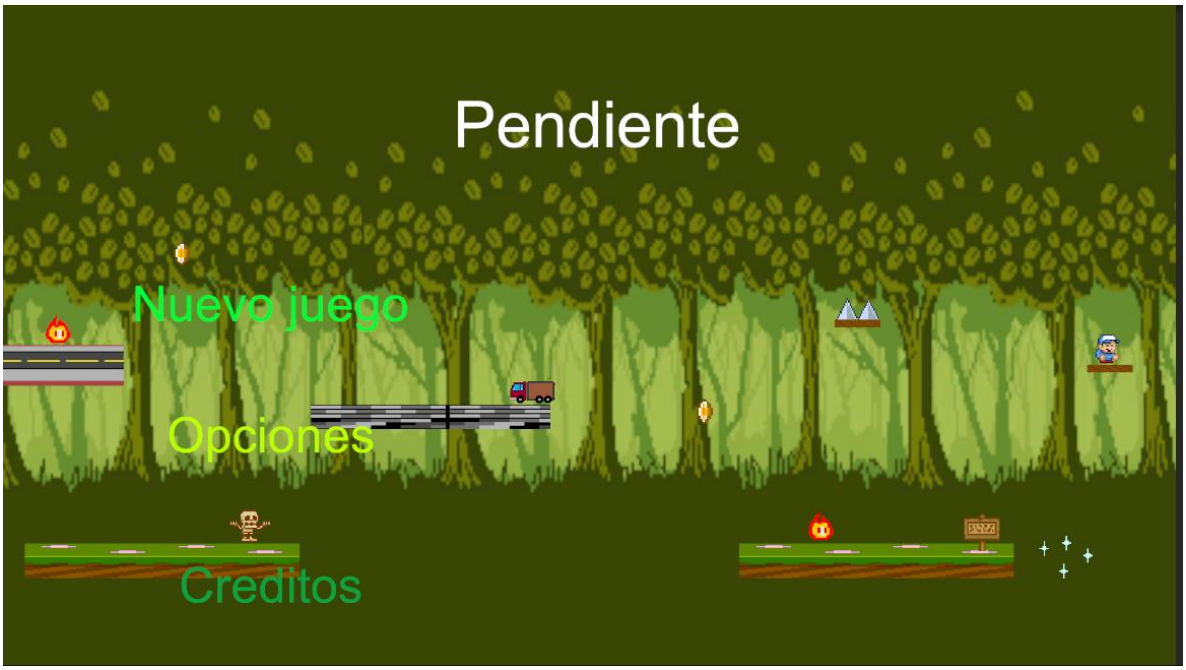


Ilustración 35 Inicio del prototipo de juego

El prototipo de videojuego 2D funciono de manera correcta en su primer apartado que sería el menú principal (ilustración 35), mediante el ratón se puede seleccionar un nuevo juego, ir a las opciones o a los créditos.

Al iniciar un nuevo juego, se empieza el tutorial por el cual el jugador conocerá cuales son los controles del juego (Ilustración 36).

Ilustración 36 Tutorial del prototipo

El tutorial explica mecánicas del prototipo para que el jugador pueda conocer cuáles son los controles del juego, así como conocer algunas mecánicas del juego, y una vez terminado el tutorial se puede jugar el prototipo con normalidad.

Ilustración 37 Prototipo nivel 1

Como se puede observar en la ilustración 37, el prototipo funciona con normalidad, se puede ver el puntaje y cuantas vidas se tienen, al igual que el mapa que se implementó para poder saber a dónde se puede ir.

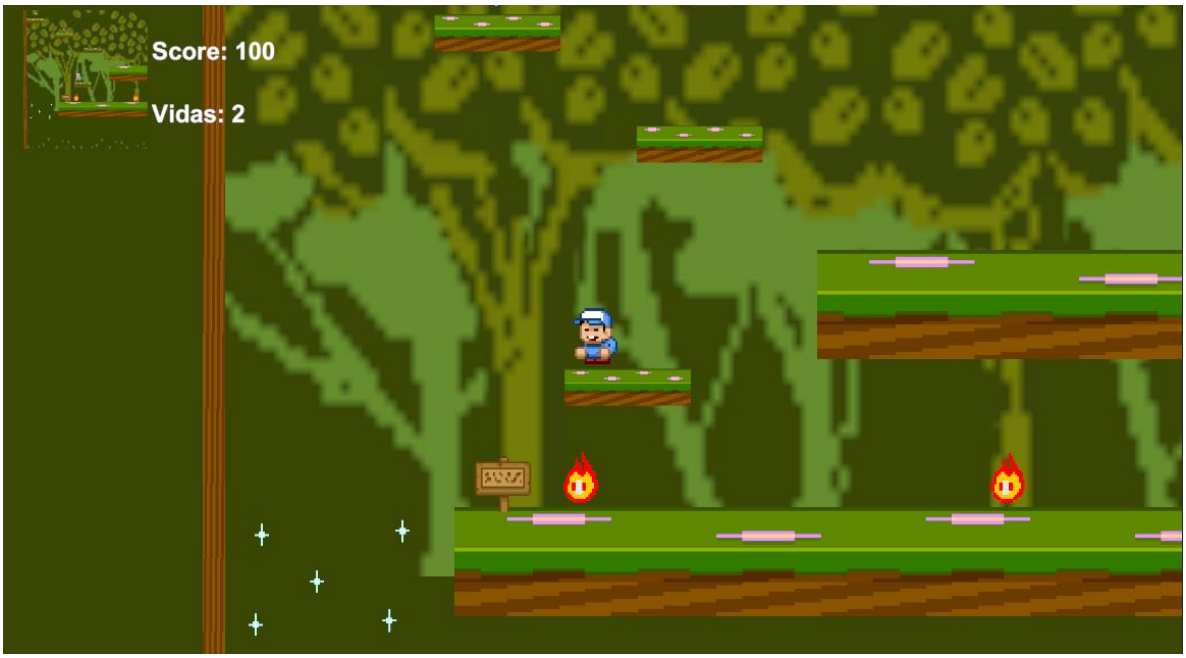


Ilustración 38 Primer punto de transporte

En la imagen 38 se puede ver uno de los puntos de transporte para poder llegar al siguiente nivel si el jugador así lo desea, o puede seguir explorando el nivel para poder encontrar el segundo punto de transporte del nivel.

