UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE QUINTANA ROO

DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

DISEÑO Y CREACIÓN DE UN VIDEOJUEGO EN DOS DIMENSIONES USANDO EL MOTOR DE VIDEOJUEGOS GODOT

TESIS
PARA OBTENER EL GRADO DE
INGENIERO EN REDES

PRESENTA
MARCO ALEJANDRO LÓPEZ CALDERÓN

DIRECTOR
M.T.I. VLADIMIR VENIAMIN CABAÑAS VICTORIA

ASESORES
DR. JAVIER VÁZQUEZ CASTILLO
M.T.I. MELISSA BLANQUETO ESTRADA
M.S.I. LAURA YÉSICA DÁVALOS CASTILLA
DR. JAIME SILVERIO ORTEGÓN AGUILAR

CHETUMAL QUINTANA ROO, MÉXICO, ABRIL DE 2024

UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE QUINTANA ROO

DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

TRABAJO DE TESIS TITULADO

"DISEÑO Y CREACIÓN DE UN VIDEOJUEGO EN DOS DIMENSIONES USANDO EL MOTOR DE VIDEOJUEGOS GODOT"

ELABORADO POR

MARCO ALEJANDRO LÓPEZ CALDERÓN

BAJO SUPERVISIÓN DEL COMITÉ DEL PROGRAMA DE LICENCIATURA Y APROBADO COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE:

INGENIERO EN REDES

COMITÉ DE TESIS

DIRECTOR: M.T.I. VLADIMIR VENIAMIN CABAÑAS VICTORIA

ASESOR: DR. JAVIER VAZQUEZ CASTILLO

ASESORA: M.T.I. MELISSA BLANQUETO ESTRADA

ASESORA SUPLENTE: MS.I. LAURA YESICA DÁVALOS CASTILLA

ASESOR SUPLENTE: DR. JAIME SILVERIO ORTEGÓN AGUILAR

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO
ÁREA DE TITULACIÓN

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO
DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

CHETUMAL QUINTANA ROO, MÉXICO, ABRIL DE 2024

Resumen

El capítulo 1 “Introducción” se explica la justificación de la tesis, en la cual se exponen todas aquellas razones que me han motivado a llevar adelante el trabajo aquí presente, además de ver los múltiples objetivos que tiene los cuales son objetivos generales y específicos, a su vez también abarca el alcance que se desea obtener con esta tesis.

En el capítulo 2 podemos encontrar, todo lo relacionado a videojuegos, desde el ¿Qué es un videojuego?, ¿Qué tipo de videojuegos existen? Hasta la historia de los videojuegos abarcando desde sus orígenes, hasta la época actual de los videojuegos con sus nuevas tecnologías, además de esto también podemos encontrar información sobre el motor que se utiliza a lo largo de esta tesis llamado “Godot”, en este capítulo vemos información importante sobre Godot, a su vez tenemos información sobre su interfaz y su uso.

El desarrollo del videojuego se encuentra en el capítulo 3, el cual describe las metodologías que se usan para crear videojuegos indies, así como toda la información necesaria paso por paso para crear el videojuego 2D, desde cómo crear un nuevo proyecto en el motor llamado Godot, todo lo relacionado a las físicas y gravedades que se utilizaran, como mejorar el videojuego en cuestión usando diversas técnicas de programación y la codificación del videojuego, toda la programación descrita y explicada paso a paso que se utiliza para la creación del videojuego en dos dimensiones.

Finalmente, el capítulo 4 muestra las conclusiones de esta tesis, a su vez que los resultados finales del videojuego 2D, a su vez también se nos presenta todo el lenguaje de programación final que se usó para la realización del juego en dos dimensiones, y la forma de exportarlo tanto a páginas web como en un archivo ejecutable.

Agradecimientos

A mi tutor

Dr. Javier Vázquez Castillo. Sin usted y sus constantes ayudas, su paciencia y constancia este trabajo no lo hubiese logrado tan fácil. Sus consejos fueron siempre útiles cuando no salían de mi pensamiento las ideas para escribir, lo que hoy he logrado. Usted formó parte importante de esta historia con sus aportes profesionales, su gran forma de ayudarme y en general por todo lo que lo caracteriza. Muchas gracias por sus múltiples palabras de aliento, cuando más las necesite; por estar allí cuando pensaba rendirme siempre animándome a continuar. Gracias por todo lo que hizo por mi".

A mis docentes

"Sus palabras fueron sabias, sus conocimientos rigurosos y precisos, a ustedes mis maestros y amigos queridos, les debo mis conocimientos. Donde quiera que vaya, los llevaré conmigo en mí vida profesional. Sus conocimientos son como una semilla que ya germinó en mí. Gracias por su paciencia, por compartir sus conocimientos de manera profesional e invaluable, por su dedicación perseverancia y tolerancia".

A mi familia

"Ustedes han sido siempre el motor que impulsa mis sueños y esperanzas, quienes estuvieron siempre a mi lado en los días y noches más difíciles durante mis horas de estudio. Siempre han sido mis mejores guías de vida. Hoy cuando concluyo mis estudios, les dedico a ustedes este logro amada familia, como una meta más conquistada. Me siento muy afortunado de tener esta familia que siempre creyó en mí, sin ustedes no sería lo que soy.

Gracias por ser quienes son y por creer en mí".

A mis compañeros:

"Mis amigos y compañeros de viaje hoy culminan esta maravillosa aventura y no puedo dejar de recordar cuántas tardes y horas de trabajo nos juntamos a lo largo de nuestra formación cómo nos unimos, como un grupo fuerte y siempre nos apoyamos entre todos.

Dedicatoria

Dedico esta tesis a mis amigos del grupo de Ingeniería en Redes quienes fueron un gran apoyo emocional durante el tiempo en que escribía esta tesis, y en general durante toda la carrera, siento que nos unimos como una familia, sin todos ustedes chicos, toda la carrera, esta tesis y en general toda mi aventura en la universidad, no hubiera sido la misma.

A mi mamá Liliana Calderón Ávila quien me apoyo todo el tiempo, quien siempre a pesar de todo lo que pasa siempre cree en mí, mamá tu siempre serás mi ejemplo de superación y te dedico esta tesis y este éxito cumplido.

A mi hermana Paulina López Calderón quien me apoyó y alentó para continuar, cuando parecía que me iba a rendir, quien a pesar de todo siempre está conmigo.

A mis abuelos: mi abuela Candelaria y mi abuelo Dimas, quienes son parte importante de la familia, pues su conocimiento y sabiduría me han ayudado demasiado a cumplir estas metas.

A mis maestros, quienes nunca desistieron al enseñarme, aún sin importar que muchas veces no entendiera algo de la clase, cometiera algún error o simplemente tuvieran que explicarme algo más de 2 veces, a ellos que continuaron depositando su esperanza en mí.

A todos los que me apoyaron para escribir y concluir esta tesis.

Para ellos es esta dedicatoria de tesis, por su apoyo incondicional.

Índice

Tabla de contenido

Tabla de Ilustraciones

Capítulo 1 Introducción

Este proyecto de tesis tiene como objetivo desarrollar un videojuego en dos dimensiones, describiendo los componentes de un motor de juegos e implementando un proceso de creación y diseño.

Se pretende hacer una difusión del proceso desde el punto de vista tecnológico, en el cual se abordarán los temas del uso estricto de matemáticas, algoritmos, arte visual y sobre todo un gran lenguaje de programación que puede ser utilizado para videojuegos en dos o en tres dimensiones.

Con este trabajo quiero resaltar las oportunidades de desarrollo profesional para los egresados del Programa Educativo de Ingeniería en Redes, el cual es un tema muy poco tratado. Los tópicos del área de programación brindan los conocimientos necesarios en cuanto a la creación de videojuegos y esto puede generar opciones profesionales tales como las de un programador especializado.

Otro de los objetivos es identificar la herramienta que considero idónea para creación de videojuegos y que cómo se observará más adelante, es óptima en cuanto a procesamiento, facilidad de manejo, utilización del entorno y que tiene su propio lenguaje de programación fácil y dinámico en cuanto a su uso, además de ser totalmente gratuito.

1.1 Justificación

Los videojuegos son parte de nuestro día a día, además de ser reconocidos dentro del ámbito cultural, tecnológico e incluso educativo, el existir desde hace bastante tiempo, y mejorar cada vez con cada generación, aumentando su nivel de desarrollo y diseño y adaptándose siempre a las nuevas tecnologías, es por esto que es de vital importancia, el tener los conocimientos para diseñar y crear los videojuegos, sin importar si son en 3 dimensiones o 2 dimensiones, ya que el adquirir estos conocimientos, puede abrir muchas puertas en el ámbito profesional de un ingeniero.

1.2 Objetivo General

Diseñar y crear un videojuego en dos dimensiones utilizando el motor de creación de videojuegos Godot.

1.3 Objetivos Específicos

1. Identificar los principales componentes del motor GODOT.
2. Describir el proceso de creación de proyectos para videojuegos.
3. Diseño de escenarios, personajes y acciones.
4. Codificar los componentes del videojuego.
5. Publicar el proyecto en repositorio público.

1.4 Alcance

El alcance de este proyecto de tesis será la creación de un videojuego corto en dos dimensiones utilizando todas las herramientas que nos brinda el motor de creación de videojuegos Godot.

Capítulo 2 Marco Conceptual

En este segundo capítulo, se describen los conceptos principales que rodean el mundo de los videojuegos con el objetivo de establecer un lenguaje común que permita avanzar con el resto del trabajo.

Se darán definiciones de lo que es un juego, qué es un videojuego, cuáles son las características de éste, se hará un breve repaso sobre la historia de los videojuegos y su industria; así como las disciplinas que abarca que el desarrollo de los videojuegos

2.1 ¿Qué es un juego?

Existen diversas definiciones provenientes de disciplinas distintas acerca del concepto del juego. Se destacarán a continuación a los autores más relevantes en este campo:

Según el autor Roger Caillois, define al juego como una actividad voluntaria e improductiva que es gobernada por reglas que definen a un universo ficticio separado de la realidad del jugador (Kent, 2016), Esto quiere decir que para él, el juego es una actividad que uno como individuo realiza de forma voluntaria y que tiene ciertas reglas que definen el universo del juego, estas reglas pueden ser interpretadas como el conjunto de instrucciones a seguir para poder jugar.

Para Johan Huizingas, historiador y fisiólogo (Kent, 2016), un juego es una actividad propia de la vida animal, que escapa de la vida ordinaria del jugador y que lo sumerge por propia voluntad a una competencia o una representación de algo sin tener por objetivo de satisfacer una necesidad básica.

Sin embargo, para el autor Jesper Juuls (Kent, 2016), un teórico influyente del campo del estudio de los videojuegos, un juego es un sistema basado en reglas con resultado variable y cuantificable, donde el jugador ejerce esfuerzo por influir en el resultado y se siente emocionalmente ligado al mismo. Esta actividad es opcional y los resultados de esta pueden ser negociables.

Para finalizar, si consideramos al autor Chris Crawford (Kent, 2016), un diseñador y teórico de los videojuegos describe al juego en cuatro partes estas son:

1. Representación. La representación se refiere a dos aspectos del juego: uno objetivo, que es el sistema de reglas cerrado por el cual está formado, y otro un aspecto subjetivo, que implica que el juego es una versión simplificada y subjetiva de una realidad emocional.
2. Interacción. La actividad de jugar es un proceso activo, y es lo que lo diferencia de otros medios de entretenimiento.
3. Conflicto. Surge naturalmente de la interacción del jugador y es una característica intrínseca del juego.
4. Seguridad. El jugador está a salvo de las consecuencias del juego y de los conflictos que este presenta. Son una manera segura de experimentar la "realidad".

Ahora que sabemos desde el punto de vista de diversos autores en el ámbito de los videojuegos qué es un "juego", podemos definir lo que es un "videojuego".

2.2 Los videojuegos

Según Chris Crawford, los videojuegos representan un subconjunto de la realidad, en donde se coloca al jugador en una situación de conflicto definido por las reglas del juego e interactúa a través de un dispositivo electrónico. (Belli & López, 2008)

De acuerdo con Crawford, es un subconjunto de la realidad, entonces podemos definir a un videojuego como un mundo propio donde el programador es el creador del mundo y el jugador es la persona que habita ese mundo; donde tiene ciertos conflictos que podemos interpretar como los peligros o retos del juego, ya sean estos los enemigos que debemos superar, los rompecabezas que debemos de armar, los mundos que debemos de pasar, entre otros desafíos del videojuego. Además se define por las reglas del juego, que pueden ser interpretadas como todo el lenguaje de programación que hace funcionar al juego, desde los códigos que mueven a los enemigos; al propio jugador, los escenarios, el menú del jugador, por último se encuentra la interacción a través de un

dispositivo electrónico, sea este una consola, una computadora o un dispositivo móvil. (Lacasa, 2011)

2.3 Juegos indie

Los videojuegos independientes son creados por personas o pequeños grupos sin apoyo financiero de distribuidores. Estos juegos se centran sobre todo en la innovación y tienen como base una distribución digital. Aunque no existe una descripción exacta, estos videojuegos comparten una serie de rasgos en común: son desarrollados por personas o compañías pequeñas de manera independiente. Por lo general, este tipo es más pequeño al resto, debido a la falta económica en comparación con las grandes marcas, ya que están a falta de respaldo económico debido a que cuentan con pequeños presupuestos y por lo tanto necesitan una distribución a través de internet. Al ser juegos independientes, los desarrolladores y los creativos no tienen límites creativos o intereses por controlar y no requieren una aprobación editorial como sucede con los juegos tradicionales. Al tener una limitación económica, los gráficos también suelen ser limitados por lo que necesitan basarse en una innovación del juego, aunque independiente no significa que tenga que basar el juego en una innovación. La industria de los videojuegos indie tiene, actualmente, un creciente interés y una gran popularidad. Esta industria registró un aumento pronunciado en la segunda mitad de la década de los 2000, sobre todo gracias a la creciente expansión de internet a los hogares, de la cual se aprovechó para distribuir estos juegos por la red, con plataformas de juegos como Xbox Live Arcade, Steam u OnLive. Aunque estos portales de distribución han sido muy criticados por cobrar una gran parte de los ingresos del juego. Por ejemplo, en 2008 un desarrollador podría ganar un 17% del precio de venta del juego y en torno al 85% si era vendido en formato digital, lo cual ha dado paso a proyectos de distribución propios a través de webs asociadas. La viabilidad en el juego Indie. María Simón Tudanca 15 También la capacidad de los desarrolladores a tener acceso a herramientas como Adobe Flash o Microsoft XNA y paquetes de software como Game Maker o GameSalad, hacen que se tenga al alcance lo necesario para poder avanzar. Tres grandes juegos indies que han demostrado la capacidad de esta nueva sección a triunfar son Braid, World of Goo y Minecraft. Sin ir más lejos, este último juego, Minecraft, lanzado

en mayo de 2009, llegó a obtener unas ganancias superiores a 24 millones de euros en 2011. Un juego que hoy en día sigue creciendo sin problemas. (Simón Tundaca, 2014)

FIGURA 1 JUEGOS INDIE

2.4 Resumen de la historia de los videojuegos.

A continuación, se verá en este apartado un breve resumen sobre la historia de los videojuegos y como han ido evolucionando a lo largo del tiempo.

2.4.1 Primeros experimentos.

Los primeros videojuegos fueron creados en universidades con fines de investigación o por visionarios experimentando con dispositivos electrónicos, y no fue hasta 1971 que llegaron al público con el juego "Computer Space". Algunos historiadores de videojuegos como D.S. Cohen no consideran a estos como los primeros videojuegos (Kent, 2016), sin embargo, constituyen un claro antecedente. Entre estos primeros experimentos se encuentran:

1. En 1947 con "Cathode Ray Tube Amusement Device" desarrollado por Thomas T. Goldsmith Jr. y Estle Ray Mann, donde se controlaba por medio de un circuito analógico el brillo y la

posición de un punto con un tubo de rayos catódicos intentando crear un simulador de misiles. Los objetivos debían ser superpuestos dado que no era posible dibujar gráficos todavía. (Marqués)

2. En 1952 Alexander S. Douglas creó para su tesis de doctorado en la Universidad de Cambridge una versión del Ta-te-ti programada para el EDSAC (La primera computadora con posibilidad de almacenar programas). Así nació OXO, un juego programado para el ESDAC (y, lógicamente, sólo funcionaba para esta computadora) que utilizaba como control un dial telefónico y, como salida, una pantalla de osciloscopio. El código del juego era bastante sencillo (se introducía en la máquina codificándolo en una tarjeta perforada) y que la computadora, jugaba al juego. Una vez que el juego era cargado, la pantalla mostraba un mensaje en el que se indicaba al usuario que eligiese quién haría el primer movimiento, si el humano o la computadora, una vez lanzado el juego, el usuario indicaba la posición de la casilla que iba a ocupar mediante el número del dial telefónico (del 1 al 9) y, acto seguido, la computadora respondía colocando su ficha en el tablero. (Kent, 2016)

3. En 1958 William Higinbotham creó "Tennis for Two" como un experimento en el laboratorio de Brookhaven. El objetivo del experimento era poder demostrar que era posible el control interactivo en pantalla. El 18 de octubre de 1958, Tennis for Two causó sensación entre los visitantes del Laboratorio Nacional de Brookhaven durante el día de puertas abiertas, donde cientos de personas hicieron cola para jugar en este partido de tenis virtual. Al año siguiente, en 1959, Higinbotham mejoró el juego y, además de utilizar un osciloscopio con una pantalla más grande, fue capaz de añadir distintos niveles de gravedad que complicaban más el juego. Al terminar esta segunda sesión de puertas abiertas, Higinbotham desmanteló la máquina de Tennis for Two para utilizar los componentes en otros proyectos del Laboratorio, por lo que la máquina original se perdió. Además, dado que William Higinbotham no patentó la idea (y si lo hubiese hecho, la patente hubiese pertenecido al gobierno estadounidense), no llegó a hacerse rico, es más, Tennis for Two fue olvidado durante mucho tiempo. (Kent, 2016)

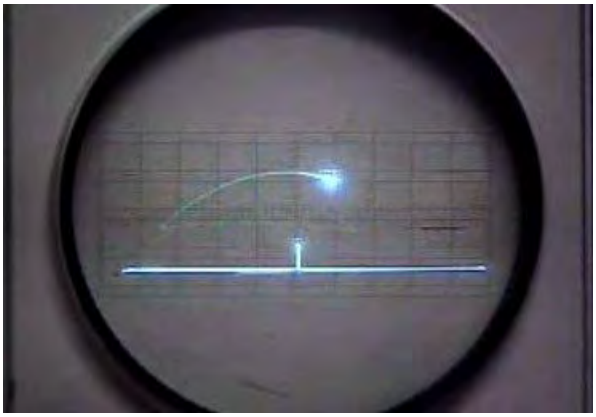

FIGURA 2: TENNIS FOR TWO

2.4.2 Primeros Videojuegos y el nacimiento de la industria.

Computer Space fue un videojuego implementado íntegramente en hardware, al igual que Tennis for Two, eso sí, en esta ocasión era un sistema totalmente digital que se basaba en tres placas conectadas a un bus común y un monitor CRT de General Electric que mostraba la interfaz gráfica de la máquina. ¿Y en qué consistía el juego? El jugador controlaba un cohete que debía disparar a un par de platillos voladores (que eran controlados por el sistema) en un tiempo acotado (100 segundos). La dinámica era bastante sencilla, había que destruir los platillos voladores, y evitar ser destruidos por ellos, dentro del tiempo que duraba el juego. El jugador podía ser derribado infinitas veces, sin embargo, el marcador del jugador debía que ser mayor que el de los platillos voladores para ganar un bonus de tiempo de 90 segundos extras. (Kent, 2016)

La máquina se fabricó en serie, pero tuvo un éxito irregular (no se llegaron a vender más de 1.000 unidades). Por un lado, había colas para jugar en ellas en las universidades en las que se instalaron, sin embargo, este arcade no era nada popular en bares y otros establecimientos porque no era nada fácil jugar y, por supuesto, ganar. (Kent, 2016)

El juego "Table Tennis" o mejor conocido como Pong fue un videojuego de la primera generación de videoconsolas publicado por Atari, creado por Nolan Bushnell y lanzado el 29 de noviembre de 1972. Pong está basado en el deporte de tenis de mesa o ping pong desarrollado como videojuego

en 1958. ¡Aunque existieron con anterioridad otros videojuegos de las dos décadas anteriores como OXO (ejecutado en una computadora única en el mundo) y posterior a éste Spacewar! bajo la PDP-1 de DEC, eran, en su mayoría, proyectos experimentales. (Contreras, 2012)

Pong está considerado por muchos como el más importante de entre la primera generación de videojuegos modernos, debido a que fue el primero en comercializarse a nivel masivo y no ejecutarse en máquinas únicas. (Wolf, 2002)

FIGURA 3 PONG

2.4.3 "La época dorada".

Considerada el origen de la industria actual del videojuego, la Atari 2600 (que originalmente fue llamada VCS – Video Computer System) fue la primera en llegar de forma significativa a los televisores domésticos. Sus ventas empezaron lentamente en 1977 (EE. UU.), pero su popularidad explotó con conversiones de exitosas arcade como Space Invaders o Asteroids. Llegó a Europa (también a España, de la mano de Audelec, su distribuidora oficial, ubicada en Málaga) en 1978. Atari abrió una fábrica para darle soporte en Tipperary, en Irlanda, dónde Atari disfrutó de grandes ventajas fiscales, principalmente para fabricar los cartuchos. (Kent, 2016)

La consola popularizó el uso de los cartuchos intercambiables, innovó con los mandos (que se podían desenchufar de la consola y presentaban multitud de formas) y con la habilidad de seleccionar "variaciones" de cada uno de los juegos. Aunque un buen puñado de consolas técnicamente superiores se lanzarían en los 80 (como la Intellivision o la Colecovision), la Atari 2600 seguiría en producción de una forma u otra hasta mucho más tarde de las quiebras de muchos de sus competidores, hasta 1991. (Kent, 2016)

Para ponernos en situación, a principios de los 70, los videojuegos empezaron a tener éxito comercial por primera vez. Los americanos, especialmente, empezaron a disfrutar de recreativas como Pong, Tank y similares, que llenaban centros recreativos y bares. Fue en ese contexto en el

que se empezó a visualizar el posible mercado de las consolas de sobremesa. En 1975 Atari lanzó una versión de Pong para jugar en casa, diseño que pronto copiarían centenares de compañías en Estados Unidos. No obstante, aunque probablemente sean líneas paralelas, en este punto se suele obviar un hecho importante: Europa fue primero. Existen consolas de 1974 que reproducen versiones caseras de Pong, como la Videomaster Home TV Game. En todo caso, ese boom provocó la primera crisis de los videojuegos en EE. UU.; demasiadas consolas inundaron el mercado, llevando a una bajada de precios que dejó en la ruina a la mayoría de las fabricantes. Atari, no obstante, vislumbró la posibilidad de diseñar un sistema que pudiese cambiar de juegos con cartuchos. Si bien es cierto que la Channel F de Fairchild y la RCA Studio II (ambas únicamente en EE. UU.) se adelantaron, ninguna de ellas podría hacer sombra a la llegada de Atari y, sobre todo, a sus juegos exclusivos. (Kent, 2016)

Atari, que hasta el lanzamiento de la VCS había vivido en la montaña rusa financiera típica del grupo de yuppies que la formaban, no disponía de cash suficiente como para lanzar la consola. Nolan Bushnell finalmente aceptó una oferta de compra de Warner Communications, y vendió la compañía en 1976 por 28 millones de dólares, bajo la condición de acelerar el lanzamiento de la 2600. Ese movimiento hizo a Bushnell millonario de facto, pero sería el punto de partida que llevaría a la desaparición de Atari en los 90. Los años siguientes serían de oro para la compañía, pero las decisiones que seguirían al crash del 83 les condenarían para siempre. (Iturriaga Barco & Medel Marchena, 2017)

La Atari 2600 pasó por múltiples iteraciones: desde la “Heavy Sixer”, de seis botones y fabricada en California, hasta la Atari 2600 Jr., de la que hablaremos en su momento. La segunda iteración (como la primera), tenía un frontal de falsa madera, pero con 4 botones en vez de las 6 iniciales. El toque de los 70 que le da ese frontal es lo que le da su encanto. La versión de 4 botones es la que más se vendió en España. Más tarde se sustituyó por un frontal de plástico negro; esa es la llamada “Darth Vader” – la VCS coincide en el tiempo con las películas de Star Wars. (Iturriaga Barco & Medel Marchena, 2017)

También se crearon otras consolas hogareñas para competir con la Atari. Éstas fueron la Coleco Vision (1982) y la Intellivision (1979) de Mattel. Coleco Vision licenció el juego "Donkey Kong" a Nintendo, el cual fue su mayor éxito. Considerando la competencia Atari lanzó una versión mejorada de su consola, la Atari 5200, lanzada en 1982 (Iturriaga Barco & Medel Marchena, 2017)

Hacia el año 1982 la mayoría de los géneros de videojuegos fueron creados para distintas versiones de consolas y arcades. En cuanto a los juegos para las computadoras personales de la época como la Comodore 64, Amiga, ZX Spectrum, Apple II, al inicio la distribución de los juegos estuvo a cargo de aficionados que clonaban los juegos más exitosos de los arcades como "Donkey Kong", "Pac Man", "Frogger" y "Space invaders". Revistas especializadas comenzaron a formarse y tomaron el papel de distribuir estos códigos fuentes. Con el tiempo surgieron las empresas distribuidoras que introdujeron los primeros juegos comerciales para PC con títulos originales como: "Zork", "Roberta Williams' Mystery House para Apple II y "King's Quest" de la empresa Sierra. (Kent, 2016) En cuanto a consolas de videojuegos portátiles existían los denominados "Handled electronic Games". Estos consisten en versiones reducidas de consolas que pueden ser llevadas en el bolsillo. Los "Handled electronic Games" de esta época consistían en juegos muy pequeños, cuyas consolas eran diseñadas específicamente para cada juego y con el display generalmente hecho a medida. El más famoso de ellos es la colección "Game and Watch" de Nintendo con juegos como "Donkey Kong", "Egg" y "Ballon Figth". (Kent, 2016)

FIGURA 4 CONSOLA ATARI

2.4.5 El crash de la de industria.

El plan de Atari era ganar dinero a través de la venta de los cartuchos de juegos, y no a partir de la venta de la consola misma. No obstante, la decadencia en calidad de sus juegos frente a la competencia y a las nuevas empresas distribuidoras quienes producían juegos de una calidad superior, terminaron por llevar al fracaso esta estrategia.

