

UNIVERSIDAD DEL ISTMO

**Prototipo didáctico de un videojuego para el aprendizaje de estructuras secuenciales, selectivas e iterativas orientado a niños**

PROTOCOLO DE TESIS QUE PARA OBTENER EL TÍTULO DE

**INGENIERO EN COMPUTACIÓN**

Presenta

**Carlos Esteban Sánchez Jiménez**

Directora de tesis

M.A.G.N. Silvia Reyes Jiménez

Santo Domingo Tehuantepec, Oaxaca, noviembre 2021

Índice

[1. Generalidades de la investigación 3](#_Toc122045406)

[1.1. Antecedentes 3](#_Toc122045407)

[1.2. Planteamiento del problema 5](#_Toc122045408)

[1.3. Justificación 6](#_Toc122045409)

[1.4. Objetivos 7](#_Toc122045410)

[1.4.1. Objetivo general 7](#_Toc122045411)

[1.4.2. Objetivos específicos 7](#_Toc122045412)

[1.5. Alcances 7](#_Toc122045413)

[1.6. Limitaciones 7](#_Toc122045414)

[1.7. Trabajos relacionados 8](#_Toc122045415)

[1.8. Metodología 12](#_Toc122045416)

[2. Marco teórico 14](#_Toc122045417)

[2.1. Videojuego 14](#_Toc122045418)

[2.1.1. Tipos de videojuegos 15](#_Toc122045419)

[2.1.2. Videojuego educativo 16](#_Toc122045420)

[2.1.3. Motor de videojuego 16](#_Toc122045421)

[2.2. Estructuras de control 17](#_Toc122045422)

[2.2.1. Estructura secuencial 18](#_Toc122045423)

[2.2.2. Estructura selectiva 18](#_Toc122045424)

[2.2.3. Estructura iterativa 18](#_Toc122045425)

[2.3. Pensamiento computacional 18](#_Toc122045426)

[2.4. Metodología de desarrollo de videojuegos SUM 19](#_Toc122045427)

[2.4.1. Concepto 20](#_Toc122045428)

[2.4.2. Planificación 21](#_Toc122045429)

[2.4.3. Elaboración 21](#_Toc122045430)

[2.4.4. Beta 22](#_Toc122045431)

[2.4.5. Cierre 22](#_Toc122045432)

[2.4.6. Gestión de riesgos 22](#_Toc122045433)

[2.5. Modelo conceptual para el diseño de videojuegos educativos 22](#_Toc122045434)

[2.5.1. Mecánica 23](#_Toc122045435)

[2.5.2. Retos 23](#_Toc122045436)

[2.5.3. Escenario 23](#_Toc122045437)

[2.5.4. Caracterización de personajes 24](#_Toc122045438)

[2.5.5. *Feedback* 24](#_Toc122045439)

[2.5.6. Socialización 24](#_Toc122045440)

[2.5.7. *Debriefing* 24](#_Toc122045441)

[2.5.8. *Storyline* 25](#_Toc122045442)

[2.5.9. Recompensa 25](#_Toc122045443)

[2.5.10. Persistencia 25](#_Toc122045444)

[2.5.11. Servicios 25](#_Toc122045445)

[2.5.12. Interfaz del juego e interacción 26](#_Toc122045446)

[3. Cronograma de actividades 27](#_Toc122045447)

[4. Desarrollo 28](#_Toc122045448)

[4.1. Conceptualización y diseño 28](#_Toc122045449)

[4.1.1. Mecánica 28](#_Toc122045450)

[4.1.2. Retos 31](#_Toc122045451)

[4.1.3. Escenario 31](#_Toc122045452)

[4.1.4. Caracterización de personajes 32](#_Toc122045453)

[4.1.5. *Feedback* 33](#_Toc122045454)

[4.1.6. *Debriefing* 33](#_Toc122045455)

[4.1.7. *Storyline* 33](#_Toc122045456)

[4.1.8. Recompensa 34](#_Toc122045457)

[4.1.9. Persistencia 34](#_Toc122045458)

[4.1.10. Interfaz del juego e interacción 34](#_Toc122045459)

[4.2. Planificación 34](#_Toc122045460)

[4.3. Elaboración 35](#_Toc122045461)

[4.3.1. Planificación de la iteración 35](#_Toc122045462)

[4.3.2. Seguimiento de la iteración 35](#_Toc122045463)

[4.3.3. Desarrollo de características 36](#_Toc122045464)

[4.4. Beta 68](#_Toc122045465)