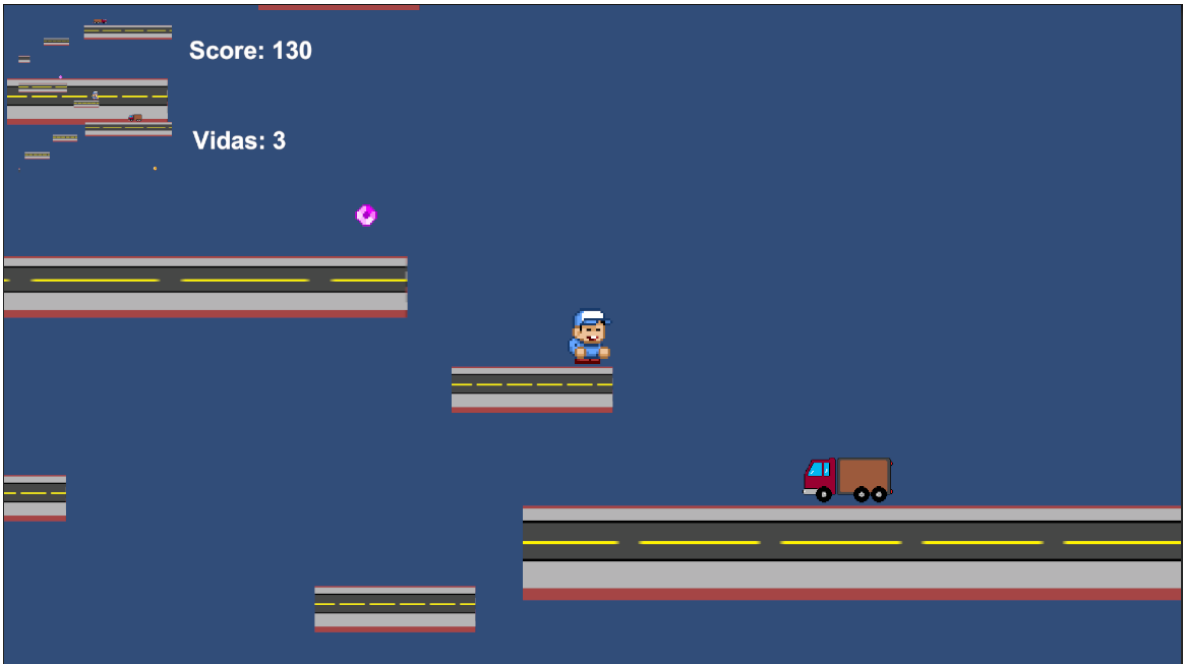


Ilustración 39 Prototipo Nivel 2

En la ilustración 39 se muestra el segundo nivel, aquí se busca aclarar que se tuvo un problema con algunos assets del prototipo y se perdieron los fondos del segundo y tercer nivel, pero se pudo rescatar el nivel funcional.

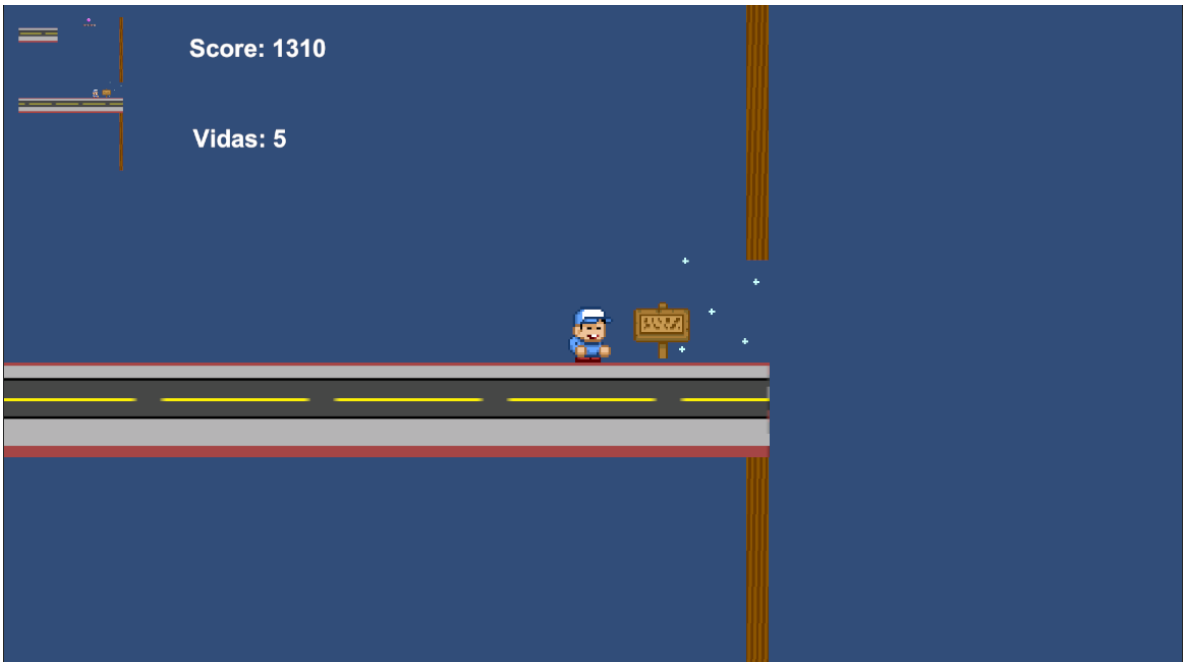


Ilustración 40 Punto de transporte del segundo nivel

Ilustración 41 Prototipo Tercer nivel

Como se puede ven en las ilustraciones 40 y 41, al usar los puntos de transporte y puede llegar a otro nivel, no siendo al azar, si no teniendo ese nivel vinculado para poder saber a dónde se está yendo cuando se vuelva a llegar a ese punto de transporte.