El mercado de videojuegos había sido inundado con muchos juegos de poca calidad, adaptaciones y secuelas de otros juegos viejos pero creados con presupuestos reducidos. Los consumidores empezaron a ser decepcionados por los productos. Se desató un efecto en cadena que afectó a toda la industria, Atari comenzó a perder dinero, Mattel decidió retirarse de la industria de juegos para siempre terminando con su consola Intellivision. En total la industria de los videojuegos paso a tener una ganancia de 2.9 miles de millones de dólares en 1983, es decir un 35 por ciento menos que el año anterior. En total la industria había perdido cerca de 1.5 miles de millones de dólares (Kent, 2016)

ILUSTRACIÓN 5 CONSOLA INTELLIVISION

2.4.6 El resurgimiento de la industria y la era de los 8 bits.

El crash terminó cuando el éxito de Japón, la Nintendo Enterteiment System (NES), llego para revitalizar a la industria. La NES fue una consola con un procesador de 8 bits con sistema de almacenamiento de cartuchos. (Altice, 2017)

Alcanzó un éxito inmediato vendiendo cerca de 60 millones de unidades con juegos populares como "Super Mario Bross", "The legend of Zelda" y "Metroid". Nintendo había aprendido del "crash" que le antecedió, y había desarrollado una política estricta para el licenciamiento de los juegos y de esta

manera se aseguraba que la consola no sería invadida con juegos de calidad cuestionable. Los juegos sin licencia de Nintendo no serían tolerados por la consola. La licencia de Nintendo limitaba la cantidad de juegos que una empresa de terceros podía crear por año y Desarrollo de Videojuegos. además, los mismos juegos no podían ser hechos para otras consolas hasta pasado un periodo de 2 años. (Altice, 2017)

En cuanto a los videojuegos para PC se estaba dando a conocer lo que se llama el "Shareware". El shareware consiste en una licencia de software que entrega una versión reducida del programa en forma gratuita, e invita a los usuarios a comprar la versión completa. (Ángel Gutiérrez, 2007)

Los juegos online también comenzaron a ganar un poco de terreno gracias a Internet, aunque se seguía manteniendo como un terreno poco explorado. Estos juegos eran generalmente juegos basados en texto (Kent, 2016)

FIGURA 6 LICENCIA SHAREWARE

2.4.7 El período 1990-1995 y las consolas de 16 bits.

En 1989 se lanzó el Game Boy, un sistema portátil de juego con una pantalla LCD con juegos almacenados en cartuchos Su popularidad fue muy grande, y logró vender un total de 30.3 Millones de unidades. Sus títulos más conocidos son "Tetris" (Nintendo negoció la Licencia con la Unión Soviética para poder distribuirlo legalmente con su consola), y "Pokemon Blue" y "Pokemon Red" dos juegos que impactaron en la cultura japonesa, llegando a producirse series animadas, películas, juegos de cartas y una infinidad de productos como muñecos y ropa. (Custodio, 2020)

La primera consola de 16 bits fue de la empresa NEC, llamada TurboGrafx 16 cuya sucesora fue la primera en utilizar CD-ROMs. Sega, con una mejor estrategia comercial liberada en 1989 comenzó

a ganar popularidad. La competencia con la NES de Nintendo fue creciendo hasta superar a esta última. Es por eso por lo que en 1991 Nintendo lanzo la Super Nintendo Enterteiment System (SNES) con un procesador de 16 bits que seguía utilizando cartuchos. (Donovan, 2010)

Sega CD fue lanzada en 1992 (Mega drive en Japón lanzada en 1991) con una gran capacidad de almacenamiento para los juegos, 2.000 veces superior a la de un cartucho gracias al uso de los CD-ROMs. (Paris & Paris, 2016)

En esa misma época surgió la máquina de arcade SNK. Una máquina de 21 bits que superaba ampliamente en poder de procesamiento y gráficos a las consolas de la época. Su versión de consola no tardó en salir, y fue llamada Neo Geo. Si bien los Desarrollo de Videojuegos. (Hennessey, 2017)

juegos lanzados para esta consola fueron muy populares, el alto precio impidió que se convirtiera en un éxito comercial (Kent, 2016)

Aun así, con la llegada de las consolas de 16 bits la diferencia con la calidad de los juegos con las máquinas de arcade fue disminuyendo y estás ultimas comenzaron a decaer en ventas, generando cada vez menos ganancias. El último gran éxito del arcade fue el Street Figther I un juego de pelea para dos jugadores. (Hennessey, 2017)

En PC la capacidad de procesamiento había crecido notablemente, en el mercado se contaba con los procesadores Intel 80386, 80486 y Motorola 68000, 68030. Al mismo tiempo se vislumbró el nacimiento del 3D, y las capacidades multimedia con las tarjetas de sonido y los CD-Roms. (McMichael, 2007)

La distribución de los juegos se realizó bajo la licencia de shareware, el cual se volvió usual gracias a que los juegos eran desarrollados por pequeños equipos como Apogee (ahora 3D Realms) Epic Megagames (Epic games) e ID software. Dado que el tamaño de los juegos creció, se fue convirtiendo impráctico utilizar disquetes para distribuir los juegos y se comenzaron a utilizar los CD-Roms. Las revistas especializadas distribuían CDs llenas de versiones Shareware de los juegos. (McMichael, 2007)

Surgieron juegos como "Wolfestein 3D" y "Doom" los cuales fueron los primeros juegos del género en primera persona modernos (FPS por sus iniciales en inglés). Los FPS son un género muy popular de videojuegos en la actualidad. (McMichael, 2007)

En 1992 se liberó el juego "Dune II" que fue el primer juego de estrategia moderno (RTS por las siglas en inglés de Real Time Strategy). Su formato marco tendencia en los RTS que le siguieron como "Comand and Conquer", "Warcraft" de Blizzard y "Age of Empires" de Microsoft Games. (McMichael, 2007)

En 1990 Maxis, comenzó a publicar su línea exitosa de juegos: "Sim". Comenzando con "SimCity", "Sim Earth" y "SimCity 2000". Muchos juegos fueron lanzados con la opción de juego online, juegos como "Age of Empires", "Startcraft", y "Ultima Online" (Kent, 2016)

FIGURA 7 COMPAÑÍA MAXIS

2.4.8 El periodo 1995-2000 y las consolas de 32 y 64 bits.

En 1996 se lanzó la 3dfx Voodo Chipset, la primera tarjeta aceleradora de gráficos para computadoras personales. Este procesador permitía mejores gráficos con resoluciones mayores gracias a un procesador y memorias dedicadas para el procesamiento de gráficos en la tarjeta aceleradora. Juegos como "Quake" fueron los primeros en hacer uso de estas capacidades mostrando gráficos totalmente en 3D. (Simone & López Raventós, 2008)

La popularidad de los juegos Online creció, juegos como "EverQuest" permitían jugar con un número limitado de jugadores al mismo tiempo bajo el nuevo concepto de mundos persistentes para juegos multiplayers gráficos. (Belli & López, 2008)

Más adelante el elemento social se hizo notar aún más con los juegos online. Los juegos "Ultima Online", "Asheron's Call", "Star Wars Galaxies", y "World of Warcraft" alcanzaban los millones de jugadores alrededor del mundo. El desarrollo de los plugin de Flash y de Java para los exploradores web abrió paso a juegos simples ejecutados directamente en el explorador. Estos juegos no requieren de instalación y se pueden jugar sin representar riesgos de seguridad a los jugadores ya que son ejecutados en máquinas virtuales. (Belli & López, 2008)

Surgió una nueva modalidad en los juegos y esta fue la posibilidad de hacerles modificaciones. El juego más exitoso con esta característica fue "Half Life". Uno de sus Mods (modificaciones) más populares fue el "Counter Strike" un juego multiplayer de gran popularidad. (Kent, 2016)

Este período se caracterizó por la popularización de los juegos 3D y el soporte de CDROMs para los juegos, y esto se dio tanto en computadores personales, como en consolas y arcades.

Nintendo y Sony comenzaron negociaciones para desarrollar un accesorio para la SNES que le permitiera leer CD-ROMs. Sin embargo, Sony consideró que la SNES era tecnología obsoleta, y diseñó una consola nueva por completo llamada PlayStation. Esta consola tendría una ranura para los cartuchos de SNES y podrían ser jugados en ella los juegos diseñados para SNES CD. Esto le daría control completo a Sony de los títulos que podrían ser desarrollados con los CD-ROMs, al contrario de lo buscaba Nintendo con su consola. Al comprender esto el presidente de Nintendo, Hirosi Yamauchi, dio fin a las negociaciones. (Iturriaga Barco & Medel Marchena, 2017)

Nintendo volvió a intentarlo con la empresa Philips, pero aconteció una historia similar. Philips desarrolló su propia consola, la CDTVi, la cual terminó en un fracaso comercial. Finalmente, Nintendo desistió de agregar una lectora de CD-ROMs a la consola SNES al ver el poco éxito conseguido por Sega CD y terminó por desarrollar la Nintendo 64, una consola de 64 bits con poco éxito ya que seguía utilizando cartuchos, cuyos precios resultaban muchísimo más caros que los de los almacenados en CD-ROMs. No obstante juegos como "Super Mario 64" marcaron un nuevo rumbo en el género de juegos de plataformas en el mundo 3D, así también como otros juegos recordados de la consola (Kent, 2016)

Por su lado la Playstation de Sony lanzada en 1994 tuvo un gran éxito comercial. Mundialmente se distribuyeron más 100 Millones de unidades (Kent, 2016) convirtiéndose en la primera consola en vender tal cantidad. La Playstation contaba con un procesador 32 bits y 2 MB de memoria RAM de propósito general, 1MB de memoria RAM para video, y 512 KB de RAM para sonido. Utilizaba soporte de CD para sus juegos y una memoria especial para poder almacenar las partidas llamada Playstation Memory card. En PC, se desarrollaron múltiples placas aceleradoras cuyos mayores exponentes fueron las series GeForce de Nvidia, y las placas aceleradoras de la serie Raedon de ATI (Kent, 2016) Cada nuevo juego en PC requería de un poder de procesamiento mayor, lo que empujó a una relación entre fabricantes de hardware y las empresas desarrolladoras de juegos.

Game Boy Color de Nintendo, en el marco de las consolas portátiles, fue lanzada 1998 con 118 millones de unidades vendidas y destacándose nuevamente juegos de la serie Pokemon. Tenía un procesador de 8 bits, y utilizaba soporte de cartuchos para sus juegos. (Custodio, 2020)

Neo Geo Pocket fue lanzada en ese mismo año, repitiendo el error de sucesora en consola de sobremesa. Si bien, la Neo Geo Poket, era superior técnicamente a su contraparte de Nintendo, su precio era muy elevado y solo alcanzó a vender 2 millones de unidades. (Custodio, 2020)

FIGURA 8 NEO GEO POCKET

2.4.9 El periodo 2000-2005 y la sexta Generación de consolas.

En el año 2000, Sony libera su segundo gran éxito, la Playstation 2 (PS2), la cual alcanzó un récord de ventas llegando a vender 100 millones de unidades solamente a 5 años de su lanzamiento y un total de 150 millones de unidades hasta la fecha Pese a ser una consola de más de 10 años de antigüedad y de competir con su propia sucesora, se continúa fabricando. (Kent, 2016)

La PS2 fue la primera consola en utilizar soporte de DVD para sus juegos, y también era retro compatible con la Playstation (Se pueden jugar juegos de su predecesora). Los juegos más vendidos para esta consola son “God of War”,” Gran Turismo”, “SingStar”. “Ratchet & Clank” y “Grand Theft Auto: San Andreas” (Kent, 2016) La elección de Sony por el soporte de DVD estuvo fundada por dos razones: los DVD podían almacenar seis veces más información que los CDs, y además le permitiría a la consola reproducir películas, algo que ninguna otra consola podía hacer . La PS2 contaba con un procesador de 128 bits de 294 MHz, un procesador gráfico de 147 MHz, y con una memoria RAM de 32 MB.

Nintendo lanzó su primera consola no basada en cartuchos denominada Nintendo Game Cube. Los juegos de la Game Cube utilizaban un soporte de almacenamiento propietario que podía almacenar 4 o 5 veces la capacidad de un DVD. Sin embargo, Nintendo solamente logró alcanzar la venta de 21 millones de unidades.

En el 2001, Microsoft hace su aparición en el mercado de las consolas con la Xbox. La Xbox tenía un procesador Intel Pentium III de 733 MHz, un procesador gráfico de Nvidia llamado NV2 con 250 MHz de procesador y 64 MB de RAM, soporte para conexión a Internet y un disco de 8GB. Su mayor éxito fue “Halo 2” que vendió 8 millones de copias, mientras que la consola misma, solamente vendió 28 Millones de unidades (Kent, 2016)

Si bien la consola no fue del todo un éxito comercial, introdujo el concepto de juego online en consolas con el sistema por subscripción de Xbox Live. El Xbox live permitía a los jugadores competir y compartir jugadas con personas a través del mundo. Sony lanzó su propio sistema de juego online, pero a diferencia de Microsoft, el servicio era gratuito una vez comprado el accesorio que permitía a la PS2 la conexión a Internet. (Kent, 2016)

En PC los juegos multijugadores online se hicieron aún más populares. En general, la mayoría de los juegos incluía ahora un modo de juego multiplayer para jugar en línea. El juego “World of Warcraft” con millones de jugadores representa el 60% del mercado de subscripciones online de juegos.

En el 2003 el juego "Second Life" hace su aparición. Es esencialmente un mundo virtual persistente donde los jugadores pueden crear contenido y son los propietarios de los derechos de propiedad intelectual de estas creaciones. El juego cuenta con un sistema de economía propia basada en la moneda Linden dólar el cual puede ser utilizado para comprar tierras y bienes virtuales.

En el año 2000 Maxis lanza el juego "The Sims" y se convierte en un éxito instantáneo y uno de los juegos de computadora mejores vendidos de todos los tiempos (Kent, 2016) Otros juegos como "Doom 3" y "Half Life 2" hacen su aparición como secuelas de juegos anteriormente exitosos.

FIGURA 9 SECOND LIFE

2.4.10 2005-Actualidad: La séptima y octava generación de consolas.

En el 2005, Microsoft lanza la sucesora de la Xbox, la Xbox 360. Ésta es la primera consola de la séptima generación, con un procesador gráfico más potente y más espacio en disco. Dicho espacio en disco fue necesario ya que las funciones de su sistema online no solamente se limitan a conectar jugadores y partidas, sino también con el lanzamiento del Xbox Live Arcade (XBLA) le permite la compra de juegos de manera online. También fue la primera consola en poseer salida HDMI para ser utilizada en televisores de alta definición. Esto generó el descontento de algunos usuarios, ya que solamente las mejoras gráficas de la consola podían ser aprovechadas al máximo teniendo un televisor con esta característica. La Xbox 360 alcanza hasta la fecha un total de 53 millones de unidades vendidas desde su aparición.

En el 2006 Sony lanza su tercera consola de sobremesa, la Playstation 3 con características muy similares a la Xbox 360, y alcanzando hasta la fecha un total de 57 millones de unidades vendidas.

En ese mismo año Nintendo lanza su nueva consola, esta vez apuntando a una dirección distinta. El nombre de esta consola es Wii, que en ingles tiene la misma pronunciación que la palabra “We” (nosotros). La característica principal de la Wii no Desarrollo de Videojuegos. reside en su capacidad de procesamiento ni en el desempeño gráfico sino en su forma innovadora de control. Sus controles inalámbricos llamados Wii Remote interpretan la aceleración de los movimientos en tres dimensiones, permitiendo utilizar el control de diversas formas. La consola alcanza un total de 83 millones de unidades vendidas, superando a sus competidores.

El Wii Remote no fue el único cambio que introdujo Nintendo con la Wii, sino el hecho de apuntar a un público más casual y no tanto a los jugadores hardcore (Este tema se explicará con mayor detenimiento más adelante). (Kent, 2016)Esta fue la estrategia que le permitió a la Wii destacarse por sobre sus competidores, quienes no tardaron en imitarlo.

En el 2010 Microsoft lanza Kinectic, un accesorio para la Xbox 360 que incorpora tecnología de visión artificial y el procesamiento de imagen y sonido, utilizando las cámaras incluidas en el accesorio para convertir al jugador en el control del juego. (Kent, 2016)

Sony por su parte, lanza un hibrido entre la tecnología utilizada en el Wii Remote y el Kinectic de Microsoft llamado PS Move y también desarrolla un catálogo de juegos orientado a los jugadores casuales. Al igual que la Xbox 360, las consolas Wii y PlayStation 3 cuentan con su propio sistema online de ventas de juegos descargables. WiiWare Para Wii, Playstation Network para Playstation 3.

El mercado de consolas portátiles se ve caracterizado por las consolas de Nintendo y Sony. Al igual que con la Wii la consola portátil de Nintendo, la Nintendo DS (NDS), busca la innovación en los controles y se presenta como una consola portátil con doble pantalla, una de las cuales es táctil. Sony por su parte lanza la Playstation Portátil (PSP) superior en cuanto capacidad de procesamiento y almacenamiento de su competidora (Kent, 2016)

En cuanto a PC grandes secuelas vuelven a hacer su aparición. “Half Life 2: Episodio I”, “Half Life 2: Episodio II” y “StartCraft II” bajo las expectativas de millones de jugadores. El concepto de pre-

orden comienza a hacerse presente al mismo tiempo que se hace más usual la distribución digital. La pre-orden consiste en la compra anticipada del producto antes de que este sea lanzado obteniendo como beneficio para el comprador descuentos.

La plataforma de distribución Steam toma un papel importante en la distribución digital de los juegos en PC, y se convierte en la plataforma online de juegos más grande del mundo. Este servicio ofrece una gran cantidad de títulos tanto a usuarios de Mac como de PC, y contiene una base de usuarios activos de 30 millones. (Kent, 2016)

El mercado de juegos online, protagonizado por los juegos Flash, basan su rentabilidad en la publicidad. Según, Mochi Ads, una red de avisos publicitarios que conecta a desarrolladores independientes con anunciantes, un tercio de los usuarios de Internet visitan los sitios de juegos online (llamados portales) lo que representa una audiencia creciente de 237 millones de personas en el mundo entero. (Kent & Andres, 2004)

Los juegos casuales, tanto en consolas, como en PC y juegos online, toman importancia en el mercado como resultado de una búsqueda de expandir los videojuegos hacia nuevos horizontes. El público de los videojuegos se amplía, y en el caso de los jugadores casuales son caracterizados en cuando a edad, un 65% mayores de 35 años, y en cuanto a género un 29% son hombres y el otro 71% son mujeres. Por el contrario, los jugadores en general de videojuegos tienen una edad promedio de 33 años con un 62% de hombres y un 38% de mujeres. (Kent, 2016)

En febrero de 2011 Nintendo lanzó la consola portátil 3DS (La última consola lanzada hasta la fecha de la construcción de este trabajo). Entre sus características se encuentran: una placa WIFI para permitir conectar la consola a internet, acelerómetros para detectar los movimientos y la inclinación de la consola, y 2 cámaras (para lograr el efecto 3D con fotografías). La novedad de esta consola radica en la pantalla superior, que posee la cualidad de mostrar imágenes con efecto estereoscópico, también conocidas como imágenes 3D, sin la necesidad de utilizar lentes.

ILUSTRACIÓN 10 CONSOLAS DE ÚLTIMA GENERACIÓN.

2.5 GODOT

Godot Engine es un motor de videojuegos multiplataforma con múltiples características para crear juegos 2D y 3D desde una interfaz unificada. Él provee un conjunto exhaustivo de herramientas comunes para que los usuarios puedan enfocarse en crear juegos sin tener que reinventar la rueda. Juegos que pueden exportarse en un sólo clic a numerosas plataformas, incluyendo las principales plataformas de escritorio (Linux, macOS, Windows), móviles (Android, iOS) y basadas en la web (HTML5). (Bradfield, 2018)

Godot es completamente gratuito y de código abierto bajo la licencia permisiva del MIT. Sin condiciones, sin regalías, nada. Los juegos de los usuarios son suyos, hasta la última línea del código del motor. El desarrollo de Godot es totalmente independiente y dirigido por la comunidad, lo que permite a los usuarios ayudar a dar forma a su motor para que coincida con sus expectativas. Está respaldado por Software Freedom Conservancy sin fines de lucro. (Manzur, s.f.)

FIGURA 11 LOGO DE GODOT

2.5.1 ¿Qué puede hacer el motor?

Godot fue desarrollado inicialmente por un estudio de juegos argentino. Su desarrollo comenzó en 2001, y el motor se reescribió y mejoró enormemente desde su lanzamiento de código abierto en 2014. (Pacheco, 2022)

Algunos ejemplos de juegos creados con Godot incluyen Ex-Zodiac y Helms of Fury.

FIGURA 12 EJEMPLO DE JUEGO CREADO POR GODOT

En cuanto a las aplicaciones, el programa de dibujo de píxel art de código abierto Pixelorama es impulsado por Godot, y también lo es el creador de juegos de rol voxel RPG en la caja.

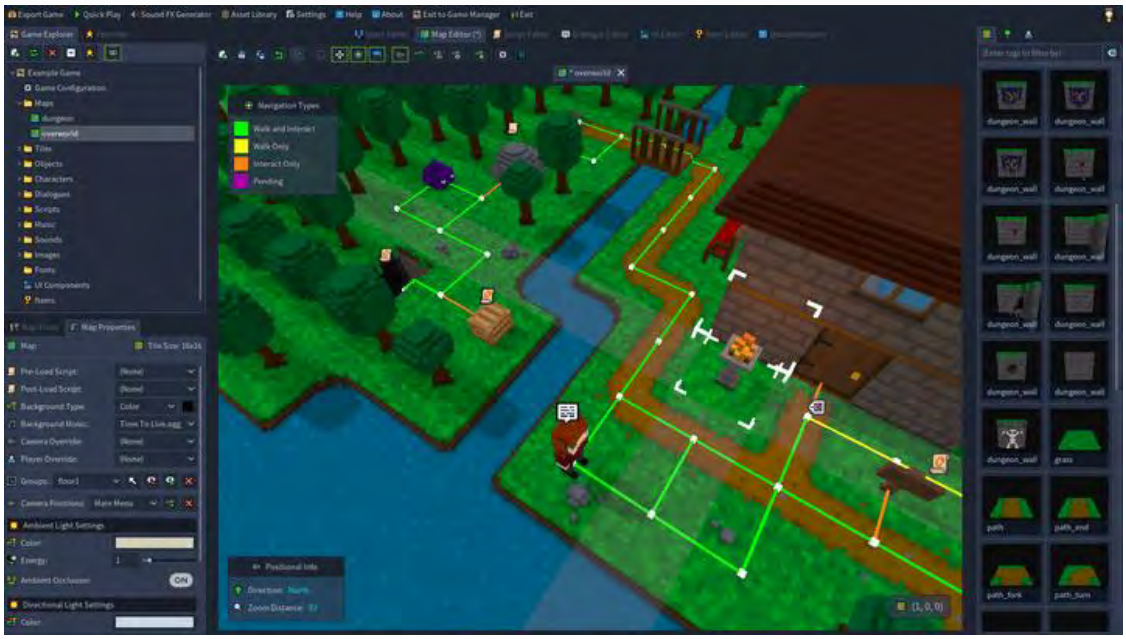

FIGURA 13 EJEMPLO DE PÍXEL ART

2.5.2 ¿Cómo funciona y cómo se ve?

Godot contiene un editor de juego completo con herramientas integradas para satisfacer las necesidades más comunes. Incluye un editor de código, de animaciones, de tilemaps, de shaders, un depurador, un perfilador y más. (Pacheco, 2022)

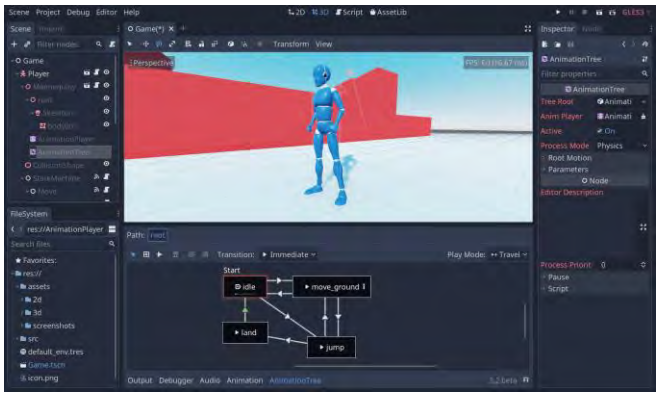

FIGURA 14 EJEMPLO DE EDITOR DE JUEGO

2.5.3 Lenguaje de programación

Godot Puede codificar sus juegos usando: GDScript un lenguaje específico de Godot y estrechamente integrado con una sintaxis ligera, o C #, que es popular en la industria de los juegos. Estos son los dos lenguajes de programación principales que Godot Admite. (Dhule, 2022)

Godot también soporta un lenguaje de programación visual basado en nodos, llamado VisualScript.

Con la tecnología GDNative, también puedes escribir lógica de juego o algoritmos de alta performance en C o C++ sin recompilar el motor. Puedes usar esta tecnología para integrar librerías de terceros y otros Kits de Desarrollo de Software (SDK) en el motor.

Por supuesto, también cuenta con la capacidad de añadir módulos directamente y características al motor, el cual es gratis y de código abierto.

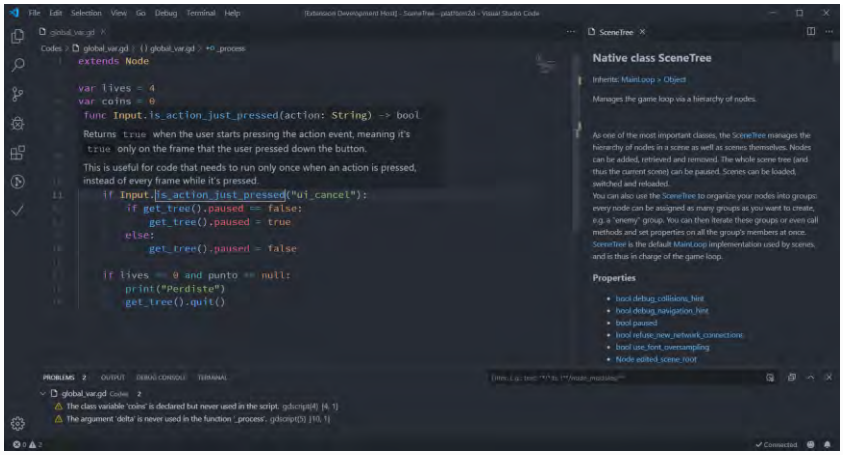

FIGURA 15 EJEMPLO LENGUAJE DE PROGRAMACIÓN

2.5.4 Escenas

En Godot, tu repartes el juego en escenas reutilizables. Una escena puede ser el personaje, un arma, el menú en las interfaces del usuario, una sola casa, un nivel entero, o cualquier cosa que tu puedas pensar. Las escenas de Godot son flexibles.

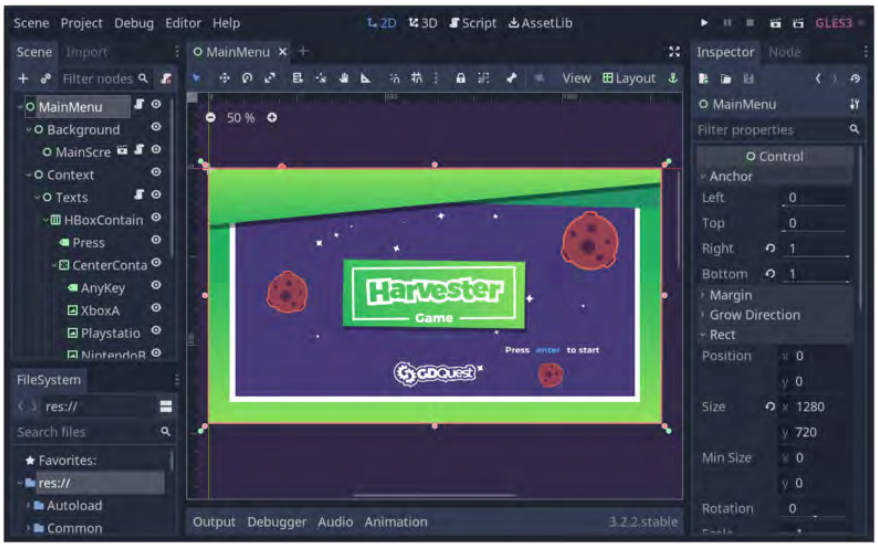

FIGURA 16 EJEMPLO DE ESCENAS

2.5.5 Nodos

Una escena es compuesta de uno o más nodos, Los nodos son pequeños bloques de construcción de tu juego que tu acomodas en árboles. Cuando se guarda un árbol de nodos en una escena, este se muestra como un solo nodo, el cual esta internamente escondido y estructurado en el editor.

Godot te provee de una extensa librería base de distintos tipos de nodos que tú puedes combinar y extender para construir unos más poderosos. 2D, 3D, o la interfaz del usuario.

FIGURA 17 EJEMPLO DE NODOS

2.5.6 El árbol de escenas

Todas las escenas de tu juego vienen juntas en el árbol de escenas, y las escenas son arboles de nodos, el árbol de escena también es un árbol de nodos. Pero hay una manera más fácil de pensar en tu juego en términos de escenas así ellos pueden representar los personajes, armas, puertas, o la interfaz del usuario.

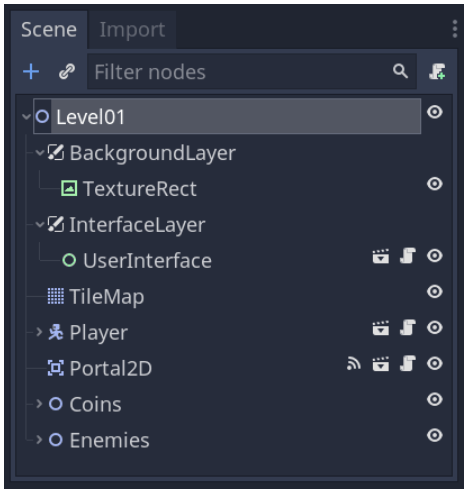

FIGURA 18 EJEMPLO ÁRBOL DE ESCENAS

2.5.7 Señales

Los nodos emiten señales cuando un evento ocurre. Esta característica te permite hacer que tus nodos se comuniquen sin dificultarte de esta forma no es necesario escribirlos en el código. Eso te da un montón de flexibilidad en como estructuras tu escena.

Por ejemplo, puedes usar señales para hacer que tus botones emitan una señal cuando son presionados. Tú también puedes conectar esa señal para correr el código a relación a este evento, como empezar el juego o abrir el menú.

Otra incorporación de las señales que puedes incluir es la señal de los objetos que colisionan, cuando un personaje o monstruo entran a un área, y mucho más. Tú también puedes definir nuevas señales justo a la medida de tu juego.

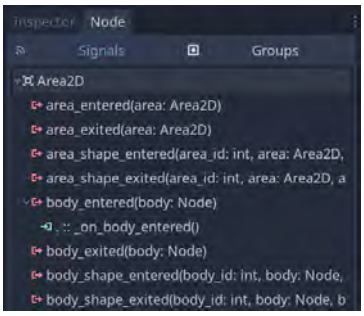

ILUSTRACIÓN 19 EJEMPLO SEÑALES

2.6 Interfaz Godot

2.6.1 Una primera mirada al editor de Godot

A continuación, veremos a detalle la interfaz de Godot.

2.6.2 El administrador de proyectos

Cuando inicies a Godot, la primera ventana que verás es la del Administrador de Proyectos. En la pestaña por defecto, "Proyectos", puedes administrar los proyectos existentes, importar o crear nuevos y más.

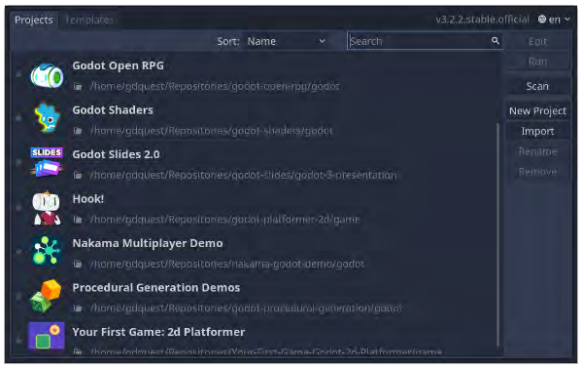

FIGURA 20 ADMINISTRADOR DE PROYECTOS

Encima de la ventana, hay otro botón llamado "Plantillas". Tú puedes buscar por proyectos demo en la librería de assets de código abierto, el cual incluye muchos proyectos desarrollados por la comunidad.