[Pruebas 68](#_Toc122045466)

[Errores encontrados 69](#_Toc122045467)

[Resolución de errores 69](#_Toc122045468)

[5. Referencias 70](#_Toc122045469)

# Generalidades de la investigación

## Antecedentes

La tecnología ha sido parte fundamental para el desarrollo de la sociedad, se han creado medios de comunicación que nos permiten estar conectados con el mundo de distintas maneras. Uno de estos medios son los dispositivos móviles, que también pueden ser usados para diferentes fines, siendo uno de ellos el ocio y más en específico, jugar videojuegos. Los videojuegos, “son una manifestación de las nuevas tecnologías que caracterizan a nuestra sociedad y cuyas transformaciones son consideradas por muchos autores como más profundos y con mayor impacto en nuestras vidas cotidianas de las que tuvieron lugar con la Revolución Industrial” en palabras de (Zhao , 2017, p.15). La popularidad de los videojuegos para dispositivos móviles ha ido en aumento (tal y como se muestra en la Figura 1), siendo más accesibles que los juegos de consola o de computadoras personales, además de que se pueden jugar en cualquier sitio.

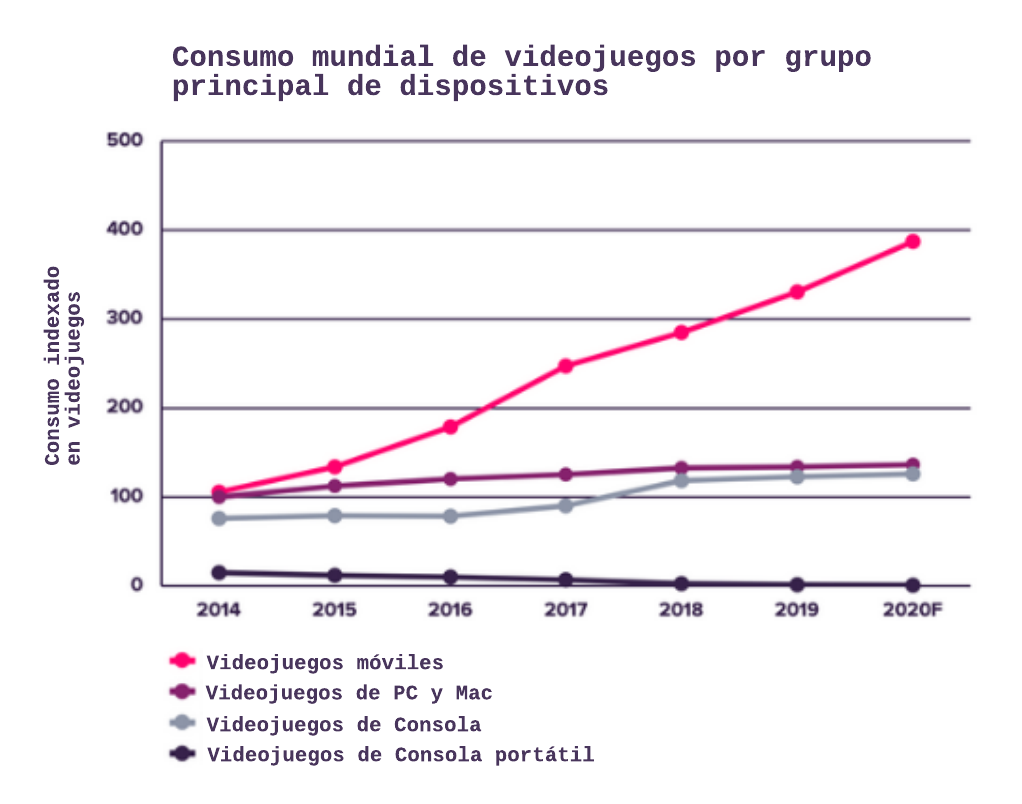


Figura 1. Consumo de videojuegos por plataforma. Fuente: App Annie & IDC, 2020, App Annie (https://www.appannie.com/en/insights/market-data/gaming-spotlight-2020-review/)

Durante mucho tiempo los videojuegos han sido sólo herramientas para el ocio y el entretenimiento, sin hacer distinción de edades ni de géneros, debido a esa razón se tiene una idea de que los mismos son carentes de valor productivo y educativo, siendo sólo medios de distracción. Esta creencia ha estado siendo refutada desde los 70’s por autores como Clark Abt (1970), quien plantea formas de usar los juegos de mesa y de cartas para aprender, y no con el único objetivo de divertirse, bajo la denominación de Juegos Serios (*Serious Games*).