4.1- Encuestas Realizadas

Se realizo una encuesta en Google Forms a personas que ocuparon la metodología propuesta, para poder saber en que pudiera mejorar y en qué aspectos pudo ayudar más a los desarrolladores que optaron por el uso de esta metodología en sus proyectos.

Se usaron diferentes preguntas de control, tales como:

- ¿El DCA se acoplo a tu metodología de trabajo?

- ¿Facilito el proceso creativo dentro de tu proyecto?

Y algunas otras preguntas de satisfacción:

-En escala de 1 a 10, ¿Qué tan probable es que recomiendes el uso de este tipo de desarrollo?

- ¿Usarías el DCA para futuros proyectos tuyos donde se vea involucrado más el proceso creativo?

Se hizo esto con el fin de poder saber que beneficios tiene la metodología propuesta, en que partes puede no ser muy apta para el desarrollo de aplicaciones, y en que otras partes puede ser muy buena para este tipo de propósito.

Se buscaron desarrolladores que iniciaban con algún proyecto pequeño, o que ya tenían un proyecto iniciado y que pudieran incorporar partes de esta metodología, esto con el fin de que se pudiera tener una clara idea de en qué ayudo, y en que beneficio a un desarrollo, así como también poder saber las debilidades que tiene esta metodología y para que también se pudiera poner a prueba en otro tipo de proyectos que no sean solo videojuegos.

Aparte de que no se buscaba que este trabajo estuviera solo fundamentado en el trabajo que se propuso en el aplicativo que se realizó, si no tener más feedback sobre esta metodología de diferentes tipos de desarrolladores, ya que no todos trabajan de la misma manera, o trabajan solos. Siendo eso una gran ayuda para poder impulsar el crecimiento de esta metodología enfocada al proceso creativo.

4.2- Estadísticas

Se les dio a 5 desarrolladores independientes contestar la encuesta después de que pudieran implementarla en algún proyecto pequeño, o introducirlas dentro de proyectos que ya tuvieran algún tiempo de desarrollo para saber que tan flexible fue el trabajar con la metodología.

De los 5 desarrolladores 3 proyectos integraron la metodología desde 0, se desconoce si fue solo esa metodología o la combinaron con otro tipo de metodologías que pudieran afectar o beneficiar a la productividad de la metodología propuesta en este documento.

La primera pregunta fue:

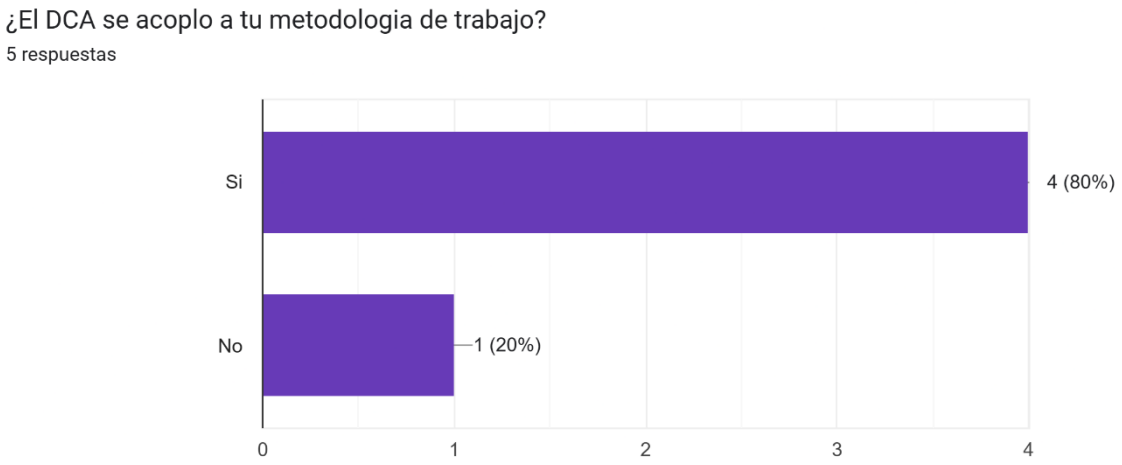


Ilustración 42 Primera pregunta

Como se puede apreciar en el gráfico, 4 de los 5 desarrolladores pudieron acoplar la metodología al tipo de proyecto que estaban trabajando.

De las 4 respuestas afirmativas 3 fueron de desarrollos iniciales y la restante fue de un desarrollo que ya estaba en curso.

La única respuesta negativa fue de un proyecto que igual estaba en pleno desarrollo y fue muy difícil poder hacer la introducción en ciertas partes para el desarrollo por la metodología que se manejó.

La segunda pregunta fue sobre la facilidad del proceso creativo con la metodología

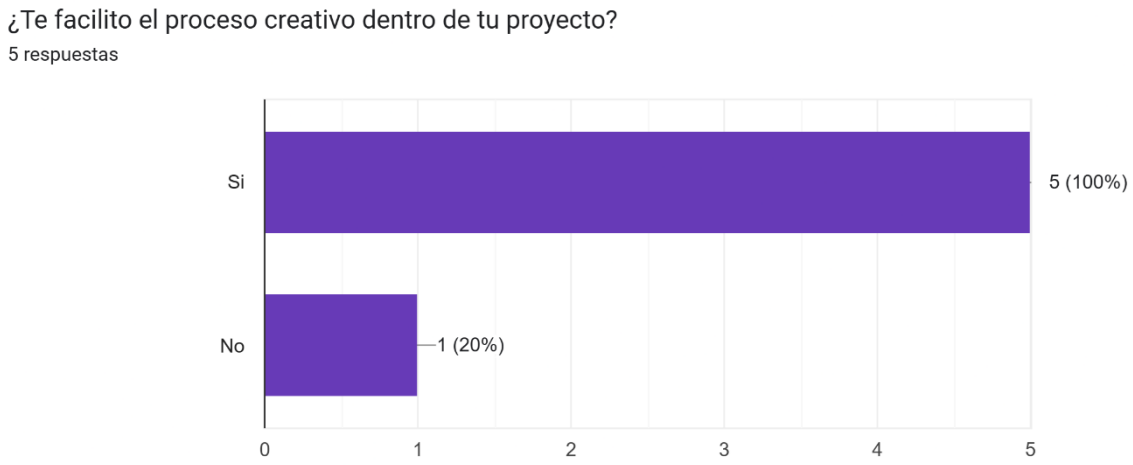


Ilustración 43 Segunda pregunta

Como se puede observar se respondió solo con un no a la pregunta sobre la facilidad con la que el proceso creativo se pudo trabajar dentro del proyecto, dando a entender que la metodología propuesta ayuda mucho para el proceso creativo.