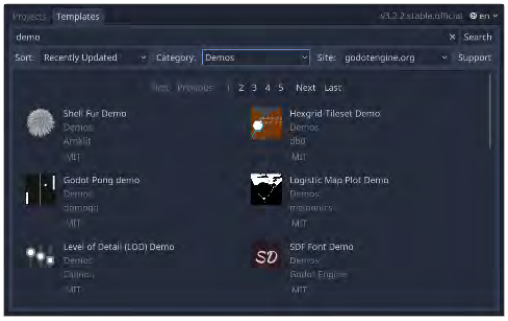

FIGURA 21 PLANTILLAS

Se puede cambiar el idioma del editor usando el menú desplegable a la derecha del motor en parte de arriba a la derecha de la ventana. Por defecto, este está en inglés (EN).

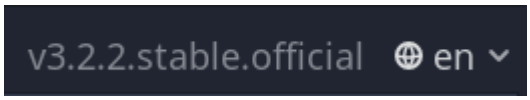

FIGURA 22 IDIOMA

2.6.3 Principales áreas del editor de Godot

Cuando abres un nuevo o un proyecto existente, la interfaz del editor aparece. Vamos a ver cuáles son sus principales áreas.

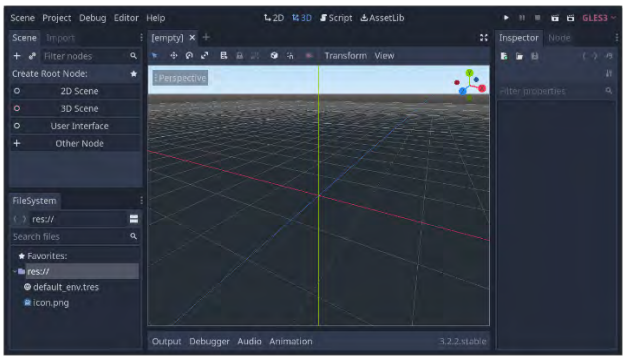

FIGURA 23 EDITOR DE GODOT

De forma predeterminada tendrás las funciones menú, **pantalla principal**, y el botón de probar juego por el borde arriba de la ventana.

FIGURA 24 MENÚ PRINCIPAL

En el centro está el viewport con su Barra de herramientas arriba, allí es donde se encuentran las herramientas para mover, escalar o bloquear objetos de la escena.

A los lados del viewport, se encuentran los paneles empotrables. En la parte inferior se encuentra el panel inferior.

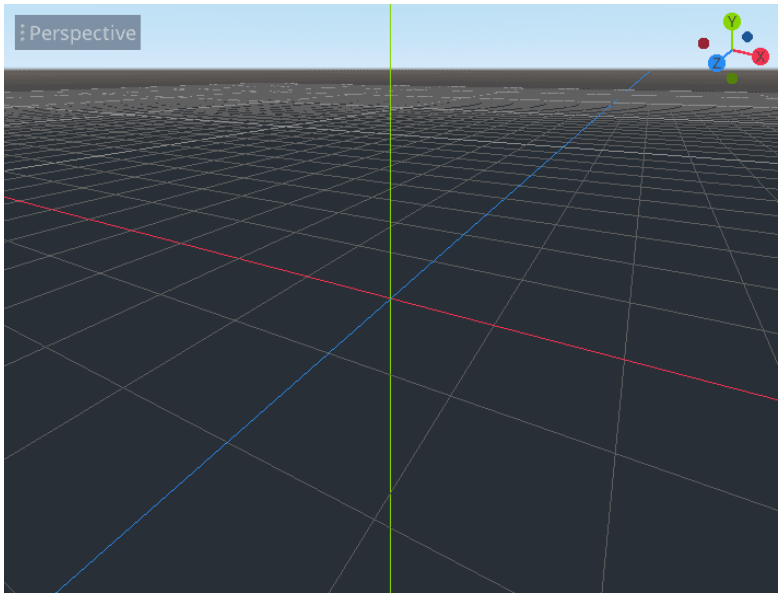

FIGURA 25 VIEWPORT

La barra de herramientas cambia con base en el contexto del nodo seleccionado. Aquí está la barra de herramientas 2D.

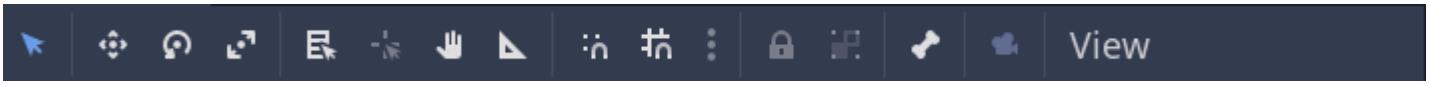

FIGURA 26 BARRA 2D

Esta es la barra de herramientas 3D.

FIGURA 27 BARRA 3D

Veamos los paneles empotrables. El panel del Sistema de archivos lista los archivos del proyecto, como pueden ser scripts, imágenes, muestras de audio y más.

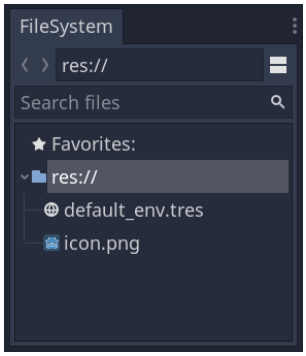

ILUSTRACIÓN 28 SISTEMA DE ARCHIVOS

El panel **Escena**, lista los nodos de la escena activa.

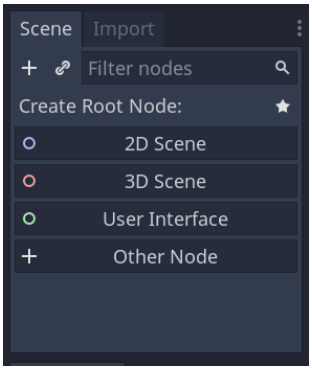

FIGURA 29 PANEL ESCENA

El **Inspector** permite editar las propiedades del nodo seleccionado.

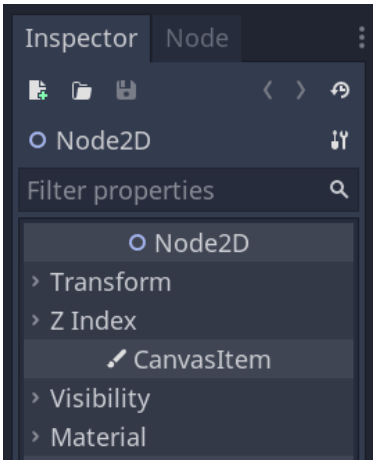

FIGURA 30 INSPECTOR

El **panel inferior** es el contenedor de la consola de depuración, el editor de animaciones, el mezclador de audio y más. Estos elementos pueden ocupar espacio valioso, es por eso por lo que se encuentran ocultos por defecto.

FIGURA 31 EL PANEL INFERIOR

Cuando haces clic en uno, se expande verticalmente. A continuación, puedes ver el editor de animación abierto.

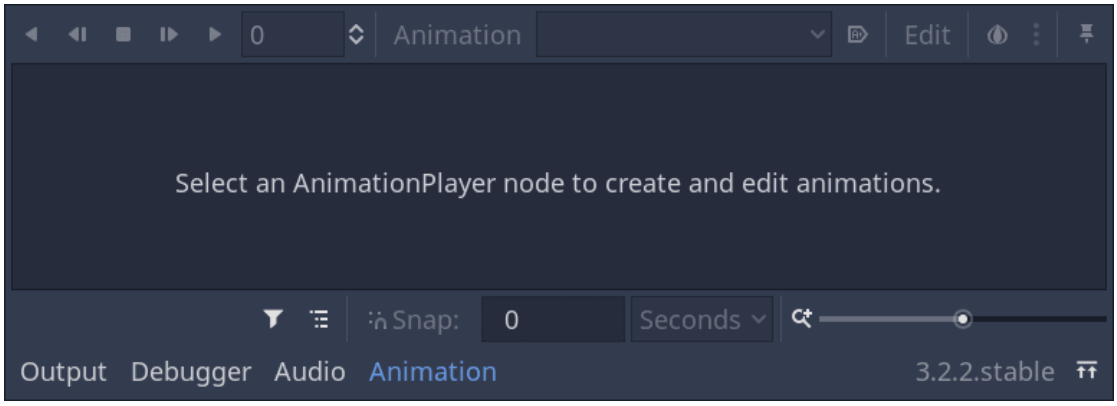

FIGURA 32 EDITOR DE ANIMACIÓN

2.6.4 Los cuatro espacios de trabajo

Hay cuatro botones de pantalla principal centrados en la parte superior del editor: 2D, 3D, Script y AssetLib. Se utiliza el **Espacio de trabajo 2D** para todo tipo de juegos que requieran 2 dimensiones. Además de los juegos en 2D, es allí donde se construyen las interfaces.

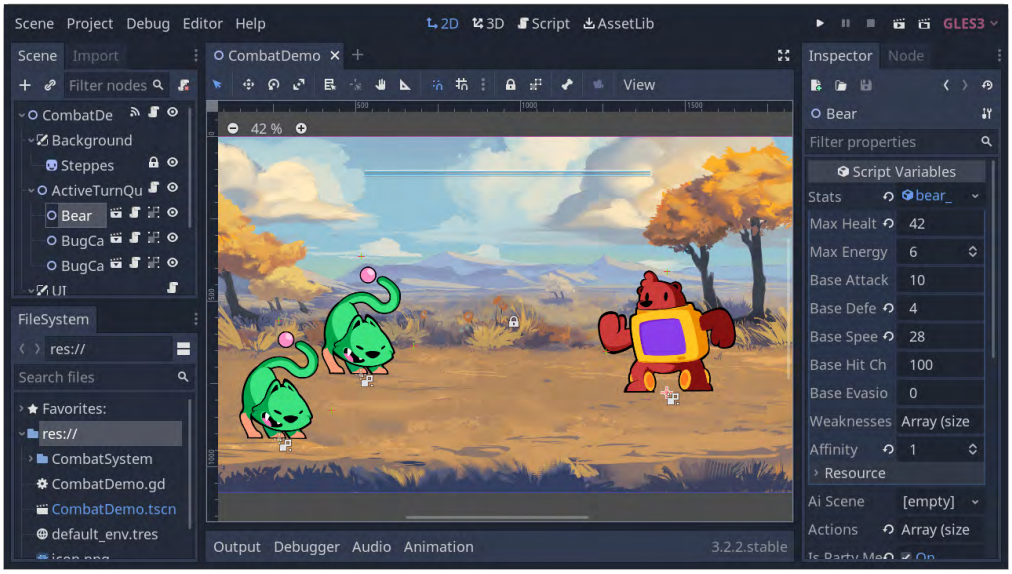

FIGURA 33 PANTALLA 2D

En la **pantalla 3D**, puede trabajar con mallas, luces y niveles de diseño para juegos 3D.

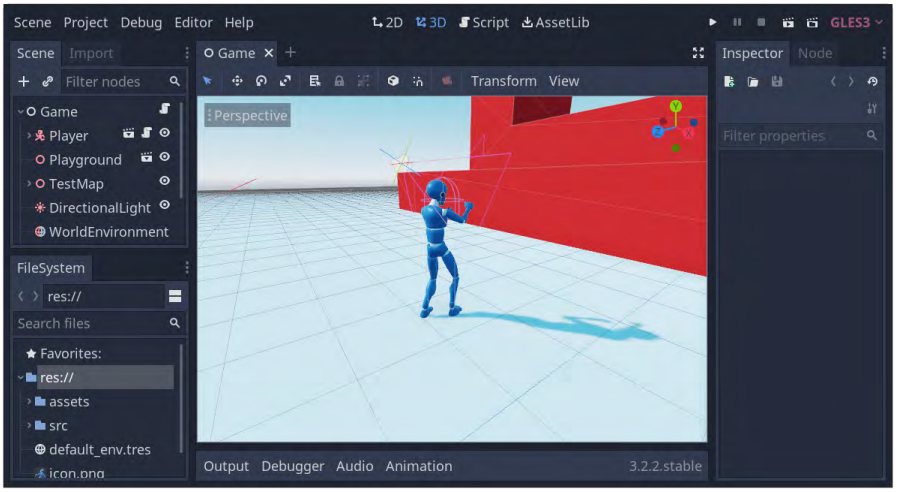

FIGURA 34 PANTALLA 3D

Nota el botón perspectiva ubicado en la barra de herramientas. Este abre una lista de opciones relacionadas con la vista 3D.

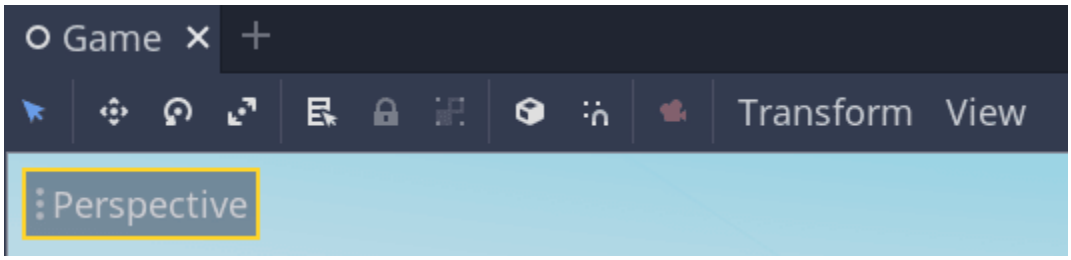

FIGURA 35 BOTÓN PERSPECTIVA

La "pantalla de scripts" es un editor completo con un depurador, autocompletado y referencia de código integrada.

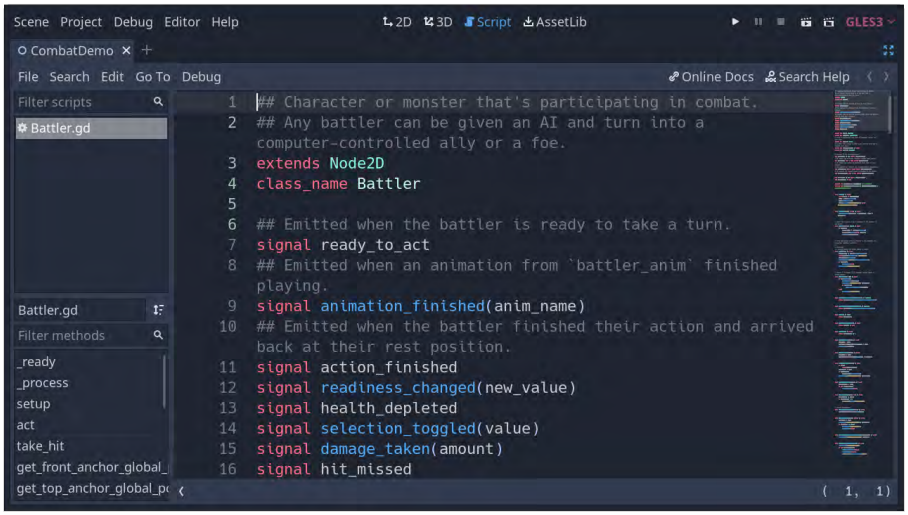

FIGURA 36 PANTALLA DE SCRIPTS

Finalmente, AssetLib es la librería de add-ons, scripts y recursos de código abierto y gratuitos para utilizar en los proyectos.

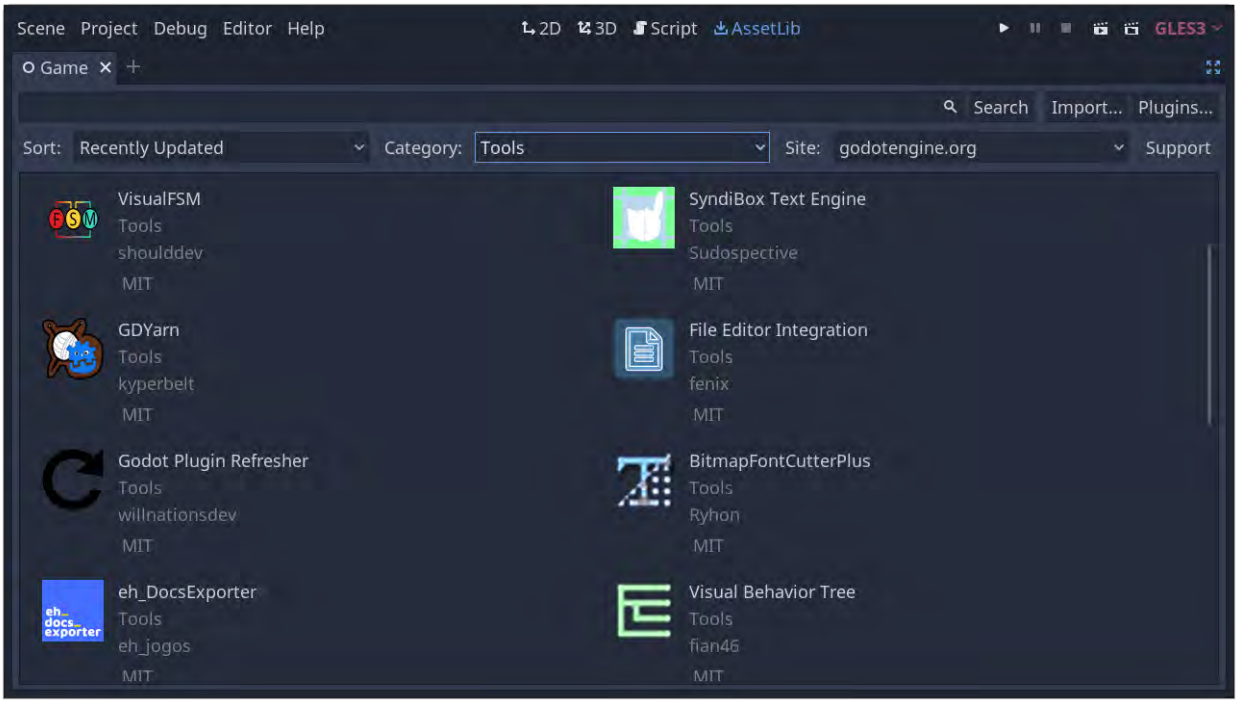

FIGURA 37 PANTALLA DE ASSETLIB

Capítulo 3 Desarrollo

3.1 Metodología del desarrollo del videojuego indie

Antes de empezar a desarrollar el videojuego 2D indie de plataformas en Godot primero veremos la metodología que se implementó en el momento de su desarrollo la metodología implementada fue la de las 6 etapas de diseño la cual consiste en lo siguiente:

Antes del lanzamiento, los videojuegos deben de pasar por varias etapas durante su desarrollo. No importa qué tipo de juego sea, o para cuál plataforma esté planeado, todos pasan por procesos similares. (Uniat, 2020)

Con una industria creciente y de alta demanda, los videojuegos se han convertido en un espacio de profesionalización para los artistas digitales. Es por ello por lo que los futuros diseñadores y programadores de videojuegos deben estar familiarizados con las etapas del desarrollo de juegos. (Uniat, 2020)

FIGURA 38 6 ETAPAS DEL DESARROLLO DE UN VIDEOJUEGO.

3.1.1 Planeación

Un videojuego nace de una idea. A partir de este momento inicia la etapa de planeación y los desarrolladores se enfrentan a todo tipo de preguntas: ¿Qué tipo de juego desarrollaremos? ¿Quiénes son los personajes? ¿Será en 2D o 3D? ¿Quién es nuestra audiencia?, en este caso el juego realizado es un juego de plataformas, cuyos personajes están realizados con el arte Sprite art en 2D, la audiencia en específico son jóvenes y adultos es decir un juego apropiado para todas las edades.

Sin duda, aterrizar y trabajar la idea de un videojuego implica un arduo trabajo pues es la antesala al concepto del juego.

Una vez establecido, todos las personas y equipos de los departamentos involucrados pueden empezar a probar el concepto. Aquí se toman las ideas que se han compartido y se analiza la viabilidad del proyecto para ser producido por el estudio. En este punto surgen preguntas como:

- ¿Cuál es el costo aproximado del proyecto?
- ¿Es necesario un nuevo motor de juego?
- ¿Cuántas personas necesitamos en el equipo?
- ¿Cuál es la fecha estimada de lanzamiento?

La prueba de concepto es vital para el éxito de un juego, pues pone en perspectiva cuáles son las posibilidades del proyecto y avanzar, con ellas en mente, el desarrollo.

3.1.2 Preproducción

En esta etapa, los escritores, diseñadores, artistas, ingenieros, leads de proyecto y otros departamentos trabajan y colaboran entre sí para definir la manera en que darán vida al videojuego.

Durante este proceso colaborativo se llevan a cabo juntas y discusiones entre varios equipos. Por ejemplo: escritores y leads trabajando la narrativa de la historia; ingenieros estableciendo qué pueden realizar con la tecnología disponible; diseñadores y artistas asegurándose que la paleta de colores, visuales y arte sigan la misma línea preestablecida; etc.

Es en este proceso donde los desarrolladores trabajan prototipos de personajes, ambientes, interfaces, etc., de la próxima obra artística que planean. De esta forma pueden saber con qué están trabajando y avanzar a la siguiente etapa.

3.1.3 Producción

Ciertamente la etapa de producción es la más compleja. La mayoría de los recursos, tiempo y trabajo que conlleva desarrollar un videojuego se los lleva la producción. Usualmente, en esta etapa:

- Los modelos de personajes se diseñan y renderizan hasta que luzcan como deben.
- El diseño de audio crea todos los sonidos del mundo del juego.
- Los diseñadores de nivel crean los ambientes de forma que sean atractivos para los jugadores.
- Se graba el doblaje con los actores en caso de que el juego lo requiera.
- Los programadores escriben los códigos necesarios para darle vida a los elementos del juego.

Es importante mencionar que esta etapa no tiene un tiempo definido. Dependiendo del tipo de producción, esta etapa puede tomar muchos años para completarse debido a muchos factores, como, por ejemplo: cambios de forma constante; rehacer partes ya terminadas que no satisfacen los objetivos del proyecto.

3.1.4 Periodo de prueba

Cada elemento, detalle y mecánica del juego debe someterse al control de calidad antes del lanzamiento. Para que un videojuego esté listo para su versión alpha, primero debe pasar por las manos de *testers* para que se identifiquen cuestiones como:

- ¿Hay áreas o niveles con muchos *bugs*?
- ¿Todo se está renderizando correctamente?
- ¿El personaje se queda atorado permanentemente en un lugar?
- ¿Los diálogos son atrapantes y realistas?

Generalmente hay distintos tipos de *testers*. Unos se enfocan en tratar de “romper” el juego; otros analizan la dificultad del juego para ver si es muy fácil o difícil. Este equipo tiene que asegurarse que el videojuego sea divertido y atractivo para generar ventas.

Después del periodo de prueba, el juego debería de estar listo para una versión beta.

3.1.5 Pre-lanzamiento

En esta etapa se suele elaborar una estrategia de mercadotecnia para vender el juego. Generalmente se usan teasers, gameplays, trailes entre otros métodos de mercadotecnia.

3.1.6 Lanzamiento

Durante esta etapa, los desarrolladores hacen una lista de *bugs* que tienen que erradicar, de mayor a menor gravedad. Comienzan con aquellos *bugs* que pueden "crashear" el juego, hasta llegar a aquellos que representan problemas menores.

Cuando el juego esté lo suficientemente limpio, es hora de lanzarlo al público.

3.1.7 Post-lanzamiento

Generalmente, en esta etapa temprana post-lanzamiento, los jugadores comienzan a identificar *bugs* u otros errores dentro del videojuego, por lo que los estudios tienen que solucionar estos problemas.

Además, se ofrecen actualizaciones de software y, en muchos casos, DLC, para que los jugadores tengan contenido extra dentro del mundo de este nuevo videojuego.

3.2 Creando un Nuevo Proyecto

Una vez definida la metodología y qué es Godot es hora de crear un nuevo proyecto y empezar a desarrollar el videojuego 2D, lo primero que tenemos que hacer es abrir Godot y crear un nuevo proyecto.

FIGURA 39 NUEVO PROYECTO

Para realizar esa acción primero debemos darle clic en donde dice crear un nuevo proyecto tal y como la imagen lo muestra, una vez realizada esa acción se nos abrirá la siguiente pantalla.

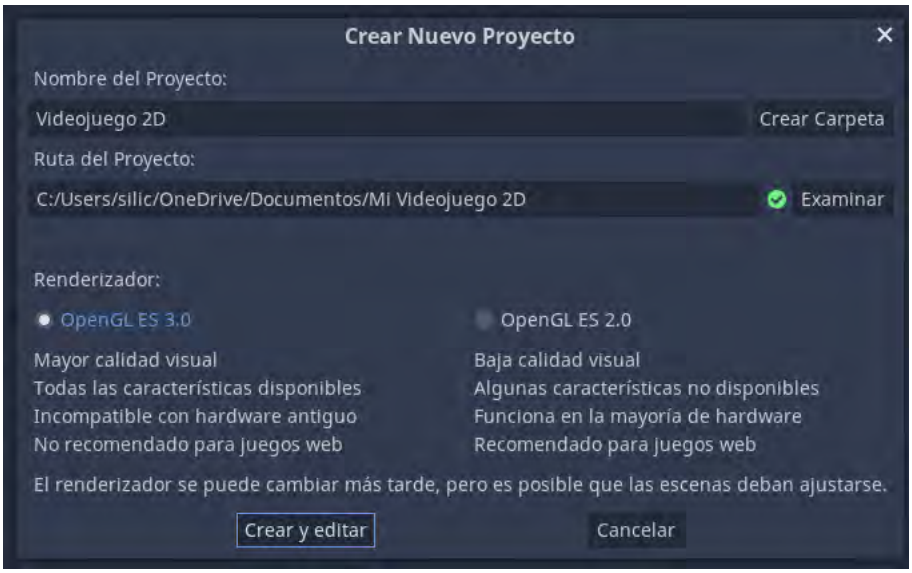

FIGURA 40 NOMBRE DEL PROYECTO

En la cual pondremos el nombre del proyecto y la ruta en la cual todos los avances realizados se guardarán, además debemos escoger el renderizado dependiendo de las capacidades de nuestra pc o laptop a la hora de utilizar Godot.

Ahora que ya conocemos la interfaz de Godot debemos configurarlo para crear un videojuego 2D, lo primero es poner la vista en 2D.

FIGURA 41 VISTA EN 2D

Ahora la vista que tendremos de Godot será la siguiente:

FIGURA 42 VISTA EN 2D

3.2.1 Añadir Sprites y Assets a Godot

Ya tenemos el nuevo proyecto creado y la vista está en 2D, es hora de añadir Sprites y Assets a nuestro videojuego 2D

3.2.2 ¿Qué es un Sprite?

La palabra Sprite viene del inglés y su traducción al español es duende. Un nombre significativo porque los videojuegos tienen mucho de magia y fantasía que se consigue, entre otras formas, con los Sprite. Este recurso gráfico de mapas de bits se popularizó a finales de los 70 y principios de los 80 de la mano de Jay Miner, un diseñador de chips. (Didactoons, 2021)

Los Sprites representan un personaje u objeto y se utilizan para que se pueda mover de forma independiente con respecto al fondo. Como los videojuegos, han ido evolucionando, en un principio eran transparentes y pequeños para que se adaptaran mejor al formato de las primeras consolas de entretenimiento, como Commodore 64 o Amiga. Con el paso del tiempo, fueron cambiando de forma y tamaño hasta la llegada de los videojuegos 3D. (Didactoons, 2021)

FIGURA 43 EJEMPLO DE SPRITE

3.2.3 ¿Qué es un Asset?

El término game assets hace referencia a los recursos que utiliza un videojuego y que forman parte de él en el momento de su creación. ¿Qué quiere decir esto exactamente? Que los assets de un videojuego son los sprites, las animaciones, los paquetes de sonido, en general todo lo relacionado al juego. (Bramble, 2023)

FIGURA 44 EJEMPLO DE ASSETS

Ahora que ya conocemos que es un Sprite y que es un Asset en general vamos a añadirlos al Godot para realizar esta acción haremos lo siguiente:

Primero una vez que ya tengamos nuestros Sprites y Assets disponibles, hay varias páginas web que tienen Sprites y Assests de libre uso, o puedes crear los tuyos propios usando diversos

programas como Aseprite (se especializa en Píxel Art y Animaciones). Photoshop o Krita para ilustraciones de alta resolución (no píxel art).

Entonces lo primero es crear 2 carpetas principales en la ruta de nuestro juego las cuales serán una carpeta llamada Assets donde pondremos todo lo relacionado al “Arte” del juego, Sprites, Música, Imágenes, fondos, entre otros recursos a utilizar y la segunda carpeta llamada Src que viene de la palabra Source o Fuente, en la cual pondremos toda la parte de programación del videojuego esto con el fin de tener una mayor organización a la hora de desarrollar el videojuego.