Según Zyda (2005), la participación de la pedagogía (actividades para educar o instruir, impartiendo así conocimiento o habilidad) es lo que hace que los juegos sean considerados como ‘serios’. La adición de características del videojuego para generar motivación y experiencias de aprendizaje inmersivas a la idea de los juegos serios, genera el concepto de los videojuegos serios, también conocidos como videojuegos educativos, y que están enfocados a enseñar mientras el jugador se divierte, haciendo uso de la gamificación (también llamado ludificación) como agente para motivar al jugador a terminar el juego.

La gamificación es un término que se utiliza para describir aquellas características de un sistema interactivo que tienen como objetivo motivar y comprometer a los usuarios finales a través del uso y la mecánica de estímulos comúnmente incluidos en los juegos. “Algunos elementos de gamificación utilizados son: puntos, recompensas, progresión, insignias y tablas de posiciones” (Zatarain, 2018, p. 116).

Por otro lado, existe un concepto que ha sido objeto de investigación de manera reciente, este es el pensamiento computacional, el cual según Basogain *et al*. (2015), es una metodología basada en la implementación de los conceptos básicos de las ciencias de la computación para resolver problemas cotidianos, diseñar sistemas domésticos y realizar tareas rutinarias. Por su parte, Zapata (2015), menciona que aprender pensamiento computacional, no es aprender a programar computadoras, incluso algunas habilidades propias del pensamiento computacional no están relacionadas con ordenadores, es una habilidad que permite resolver problemas de manera inteligente e imaginativa.

Por lo tanto, el presente trabajo busca prototipar un videojuego serio con características de gamificación para dispositivos móviles orientado a niños de quinto y sexto año de primaria, mediante temas básicos de la programación como lo son las estructuras secuenciales, iterativas y selectivas (la selección de estos temas se debe a que son los principios básicos de la programación, los cuales permiten idear un algoritmo capaz de resolver un problema de manera lógica y estructurada), para incentivar el desarrollo del pensamiento computacional como una habilidad más para resolver situaciones cotidianas con un enfoque computacional.

## Planteamiento del problema

El pensamiento computacional como área de investigación es relativamente nueva, desarrollada de manera masiva a principios de la década pasada (Téllez, 2019), por lo que, los materiales y experiencias en esta línea de investigación son recientes, lo que da a entender que los procesos para crear videojuegos educativos enfocados a desarrollar el pensamiento computacional se encuentran en la fase de exploración.

Por otro lado, los videojuegos orientados a enseñar programación han proliferado más en el mercado y al día de hoy existen muchas propuestas de este tipo, no así los videojuegos orientados al desarrollo del pensamiento computacional, lo que no quiere decir que no se fomente el pensamiento computacional en estos videojuegos, ya que al aprender programación se desarrolla indirectamente el pensamiento computacional (Álvarez, 2017; Basogain *et al.,* 2015; Compañ *et al.,* 2015), esto más bien es un señalamiento de la carencia de videojuegos destinados a desarrollar esta habilidad en particular.

Basado en lo anterior, se observó que, en los videojuegos educativos para niños orientados a enseñar programación, uno de los mecanismos más utilizados, no siendo el único, es el uso de un personaje “llamativo” al que se le tiene que asignar instrucciones secuenciales para cumplir una tarea u objetivo haciendo que el niño empiece a familiarizarse con la estructura de los lenguajes de programación. Algunos ejemplos de este tipo de juegos son: *Rabbids coding!* (<https://store.ubi.com/ofertas/rabbids-coding/5d96f9075cdf9a2eacdf68c9.html>), *Codemonkey* ([https://app.codemonkey.com](https://app.codemonkey.com/login)), *LightBot* ([https://lightbot.com](https://lightbot.com/)) y *Aprende a programar con el Chavo* (<https://civat.es/app/aprende-a-programar-con-el-chavo/>).

De igual manera se observó que los videojuegos mencionados cuentan con apartados visuales coloridos y divertidos los cuales están hechos con la intención de llamar la atención del niño, no obstante, se identificaron problemáticas relacionadas con los mecanismos de interacción entre el juego y el niño, haciendo que en algunos casos sea complicada de entender la tarea a realizar o el propósito de la misma, siendo este el principal problema al desarrollar el pensamiento computacional con esta clase de videojuegos que están orientados a enseñar programación, ya que hace que el niño pierda interés en seguir jugando, sin contar que muchos de estos juegos están en inglés lo que hace que el interés de los niños para aprender sobre computación se vea más reducida.