Como tercera pregunta se quiso analizar la productividad que tiene la metodología:

Ilustración 44 Tercera pregunta

En el caso de las 5 respuestas, todas fueron afirmativas para el ahorro de tiempo usando la metodología propuesta, en el caso especial del proyecto que se manejó en este documento, fueron un total de 2 meses y 2 semanas, o un total de 10 sprints.

Como cuarta pregunta se hizo un control sobre los diferentes proyectos con los cuales se usó, o realizo una integración de la metodología propuesta:

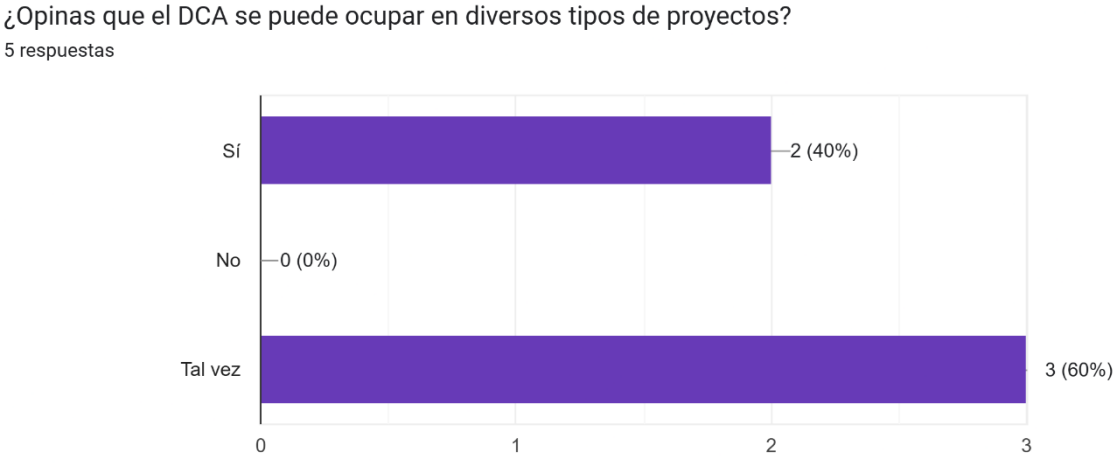


Ilustración 45 Cuarta pregunta

Como se puede observar 2 respuestas fueron afirmativas sobre proyectos que se iniciaron desde 0 con la implementación de esta metodología.

Las otras 3 respuestas fueron un tal vez, siendo 1 con desarrollo desde 0 y los 2 restantes fue con la implementación en una etapa avanzada de dicho proyecto.

Dejando sin contestar el apartado de no, demostrando que esta metodología pudiera llegar a ser más flexible en diferentes tipos de proyectos y no solo para el desarrollo de videojuegos.

Como quinta pregunta se analizó el si se recomendaría esta metodología a otros desarrolladores para que las usen en sus diversos proyectos:

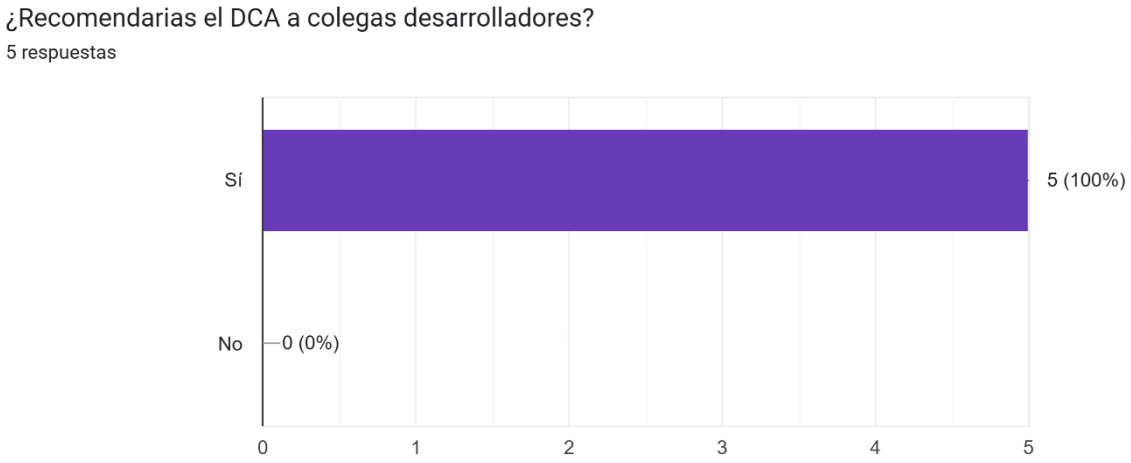


Ilustración 46 Quinta pregunta

Como se puede observar, en un 100% estadístico recomendaría esta metodología a otros desarrolladores, demostrando que puede ser muy versátil a la hora de desarrollar proyectos.

La sexta pregunta fue para poder saber que tanto los desarrolladores seguirían usando la metodología propuesta:

Ilustración 47 Sexta pregunta

Como se puede observar, un 60% usaría esta metodología para futuros proyectos, y un 40% tal vez la volvería a usar, esto demuestra que, para más de la mitad de desarrolladoras encuestados, la metodología propuesta fue cómoda para trabajar.