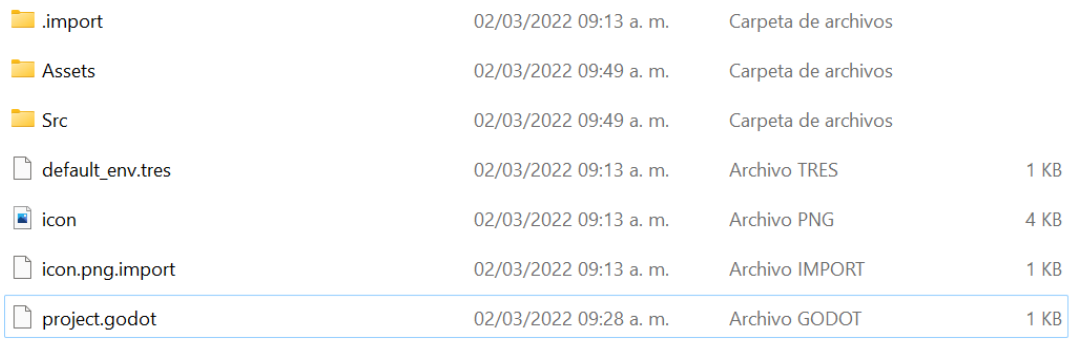

FIGURA 45 CARPETAS A UTILIZAR

Una vez creadas estas carpetas Godot si están en guardadas en la ruta principal Godot automáticamente descomprime y añade estas carpetas a la ruta principal de la herramienta, como puedes ver en la siguiente imagen.

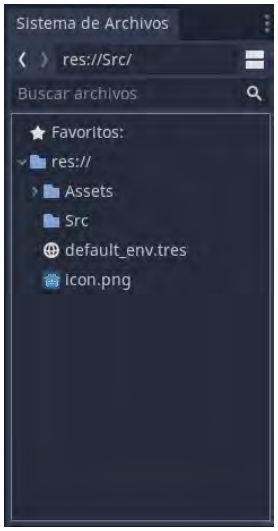

FIGURA 46 RUTA PRINCIPAL DE GODOT

3.2.4 Creación del personaje principal del juego

A continuación, vamos a crear el personaje principal del videojuego en 2D para esto vamos a empezar creando la primera escena en 2D dando clic en escena 2D.

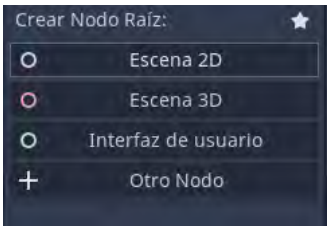

FIGURA 47 ESCENA 2D

Al crear la primera escena 2D Godot nos pedirá que creemos el primer Nodo Padre a este le pondremos el nombre de Nivel.

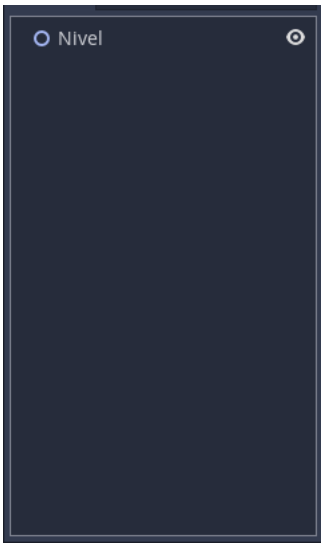

FIGURA 48 NODO NIVEL

Ya tenemos el Nodo Padre de nuestro juego ahora debemos crear los nodos hijos y a su vez crearemos el cuerpo del personaje, para hacer esto vamos a darle clic en el siguiente botón de más.

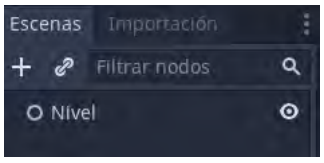

FIGURA 49 NODOS HIJOS

Al realizar esta acción se nos abrirá el Árbol de Nodos como ya hemos visto anteriormente Godot tiene diversos nodos a utilizar dependiendo del tipo de juego a desarrollar tenemos los nodos 3D, Nodos 2D y Nodos de Control.

FIGURA 50 ÁRBOL DE NODOS

De momento los nodos a utilizar serán los nodos relacionados al juego 2D, el nodo que utilizaremos a continuación será el nodo KinematicBody2D

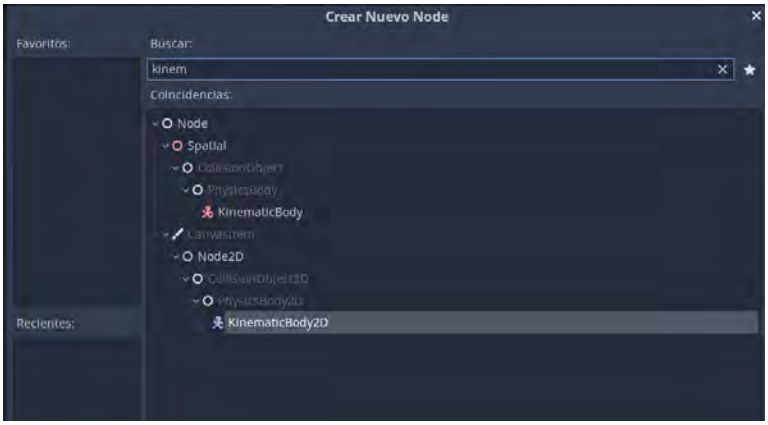

FIGURA 51 NODO KINEMATICBODY2D

3.2.5 ¿Qué es el Nodo KinematicBody2D?

Los cuerpos cinemáticos son tipos especiales de cuerpos destinados a ser controlados por el usuario. No se ven afectados por la física en absoluto; a otro tipo de cuerpos, como un personaje o

un cuerpo rígido, estos son lo mismo que un cuerpo estático. Sin embargo, tienen dos usos principales: (Manzur, s.f.)

Movimiento simulado: cuando estos cuerpos se mueven manualmente, ya sea desde el código o desde un AnimationPlayer (con AnimationPlayer.playback_process_mode establecido en "física" es decir la física establecida en el juego), la física calculará automáticamente una estimación de su velocidad lineal y angular. Esto los hace muy útiles para mover plataformas u otros objetos controlados por AnimationPlayer (como una puerta, un puente que se abre, etc.). (Manzur, s.f.)

Caracteres cinemáticos: KinematicBody2D también tiene una API para mover objetos (los métodos move_and_collide y move_and_slide es decir los movimientos y colisiones del personaje) mientras se realizan pruebas de colisión. Esto los hace realmente útiles para implementar personajes que chocan con un mundo, pero que no requieren física avanzada. (Manzur, s.f.)

Nosotros usaremos este Nodo para crear el cuerpo del personaje principal, para hacer esto lo que haremos será crear el Nodo.

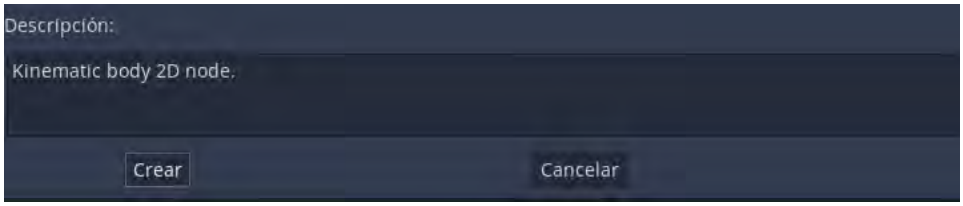

FIGURA 52 CREACIÓN DEL NODO

Una vez creado el nodo, le cambiaremos el nombre a "Jugador" como podemos observar este nodo creado es el hijo de nuestro Nodo Padre llamado "Nivel ".

FIGURA 53 NODO JUGADOR

Continuando con el proceso anterior, ahora añadiremos un nuevo Nodo llamado AnimatedSprite el cual funciona para agregar sprites como ya se mencionó en esta tesis un Sprite es una imagen que utiliza fotogramas para reproducir o realizar una animación.

FIGURA 54 NODO ANIMATEDSPRITE

Ahora que ya tenemos el nodo creado vamos a configurarlo y le agregaremos propiedades, para hacer esto hacemos clic en nuestro nodo y vamos al apartado del inspector.

FIGURA 55 INSPECTOR DEL NODO

Ahora agregaremos los frames o Fotogramas para darle la animación a nuestro Sprite, con lo cual haremos clic en frames seleccionaremos nuevo Sprite Frames y haremos clic en él.

FIGURA 56 AGREGANDO SPRITEFRAMES

Al hacer doble clic en el apartado SpriteFrames se nos abrirá el apartado de animaciones.

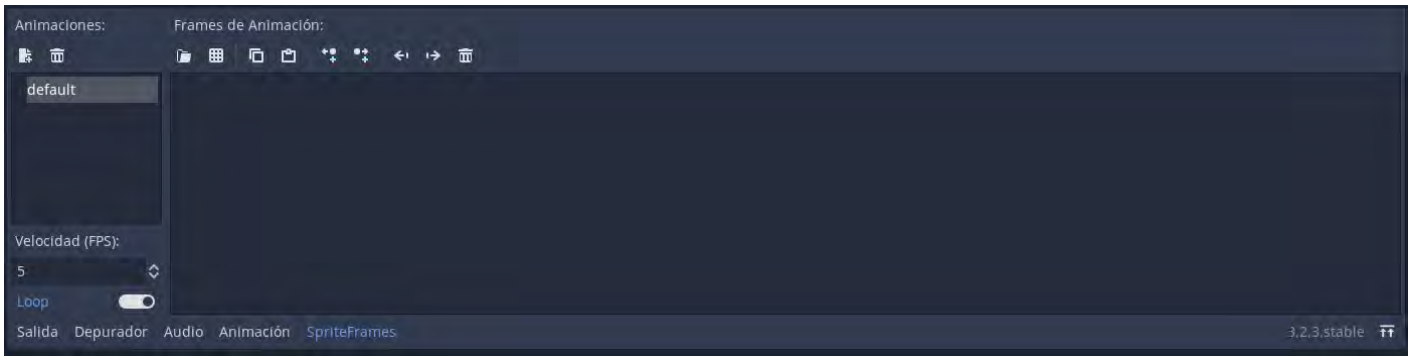

FIGURA 57 APARTADO DE ANIMACIONES

Ahora vamos a crear la primera animación o postura llamada Idle, la cual representa que el personaje esta quieto o en espera para hacer esto primero donde dice default le damos clic y cambiamos el nombre a Idle, segundo damos clic en la herramienta añadir frames.

Todos los sprites tienen diversas poses, al hacer clic en añadir frames desde un Sprite sheet buscaremos en nuestra carpeta Assets la pose idle de nuestro personaje.

Abrimos la imagen y nos saldrá el apartado de configuración de los fotogramas, en este proceso nos pide que pongamos la cantidad de imágenes que tengamos tanto verticalmente, como horizontalmente, en este caso tenemos 1 sola línea vertical y la imagen se repite 11 veces horizontalmente.

FIGURA 58 FOTOGRAMAS DEL SPRITE

Lo último que queda por definir es la velocidad de fotogramas que tendrá nuestra imagen por segundo, para tener una animación fluida, pondremos la velocidad a 24 fotogramas por segundo, además para ver la animación, vamos a nuestro apartado en el inspector y le damos clic en donde dice playing.

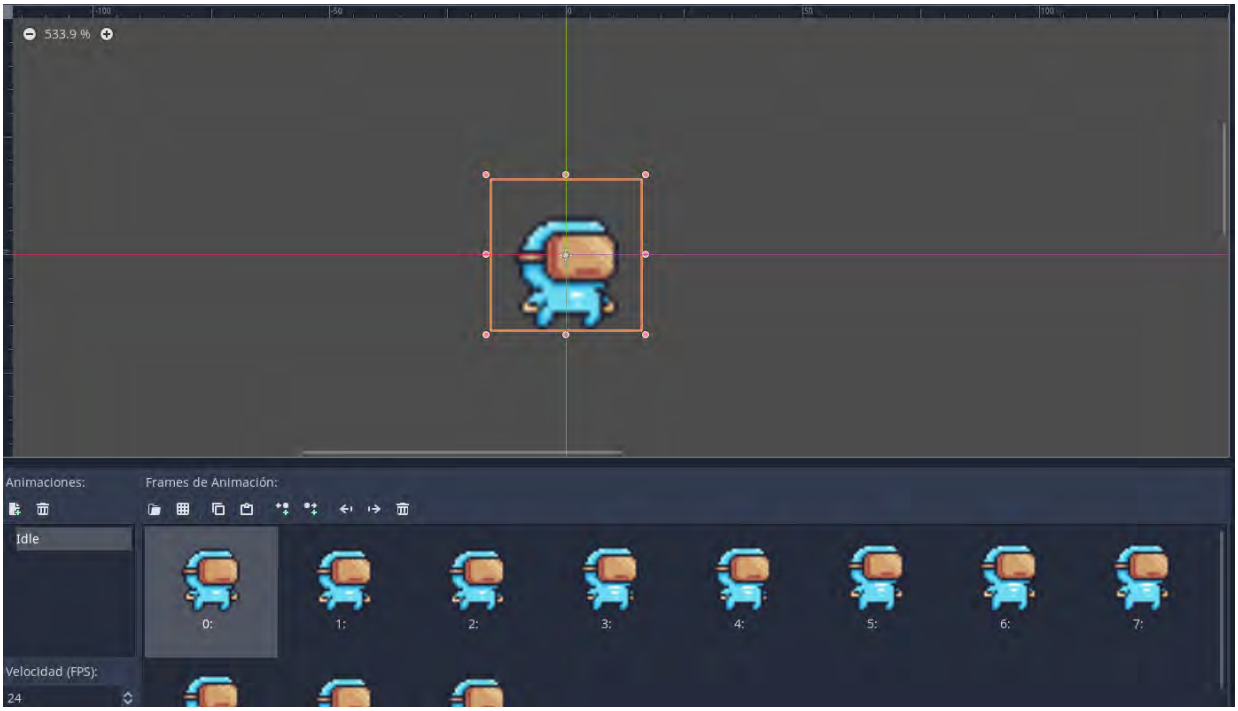

FIGURA 59 ANIMACIÓN IDLE

Tendremos que repetir este proceso para todas las poses Idle de los personajes, enemigos, objetos y todas las demás animaciones que queramos poner en nuestro juego 2D.

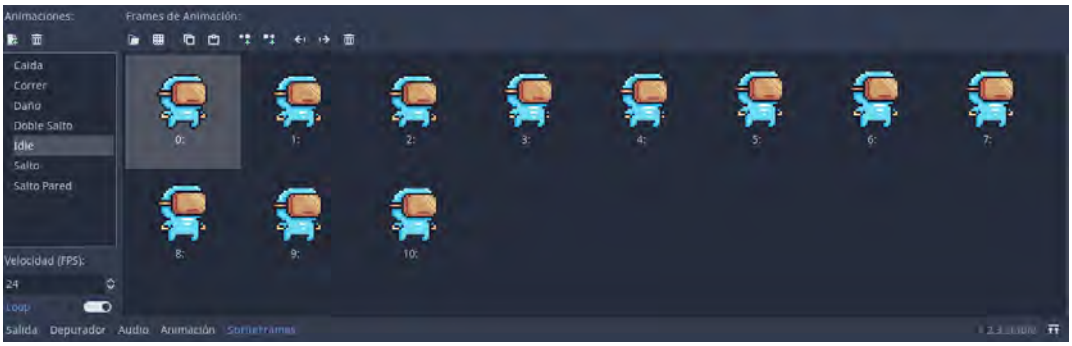

FIGURA 60 TODAS LAS ANIMACIONES PERSONAJE

3.3 Físicas 2D del cuerpo del personaje y colisiones

3.3.1 ¿Qué son las colisiones en los videojuegos?

En muchos juegos 2D se utilizan sprites, es decir, imágenes prediseñadas que se dibujan y mueven por la pantalla para representar a los personajes y demás elementos del juego. Además, suele ser necesario detectar cuando un elemento colisiona con otro para, por ejemplo, quitar vidas, destruirlo, o simplemente sujetarse en una plataforma sin caerse. (Ants, 2019)

Detectar cuando realmente se están tocando dos sprites requeriría comprobar si cada píxel visible (no los transparentes que suelen rodear al dibujo) de un Sprite está en contacto con algún píxel visible del otro Sprite con lo que, computacionalmente hablando, requeriría mucho tiempo.

Para poder detectar colisiones de una forma eficiente normalmente se recurre a suponer que cada Sprite está inscrito en alguna figura geométrica que, aunque sea más grande que el Sprite, permita aproximar su forma. Esto simplifica la detección de las colisiones entre elementos pues basta con hacer unos pocos cálculos matemáticos para determinar si las distintas figuras geométricas están solapándose o no. (Ants, 2019)

Se suelen usar círculos, rectángulos o incluso combinaciones de ambas para representar de forma aproximada el área ocupada por cada Sprite como se puede ver en las siguientes imágenes.

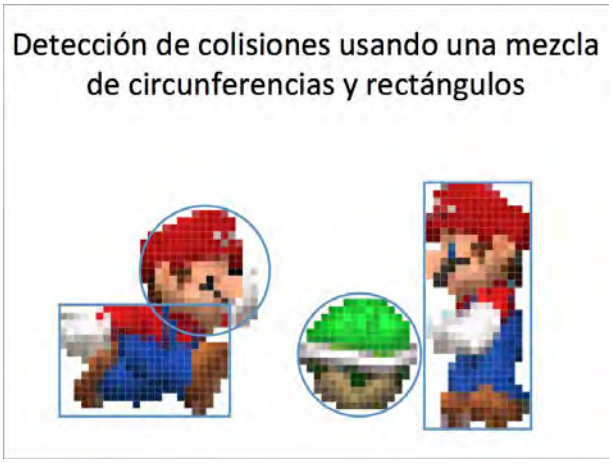

FIGURA 61 EJEMPLO DE COLISIONES

Ahora que sabemos que son las colisiones vamos a agregar un nuevo nodo en el nodo padre "Nivel" siguiendo el mismo proceso para agregar nodos, esta vez se trata del nodo ColisionShape2D el cual nos va a permitir agregar las colisiones a nuestro personaje principal.

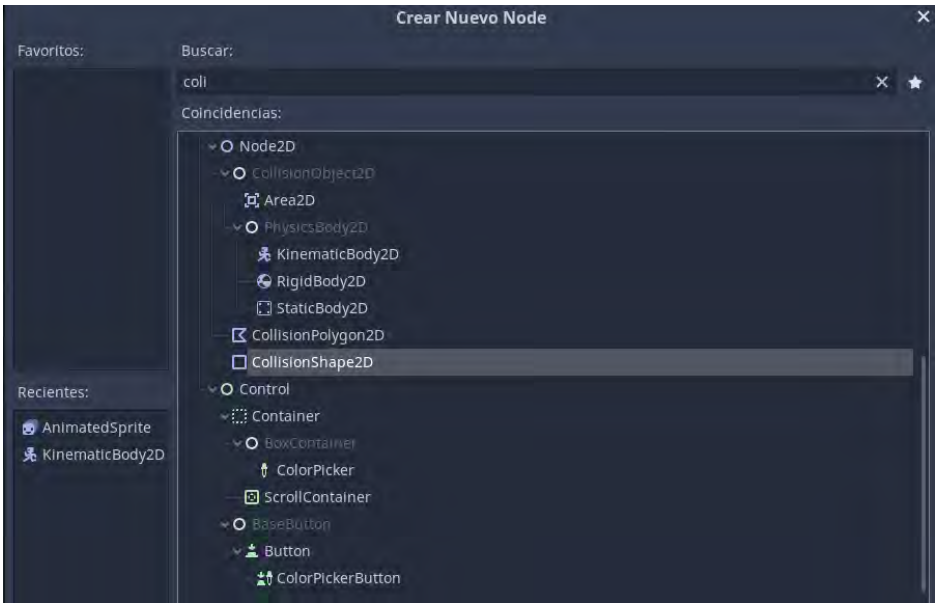

FIGURA 62 NODO COLISIONSHAPE2D

Una vez agregado el nodo de colisiones es hora de poner la figura de la colisión o común mente llamada hit box, para esto vamos al apartado del inspector de nuestro nodo de colisiones y le damos clic donde dice Shape.

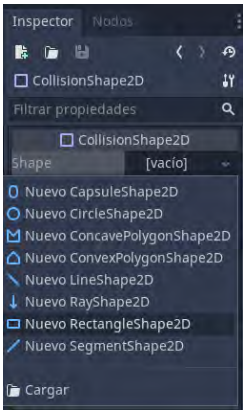

FIGURA 63 SHAPE

Seleccionamos el rectángulo y ahora debemos de poner la figura alrededor del personaje, un consejo útil es que debido a que estamos utilizando imágenes de tipo píxel art, es decir imágenes que usan pixeles podemos configurar Godot para trabajar en un ambiente relacionado a pixeles, para hacer esto damos clic en opciones de Snapping y le damos clic donde dice, usar píxel snap, esto convertirá nuestra área de trabajo en un área de trabajo para pixeles.

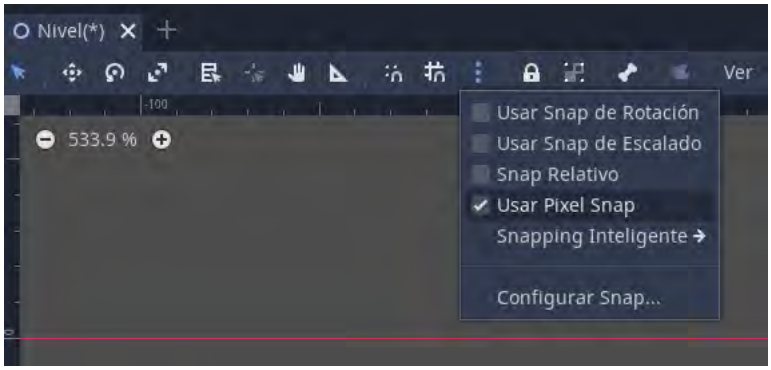

FIGURA 64 PÍXEL SNAPPING

Una vez realizado esto ahora debemos de poner la figura lo más alineada posible al personaje principal tratando de que quede justo en el cuerpo y pies del personaje.

FIGURA 65 RECTÁNGULO DE COLISIONES DEL PERSONAJE

3.3.2 Programación de Godot.

Ahora que ya tenemos las Colisiones es hora de empezar con la programación de Godot.

Primero Crearemos un Script o un Código de programación para hacer esto seleccionamos nuestro nodo llamado jugador y damos clic en añadir nuevo Script.

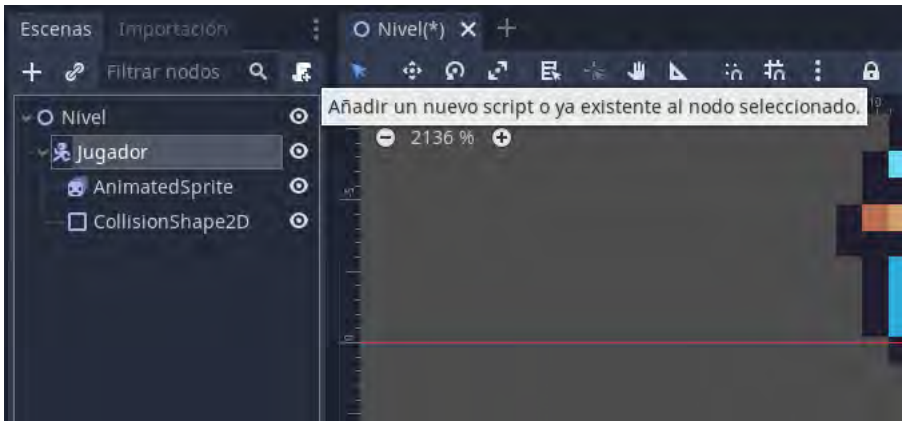

FIGURA 66 AÑADIR NUEVO SCRIPT

Nos saldrá el siguiente apartado en el cual se muestra el lenguaje de programación a usar, como ya se mencionó anteriormente Godot tiene diversos lenguajes de programación, en esta tesis utilizaremos el GDScript el cual es un lenguaje de programación de alto nivel, orientado a objetos, imperativo y tipificado gradualmente creado para Godot. Utiliza una sintaxis basada en sangría similar a lenguajes como Python. Su objetivo es optimizarse e integrarse estrechamente con Godot Engine, permitiendo una gran flexibilidad para la creación e integración de contenido.

GDScript es completamente independiente de Python y no se basa en él.

GDscript es de donde se hereda el script en este caso de nuestro nodo Jugador y la plantilla, así como el nombre del Script.

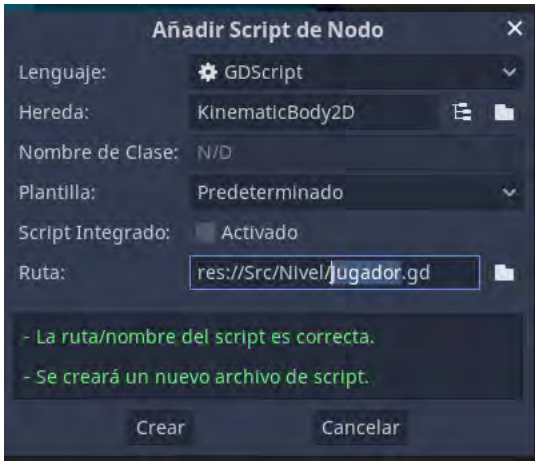

FIGURA 67 CREACIÓN DE SCRIPT

Se nos abrirá la pestaña de scripts que muestra lo siguiente:

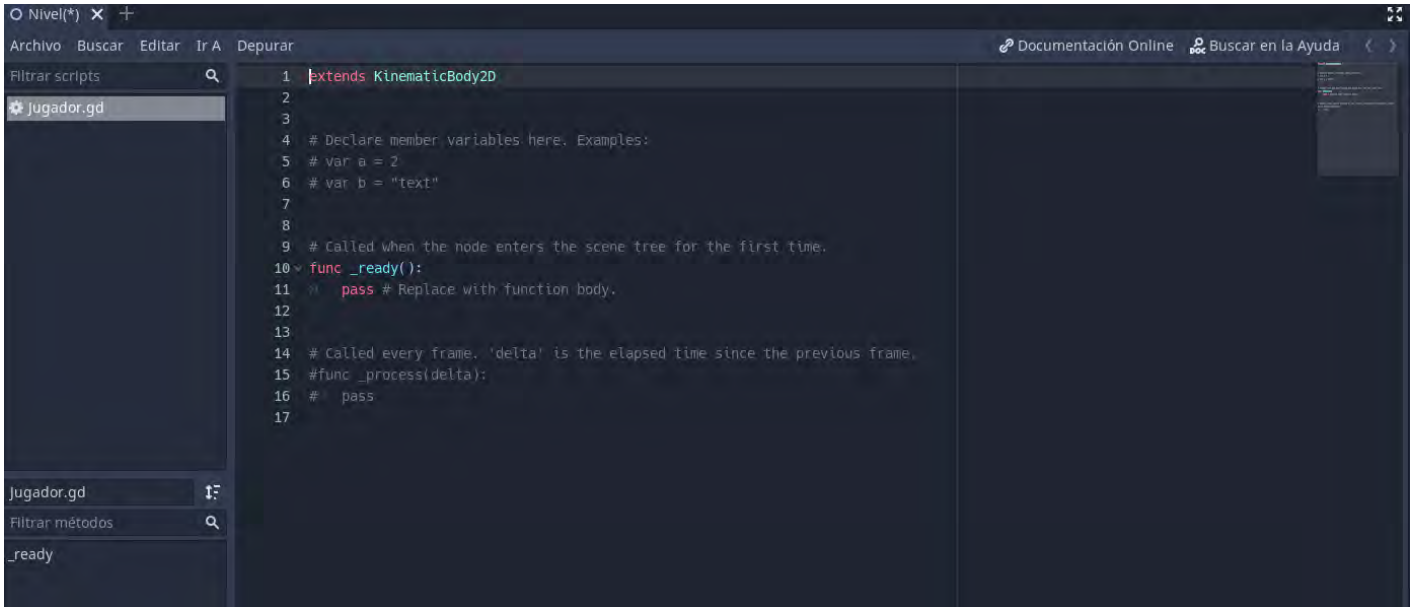

FIGURA 68 PESTAÑA DE SCRIPTS

En la primera línea la función extends trae todas las variables del nodo KinematicBody 2D, la función ready se llama justo cuando el nodo entra en escena por primera vez es decir al iniciar el juego, y dentro de esta función de ready tenemos la función Pass, la cual simplemente como su nombre lo indica pasa a la siguiente variable.

Ahora en la línea 11 tenemos la función process (delta): esta variable se reproduce cada frame para que se pueda realizar cada acción.

Una vez explicado este código básico, vamos a encapsular nuestro nodo de jugador para hacer esta acción primero realizamos los siguientes pasos:

Damos clic derecho en nuestro nodo jugador, de allí vamos a la opción que dice guardar rama como escena, una vez hecho esto guardamos nuestra escena en la carpeta src dentro de la carpeta nivel, una vez encapsulado nuestro nodo como una escena ahora solo tendremos el nodo jugador dentro de nuestro nodo padre nivel, esto sirve para tener una mayor organización y flexibilidad ya que al dividir nuestros nodos en escena ayudan a realizar cambios únicos en la escena en la cual tu deseas trabajar sin realizar cambios en otra escena.

FIGURA 69 GUARDAR RAMA COMO ESCENA.

Lo siguiente que vamos a realizar es el ajuste del proyecto para hacer esto damos clic en proyecto y luego como su nombre lo indica en ajustes del proyecto, dentro del buscador pondremos la palabra window.

FIGURA 70 PANTALLA DEL JUEGO.