A raíz de la inesperada pandemia desatada por el coronavirus SARS-COV2 en 2020, el uso de TIC’s para el trabajo y la educación se ha integrado como parte de la nueva normalidad que se vive en todo el mundo. Debido al reciente confinamiento global y, de manera específica, en México según Sada (2020) aumentó considerablemente el tiempo que las personas destinan al uso del teléfono celular, siendo gran parte de este tiempo destinado a cuestiones de entretenimiento.

Tomando en cuenta la problemática mencionada sobre el poco desarrollo de videojuegos orientados a fomentar el pensamiento computacional, el de los juegos educativos para aprender a programar, y el aumento en el uso de dispositivos móviles; resulta adecuado explorar el desarrollo del pensamiento computacional mediante videojuegos para dispositivos móviles, que sin necesidad de estar conectado a internet para su funcionamiento, permitan a los niños visualizar y aplicar este razonamiento en situaciones de su vida diaria, fomentando su interés en aprender más sobre las temáticas relacionadas con la computación y sus distintas aplicaciones.

## Justificación

La pandemia, y en consecuencia el confinamiento, también ha incrementado las actividades relacionadas a los videojuegos (Montás, 2020), bajo la idea inicial que los videojuegos no sirven más que para distraer y perder el tiempo, sin embargo, esto también ha provocado que muchos jóvenes se sientan atraídos hacia la programación de videojuegos como un posible trabajo futuro, aunque siguen carentes de información y conocimientos sobre el área, lo que causa que dejen de lado ese deseo.

Una manera de afrontar esta carencia de información desde edades tempranas, es mediante el desarrollo del pensamiento computacional, ya que como bien sugiere Wing (2006), el pensamiento computacional debería ser una habilidad que todos deberían de tener, no solamente los que se dedican a esta área. Esta sugerencia se debe a que el pensamiento computacional permite la resolución de distintos problemas mediante técnicas nativas de la informática sin importar el área de estos. De esta manera, el pensamiento computacional facilitará el aprendizaje sobre computación a los niños que quieran dedicarse a esta área.

Con el desarrollo de este trabajo se espera fortalecer el pensamiento computacional en niños mediante mecánicas implementadas en un videojuego con elementos de gamificación para motivarlos a jugar y aprender en el transcurso del juego sin recurrir a interfaces complicadas basadas en instrucciones como lo hacen los videojuegos convencionales de programación.

## Objetivos

### Objetivo general

Desarrollar un prototipo didáctico para dispositivos móviles para el aprendizaje de estructuras secuenciales, selectivas a iterativas que fomente el desarrollo del pensamiento computacional en niños de quinto y sexto grado de primaria.

### Objetivos específicos

1. Plantear el concepto y las mecánicas del juego de tal manera que fomente el desarrollo del pensamiento computacional.
2. Desarrollar un prototipo para dispositivos móviles empleando el motor de videojuegos Godot.
3. Realizar pruebas de usabilidad con niños de quinto y sexto año de primaria.

## Alcances

* El prototipo podrá ejecutarse sobre cualquier dispositivo móvil con Android.
* Será jugable sin necesidad de una conexión a internet.
* El prototipo sólo tendrá como ejes temáticos las estructuras secuenciales, selectivas e iterativas al resolver problemas, ya que las principales características del pensamiento computacional se pueden fomentar con estos tres.
* Se llevarán a cabo pruebas de usabilidad para medir la satisfacción del usuario con el prototipo sin evaluar la comprensión de los temas planteados.
* El resultado de este proyecto será un prototipo de videojuego educativo que no será lanzado al mercado.

## Limitaciones

* El prototipo desarrollado será exclusivo para dispositivos móviles con sistema operativo Android.
* El prototipo sólo podrá ser jugado por niños que ya sepan leer y ser capaces de seguir instrucciones sencillas, focalizándose en niños de quinto y sexto año de primaria.
* El trabajo no estará financiado por lo que se optará por utilizar software libre o gratuito.

## Trabajos relacionados

En el trabajo de Rahman (2017), se describe el desarrollo de un videojuego educativo en 3D para dispositivos móviles con Android para enseñar la lógica relacionada a la programación a personas de 5 a 18 años. El jugador controla a un personaje al que se le asignan instrucciones que va ejecutando una a una para derrotar enemigos y superar obstáculos con el objetivo de alcanzar un gato ubicado en el mapa.