Como séptima pregunta se quiso saber si la metodología implementaba bien el proceso creativo dentro del desarrollo y planeación del proyecto o en su caso, la introducción de este:

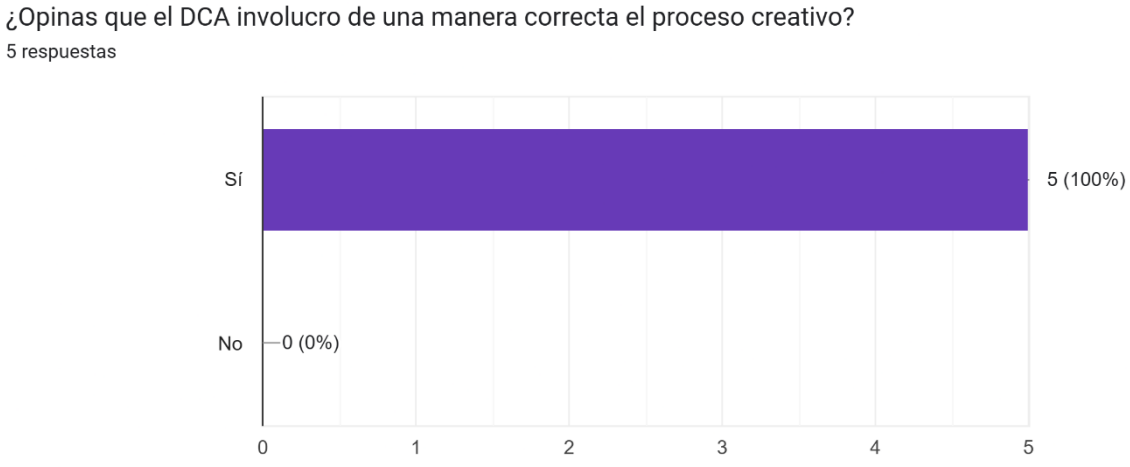


Ilustración 48 Séptima pregunta

Como se puede observar, todos los desarrolladores aceptaron que el DCA involucra de una buena manera el proceso creativo dentro del proyecto desarrollado, o lo introduce de una buena manera al desarrollo de un proyecto en curso.

Como octava pregunta se quiso conocer si los sprint propuestos en la metodología DCA estuvieron bien desarrollados:

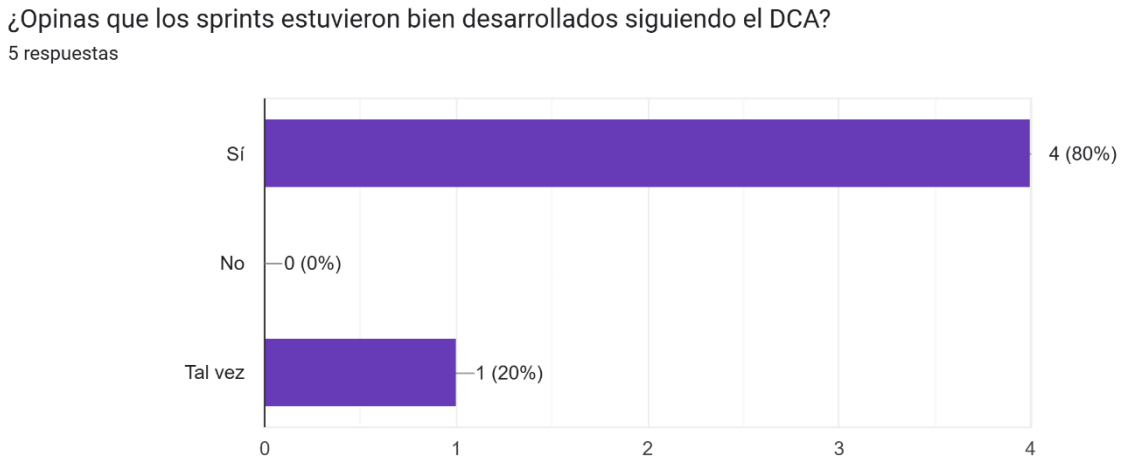


Ilustración 49 Octava pregunta

Como se puede observar 4 desarrolladores aceptaron de manera positiva a la pregunta, y solo 1 contesto con un tal vez dentro de la encuesta que se realizó, esto demuestra que la planeación de los sprints dentro de la metodología está bien desarrollada.

La penúltima pregunta se hizo con el motivo de saber si aparte de ser útil, el DCA puede hacer que el usuario se sienta cómodo, o “a gusto”:

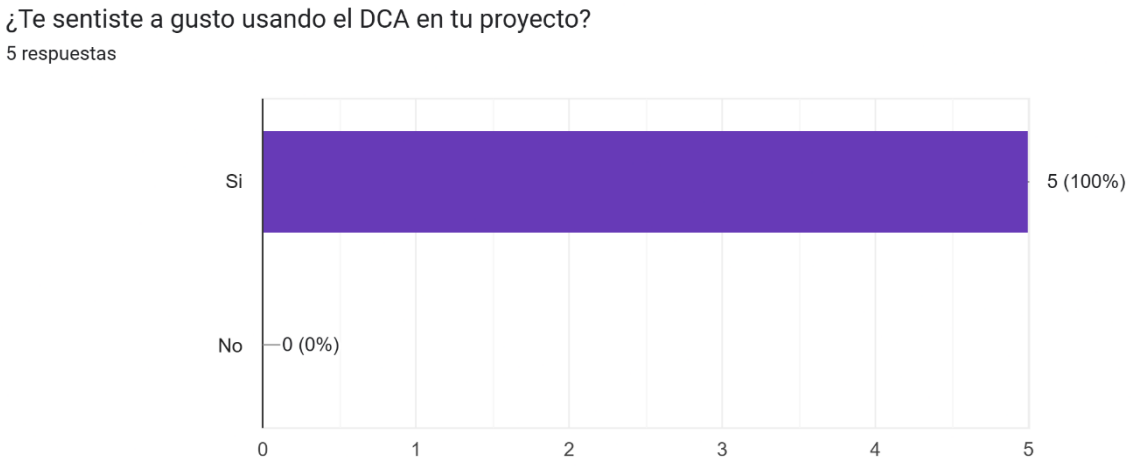


Ilustración 50 Novena pregunta

Como podemos observar tuvo una total aprobación a hacer sentir cómodos a todos los encuestados a pesar de que no todos lo usaron, o les facilito el trabajo, demostrando que no solo con más adaptabilidad, el DCA podría ser una metodología óptima para el desarrollo de varios proyectos.