Dentro de esta configuración cambiaremos nuestro ancho y largo de la pantalla a la mitad también en el apartado de escalado pondremos el modo 2D para tener una pantalla escalada y adecuada a un videojuego 2D.

FIGURA 71 TAMAÑO DE PANTALLA.

FIGURA 72 ESCALADO DE PANTALLA.

Ya que tenemos la pantalla configurada ahora tenemos que definir las teclas con las cuales nuestros jugadores podrán moverse dentro del juego para hacer esto vamos una vez más a la configuración del proyecto y damos clic en donde dice, mapa de entrada una vez dentro de esta pestaña creamos las acciones del personaje, en donde dice acción escribiremos las diversas acciones que nuestro

personaje puede tomar, una vez añadidas definiremos que tipo de teclas son si son de algún mando, si son teclas del teclado de una pc, en general el tipo de controlador del personaje.

FIGURA 73 TECLAS DEL JUEGO.

Una vez definidas las teclas es hora de agregarlas en nuestro script de programación para hacer esto vamos a nuestro script de jugador y obtendremos el siguiente código de programación.

```
1  extends KinematicBody2D
2
3
4  var velocidad = 120
5
6
7  # Called when the node enters the scene tree for the first time.
8  func _ready():
9      pass # Replace with function body.
10
11
12 # Called every frame. 'delta' is the elapsed time since the previous frame.
13 func _process(delta):
14     if Input.is_action_pressed("derecha"):
15         position.x += velocidad * delta
16
17     if Input.is_action_pressed("izquierda"):
18         position.x -= velocidad * delta
19
20
```

FIGURA 74 TECLAS DE MOVIMIENTO.

Primero tuvimos que crear una Variable llamada velocidad en la línea 4 la cual indica la velocidad de frames con la cual se mueve nuestro videojuego luego en la línea 14 se puede leer lo siguiente,

if (si) input (valor de entrada) is_action_pressed (es una acción que se presiona o se presiona la tecla de esa acción), definimos la acción derecha entonces position.x (es la posición x en el plano cartesiano) es mayor o igual que la variable velocidad por delta.

Tendremos que repetir este código para también movernos a la izquierda tal y como lo muestra la imagen. También de este modo cambiando la posición de x a y podremos movernos en el plano y del plano cartesiano.

3.3.4 Añadiendo escenario y creación del mapa del juego

Ahora vamos a añadir el escenario y el mapa del juego para hacer esto vamos a crear un nuevo nodo en nuestro nodo padre llamado nivel, y añadiremos el siguiente nodo tilemap.

Este nodo nos crea una rejilla en nuestra área de trabajo creando pequeños cuadros en los cuales podemos añadir la escenografía de los niveles del juego, entre otros objetos a utilizar en el juego también.

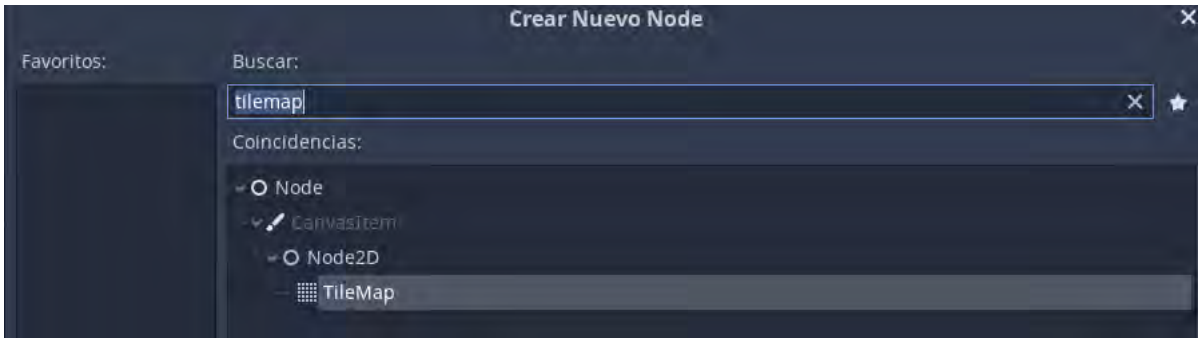

FIGURA 75 NODO TILEMAP.

Ahora en nuestro inspector vamos a agregar un tileset es decir el escenario para hacer esto vamos a nuestro inspector y damos clic en añadir nuevo tileset.

FIGURA 76 TILESET.

Una vez creado el tileset debemos de añadir las imágenes del escenario del juego para hacer esto vamos a darle clic a nuestro tileset, después vamos a la tecla de más y allí incluiremos las imágenes de nuestro background o escenario.

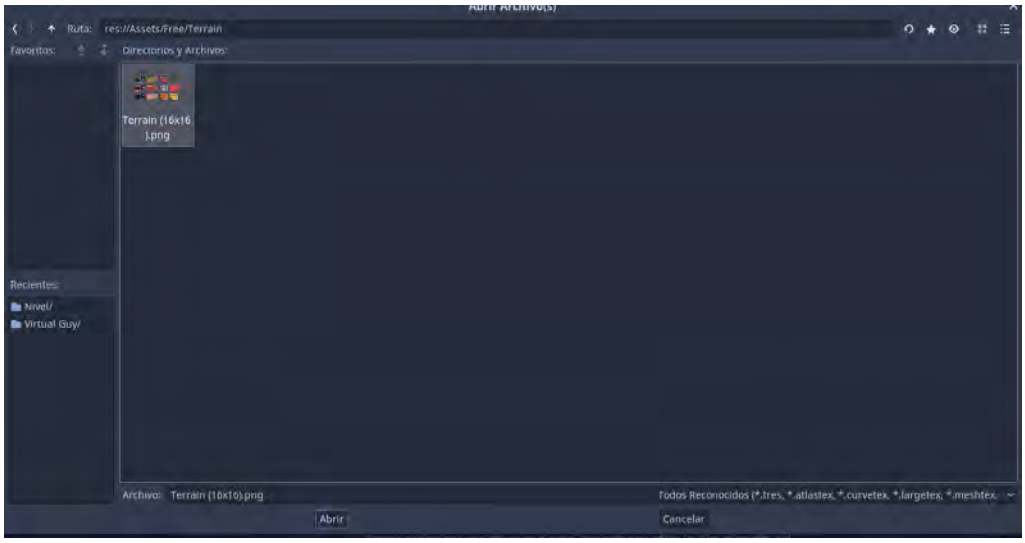

FIGURA 77 AÑADIENDO ESCENARIO.

Una vez agregadas las imágenes tendremos que seleccionar si es un tile individual es decir una escenografía sola para hacer esto debemos de dar clic en donde dice tile individual, configuramos ese tile según el tamaño apropiado de la imagen a utilizar en este caso la mía es de 16 x 16 pixeles, y lo seleccionamos.

FIGURA 78 AÑADIENDO TILEMAP.

Ahora solo falta crear el mapa utilizando los diferentes tiles que tu necesites incluir en el juego, dependiendo de tus gustos diversos o de los requerimientos del cliente entre otros factores a tomar en cuenta.

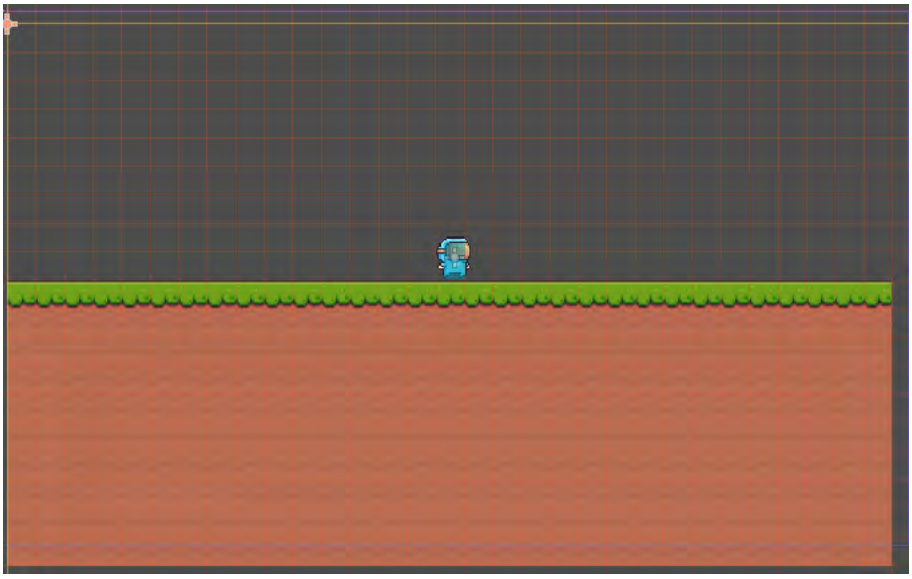

FIGURA 79 EJEMPLO DE UN TILESET.

También podemos crear un nuevo tipo de tile llamado, Autotile el cual se encarga de agrupar los tiles individuales en uno solo, para hacer esto en nuestra pestaña de tileset le damos clic en nuevo autotile y seleccionaremos el conjunto de bloques que queremos que forme el autotile, usando la

herramienta de bitmask o mascara de bit que se encargara de agrupar el autotile tal y como se muestra en la imagen.

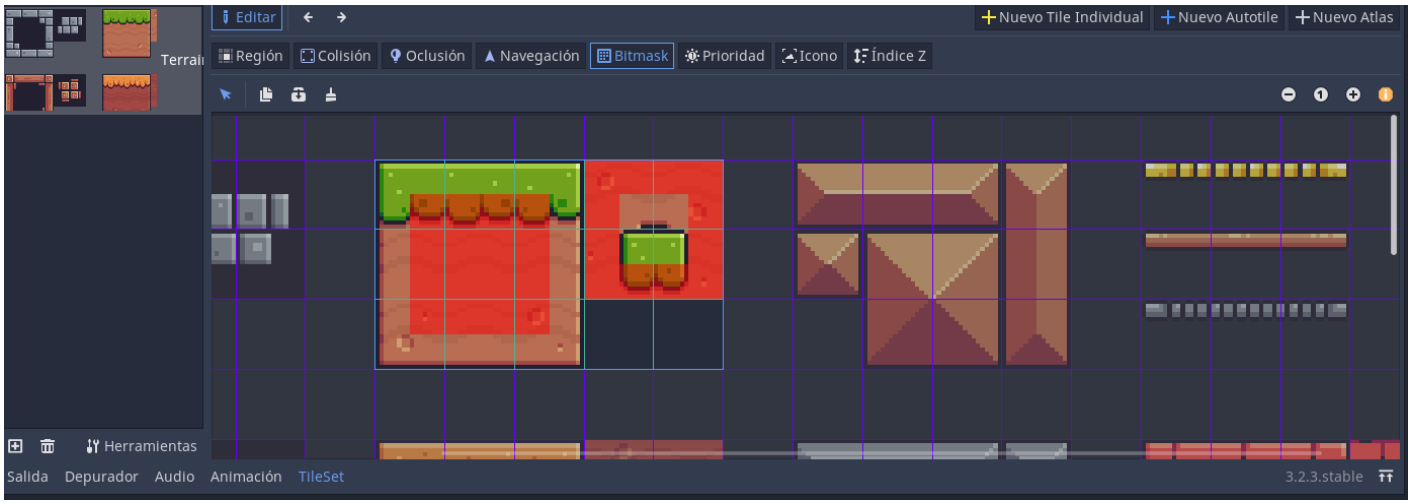

FIGURA 80 AUTOTILE.

A continuación, vamos a programar la muerte del jugador y también seguidamente comenzaremos a agregar trampas a nuestro mapa creado hasta el momento, con las cuales el jugador interactuará y este mismo a su vez, tendrá la posibilidad de perder el juego al tocarlas y activarlas.

Comenzaremos con una muerte básica, una muerte por caída, para hacer esto usaremos una herramienta nueva llamada “Ruler Mode”.

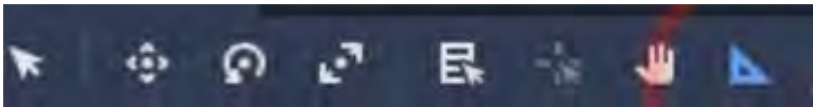

FIGURA 81 HERRAMIENTA RULER MODE.

La cual esta remarcada en azul y tiene forma de escuadra, la herramienta Ruler Mode nos permite medir en forma de pixeles, esto nos servirá mucho para medir la cantidad de caída que queremos darle a nuestro jugador, vamos a tomarla y a empezar a medir nuestro escenario de trabajo, tal y como se muestra en la siguiente imagen.

FIGURA 82 MIDIENDO EL ESCENARIO USANDO RULER MODE.

Tal y como se muestra en la imagen nuestro escenario actual tiene un aproximado de 370 píxeles

FIGURA 83 PIXELES DEL ESCENARIO.

Teniendo en cuenta estas medidas, vamos a entrar al script de nuestro jugador en el aparatado en donde tenemos nuestro código, y vamos a crear una función de muerte, para crear una función usamos el comando “func” y le pondremos de nombre muerte, a continuación usaremos la función get_tree la cual nos permitirá acceder a nuestro árbol de nodos, o en este caso a nuestro árbol de escenas, además vamos a agregarle a nuestra función el comando reload_current_scene, este comando nos permitirá recargar la escena que esta activa, también agregaremos una condicional para condicionar la muerte del jugador usando la posición de los pixeles medidos anteriormente, nuestro código quedara de la siguiente manera.

```
75    if position.y > limite_caida:
76        muerte()
77
78
79  func muerte():
80      get_tree().reload_current_scene()
81
```

FIGURA 84 CÓDIGO MUERTE POR CAÍDA.

Con este código tenemos la primera muerte a implementar, a continuación, haremos una de las primeras trampas que usaremos en el juego, los cuales serán los picos, que al tocarlos nuestro jugador morirá y el juego se reiniciará o en cuyo caso dependiendo de tus gustos o especificaciones necesarias del cliente, este perderá una vida.

Lo primero que haremos será crear una nueva escena, la cual seguirá siendo una escena en 2D y le pondremos de nombre picos, dentro de esta nueva escena añadiremos un nuevo nodo llamado “Sprite”.

FIGURA 85 NODO SPRITE.

A continuación, vamos a ir a nuestra carpeta Assets, luego vamos a la carpeta Free, luego a la carpeta Traps, a continuación, a la carpeta Spikes y allí encontraremos una imagen de nuestros picos.

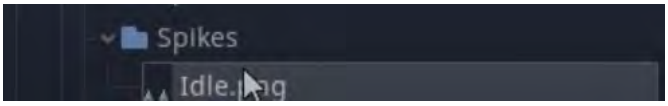

FIGURA 86 IMAGEN DE NUESTROS PICOS.

A continuación, vamos a arrastrar nuestra imagen al apartado del nodo Sprites tal y como se muestra en la imagen.

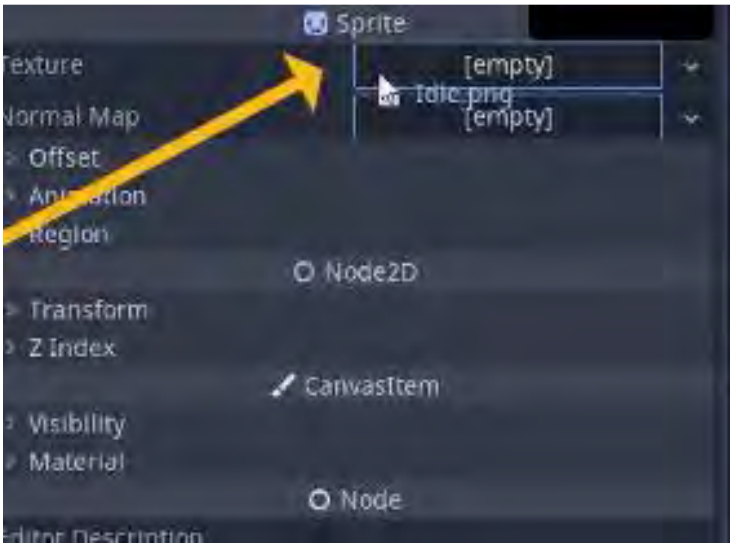

FIGURA 87 ARRASTRANDO LA IMAGEN A NUESTRO NODO.

Una vez realizado esto tendremos nuestros picos en nuestra área de escenario, sin embargo, recuerda que debemos de reimportarlos una vez más, para evitar que se vean borrosos, una realizada la reimportación tendremos nuestros picos de la siguiente manera.

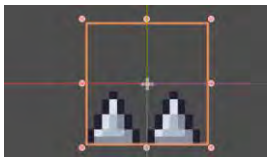

FIGURA 88 PICOS IMPORTADOS.

Guardaremos nuestra escena tal y como echo anteriormente y le pondremos el nombre de picos, ahora que tenemos nuestros picos importados, debemos añadirlos a nuestra escena principal del nivel, para hacer esto vamos a dar clic a nuestra escena llamada nivel, una vez en nuestra escena principal de nuestro juego, vamos a ir al nodo de nuestro nivel y daremos clic sobre el botón que se muestra en la siguiente imagen.

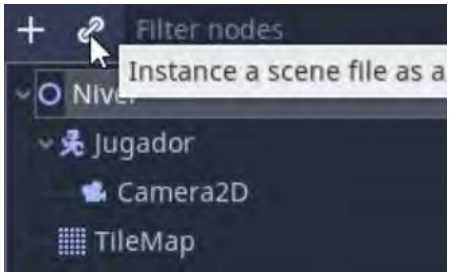

FIGURA 89 BOTÓN DE ESTANCIA.

Este botón tiene la función de instanciar un archivo de escena como si fuera un nodo, al hacer clic en él, nos abrirá todas las escenas que tenemos actualmente, nosotros abriremos la escena de picos que creamos anteriormente.

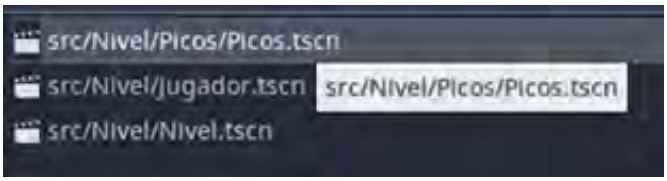

FIGURA 90 ABRIENDO LA ESCENA DE PICOS.

ahora si regresamos a nuestra escena de nuestro nivel principal veremos 2 cosas, la primera es que tenemos un nuevo nodo en nuestro apartado de escena llamado picos, y la segunda es que nuestra imagen de picos se encuentra actualmente en nuestro escenario principal de nuestro juego.

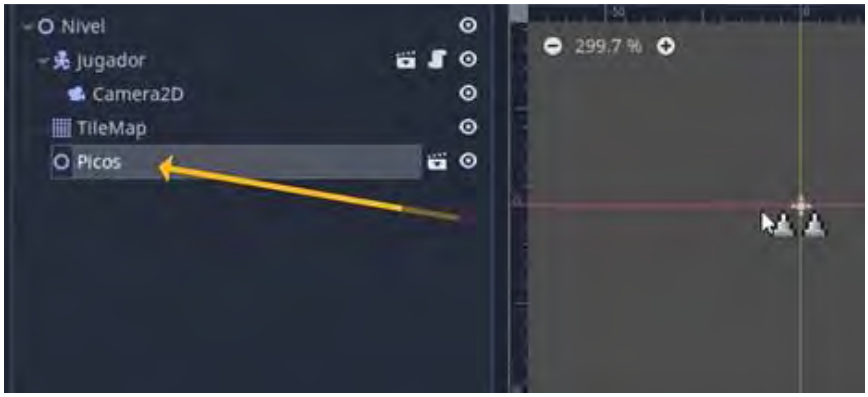

FIGURA 91 PICOS EN NUESTRO ESCENARIO PRINCIPAL.

Como último paso activaremos una herramienta para poder posicionar de una manera correcta nuestros picos, usando el siguiente botono de rejilla que se muestra en la imagen.

FIGURA 92 BOTÓN DE REJILLA.

De esta manera resulta más cómodo y eficaz acomodar los picos dentro de nuestro mapa del juego, quedando de manera final de la siguiente forma.

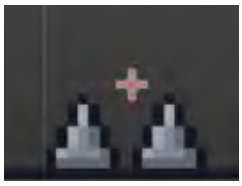

FIGURA 93 PICOS FINALES.

Ya tenemos nuestros picos de manera gráfica, sin embargo, ahora tenemos que hacer la programación de estos mismos para que interactúen con nuestro jugador y este mismo pueda morir al tocarlos, lo primero que vamos a hacer es irnos a nuestra pestaña que creamos anteriormente llamada picos, una vez estando en la escena de picos, vamos a crear un nuevo nodo que nos va a ayudar a detectar el cuerpo del jugador, este nodo se llama “Area2D”.

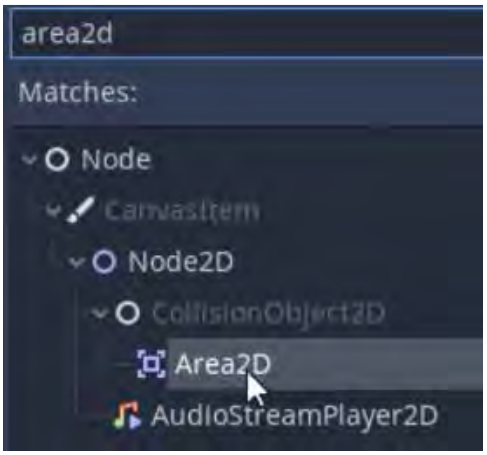

FIGURA 94 NODO AREA2D.

Ahora que tenemos nuestro nodo de area2d, lo seleccionamos y damos al botón de +, lo que vamos a hacer a continuación es crear un área de colisión para nuestro nodo, para hacer esto buscamos el nodo de colishionshape2d.

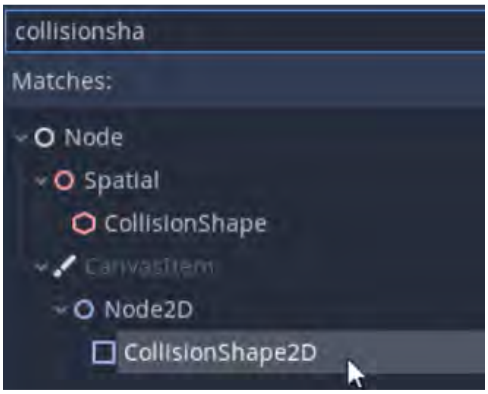

FIGURA 95 NODO COLISHIONSHAPE2D.

Una vez que tenemos nuestro nodo de Colishionshape2d ahora queda darle una forma tal y como hicimos anteriormente con nuestro personaje que maneja el jugador, para hacer esto tenemos que, irnos a nuestro apartado del inspector y en el apartado de shape, le daremos clic y seleccionaremos un rectángulo.

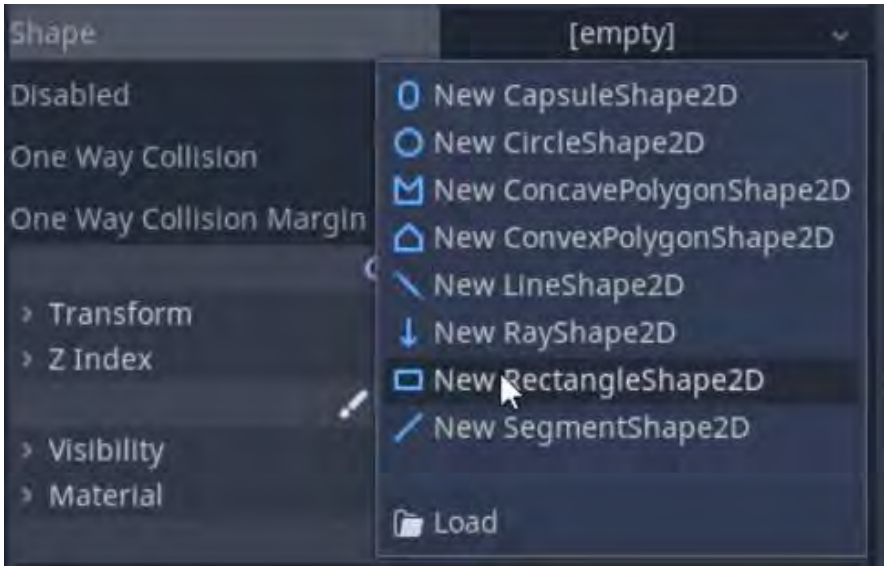

FIGURA 96 CREANDO UN ÁREA DE COLISIÓN RECTANGULAR.

El siguiente paso es ajustar el área de colisión para que quede justo en los picos, una vez ajustada quedaría de la siguiente manera.

FIGURA 97 ÁREA DE COLISIÓN DE LOS PICOS.

Lo siguiente que tenemos que hacer es empezar a programar nuestros picos para que estos interactúen con nuestro jugador, para realizar esto vamos a crear un nuevo script para nuestros picos, una vez creado el script, ahora vamos a hacer uso de otra de las grandes herramientas que nos da Godot, la cual es llamada como signals o eventos, damos clic en nuestro nodo de "area2d",a continuación del lado derecho del área del inspector, tenemos una pestaña llamada nodo, damos clic en ella, allí encontraremos las signals o los eventos.

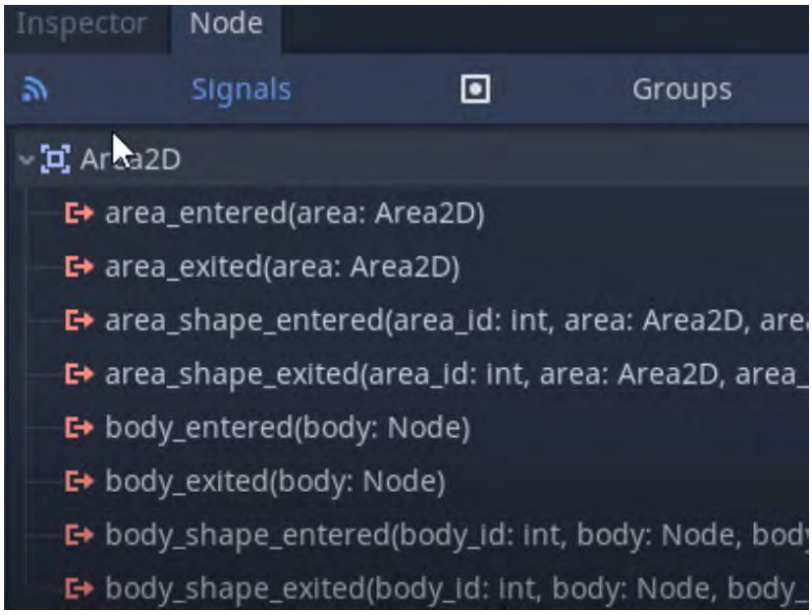

FIGURA 98 SIGNALS O EVENTOS.

Por el momento nosotros vamos a trabajar con los eventos del Area2D, usando en específico el evento de body_entered, este evento significa que se activa cuando un cuerpo entra dentro de nuestra area2d es decir cuando entre dentro del área de nuestros picos creados anteriormente, le damos clic en donde dice, body_entered y al hacer doble clic en él, Godot nos permite conectar este evento, nosotros lo vamos a conectar en el nodo de picos.

FIGURA 99 CONECTANDO EN EL NODO DE PICOS.

Al realizar esta acción Godot automáticamente abre nuestro script de picos y nos crea una función.

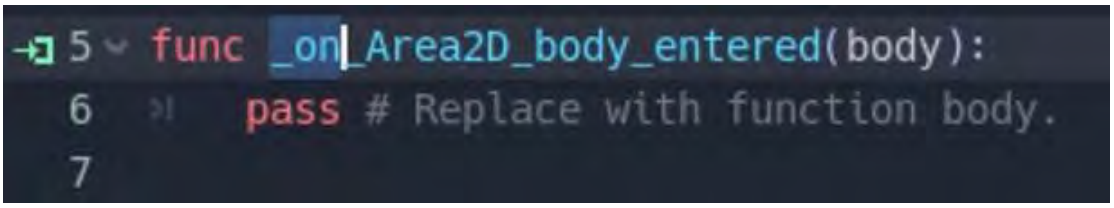

FIGURA 100 FUNCIÓN ON AREA2D.

Esta función nos dice que cuando al area2d haya entrado un cuerpo, nos pasa al área de body, ósea al área del cuerpo de nuestro jugador, sin embargo, el cuerpo del jugador aún no está etiquetado, por lo tanto, Godot, puede tomar el cuerpo del jugador o el cuerpo del escenario, para evitar que esto pase vamos a abrir nuestra escena del jugador, para hacer esto vamos al filesystem y escribimos el nombre jugador.

FIGURA 101 ABRIENDO NUESTRA ESCENA DEL JUGADOR.

Abrimos la escena de jugador. tscn y en la misma pestaña de nodo que vimos anteriormente, vamos al apartado de grupos, vamos a crear un nuevo grupo llamado "JUGADOR" es importante ponerlo todo en mayúsculas, tal y como se muestra en la siguiente imagen.

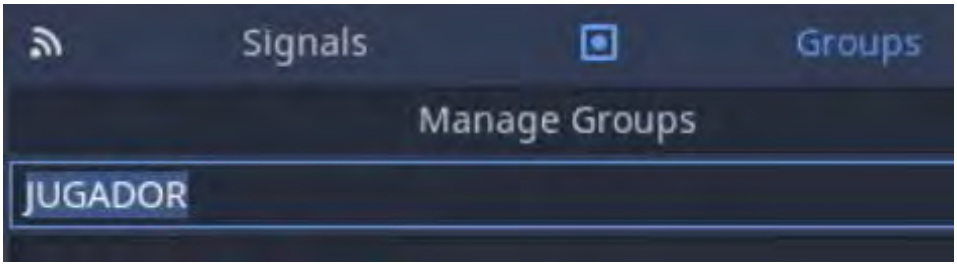

FIGURA 102 CREANDO EL GRUPO JUGADOR.