Y como última pregunta se pidió que los evaluadores calificaran el DCA, para así poder sacar un promedio de calificación de usuarios:

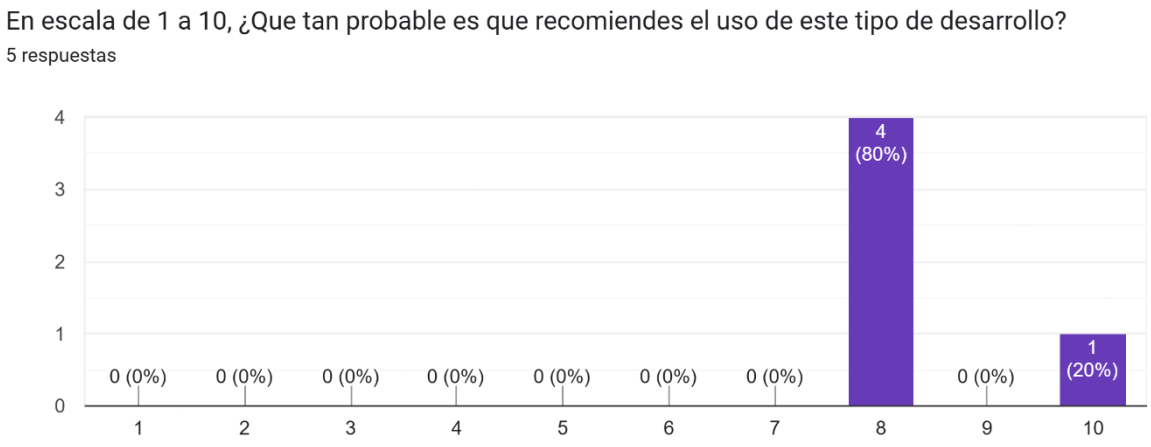


Ilustración 51Decima pregunta

Como se puede apreciar, 4 usuarios de la DCA la calificaron con un 8 y solo 1 la califico con 10, esta última pregunta nos demuestra que con una calificación promedio de 8.4, el DCA es una metodología que pudiera ser ampliamente aceptada por diferentes tipos de proyectos.

4.3.- Retrospectiva y Pruebas

Tras completar las primeras cuatro semanas de sprints, se realizó una prueba de funcionamiento e integración del código y la jugabilidad para evaluar los siguientes pasos a seguir. Sin embargo, al ejecutar el juego, el equipo de desarrollo se encontró con varios errores que afectaban la experiencia planeada.

Uno de los principales problemas fue la disposición de las plataformas en el nivel, que no estaban correctamente alineadas, lo que impedía que el jugador pudiera saltar de una a otra como se había previsto. Para corregir esto, se llevó a cabo un rediseño del nivel, aumentando el tamaño de las plataformas y ajustando ciertos parámetros del personaje para hacer la mecánica más eficiente sin que resultara excesivamente poderosa, evitando lo que el equipo denominaba "broken" o desbalanceado.

No obstante, al realizar estos ajustes, se detectó otro problema: las plataformas ubicadas por encima del personaje eran inalcanzables sin un rediseño completo del nivel. En lugar de rediseñar todo de nuevo, el equipo creativo propuso una solución: permitir que el personaje realizara un doble salto, justificándolo como si se agarrara del borde de una plataforma, lo que le permitiría alcanzar zonas más altas.

Esta idea no solo fue bien recibida por el equipo de desarrollo, sino que, tras implementarla, mejoró significativamente la jugabilidad. Se modificó el código y se ajustaron hitboxes tanto en enemigos como en plataformas, logrando una versión funcional del juego en sus primeros niveles.

El siguiente paso en el calendario del equipo creativo fue el diseño de los niveles dos y tres, además de la creación de la música que acompañaría cada nivel para enriquecer la experiencia de juego. Mientras tanto, el equipo de desarrollo depuró algunos errores adicionales y quedó a la espera de las nuevas mecánicas que se podrían incorporar en los siguientes niveles en desarrollo.

Capítulo 5 Conclusiones y trabajo futuro

El desarrollo de un videojuego en 2D desde una perspectiva de ingeniería de software y proceso creativo nos permite entender mejor los retos y beneficios de aplicar metodologías de desarrollo estructurado en un campo que a menudo depende de la innovación y la creatividad.

A través de este trabajo se exploró y aplico principios de metodologías que combinadas ofrecen un enfoque robusto y flexible, siendo esto necesario para afrontar las demandas técnicas y creativas de un videojuego moderno.

Uno de los principales aprendizajes fue la de adaptar una metodología ágil modificada para este proyecto, el uso de iteraciones cortas y ciclos de feedback continuo resulto fundamentar para ajustar el prototipo en tiempo real, permitiendo correcciones tempranas y sobre todo la construcción de este mismo en la forma y tiempo estipulado. La agilidad del desarrollo también permitió enfrentar desafíos y cambios imprevistos sin comprometer gran avance del proyecto en general.

Otro aprendizaje de gran valor fue el de una planificación inicial clara y detallada, que no solo establece las bases de los objetivos, sino que también permite la adaptación con forme se detectaban nuevas necesidades o con forme surgían nuevas innovaciones durante el proceso. La flexibilidad que se pudo conseguir fue la clave que permitió a los equipos enfocarse en crear la experiencia de juego que se buscaba desde un inicio, priorizando elementos riticos que aportaban valor directo al prototipo.

Este proyecto también permitió aprender sobre el impacto de la colaboración activa y el trabajo en equipo, sobre todo al momento de hacer las reuniones al final de cada sprint donde se demostró que el compartir ideas y buscar resolver problemas en conjunto aumenta la calidad de código y diseño y fomenta una cultura de aprendizaje continuo. La retroalimentación entre compañeros y el apoyo en el desarrollo técnico, tanto en diseño como en implementación fue esencial para mejorar la comprensión de las soluciones, para así evitar errores, y en última instancia, crear un prototipo funcional.

El uso de la metodología ágil modificada permitió no solo cumplir con los requisitos técnicos, sino que además demostró ser una herramienta poderosa para mejorar la eficiencia del equipo y potenciar la creatividad. Al balancear la estructura de la ingeniería de software con la libertad que tiene el proceso creativo, se pudo obtener una metodología capaz de maximizar el valor del producto final.

Esto mismo genera una experiencia de usuario enriquecida y que cumple con las expectativas cambiantes del mercado de los videojuegos en 2D.

Glosario

A

Asset

Recurso utilizado dentro de un proyecto digital o videojuego. Puede incluir imágenes, sonidos, modelos, animaciones, scripts o cualquier elemento necesario para la construcción del producto final.

B

BanLab

Plataforma o entorno colaborativo orientado a la creación y edición musical, que permite a los usuarios producir, compartir y gestionar proyectos de audio.

C

Caching

Técnica que almacena temporalmente datos para acelerar accesos futuros y optimizar el rendimiento de sistemas, aplicaciones y servicios web.

Check

Verificación o condición empleada en programación para validar que un proceso, función o estado se encuentre en el estado correcto antes de continuar.

CPU (Central Processing Unit)

Unidad central de procesamiento. Componente principal de un sistema informático encargado de ejecutar instrucciones, coordinar tareas y controlar el flujo de operaciones.

D

DOM (Document Object Model)

Modelo de representación estructurada de documentos HTML o XML, que permite la manipulación programática de su contenido y estilo mediante lenguajes como JavaScript.

Domestika

Plataforma de cursos en línea enfocada en disciplinas creativas como diseño, ilustración, animación, fotografía y artes digitales.

F

FL Studio

Estación de trabajo de audio digital (DAW) que permite componer, grabar, editar y mezclar música utilizando instrumentos virtuales, sintetizadores y herramientas de secuenciación.

G

Gameplay

Conjunto de mecánicas, reglas, interacciones y experiencias que definen cómo el jugador se relaciona con el videojuego y cómo se desarrolla la acción.

GPU (Graphics Processing Unit)

Unidad de procesamiento gráfico especializada en el cálculo paralelo y en el renderizado de imágenes, fundamental en aplicaciones visuales y videojuegos.

H

Hitbox

Área invisible asociada a un objeto o personaje en un videojuego que se utiliza para detectar colisiones, impactos o interacciones.

HTTP (Hypertext Transfer Protocol)

Protocolo de comunicación utilizado para el intercambio de información entre clientes y servidores en la web.

L

Latencia

Tiempo que tarda una señal o dato en desplazarse desde su origen hasta su destino. En videojuegos influye en la velocidad de respuesta y la sincronización en entornos en línea.

M

Mecánicas de juego

Reglas, acciones, sistemas y comportamientos que determinan cómo interactúa el jugador con el mundo del videojuego y qué posibilidades de acción existen.

Modularización

Proceso de dividir un sistema en partes o módulos independientes para facilitar su desarrollo, mantenimiento, reutilización y escalabilidad.

P

Píxel Art

Estilo visual caracterizado por gráficos formados por píxeles perceptibles, común en videojuegos retro y producciones inspiradas en la estética de 8 y 16 bits.

Plugins

Complementos o extensiones que añaden nuevas funciones a un software o motor sin modificar su estructura principal.

Profilers

Herramientas que permiten medir y analizar el rendimiento de un sistema o videojuego, proporcionando información sobre uso de CPU, GPU, memoria, tiempos de carga y ejecución.

R

Rigidbody

Componente utilizado en motores de videojuegos, como Unity, que permite que un objeto se comporte según las leyes de la física, incluyendo gravedad, colisiones y fuerzas.

S

Slime

En videojuegos, criatura enemiga común representada como una masa gelatinosa; también puede referirse a su estilo de animación o comportamiento dentro del juego.

Sprite

Imagen o conjunto de imágenes 2D empleadas para representar personajes, objetos, decoraciones o animaciones dentro de un videojuego.

Sprint

Periodo de tiempo corto utilizado en metodologías ágiles, especialmente en Scrum, en el que se planifican, desarrollan y entregan funcionalidades específicas del proyecto.

U

Unity

Motor de videojuegos multiplataforma que permite desarrollar experiencias 2D, 3D y de realidad interactiva mediante herramientas visuales y scripting con C#.

Referencias

“Creación de videojuegos de plataformas con Unity”. Un curso online de 3D y Animación de Steve Durán | Domestika. (2024, September 22). Domestika. https://www.domestika.org/es/courses/910-creacion-de-videojuegos-de-plataformas-con-unity/course

Arbonés, Á. (2018, July 20). *Del 2D al 3D: cómo el videojuego pasó de la imaginación a sólo valorar los números*. Canino. https://www.caninomag.es/del-2d-al-3d-o-como-el-videojuego-paso-de-la-imaginacion-a-solo-valorar-los-numeros/

Atlassian. (n.d.). *¿Qué es ágil? | Atlassian*. https://www.atlassian.com/es/agile

Campus, C. (2024, April 22). *Proceso creativo: en qué consiste y características*. Universidad Europea Creative Campus. https://creativecampus.universidadeuropea.com/blog/fases-proceso-creativo/

Caurin, J. (2019, April 26). *Videojuegos 2D | ¿Qué significa Videojuegos 2D?* Geekno. https://www.geekno.com/glosario/videojuegos-2d

"Ciclo de vida del software: todo lo que necesitas saber." (“Ciclo de vida del software: todo lo que necesitas saber - Intelequia”) (n.d.). *Intelequia*. https://intelequia.com/es/blog/post/ciclo-de-vida-del-software-todo-lo-que-necesitas-saber