Ahora que nuestro jugador ya está etiquetado, volvemos a nuestro archivo de script de picos y en nuestra función que nos creó Godot automáticamente vamos a remplazar el cuerpo de esa función por un condicional, usando nuestro nuevo grupo que creamos del jugador, el código quedaría de la siguiente manera, en la cual nos dice, que si el cuerpo que está en el grupo de nuestro jugador entra al área de los picos que nosotros creamos este muere.

```
5 func _on_Area2D_body_entered(body):
6     if body.is_in_group("JUGADOR"):
7         body.muerte()
8
9
```

FIGURA 103 CÓDIGO DEL ÁREA DE MUERTE.

Repitiendo este proceso es que crearemos nuestras diversas trampas que se encuentren en los pisos de nuestro mapa.

Ahora vamos a crear una trampa diferente, una trampa de salto, para hacer esto vamos a crear una nueva escena llamada trampolín.

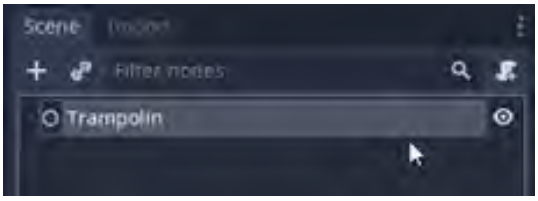

FIGURA 104 NUEVA ESCENA TRAMPOLÍN.

Tal y como hemos realizado anteriormente, añadiremos un nuevo nodo llamado animatedsprite y en nuestra área de inspector en el apartado de frames,vamos a añadirles sus frames, se nos abrirá el panel de animación, tal y como hicimos con la animación de correr, de nuestro personaje principal que maneja el jugador, animaremos los frames de nuestro trampolín, las imágenes que usaremos para nuestro trampolín se encuentran en la carpeta assest, luego en la carpeta free, luego en traps y al final en trampoline.

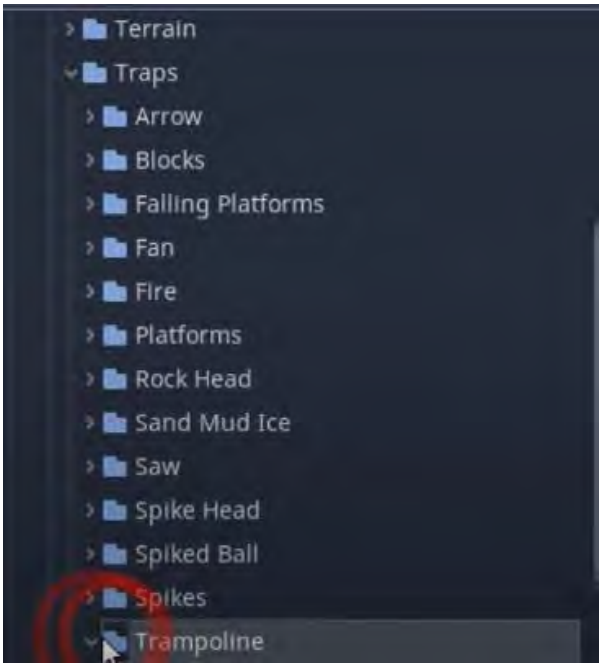

FIGURA 105 AÑADIENDO LOS FRAMES A NUESTRO TRAMPOLÍN.

Procedemos a reimportar la imagen para que no se vea borrosa como hemos hecho con todas las imágenes reimportadas hasta el momento y vamos a crear una nueva animación la cual se va a llamar activado, hacemos el mismo proceso que realizamos con el movimiento de correr de nuestro personaje quedando de la siguiente manera.

FIGURA 106 ANIMACIÓN ACTIVADA.

Configuramos la velocidad de animación a nuestro gusto, es decir la velocidad de los FPS, en mi caso la velocidad que use fue de 18 FPS, y quitamos el apartado que dice Loop, ya que para esta animación no queremos que se repita el proceso, si no que se active las veces que nuestro jugador entre en contacto con nuestro trampolín, ahora lo que haremos a continuación es crear nuestro código para nuestro trampolín, vamos a nuestro nodo y creamos nuestro script del trampolín, también una vez guardada la escena nosotros podemos incluir nuestro trampolín en nuestro nivel, tal y como hicimos con nuestros picos repitiendo el proceso anterior.

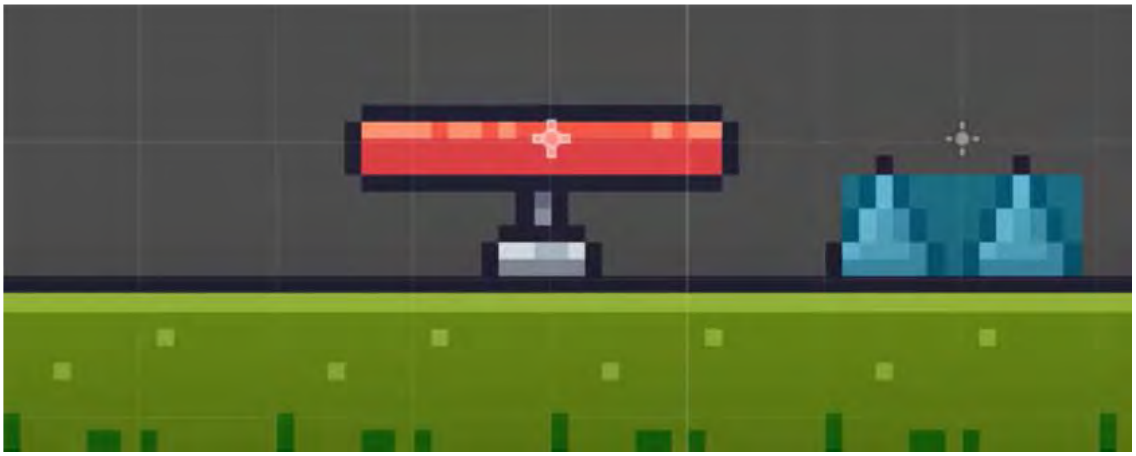

FIGURA 107 TRAMPOLÍN EN NUESTRA ESCENA DEL JUEGO.

Ahora vamos a programar nuestro trampolín para que interactúe con nuestro jugador, al igual que hicimos con los picos, vamos a seleccionar nuestro nodo de trampolín y añadiremos un nuevo nodo llamado area2d, también de la misma forma dentro de nuestro nodo de area2d, vamos a añadirle el nodo colishionshape2d, al igual que con los picos,en nuestro inspector en el área de shape, dentro de nuestro nuevo nodo de colishionshape2d, también seleccionamos el rectangleshape2d, y configuramos nuestra área de colisión al igual que hicimos con los picos.

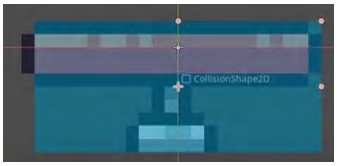

FIGURA 108 ÁREA DE COLISIÓN DEL TRAMPOLÍN.

Ahora seleccionamos area2d, vamos a nuestro inspector, en el apartado de signals o eventos y repetimos el proceso que hicimos con los picos, seleccionando el body_entered y lo conectamos a nuestro nodo de trampolín.

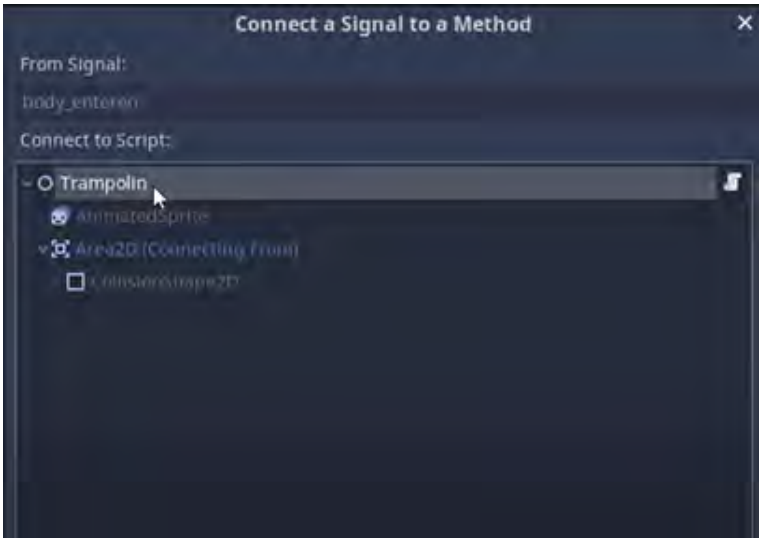

FIGURA 109 CONECTANDO NUESTRO EVENTO A TRAMPOLÍN.

Antes de configurar nuestro script de trampolín es necesario crear la función trampolín para hacer esto vamos a nuestro script de jugador y creamos la función trampolín, además también debemos crear una variable llamada impulso_trampolin.

```
12  var fuerza_salto = 390
13
14  var impulso_trampolin = 500
15
16
17  var limite_caida = 750
```

FIGURA 110 VARIABLE IMPULSO TRAMPOLÍN.

```
func trampolin():
    movimiento.y = -impulso_trampolin
```

FIGURA 111 FUNCIÓN TRAMPOLÍN.

Al hacer esto ya podemos abrir nuestro script del trampolín y tal y como hicimos con los picos pondremos el siguiente código, el cual nos dice lo siguiente, si un cuerpo que pertenece al grupo de jugador entra al área de colisión de nuestro trampolín, entonces el cuerpo interactúa con nuestro trampolín.

```
5  func _on_Area2D_body_entered(body):
6      if body.is_in_group("JUGADOR"):
7          body.trampolin()
8
```

FIGURA 112 SCRIPT DEL TRAMPOLÍN.

Repetiremos este proceso para todas las trampas que sean de salto, o que hagan saltar a nuestro jugador cuando interactúen con ellas.

A continuación, haremos otro tipo de elementos que se usan en el diseño de un videojuego los cuales son los coleccionables o los ítems, que nuestro jugador puede usar, recoger o hasta comprar dentro de un juego.

Como en todos los procesos que hemos realizado anteriormente vamos a crear una nueva escena 2D, creamos un nuevo nodo de animatedsprite, añadimos de igual forma un nodo de area2d, y de

igual forma una colishionshape2d, estos nodos que estamos añadiendo en estos momentos son los más utilizados y los más generales a la hora de realizar un videojuego en Godot, es por esto por lo que tienden a repetirse mucho, bueno cambiamos el nombre de nuestro nodo principal a Fruta.

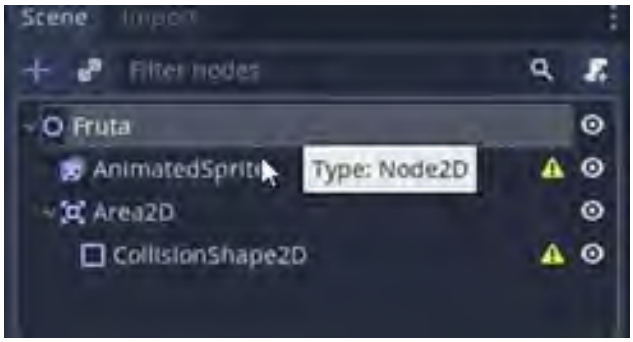

FIGURA 113 NUEVA ESCENA FRUTA.

de igual forma vamos a repetir el proceso de frames y de animación, vamos a nuestro nodo animatedsprite, vamos a nuestro inspector, seleccionamos frames y creamos new Sprite frames, cambiamos los nombres de las animaciones, la que está por defecto se llama idle, vamos a nuestra carpeta de recursos que en mi caso es assest, ítems, fruits, y seleccionamos la fruta que más nos guste.

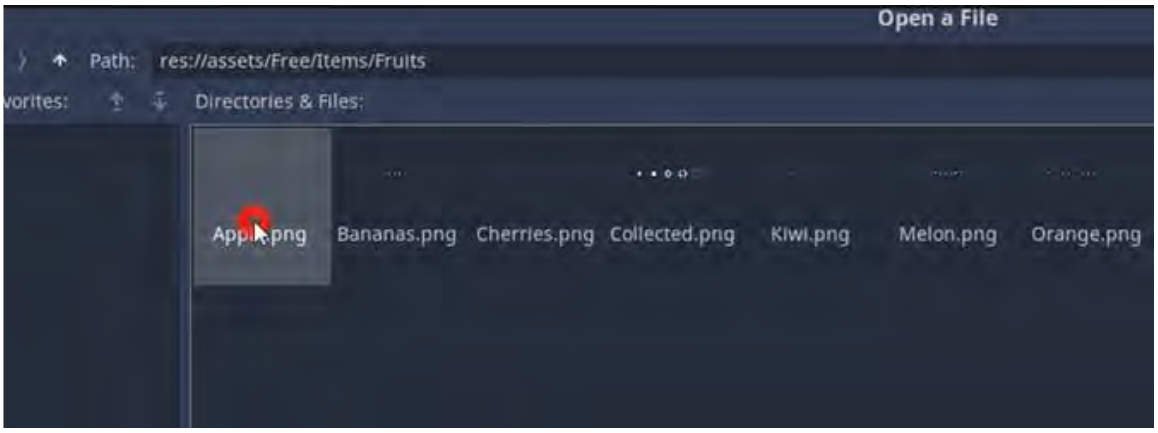

FIGURA 114 SELECCIONANDO LOS SPRITES.

Ajustamos los frames necesarios en mi caso fueron 17 horizontal y 1 vertical.

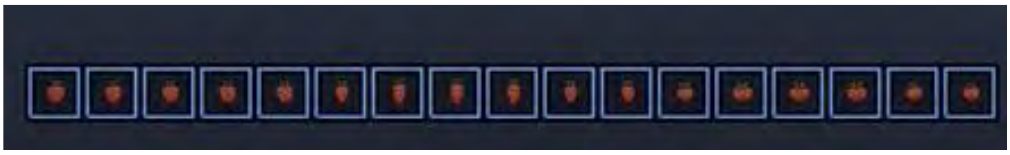

FIGURA 115 FRAMES DE LA FRUTA.

Como toda imagen la tenemos que reimportar para que no se vea borrosa y procedemos a realizar la animación de la fruta. Seleccionamos la velocidad de los FPS que más nos gusten, en mi casa la velocidad fue de 24 FPS, De igual forma vamos a crear una nueva animación para que cuando nuestro jugador pase por la fruta, la atrape o la recolecte, para hacer esto creamos una nueva animación llamada recolectada, buscamos en nuestra carpeta de assest los sprites de recolectec.

FIGURA 116 SPRITES RECOLECTEC.

Una vez más añadimos la velocidad que más nos guste en mi caso fue de 10 FPS, quitamos la función de Loop, para que nuestra animación solo se reproduzca una vez al momento, en el cual nuestro jugador toque la fruta, insertamos un fotograma vacío para que nuestra animación de la fruta desaparezca al momento de ser tocada quedando de la siguiente manera.

FIGURA 117 ANIMACIÓN RECOLECTADA.

Ahora repetimos el proceso de la colishionshape2d, creando nuestra área de colisión de la fruta, para que cuando nuestro jugador entre dentro de esta área, este interactúa con ella y se reproduzca la animación.

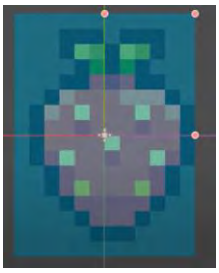

FIGURA 118 ÁREA DE COLISIÓN DE LA FRUTA.

Ahora importamos nuestra fruta a nuestra escena principal del juego, tal y como hemos realizado anteriormente con los picos y con el trampolín.

Una vez realizado esto, ahora toca hacer la programación de la fruta, para que interactué con nuestro jugador, una vez más vamos a nuestra escena principal de fruta, añadimos un nuevo script en nuestro nodo de fruta, en nuestro script vamos a poner una variable que se llama fue_recolectada y le daremos un valor de falso, ahora vamos a nuestro nodo de area2d, y vamos a agregar el evento o la signal de body_entered, una vez más usamos este evento, de la misma manera que lo usamos en los picos y en el trampolín, lo conectamos a fruta, ahora necesitamos que en nuestro código se refleje ese cambio de animación para hacer esto pondremos lo siguiente, onready var animation = $animatedsprite, este código nos permite acceder a nuestra nueva animación de recolectada, repetimos el código usado de la etiqueta de nuestro jugador, y le agregamos nuestra variable fue_recolectada y le damos el valor verdadero,por ultimo llamamos a nuestra función play y buscamos nuestra animación de recolectada, nuestro código para la fruta quedaría de la siguiente forma:

```
1  extends Node2D
2
3
4  var fue_recolectada = false
5
6
7  onready var animacion = $AnimatedSprite
8
9  func _on_Area2D_body_entered(body):
10     if body.is_in_group("JUGADOR"):
11         fue_recolectada = true
12         animacion.play("recolectada")
13
```

FIGURA 119 CÓDIGO DE FRUTA.

Repetiremos este procedo para todos los ítems o coleccionables que nosotros queramos incluir dentro de nuestro videojuego.

3.3.6 Efectos de sonido dentro de un videojuego.

En este apartado de la tesis veremos todo lo relacionado a efectos de sonido que se usan dentro de los videojuegos, el cómo utilizarlos y programarlos dentro de Godot y algunas funciones y variables específicas de Godot para manejarlos de una manera más profesional.

3.3.7¿Qué es un sonido?

En física se le llama sonido a la propagación de las ondas mecánicas que genera un cuerpo con movimiento vibratorio. Sin embargo, el sonido no implica necesariamente que escuchemos nada.

Cuando hablamos de sonidos que sí podemos oír, nos referimos a ondas sonoras. Con las vibraciones de los cuerpos y los objetos en nuestro entorno, esas ondas sonoras viajan y llegan hasta nuestros oídos, transformándose en ondas mecánicas. Dicho más simple, podemos oír

porque nuestros tímpanos recogen esa vibración en forma de ondas y envían información a nuestro cerebro. (Educaciencia, 2016)

FIGURA 120 ONDAS DEL SONIDO.

3.3.8 Propagación del sonido

Para poder oír el estridente ruido de los cláxones de los coches o el agradable ruido de una melodía de piano, el sonido tiene que propagarse: sus ondas tienen que "viajar".

El sonido viaja siempre por medios elásticos; es decir, un medio en el cual las moléculas se mueven alrededor de su posición de equilibrio y trasladan la vibración a las adyacentes. Así, el sonido puede propagarse por medios sólidos, como una pared; líquidos, cómo el agua; o gaseosos, como el aire. En esta propagación, tiene lugar un transporte de energía, pero no de materia. A diferencia de la luz y el resto de las ondas electromagnéticas, el sonido no se propaga en el vacío. (Educaciencia, 2016)

¿Viajará igual de bien el sonido en los distintos medios? No. Según el medio en el que viaje, su velocidad será distinta. Al contrario de lo que podríamos pensar, en los materiales sólidos el sonido se propaga más rápido (a más de 2.500 metros por segundo) que en líquidos como el agua (a unos 1.500 metros por segundo) y mucho más rápido que en el aire (a 340 metros por segundo). (Educaciencia, 2016)

3.3.9 ¿Cómo son los sonidos?

Los sonidos agudos son los que tienen mayor frecuencia, entre 2.000 y 20.000 Hertzios. Los sonidos graves, por su parte, son los que tienen menor frecuencia, entre 20 y 250 Hertzios. Los sonidos

medios, como son las voces humanas, por ejemplo, oscilan entre los 250 y los 2.000 Hertzios. (Educaciencia, 2016)

En general, podemos diferenciar unos sonidos de otros por tres características: el timbre, la intensidad y la duración. El timbre sería el tipo de sonido, si es un pájaro que canta, el motor de un tractor o las palmadas de un aplauso. La intensidad sería la energía que contiene un sonido y nos indica si este es fuerte o débil. Mientras que la duración hace referencia al tiempo que dura el sonido. (Educaciencia, 2016)

3.3.10 ¿Qué es Bfxr?

Bfxr es una elaboración del glorioso Sfxr, el programa elegido por muchas personas que buscan crear efectos de sonido para juegos de computadora. (Increpare, 2020)

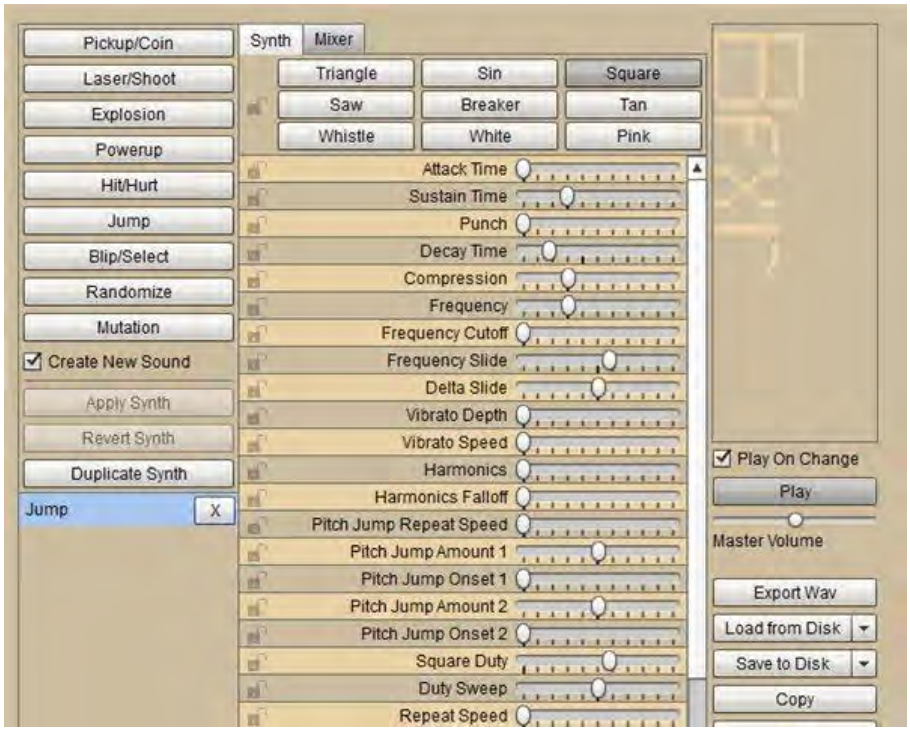

FIGURA 121 BFXR.

Nosotros usaremos el programa llamado Bfxr, para generar sonidos aleatorios usando las funciones matemáticas seno, coseno, tangente entre otras.

Tiene diversos sonidos pre generados que serán de gran utilidad por ejemplo el sonido de recoger un ítem.

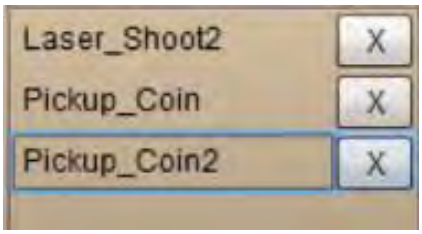

FIGURA 122 SONIDO RECOGER UNA MONEDA

Una vez que tenemos un sonido que nos gusta, lo podemos exportar para posteriormente usarlo en nuestro programa de Godot, entonces a manera de ejemplo yo exporte el sonido de recoger una moneda dando clic en la parte que dice export WAV.

Ahora para reproducir sonido dentro de nuestro Godot, iremos a nuestro nodo de jugador y dentro de nuestro nodo de jugador, vamos a añadir un nodo que se llama audiostreamplayer.

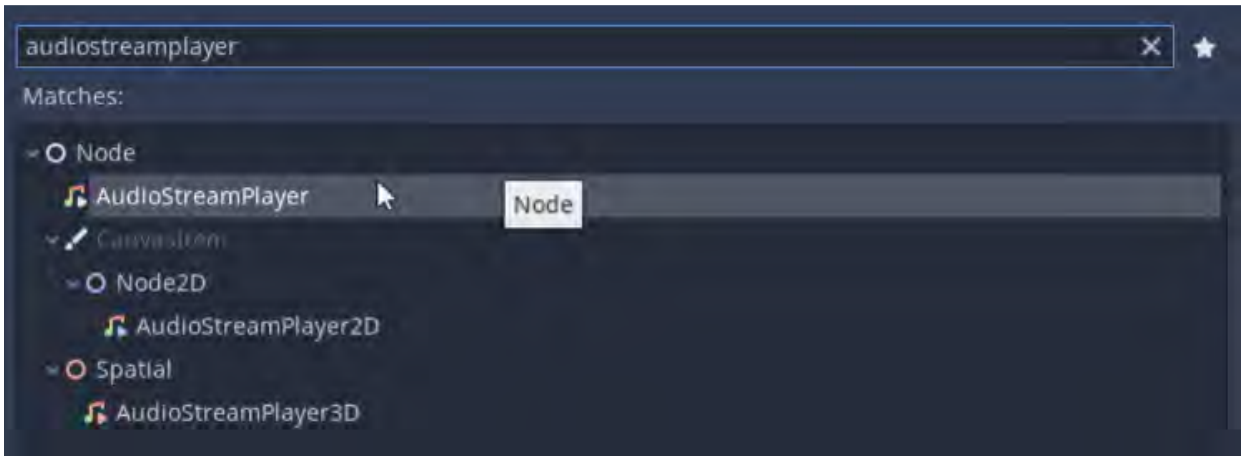

FIGURA 123 NODO AUDIOSTREAMPLAYER.

La diferencia entre audiostreamplayer normal y audiostreamplayer 2d o 3d es la posición la versión del nodo normal, solo va a reproducir un sonido, la versión del nodo 2d o 3d, va a colocar ese sonido en una posición y dependiendo de que tan cerca este el jugador de ese sonido es el volumen del audio que se va a reproducir.

Creamos el nuevo nodo de audio y le cambiamos el nombre, de tal forma que queda de la siguiente manera.

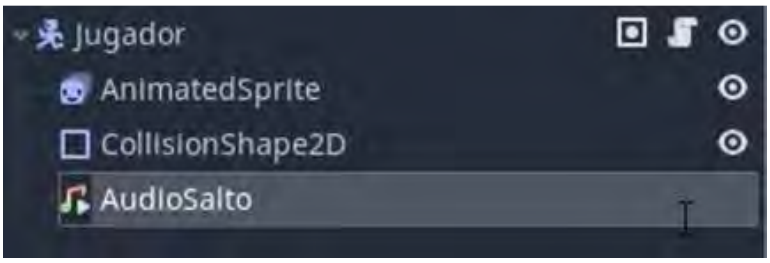

FIGURA 124 NODO AUDIO SALTO.

Seleccionamos nuestro audio de salto en el filesystem y lo arrastramos hasta nuestro inspector, en el apartado que dice Stream.

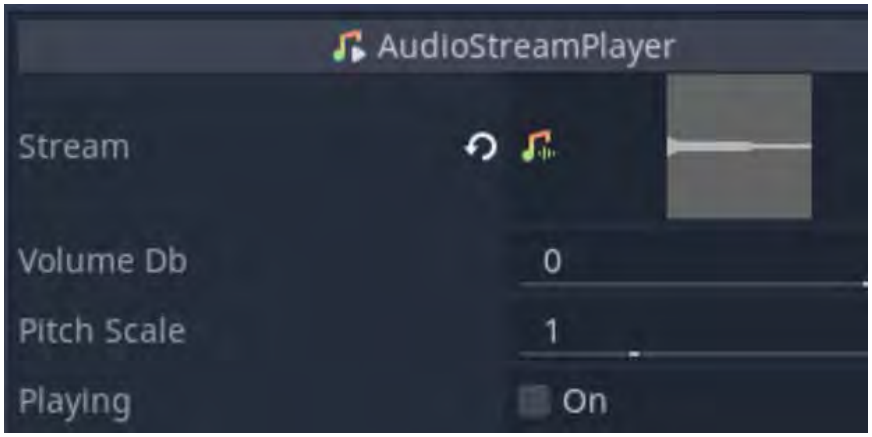

FIGURA 125 AUDIO STREAM.

Ahora en nuestro script de jugador, en el apartado de la variable salto, vamos a modificar el código para agregarle ese sonido al momento en el cual, nuestro jugador presione la tecla de salto, agregamos el siguiente código $AudioSalto.play(), de esta manera estamos accediendo a nuestro nodo de audio de salto y Godot al momento de usar la tecla salto reproducirá el sonido.

```
# Si justo acabamos de presionar SALTAR y ademas "Jugador" esta en el SUELO
if Input.is_action_just_pressed("saltar") and is_on_floor():
    # Impulsamos nuestro "Movimiento" con "Fuerza Salto"
    # Establecemos que nuestro movimiento en el eje Y va a ser igual a "Fuerza
    # Le ponemos el signo de menos "-" para movernos hacia ARRIBA en el eje Y
    movimiento.y = -fuerza_salto
    $AudioSalto.play()
```

FIGURA 126 CÓDIGO AUDIO SALTO.

Repetiremos este proceso para todos los audios en general que usemos para nuestro videojuego, en este caso le pondré audio a los picos, los trampolines, las frutas y las animaciones de muerte, quedando de la siguiente manera.

Este es el código de audio del trampolín.

```
1  extends Node2D
2
3
4
5  func _on_Area2D_body_entered(body):
6      if body.is_in_group("JUGADOR"):
7          body.trampolin()
8          $Audio.play()
9
```

FIGURA 127 CÓDIGO DE AUDIO DE TRAMPOLÍN.

Este es el código de sonido de la fruta, tuve que agregarle una variable de if, para evitar que el sonido se siguiera reproduciendo en el área de colisión de la fruta una vez que la fruta ya no esté en esa área.

FIGURA 128 CÓDIGO DE SONIDO DE LA FRUTA.

Este es el código de sonido de la muerte del jugador.

```
84
85  # Funcion basica de muerte: Solo reinicia el juego
86  func muerte():
87      $AudioMuerte.play()
88
89
90
91  # Funcion llamada por Trampolin.tscn
92  func trampolin():
93      # Impulsamos nuestro "Movimiento" con la variable "Impulso Trampolin"
94      # Establecemos que nuestro movimiento en el eje Y va a ser igual a menos "Impulso T
95      # Le ponemos el signo de menos "-" para moverno hacia ARRIBA en el eje Y
96      movimiento.y = -impulso_trampolin
97
98
99
100
101
102
103 func _on_AudioMuerte_finished():
104     get_tree().reload_current_scene()
105
```

FIGURA 129 CÓDIGO AUDIO DE MUERTE.

En este código tuve que agregar en el numero 103 la función de finished para que en el momento en el que el jugador muera, el sonido se reproduzca y luego la escena se repita, o el cuyo caso el mismo juego se reinicie.

Ahora para mejorar nuestra escena de muerte del jugador vamos a realizar lo siguiente, ¿tuve que crear una variable llamada esta?, y darle el valor de falsé, esto para representar que al principio nuestro jugador no está muerto, luego a continuación tuve que crear un condicional, tal y como se muestra en la imagen.

```
# Funcion basica de muerte: Solo reinicia el juego
func muerte():
    if esta_muerto == false:
        $AudioMuerte.play()
        esta_muerto = true
```

FIGURA 130 CÓDIGO DE MUERTE DE JUGADOR MEJORADO.

Este código representa lo siguiente, el condicional if esta_muerto == false, representa que el cuándo el jugador muera, se va a reproducir el audio de la muerte del jugador, y esta_muerto = true esto se asegura que solo se reproduzca una vez.

Por último, tuve que agregar el siguiente condicional en todo mis variables de movimiento de mi jugador, porque noté que se seguía moviendo en los picos después de muerto.

```
# Si el jugador esta presionando la DERECHA
if Input.is_action_pressed("derecha") and esta_muerto == false:
    # Nuestra velocidad de movimiento va a ser igual a "Velocidad" (120 pixeles por
    # Al ponerle a "Movimiento.X" un numero positivo ("Velocidad") nos moveremos ha
    movimiento.x = velocidad
    # "Animacion.Scale.X" al ser "1" nuestra animacion va a mirar a la DERECHA
    animacion.scale.x = 1
    # "Animacion.Play()" nos permite reproducir una animacion: "Run"
    animacion.play("run")
```

FIGURA 131 CÓDIGO DE MUERTE DE MOVIMIENTO.

Este condicional and esta_muerto == false, representa que el jugador ya no se pueda mover después de muerto.

Por último, agregue unas cuantas mejoras a la muerte del personaje, hice que al momento de colisionar con los picos de un pequeño salto hacia arriba y luego caiga fuera del mapa, el código quedo de la siguiente manera.

```
func muerte():
    if esta_muerto == false:
        $AudioMuerte.play()
        movimiento.y = -impulso_muerte
        $CollisionShape2D.free()
        esta_muerto = true
```

FIGURA 132 CÓDIGO DE MUERTE DEL JUGADOR FINAL.

3.4 Técnicas para mejorar la calidad de un videojuego en Godot.

En este apartado de la tesis daré a conocer diversas técnicas que podemos usar en nuestro Godot para mejorar la calidad de un videojuego y hacerlo de una manera más profesional.

3.4.1 Movimiento avanzado memoria de salto.

Este es el término utilizado para dar una instrucción a una entrada (en este caso, presionar un botón) para mantener en la memoria hasta que pueda realizar dicha instrucción. Un buen ejemplo de esto es incluir una pequeña cantidad de tiempo en la que el jugador puede presionar el botón de salto (mientras todavía está en el aire) haciendo que el personaje salte una vez que haya tocado el suelo. (Manthorp, 2019)

FIGURA 133 MEMORIA DE SALTO EJEMPLO.

Esta técnica le da al jugador un pequeño margen de error si el botón de salto se presiona demasiado pronto, lo que hace que cada salto previsto se realice con éxito. Imagine un nivel difícil que contenga una multitud de saltos rápidos para evitar daños, requeriría que el jugador se asegure de que cada entrada de salto se presionó mientras estaba en el suelo, lo que significa que cualquier entrada temprana (incluso un solo píxel desde el suelo) no lo haría. estar registrado. Es una característica pequeña y oculta que generalmente pasa desapercibida hasta que ya no existe. (Manthorp, 2019)

3.4.2 Tiempo de coyote (tolerancia de borde)

Coyote Time es otra característica oculta que le brinda al jugador una "red de seguridad" justa para una entrada de salto que de otro modo se perdería justo después de abandonar una repisa o plataforma. Esto es efectivamente lo contrario de una entrada de salto amortiguada, ya que la acción se ejecutará al presionar el botón más tarde de lo previsto en lugar de antes. (Manthorp, 2019)

FIGURA 134 TIEMPO DEL COYOTE EJEMPLO.

3.4.3 Memoria de salto en Godot.

A continuación, vamos a programar la primera técnica, la memoria de salto en nuestro Godot, para hacer esto primero vamos a abrir nuestra escena de jugador, y vamos a añadir un nuevo nodo, el cual es “Timer”.

FIGURA 135 NODO TIMER.

El cual es un reloj o un contador, funciona de la siguiente manera nosotros le damos un número, ejemplo que cuente hasta 10 y después de hacer el conteo, este nodo manda una señal a nuestro código, lo primero que haremos será cambiarle el nombre a nuestro nodo, por el de “MemoriaSalto”, ahora en nuestro apartado del inspector veremos que tenemos nuevas propiedades en el apartado wait time pondremos lo siguiente.

FIGURA 136 CONFIGURACIÓN DEL TIMER.

Aquí tenemos los siguientes datos, el Wait Time nos indica la cantidad de tiempo que va a contar o esperar después de mandar una señal, en este caso le pondremos 0.1, luego tenemos activada la propiedad de one shot, el one shot si no se activa el contador estará repitiendo el proceso de esperar, la cantidad de tiempo puesta en el wait time y luego mandar una señal, al activar el one shot este proceso solo se va a repetir una vez.

Ahora vamos a nuestro código de jugador y buscamos nuestro apartado del código en la línea 71, en el apartado de salto y pondremos lo siguiente.

Primero vamos a dividir nuestro código en 2 condicionales, dividendo nuestro apartado de que si está vivo, ejecutara el código normal, pero si el jugador está en el suelo el código hará que este mismo salte, también agregaremos un else, para implementar que si no está en el suelto, más bien si nuestro jugador se encuentra en el aire,y el botón de salto fue presionado entonces, vamos a iniciar nuestro timer.

Para hacer esto vamos a acceder a nuestro nodo de memoria el cual se llama “MemoriaSalto”, y le vamos a agregar el “.start”, con esta función vamos a iniciar nuestro contador, también vamos a

agregar el comando "or", accediendo una vez mas a nuestro nodo de "MemoriaSalto" y le vamos a preguntar si esta detenido usando la función, "is_stopped" y la vamos a dar el valor de igual a falso, para que de esta manera nuestro timer este activo, nuestro código quedaría de la siguiente manera.

```
# Si justo acabamos de presionar SALTAR y ademas "Jugador" esta en el SUELO
if Input.is_action_just_pressed("saltar") or $MemoriaSalto.is_stopped() == false and esta_muerto == false:
    if is_on_floor():
        # Impulsamos nuestro "Movimiento" con "Fuerza Salto"
        # Establecemos que nuestro movimiento en el eje Y va a ser igual a "Fuerza Salto"
        # Le ponemos el signo de menos "-" para movernos hacia ARRIBA en el eje Y
        movimiento.y = -fuerza_salto
        $AudioSalto.play()
    elif $MemoriaSalto.is_stopped():
        $MemoriaSalto.start()
```

FIGURA 137 CÓDIGO DE MEMORIA DE SALTO.

Al final después de hacer pruebas, decidí poner el Wait time en 0.14 por cuestiones de fluides en el salto, además en este código decidí agregar el condicional elif en vez del else.

Antes de continuar con la siguiente técnica, decidí mejorar un poco el código de jugador que tenemos hasta el momento de la siguiente manera, el código relacionado al movimiento del jugador y animaciones quedo de la siguiente forma.

```
func movimiento_de_jugador():
    var direccion = 0
    if Input.is_action_pressed("derecha"):
        direccion = 1
    elif Input.is_action_pressed("izquierda"):
        direccion = -1


    if direccion != 0:
        movimiento.x = velocidad * direccion
        animacion.scale.x = direccion
        animacion.play("run")
    else:
        movimiento.x = 0
        animacion.play("idle")
```

FIGURA 138 MOVIMIENTO DEL JUGADOR MEJORADO.

Modifique las direcciones usando el sistema cartesiano dando valor de 1 ala derecha y -1 a la izquierda, usando la variable de dirección, además usando el condicional elif, hice que el movimiento sea uno u otro, en el apartado de animación use el condicional else, para que la animación pase de correr si se están presionando las teclas a que se quede estático sino lo están haciendo.

También decidí agregar variables a mi salto de jugador y al código relacionado a la gravedad que se agrega dentro del juego quedando de la siguiente forma.

```
func salto_de_jugador():
    if Input.is_action_just_pressed("saltar") or $MemoriaSalto.is_stopped() == false:
        if is_on_floor():
            movimiento.y = -fuerza_salto
            $AudioSalto.play()
        elif $MemoriaSalto.is_stopped():
            $MemoriaSalto.start()


func agregar_gravedad(delta):
    movimiento.y += gravedad * delta
```

FIGURA 139 SALTO DEL JUGADOR Y GRAVEDAD MEJORADO.

También me di cuenta de un pequeño error a la hora de que el personaje muere, se puede seguir moviendo a la izquierda, para corregir este error simplemente copié el código de muerte del personaje, con el código de movimiento del personaje, quedando de la siguiente forma:

```
func muerte():
    if esta_muerto == false:
        $AudioMuerte.play()
        movimiento.x = 0
        movimiento.y = -impulso_muerte
        $CollisionShape2D.free()
        esta_muerto = true
```

FIGURA 140 CÓDIGO DE MUERTE MEJORADO.

3.4.5 Salto del coyote en Godot.

A continuación vamos a programar la técnica del salto del coyote en nuestro Godot, para hacer esto vamos a crear un nuevo nodo, este siendo un timer al igual que el anterior, y le vamos a cambiar el nombre a "MemoriaSuelo".

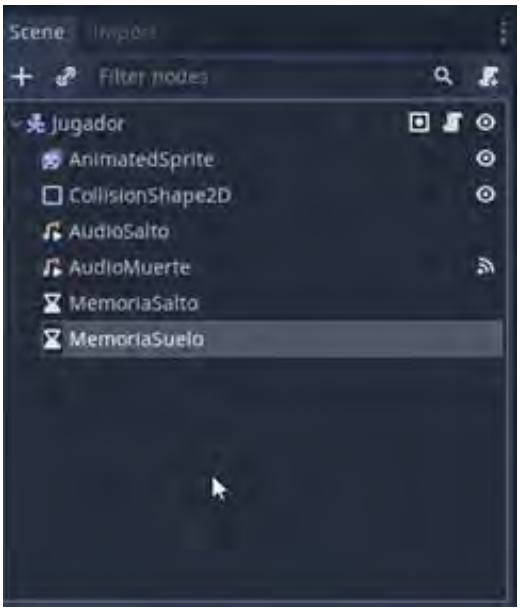

FIGURA 141 NODO MEMORIASUELO.

Al igual que con el otro timer, nosotros vamos a configurar el wait time a unos 0.18, y también activamos el one shot.

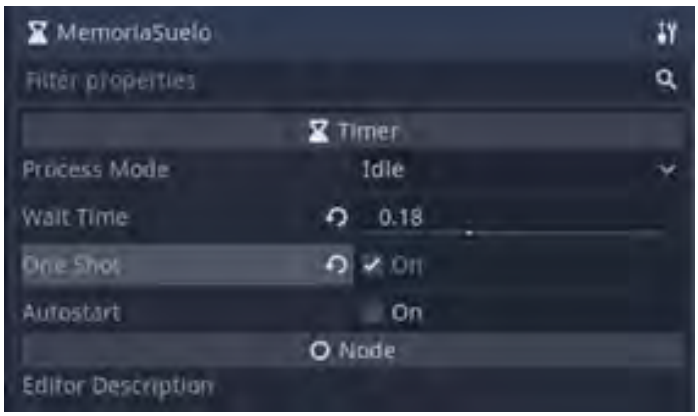

FIGURA 142 CONFIGURACIÓN MEMORIA SUELO.

Ahora para configurar el salto del coyote nosotros vamos a abrir nuestro código de jugador y en el apartado de físicas vamos a poner, con el condicional if, con la variable is_on_floor_, luego accedemos a nuestro nodo "MemoriaSuelo", y le agregamos la variable start, también en nuestro código de salto de jugador llamamos a nuestro nodo "MemoriaSuelo", teniendo el código de la siguiente manera.

FIGURA 143 LLAMANDO AL NODO MEMORIA SUELO.

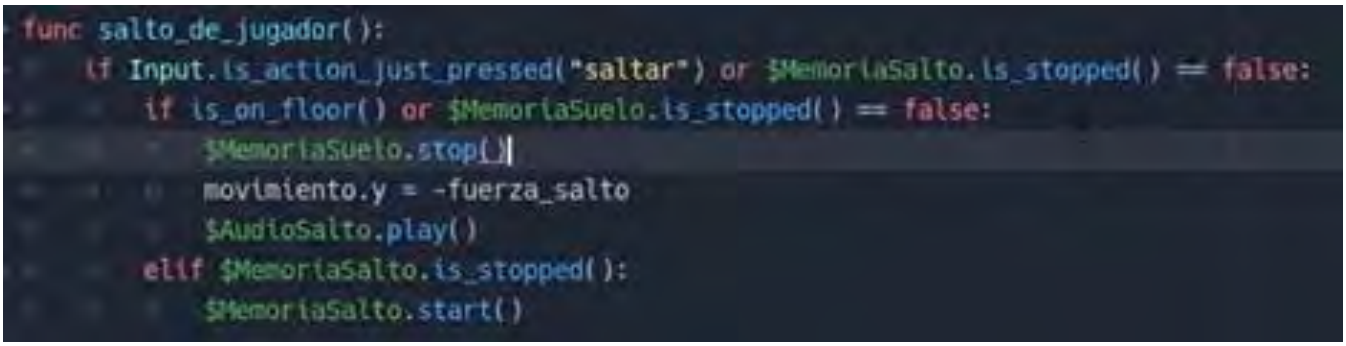

FIGURA 144 CÓDIGO SALTO DEL COYOTE.

De esta forma primero agrego un or y llamo al nodo MemoriaSuelo y le agrego el valor de falso para que este funcione, también agrego la variable stop, en este caso deteniendo el salto del coyote.

Estas técnicas vistas en este apartado le darán un mayor control al jugador de su personaje a la hora de realizar los saltos necesarios, además de que hace que el juego se sienta de una manera más fluida.

3.4.6 Agregando un fondo a nuestro juego utilizando Godot.

A continuación, vamos a crear un fondo, para nuestro juego, primero vamos a ir a nuestro nodo de nivel, y vamos a añadir un nuevo nodo llamado "Canvaslayer".

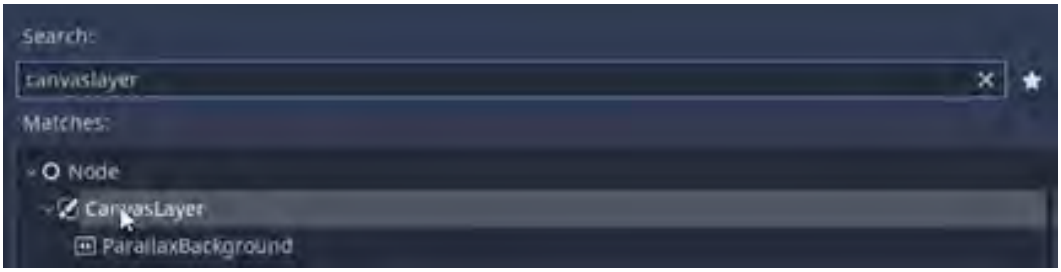

FIGURA 145 NODO CANVASLAYER.

De nuestro lado del inspector vamos a agregarle un valor a la layer de -1 para que Godot detecte que es un fondo.

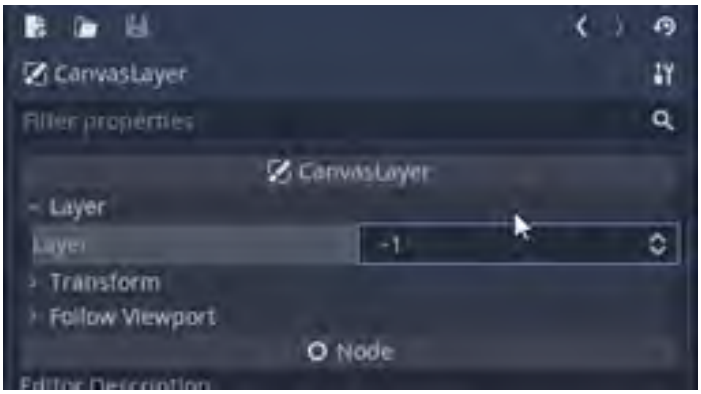

FIGURA 146 AGREGANDO VALORES AL CANVASLAYER.

Ahora vamos a agregar un nuevo nodo llamado "textureRect" dentro de la capa de Canvaslayer.

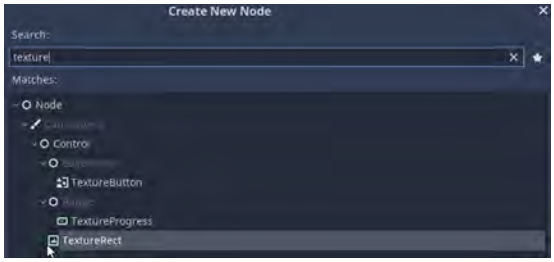

FIGURA 147 NODO TEXTURERECT

Este nuevo nodo nos va a permitir abarcar toda el área de la cámara, para hacer esto vamos a la parte de arriba de nuestro Godot, usando la herramienta layout y damos clic en full rect de esta forma abarcara toda el área de la cámara de nuestro juego.

FIGURA 148 FULL RECT.

Ahora vamos a seleccionar el fondo que nosotros queramos de nuestros Assets.

FIGURA 149 SELECCIONANDO UN FONDO.

Ya que tenemos un fondo seleccionado lo arrastramos a nuestro inspector y a continuación y seleccionamos la casilla que dice expand, para expandir nuestro fondo alrededor de toda la cámara

de nuestro juego y también,en donde dice Strech Mode, vamos a poner tile, para convertir nuestro fondo en un tile.

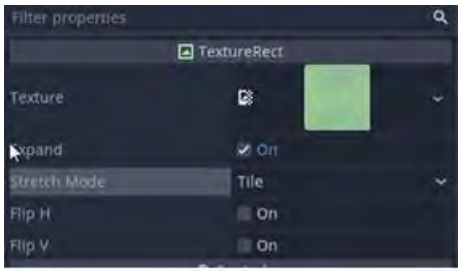

FIGURA 150 CREANDO EL FONDO.

y con esto ya tenemos un fondo listo en nuestro juego.

3.4.7 Salto dependiendo de cuanto presionemos el botón con Godot.

Ahora veremos la última técnica para mejorar nuestros saltos de nuestro juego para hacer esto vamos a nuestro código de jugador, en nuestro código de jugador vamos a ir a la función de salto del jugador, una vez dentro vamos a agregar el condicional if y la variable is_action_just_released, esta variable significa que si el jugador, acaba de soltar la tecla de saltar y luego agregamos la condicional and, y le agregamos movimiento en el eje y, que sea menor a 0,y por ultimo multiplicamos nuestro movimiento y por 0.6.

```
58 - func salto_de_jugador():
59 -     if Input.is_action_just_released("saltar") and movimiento.y < 0:
60           movimiento.y *= 0.6
```

FIGURA 151 SALTO MANTENIDO.

3.4.8 Pasando al siguiente nivel de nuestro juego en Godot.

Para pasar de un nivel a otro vamos a usar un singletone el cual es lo siguiente:

El Singletone es un patrón de diseño creacional que le permite garantizar que una clase tenga solo una instancia, al tiempo que proporciona un punto de acceso global a esta instancia.

Es una herramienta útil para resolver el caso en el que necesita almacenar información persistente entre escenas. En nuestro caso es posible reutilizar la misma escena o clase para múltiples singletons mientras posean nombres distintos. (Manzur, s.f.)

Usando este concepto, puede crear objetos que:

- Siempre están cargados, no importa qué escena esté en ejecución.
- Puedes almacenar variables globales como la información del jugador.
- Puedes manejar el cambio de escenas y las transiciones entre escenas.
- *Actúa* como un singleton, ya que GDScript no soporta variables globales por diseño.

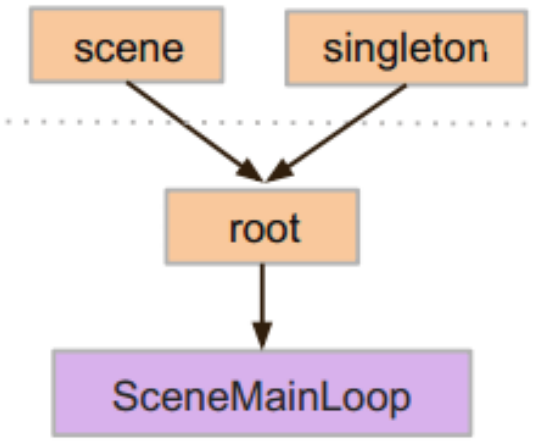

FIGURA 152 EJEMPLO DE SINGLETONE

Ahora que sabemos lo que es un singleton vamos a crear uno en Godot, lo primero que hay que hacer es crear un script nuevo llamado "Singleton" y una nueva carpeta llamada administrador nivel.

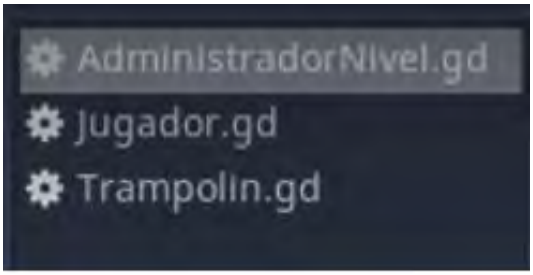

FIGURA 153 ADMINISTRADOR DE NIVEL.

Ahora a continuación vamos a hacer que nuestro singletone, se cargue una sola vez para realizar esta acción vamos a la pestaña de proyecto, nos dirigimos a la pestaña de configuraciones de nuestro proyecto, y en el apartado donde dice Autoload, en donde dice Path, buscamos el archivo que acabamos de crear.

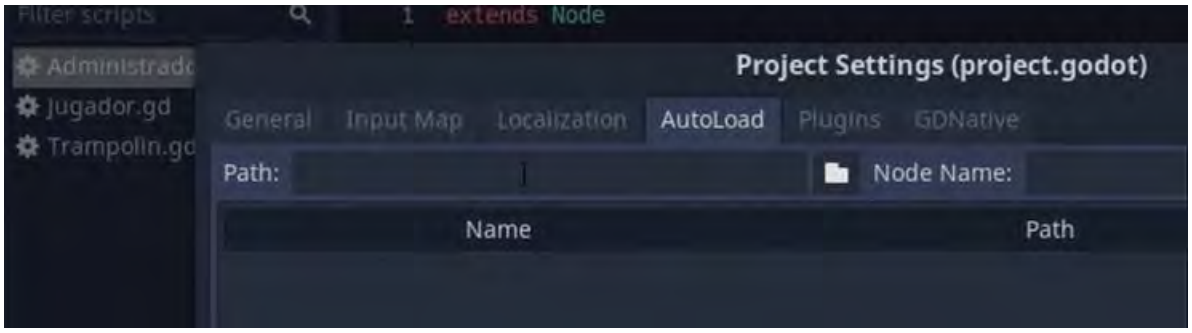

FIGURA 154 AUTOLOAD.

FIGURA 155 CARGANDO ADMINISTRADOR NIVEL EN EL AUTOLOAD.

Lo que acabamos de realizar aquí es una variable global, la cual nos permite acceder desde cualquier nodo con el nombre puesto, en este caso el nombre que se uso, fue el de AdministradorNivel, y nuestra variable global, va a tener automáticamente el script que nosotros pusimos en donde dice path.

Lo que haremos a continuación será crear una carpeta llamada listadeniveles en la cual colocaremos todos nuestros niveles siguientes de nuestro juego.

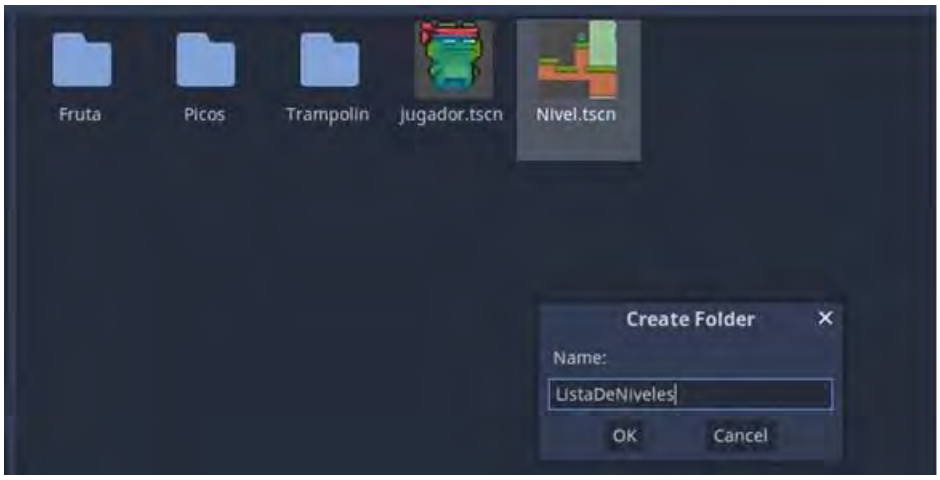

FIGURA 156 LISTA DE NIVELES.

Ahora pasaremos a la ventana de nuestro script creado, el cual se llama administradornivel, y crearemos una variable, llamada lista de niveles y abriremos nuestros caracteres de lista, también usaremos una función nueva llamada "Preload" la cual se encarga de cargar nuestros niveles en forma de código, para que de esta forma ya estén listos, ahora solo queda crear una función que nos permita pasar al siguiente nivel, quedando nuestro código de la siguiente manera.

```
1  extends Node
2
3
4  var lista_de_niveles = [
5      preload("res://src/Nivel/ListaDeNiveles/Nivel01.tscn")
6  ]
7
8
9  func siguiente():
10     var nivel = lista_de_niveles.pop_front()
11     get_tree().change_scene_to(nivel)
```

FIGURA 157 CÓDIGO PARA CAMBIAR DE NIVEL.

La función que se utilizó al final del código nos dice lo siguiente, la función se llama siguiente, la variable creada dentro de la función, toma la lista de niveles y le aplica el método pop front, el cual quita un elemento de la lista y regresa ese primer elemento, y si no tiene elementos regresa una señal de null, es decir que en este caso va a quitar el nivel 1 y lo va a regresar, de esa forma lo podemos almacenar en el nivel, también usamos el get tree para obtener el árbol completo de Godot, es decir su nodo base y usamos la función change scene to, para cambiar la escena de nivel, es decir cambiar a la escena del siguiente nivel.

Ahora vamos a usar nuestra fruta creada para que cuando el jugador tome la fruta, se llame a nuestra variable del siguiente nivel, para hacer esto vamos al script de nuestra fruta y vamos a usar nuestra variable global, creada anteriormente llamada administradornivel, y llamaremos a nuestra función de siguiente nivel, quedando el código de la siguiente manera.

FIGURA 158 CÓDIGO DE FRUTA SIGUIENTE NIVEL.

Ahora en nuestro nodo de fruta vamos a agregar un timer, y el tiempo de espera será de 2 segundos, también usaremos las señales en este caso el timer tiene una señal llamada timeout.

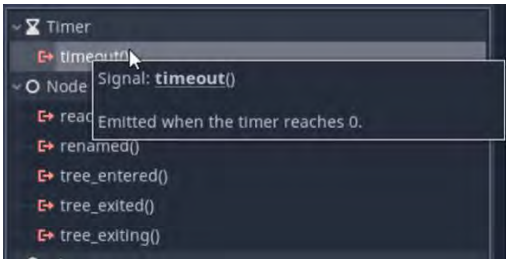

FIGURA 159 SEÑAL TIMEOUT.

La cual es un conteo y se detendrá cuando llegue a los 2 segundos, la vamos a conectar con nuestro script de fruta y Godot nos va a crear la función de timeout, quedando nuestro código de la fruta de la siguiente manera.

```
func _on_Area2D_body_entered(body):
    if fue_recolectada == false and body.is_in_group("JUGADOR"):
        fue_recolectada = true
        animacion.play("recolectada")
        $Audio.play()
        $TiempoEspera.start()
                void start(time_sec: float = -1)

func _on_TiempoEspera_timeout():
    AdministradorNivel.siguiente()
```

FIGURA 160 CÓDIGO FRUTA TIEMPO DE ESPERA.