Cortizo, J. C. (2022, October 20). *"Si hay una industria que no es un juego, esa es la de los Videojuegos*." (“Si hay una industria que no es un juego, esa es la de los Videojuegos ...”) Product Hackers. https://producthackers.com/es/blog/industria-videojuegos

Definición de Optimizar software o hardware (informática). (2023, July 9). Alegsa.com.ar. https://www.alegsa.com.ar/Dic/optimizar.php#gsc.tab=0

Delgado, M. (2022, September 8). *Videojuegos y creatividad - The Good Gamer*. The Good Gamer. https://thegoodgamer.es/videojuegos-y-creatividad

Diaz, D. (2024, May 17). *11 herramientas de ingeniería de software que debe conocer como programador*. Geekflare Spain. https://geekflare.com/es/software-engineering-tools/

Estos son los 10 mejores videojuegos 2D de la historia | Tokio. (n.d.). Tokio School. https://www.tokioschool.com/noticias/videojuegos-2d/

Estrada, J., Salazar, A., & Wightman, P. (2017, June 2). *Diseño e Implementación de un videojuego bajo la propuesta de La Leyenda del Guerrero Malibú*. (“Diseño e

Implementación de un videojuego bajo la propuesta de La ...") Universidad Del Norte. https://manglar.uninorte.edu.co/handle/10584/7298

Evad. (2020, October 12). *EL PROCESO CREATIVO DE UN VIDEOJUEGO*. EVAD Escuela Superior De Videojuegos Y Arte Digital. https://evadformacion.com/proceso-creativo-videojuego/

Feed. (n.d.). BandLab. https://www.bandlab.com/feed

Global, R. A., & Global, R. A. (2018, May 22). *La arquitectura y su importancia en los videojuegos | Arcus Global*. Arcus Global. https://www.arcus-global.com/wp/la-arquitectura-y-su-importancia-en-los-videojuegos/

Guevara, L. A., & Peña, F. F. (2024). "Desarrollo de un Algoritmo Procedimental de Generación de Estructuras para su Implementación en el Desarrollo de un Videojuego 2D." ("Desarrollo de un algoritmo procedimental de generación de estructuras ...") *Ciencia Latina Revista Científica Multidisciplinar*, *7*(6), 7736–7748. https://doi.org/10.37811/cl_rcm.v7i6.9305

Guzman, H. C. (n.d.). *Las 7 fases más importantes en el desarrollo de juegos | Escuela de Videojuegos | Hektor Profe*. https://docs.hektorprofe.net/escueladevideojuegos/articulos/fases-del-desarrollo-de-videojuegos/

Historia de los videojuegos. (n.d.). https://www.fib.upc.edu/retro-informatica/historia/videojocs.html

https://dialnet.unirioja.es/servlet/tesis?codigo=312707

Laoyan, S. (2024, February 6). Qué es la metodología waterfall y cuándo utilizarla [2024] • Asana. *Asana*. https://asana.com/es/resources/waterfall-project-management-methodology

Ledmon. (2022, February 3). Herramientas del proceso creativo para optimizar resultados | Ledmon. *Ledmon*. https://ledmon.com/6-herramientas-del-proceso-creativo-para-optimizar-resultados/

Los 6 niveles de optimización de software | Mejora tu código | Go4IT Solutions. (n.d.). https://www.go4it.solutions/es/blog/los-6-niveles-de-optimizacion-de-software-mejora-tu-codigo

Miguel, P. D. S. (2019, August 6). *Las plataformas 2D - 33bits*. 33bits. https://portal.33bits.net/los-plataformas-bidimensionales/

Porto, J. P., & Gardey, A. (2023, May 17). *Optimización - Qué es, definición y concepto*. Definición.de. https://definicion.de/optimizacion/

Ros, I. (2021, November 18). *"Cinco problemas a los que nos han acostumbrado los desarrolladores de videojuegos, y que deberíamos superar cuanto antes*."

(“Cinco problemas a los que nos han acostumbrado los desarrolladores de ...”) MuyComputer. https://www.muycomputer.com/2021/07/01/errores-problemas-videojuegos/

Sanz, C. V., & Suemay, M. Y. C. (2020). *"Metodologías de diseño y desarrollo para la creación de juegos serios digitales*." (“Descripción: Metodologías de diseño y desarrollo para la creación de ...”) https://sedici.unlp.edu.ar/handle/10915/111123

Sara, C. G. (2022). *Diseño creativo en la preproducción y difusión de videojuegos independientes: estudio de caso de Hollow Knight (Team Cherry)*. Dialnet. https://dialnet.unirioja.es/servlet/tesis?codigo=312707

Stsepanets, A., Stsepanets, A., & Stsepanets, A. (2024, March 26). *Modelo cascada, qué es y cuándo conviene usarlo*. Gantt Chart GanttPRO Blog. https://blog.ganttpro.com/es/metodologia-de-cascada/

Unity Essentials Pathway - Unity Learn. (n.d.). Unity Learn. https://learn.unity.com/pathway/unity-essentials

VivesBarcelona, J. (2020, February 21). La parte creativa de los videojuegos. *La Vanguardia*. https://www.lavanguardia.com/vida/junior-report/20200219/473657943447/parte-creativa-videojuegos.html

Wilsom, N. B., & Armando, R. P. D. (2020, May 28). *"Desarrollar una herramienta software para atraer potenciales aspirantes al programa de ingeniería de sistemas y computación*." (“Desarrollar una herramienta software para atraer potenciales aspirantes ...”) Universidad Del Norte. https://manglar.uninorte.edu.co/handle/10584/8864

Zendesk. (2023, February 15). *¿Qué es la metodología ágil? ¿Para qué sirve?* https://www.zendesk.com.mx/blog/metodologia-agil-que-es/

Félix, E. B. (2001). *Videojuegos y educación*. Universidad De Salamanca. https://gredos.usal.es/handle/10366/56438

Pereira, A. M. M. (2014). El proceso productivo del videojuego: fases de producción/The production process of the game: production phases. *Historia y comunicación social*, *19*, 791-805.