Lo que se realizó en este código fue iniciar nuestro tiempo de espera con la función start, luego se está agregando la función del tiempo de espera y se está iniciando el contador, el cual se detendrá cuando llegue a 0, y por último se está llamando a la función de siguiente nivel, esto quiere decir que cuando nuestro jugador, tome la fruta, esta sonara, se agregara a su inventario y luego se cambiara de nivel.

Ahora solo queda hacer cuantos niveles queramos tener en nuestro juego, creando diversas escenas de nivel, y al final crearemos una escena de ganaste, la cual se pondrá cuando el jugador termine nuestro juego, la escena final de los créditos que todo juego tiene cuando este es acabado.

Para hacer esto vamos a crear una nueva escena llamada "Ganaste", usaremos un nodo especial que tiene Godot, llamado Label, el cual nos permite agregar texto a nuestro videojuego.

FIGURA 161 NODO LABEL

Usando este nodo, podemos poner el texto que necesitemos y centrándolo a nuestra pantalla del juego quedaría de la siguiente forma.

FIGURA 162 PANTALLA DE VICTORIA.

Ahora vamos al script de nuestra lista de niveles y vamos a agregar nuestra pantalla de victoria al igual que agregamos los demás niveles.

```
var lista_de_niveles = [
    preload("res://src/Nivel/ListaDeNiveles/Nivel01.tscn"),
    preload("res://src/Nivel/Ganaste.tscn"),
]
```

FIGURA 163 CÓDIGO DE VICTORIA.

3.4.9 Como exportar nuestro videojuego

Con el videojuego ya creado lo que realizaremos a continuación será la exportación de nuestro juego finalizado para realizar esto primero debemos, de ir al apartado que dice Project y dentro de ese apartado le vamos a dar clic en donde dice export.

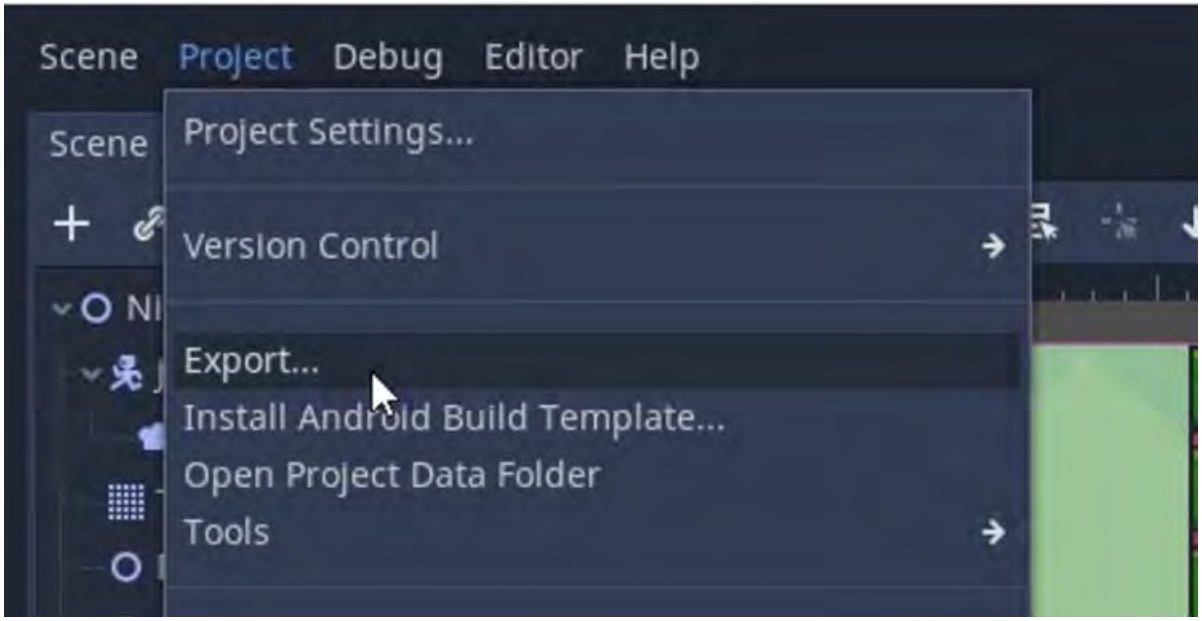

FIGURA 164 APARTADO EXPORT.

Al hacer clic en el apartado de export nos saldrá las siguientes opciones mostrándonos los presets y si le damos en el apartado de add, tendremos las siguientes opciones en este caso exportaremos nuestro juego a Windows.

FIGURA 165 EXPORTANDO NUESTRO JUEGO EN WINDOWS.

Al hacer esto nos aparecerán las siguientes opciones de importación, de las cuales iremos hasta la parte de abajo y el mismo Godot nos dirá que nos faltan los templates para exportar nuestro videojuego, los templates son archivos ya creados por Godot, los cuales se funcionan con nuestro videojuego para que Godot pueda exportar nuestro juego y hacer las conversiones necesarias para que se puede jugar dentro de Windows.

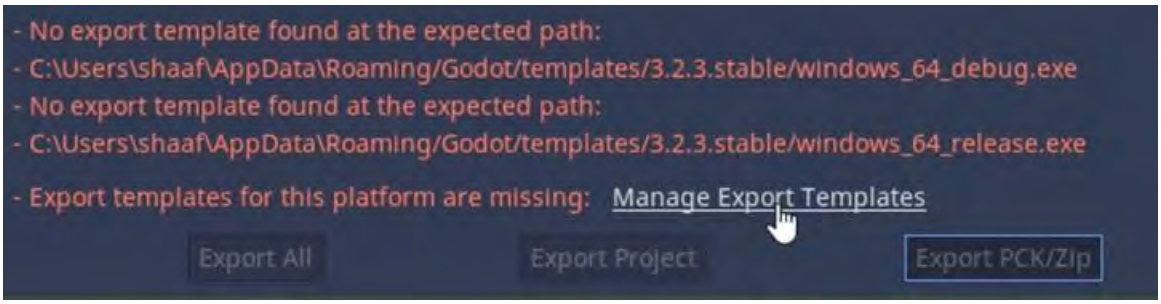

FIGURA 166 TEMPLATES DE GODOT.

Para descargar los templates simplemente le damos clic en Manage Export Templates y descargamos los templates faltantes, una vez realizada esta acción, volvemos a Project a export y veremos que ya tenemos el preset de Windows.

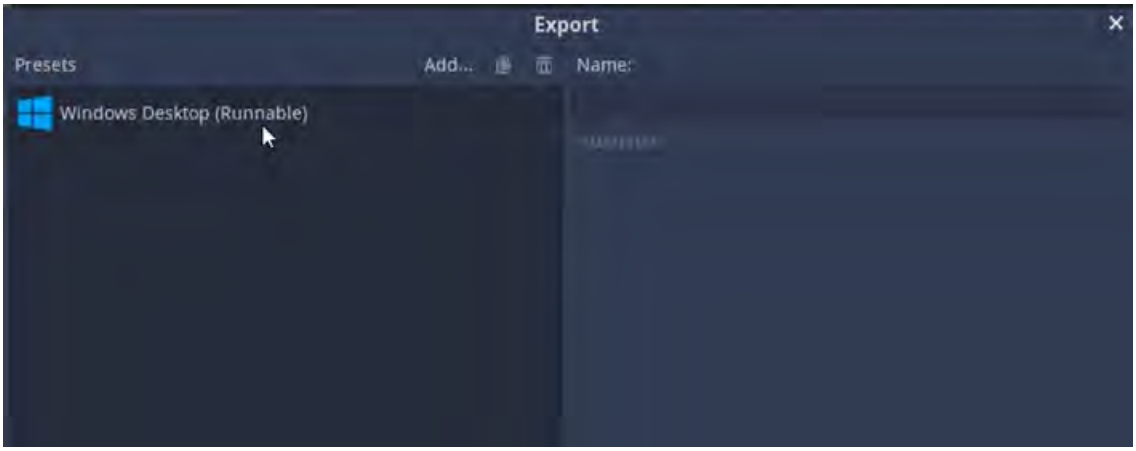

FIGURA 167 PRESET WINDOWS.

Ahora ya podremos exportar nuestro juego, simplemente dando clic en Export Project.

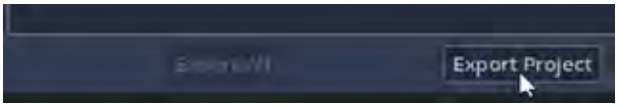

FIGURA 168 EXPORTANDO NUESTRO JUEGO.

Ahora simplemente vamos a seleccionar la carpeta en la cual nosotros queramos tener nuestro videojuego, esperamos el proceso de conversión y exportación y una vez terminado, tendremos lo siguiente un archivo exe, es decir un archivo ejecutable de nuestro juego.

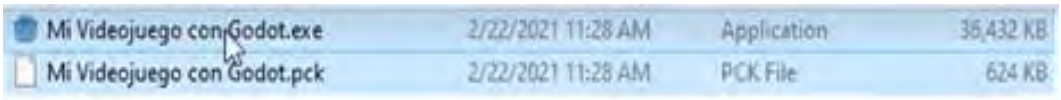

FIGURA 169 ARCHIVO EJECUTABLE.

Listo ahora tenemos nuestro juego completo y listo para ser jugado, ahora solo quedaría subirlo a diversas páginas de juegos indie, subirlo a una playstore o lo que el cliente o tu mismo quieras y requieras en ese momento.

3.5 Codificación

3.51 Código de la fruta.

```
1.  extends Node2D
2.
3.
4.  var fue_recolectada = false
5.
6.
7.  onready var animacion = $AnimatedSprite
8.
9.  func _on_Area2D_body_entered(body):
10.     if fue_recolectada == false and body.is_in_group("JUGADOR"):
11.         fue_recolectada = true
12.         animacion.play("recolectada")
13.         $Audio.play()
14.         $TiempoEspera.start()
15.
16.
17. func _on_TiempoEspera_timeout():
18.     AdministradorNivel.siguiente()
```

3.5.2 Código Altura del Salto.

```
1.  extends KinematicBody2D
2.
3.
4.  var velocidad = 185
5.
6.  var gravedad = 1000
7.
8.  var fuerza_salto = 390
9.
10. var impulso_muerte = 200
11. var impulso_trampolin = 500
12. var limite_caida = 750
13.
14. var movimiento = Vector2(0, 0)
15.
16.
17. var esta_muerto = false
18.
19.
20. onready var animacion = $AnimatedSprite
21.
22.
23.
24. func _physics_process(delta):
25.     if is_on_floor():
26.         $MemoriaSuelo.start()
```

```
27.
28.     if esta_muerto == false:
29.         movimiento_de_jugador()
30.         salto_de_jugador()
31.
32.     agregar_gravedad(delta)
33.
34.     movimiento = move_and_slide(movimiento, Vector2.UP)
35.
36.     if position.y > limite_caida:
37.         muerte()
38.
39.
40. func movimiento_de_jugador():
41.     var direccion = 0
42.
43.     if Input.is_action_pressed("derecha"):
44.         direccion = 1
45.     elif Input.is_action_pressed("izquierda"):
46.         direccion = -1
47.
48.
49.     if direccion != 0:
50.         movimiento.x = velocidad * direccion
51.         animacion.scale.x = direccion
52.         animacion.play("run")
53.     else:
54.         movimiento.x = 0
55.         animacion.play("idle")
56.
57.
58. func salto_de_jugador():
59.     if Input.is_action_just_released("saltar") and movimiento.y < 0:
60.         movimiento.y *= 0.6
61.
62.     if Input.is_action_just_pressed("saltar") or $MemoriaSalto.is_stopped() == false:
63.         if is_on_floor() or $MemoriaSuelo.is_stopped() == false:
64.             $MemoriaSuelo.stop()
65.             movimiento.y = -fuerza_salto
66.             $AudioSalto.play()
67.         elif $MemoriaSalto.is_stopped():
68.             $MemoriaSalto.start()
69.
70.
71. func agregar_gravedad(delta):
72.     movimiento.y += gravedad * delta
73.
74.
75. func muerte():
76.     if esta_muerto == false:
77.         $AudioMuerte.play()
78.         movimiento.x = 0
79.         movimiento.y = -impulso_muerte
80.         $CollisionShape2D.free()
81.         esta_muerto = true
82.
```

```
83.
84. func trampolin():
85.     movimiento.y = -impulso_trampolin
86.
87.
88. func _on_AudioMuerte_finished():
89.     get_tree().reload_current_scene()
```

3.5.3 Código salto del coyote.

```
1.  extends KinematicBody2D
2.
3.
4.  var velocidad = 185
5.
6.  var gravedad = 1000
7.
8.  var fuerza_salto = 390
9.
10. var impulso_muerte = 200
11. var impulso_trampolin = 500
12. var limite_caida = 750
13.
14. var movimiento = Vector2(0, 0)
15.
16.
17. var esta_muerto = false
18.
19.
20. onready var animacion = $AnimatedSprite
21.
22. func _physics_process(delta):
23.     if is_on_floor():
24.         $MemoriaSuelo.start()
25.
26.     if esta_muerto == false:
27.         movimiento_de_jugador()
28.         salto_de_jugador()
29.
30.     agregar_gravedad(delta)
31.
32.     movimiento = move_and_slide(movimiento, Vector2.UP)
33.
34.     if position.y > limite_caida:
35.         muerte()
36.
37.
38. func movimiento_de_jugador():
39.     var direccion = 0
40.
41.     if Input.is_action_pressed("derecha"):
42.         direccion = 1
43.     elif Input.is_action_pressed("izquierda"):
```

```
        direccion = -1


    if direccion != 0:
        movimiento.x = velocidad * direccion
        animacion.scale.x = direccion
        animacion.play("run")
    else:
        movimiento.x = 0
        animacion.play("idle")


func salto_de_jugador():
    if Input.is_action_just_pressed("saltar") or $MemoriaSalto.is_stopped() == false:
        if is_on_floor() or $MemoriaSuelo.is_stopped() == false:
            $MemoriaSuelo.stop()
            movimiento.y = -fuerza_salto
            $AudioSalto.play()
        elif $MemoriaSalto.is_stopped():
            $MemoriaSalto.start()


func agregar_gravedad(delta):
    movimiento.y += gravedad * delta


func muerte():
    if esta_muerto == false:
        $AudioMuerte.play()
        movimiento.x = 0
        movimiento.y = -impulso_muerte
        $CollisionShape2D.free()
        esta_muerto = true


func trampolin():
    movimiento.y = -impulso_trampolin


func _on_AudioMuerte_finished():
    get_tree().reload_current_scene()
```

3.5.4 Código del Jugador.

```
extends KinematicBody2D


var velocidad = 185

var gravedad = 1000

```

```
8.  var fuerza_salto = 390
9.
10. var impulso_muerte = 200
11. var impulso_trampolin = 500
12. var limite_caida = 750
13.
14. var movimiento = Vector2(0, 0)
15.
16.
17. var esta_muerto = false
18.
19.
20. onready var animacion = $AnimatedSprite
21.
22. func _physics_process(delta):
23.     if esta_muerto == false:
24.         movimiento_de_jugador()
25.         salto_de_jugador()
26.
27.     agregar_gravedad(delta)
28.
29.     movimiento = move_and_slide(movimiento, Vector2.UP)
30.
31.     if position.y > limite_caida:
32.         muerte()
33.
34.
35. func movimiento_de_jugador():
36.     var direccion = 0
37.
38.     if Input.is_action_pressed("derecha"):
39.         direccion = 1
40.     elif Input.is_action_pressed("izquierda"):
41.         direccion = -1
42.
43.
44.     if direccion != 0:
45.         movimiento.x = velocidad * direccion
46.         animacion.scale.x = direccion
47.         animacion.play("run")
48.     else:
49.         movimiento.x = 0
50.         animacion.play("idle")
51.
52.
53. func salto_de_jugador():
54.     if Input.is_action_just_pressed("saltar") or $MemoriaSalto.is_stopped() == false:
55.         if is_on_floor():
56.             movimiento.y = -fuerza_salto
57.             $AudioSalto.play()
58.         elif $MemoriaSalto.is_stopped():
59.             $MemoriaSalto.start()
60.
61.
62. func agregar_gravedad(delta):
63.     movimiento.y += gravedad * delta
```

```
64.
65.
66. func muerte():
67.     if esta_muerto == false:
68.         $AudioMuerte.play()
69.         movimiento.x = 0
70.         movimiento.y = -impulso_muerte
71.         $CollisionShape2D.free()
72.         esta_muerto = true
73.
74.
75. func trampolin():
76.     movimiento.y = -impulso_trampolin
77.
78.
79. func _on_AudioMuerte_finished():
80.     get_tree().reload_current_scene()
81.
82.
```

3.5.5 Código de picos.

```
1.  extends Node2D
2.
3.
4.
5.  func _on_Area2D_body_entered(body):
6.      if body.is_in_group("JUGADOR"):
7.          body.muerte()
```

Capítulo 4 Resultados

En este capítulo veremos los Resultados finales del videojuego y como incluirlo y ponerlo en páginas web y en Android e IOS para que los usuarios puedan descargarlo o en cuyo caso comprarlo.

4.1 Pagina HTML5

Para exportar nuestro videojuego concluido a una página web, dentro de la cual se pueda jugar a través de un navegador primero lo que haremos será ir al apartado de exportación y seleccionar la opción de HTML5.

FIGURA 170 EXPORTACIÓN HTML5.

Una vez exportado tendremos el siguiente archivo.

FIGURA 171 ARCHIVO HTML5.

Ahora lo que tendríamos que hacer seria exportar estos archivos a un servidor.

4.1.1 Página de juegos indie comerciales gratuita

La página comercial a la cual subiremos nuestro juego indie 2D o también llamado expóngame, ya que su finalidad es dar a conocer al equipo de programación indie que trabajo en este proyecto es decir el autor de esta tesis, es decir yo mismo, parte de los objetivos finales es dar a conocer la programación de videojuegos indie y que se dé más atención a los programadores independientes.

La página utilizada se llama itch.io.

4.1.2 Itch.io

Itch.io es una tienda digital de videojuegos para PC y móviles, un marketplace enfocado a obras independientes en el que sus creadores pueden decidir cuánto quieren cobrar por su trabajo y qué porcentaje le dan a la plataforma. También los jugadores podemos escoger cuánto pagar por cada título, con un mínimo establecido que puede llegar ser incluso de cero, con la posibilidad de pagar de más para apoyar a algún estudio que nos guste especialmente. (Delgado, 2021)

Este lugar se define a sí mismo como "una colección de algunas de las creaciones más singulares, interesantes e independientes que encontrarás en la web", un espacio en el que bucear buscando obras experimentales o en el que comprar juegos indies sin DRM. Una tienda digital en la que todo tiene cabida, en la que sus responsables seleccionan semanal y manualmente los juegos más interesantes, y aun así una no muy utilizada por los jugadores, más acostumbrados a comprar en otras páginas más populares. (Delgado, 2021)

El creador de Itch.io, Leaf Corcoran, abrió en 2013 un *marketplace* alternativo para que los desarrolladores independientes pudieran publicar su contenido sin tener que pagar por ello y pudiendo establecer un método de compra de videojuegos basado en el 'paga lo que quieras', lo que permitía a los autores establecer un precio mínimo para sus juegos -que podría ser cero- dejando a los jugadores la decisión de abonar la cantidad que quisieran, incluso más que ese mínimo establecido. (Delgado, 2021)

Desde entonces, Itch.io se ha convertido en una plataforma ideal para adentrarse en el mundillo del desarrollo indie. Actualmente cuenta con más de 140.000 videojuegos para comprar, jugar o

descargar de manera gratuita, desde los proyectos más experimentales hasta las obras independientes que más fama han cosechado en la industria. Cualquiera puede publicar su videojuego en Itch.io, aunque la plataforma, que ya cuenta con un equipo de personas dedicado a su gestión, realiza labores de selección y recomendación de aquellos nuevos proyectos que más interés suscitan. (Delgado, 2021)

FIGURA 172 ITCH.IO.

A lo largo de estos años también ha ido ampliando sus fronteras hacia otros derroteros: en las secciones de Itch.io no sólo podemos encontrar videojuegos, sino también juegos de mesa, para imprimir y jugar en casa, o eventos como *game-jams*, donde los desarrolladores pueden unirse libremente para crear videojuegos sobre un tema concreto durante un tiempo estipulado. También pueden descargarse bandas sonoras, libros, *fanzines*, cómics, herramientas y *assets* para crear tus propios juegos.

Para subir nuestro juego a itch.io, lo único que tenemos que hacer es registrarnos a la pagina utilizando un usuario y contraseña al igual que un correo electrónico.

FIGURA 173 REGISTRO EN ITCH.IO

Una vez creado el perfil simplemente tendremos que darle clic en donde dice upload new Project

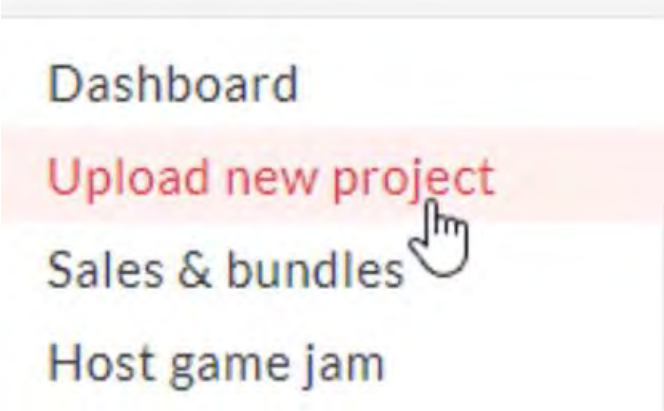

FIGURA 174 UPLOAD NEW PROJECT.

Una vez llenado los datos que te pide itch.io y habido subido el videojuego con su precio de descarga o en cuyo caso de forma gratuita nos quedaría de la siguiente forma.

FIGURA 175 MI VIDEOJUEGO EN ITCH.IO.

Capítulo 5 Conclusiones

Los videojuegos son parte de la cultura moderna, no solo en un ámbito de entretenimiento, sino que también en un ámbito educativo lo cual conlleva a una gran oportunidad de trabajo laboral como lo es un programador de videojuegos. Actualmente estos perfiles profesionales tienen una gran demanda en el mercado, a su vez como queda demostrado en esta tesis; cada vez hay más géneros y subgéneros de videojuegos, como son los juegos tipo indie, o juegos independientes. Esta diversificación en los videojuegos está incrementándose e incrementa el número de usuarios que los prefieren, ya que son creados por programadores independientes.

Para acompañar el proceso de creación de los videojuegos, las herramientas tecnológicas son primordiales, por la evolución constante que experimenta esta industria. La herramienta Godot, que se utilizó en este trabajo, destaca por la facilidad de acceso, el manejo de la programación, creación, diseño de personajes, mapeo, entre otras características, siendo amigable para todo tipo de usuarios. Además, otra de sus ventajas son las actualizaciones frecuentes, con una licencia libre de uso, lo cual permite que los usuarios puedan migrar a un perfil de desarrollador de videojuegos en dos o incluso en tres dimensiones, ya que la plataforma Godot está especializada para todo tipo de géneros y subgéneros de los videojuegos. Gracias a su sistema de programación basado en nodos, las múltiples herramientas que nos brinda, sus facilidades para hacer un mapa o todos los escenarios que nuestro videojuego requiera, así como la interfaz nada complicada e intuitiva, Godot es la herramienta ideal para todos aquellos programadores que se dediquen al ámbito de los videojuegos. (Belli & López, 2008)

En esta tesis se presentaron las diversas fases para la creación de un videojuego, estas fases se incluyen en una metodología de desarrollo, dependiendo del tipo de juego que se requiera realizar. Los videojuegos no se enfocan sólo hacia el entretenimiento, sino que también se han orientado hacia actividades lúdicas para el aprendizaje de niños, personas con capacidades diferentes o adultos mayores, inclusive para fines terapéuticos, ya que el uso adecuado puede evitar la depresión, el estrés, la ansiedad entre otros factores.

En este trabajo de tesis también se abordaron temáticas a través de la búsqueda de información en diversas fuentes independientes, comerciales y de grandes marcas como son: Steam, Epic Games; donde se apoya a los creadores de videojuegos independientes, les permiten publicar sus proyectos para promocionarse como un desarrollador de videojuegos, e ir escalando poco a poco hasta tener un nombre reconocido dentro de la comunidad.

Para finalizar, los videojuegos destacan en diferentes ámbitos dentro de la vida diaria, con fines de entretenimiento, educativos y hasta terapéuticos; ofrecen una gran oportunidad dentro del mercado laboral, usan diferentes metodologías dentro de su desarrollo dependiendo del tipo de juego que se desea realizar, el tipo de jugador que se desea alcanzar, y a su vez el tipo de historia o escenario que se desea plantear. Los videojuegos tienen una larga historia de desarrollo, desde las actualizaciones de las consolas de videojuegos, hasta las herramientas de programación para su creación; forman parte de la cultura moderna y se van promocionando en diversas plataformas digitales y/o presenciales como grandes convenciones o a través de grupos sociales. La capacidad para lograr desarrollar videojuegos es uno de los perfiles que tiene demanda en el ámbito profesional, y en este trabajo de tesis se demostró cómo es el proceso de creación a través de la herramienta Godot.

Bibliografía

Altice, N. (2017). *I Am Error.* MIT Press.

Ángel Gutiérrez, D. Z. (2007). *Freeware y shareware.* Creaciones Copyright.

Ants. (2019). *Ants Info*. Obtenido de https://ants.inf.um.es/staff/jlaguna/tp/tutoriales/colisiones/index.html

Barco, D. I. (2017). La historia a través de los videojuegos. *Iber: Didáctica de las ciencias sociales, geografía e historia*, 30-36.

Belli, S., & López Raventós, C. (otoño, 2008). *Breve historia de los videojuegos.* Barcelona: Athenea Digital.

Bradfield, C. (2018). *Godot Engine Game Development Projects: Build five cross-platform 2D and 3D.* Birmingham: Packt.

Bramble, R. (enero de 2023). *EVERYTHING YOU NEED TO KNOW BEFORE YOU MAKE A 2D GAME*. Obtenido de https://gamemaker.io/es/blog/slash-how-to-make-a-2d-game

Cervera, I. (9 de Abril de 2019). *Geekno*. Obtenido de https://www.geekno.com/glosario/asset

Contreras, R. S. (2012). *Videojuegos : conceptos, historia y su potencial como herramientas para la educación.* Eguia.

Custodio, A. (2020). *Who Are You?* MIT Press.

David Paris, S. H. (2016). *History of Video Games.* Teacher Created Materials.

Delgado, M. (13 de febrero de 2021). *vandal.elespanol.com*. Obtenido de https://vandal.elespanol.com/reportaje/itchio-el-mejor-sitio-para-descubrir-videojuegos-indies

Dhule, M. (2022). *Getting Started with Godot.* Springer Nature.

Didactoons. (16 de Febrero de 2021). *Didactoons*. Obtenido de https://www.didactoons.com/que-es-un-sprite-y-para-que-sirve-en-el-desarrollo-de-videojuegos/

Donovan, T. (2010). *Replay The History of Video Games.* Yellow Ant.

Educaciencia. (2016). *https://educa-ciencia.com*. Obtenido de https://educa-ciencia.com/sonido/

Hennessey, J. (2017). *The Comic Book Story of Video Games The Incredible History of the Electronic Gaming Revolution.* Clarkson Potter/Ten Speed.

https://www.bfxr.net/. (2020). *https://www.bfxr.net/*. Obtenido de https://www.bfxr.net/

K. B. (2004). *Kent Beck and Cynthia Andres. Extreme Programming Explained: Embrace.* Nueva York: Addison-Wesley Professional.

Lacasa, P. (2011). *Los videojuegos : aprender en mundos reales y virtuales.* Madrid España: Ediciones Morata.

Manthorp, R. (05 de diciembre de 2019). *FLYNN: ADVANCED JUMP MECHANICS.* Obtenido de https://gamemaker.io/es/blog/flynn-advanced-jump-mechanics

Manzur, J. L. (s.f.). *Godot Docs*. Obtenido de Godot Docs: https://docs.godotengine.org/es/stable/about/introduction.html

Mark.JP.Wolf. (2002). *The Video Game Explosion: A History from PONG to Playstation and Beyond.* Westport, Connecticut: GreenWood.

Marqués, S. F. (s.f.). *LOS VIDEOJUEGOS.*

McMichael, A. (2007). PC Games and the Teaching of History. *The History Teache*, 203-218 (16 paginas).

Pacheco, F. (2022). *Desarrollo de videojuegos accesibles en Godot Engine: la problemática de los subtítulos.* SAEI.

Raventós, S. B. (2008). *Breve historia de los videojuegos.* Barcelona: Athenea Digita.

Simón Tudanca, M. (20-ene-2014). *La viabilidad del juego indie.* Catalunya: Universitat Oberta de Catalunya.

Steven.L.Kent. (2016). *La Gran Historia de los Videojuegos.* Barcelona: B.S.A.

Uniat. (15 de Enero de 2020). *Uniat.Edu*. Obtenido de https://www.uniat.edu.mx/etapas-del-desarrollo-de-videojuegos/