Manquilef (2021), redacta en su trabajo el desarrollo de un videojuego educativo en 3D para computadoras personales, enfocado a desarrollar el pensamiento computacional en los jugadores mediante la resolución de puzles. Utiliza un sistema de comandos introducidos en una consola gráfica dentro del juego que afectan principalmente a unas plataformas que están repartidas en el mapa con el objetivo de escapar de la sala del nivel, las instrucciones emulan estructuras cíclicas y selectivas básicas de programación.

Por otro lado, Ticante *et al*. (2019), en su trabajo desarrollan un videojuego educativo de tipo RPG (juego de rol por sus siglas en inglés) de aventuras en 2D, para computadoras personales, como apoyo para comprender principios básicos de programación y desarrollar habilidad lógica en niños. La interfaz está constituida por una consola en la que el jugador tendrá que escribir instrucciones para que el personaje recorra el nivel y derrote a los enemigos que se vaya encontrando.

Además de los trabajos mencionados anteriormente, existen videojuegos educativos orientados a la computación creados por desarrolladoras para distintas plataformas. Estos tienen un objetivo más comercial y algunos no son precisamente trabajos de investigación, por lo tanto, el desarrollo de estos no está documentado lo que dificulta adquirir nuevo conocimiento sobre el desarrollo de videojuegos educativos. En la Tabla 1 se listan algunos videojuegos educativos disponibles en el mercado, su información y algunos puntos buenos y carencias respecto a cada uno.

Tabla 1. Videojuegos educativos publicados orientados a programación.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Videojuego | Enlace | Plataforma | Características | Observaciones |
| LightBot | <https://lightbot.com/> | * En línea * iOS * Android | * Videojuego para aprender programación basado en la resolución de puzles. * El personaje debe de pasar por los cuadros apagados para prenderlos. * Implementa el uso de instrucciones a través de comandos gráficos. | * Interfaz amigable * Traducción a varios idiomas * Tiene costo * La versión en línea requiere de la obsoleta tecnología Flash para poder jugar. * Requiere de una conexión a internet. |
| Code Monkey | <https://www.codemonkey.com/> | * En línea | * Videojuego para aprender programación basado en la resolución de puzles. * El juego consiste en conseguir el plátano del monito. * Las instrucciones se asemejan a un lenguaje de programación escrito. | * Entorno muy amigable para los niños. * Tiene mucha ayuda y diferentes planes de aprendizaje. * Tiene planes para maestros y padres. * Incluye niveles de prueba sin costo, pero se tendrá que pagar por un plan para seguir jugando. * Las instrucciones que se le dan al monito están en inglés. * Requiere de una conexión a internet. |
| Code Combat | <https://codecombat.com/> | * En Línea | * Videojuego de aventura, basado en la exploración de mazmorras para aprender programación. * El objetivo del juego es obtener el tesoro en cada nivel evitando los peligros de la mazmorra. * Las instrucciones se asemejan a un lenguaje de programación escrito. | * El entorno se ve muy llamativo parecido a un juego de aventuras. * Permite la mejora y elección del personaje. * No tiene costo. * Permite utilizar Python, JavaScript, Coffe Script o C++ como sintaxis para la programación. * Para utilizar algunas sintaxis es necesario estar registrado en la página del juego. |
| Cargo-Bot | <https://apps.apple.com/es/app/cargo-bot/id519690804> | * iOS | * Videojuego de rompecabezas para aprender a programar instrucciones. * El objetivo del juego es introducir instrucciones a un brazo robótico para replicar un patrón. * Las instrucciones están representadas de manera gráfica para que el niño pueda guiarse sin necesidad de escribir nada. | * El juego está para su descarga gratuita en la tienda de Apple. * El ambiente es muy colorido con una temática caricaturesca. * Está disponible únicamente en inglés. * No requiere de una conexión a internet para jugar. |
| RoboZZle | * <https://apps.apple.com/es/app/robozzle/id350729261> * <https://play.google.com/store/apps/details?id=com.team242.robozzle> * <https://amzn.to/2HGC4gC> | * iOS * Android * Kindle Fire | * Videojuego para aprender principios de programación basado en la resolución de puzles. * El objetivo es recorrer el nivel mediante instrucciones recogiendo todas las estrellas del camino. * Las instrucciones son flechas que se arrastran al área de instrucciones. | * El videojuego está disponible de manera gratuita en la tienda de Apple y la App Store de Android, con compras dentro de la app. * La versión para Kindle Fire tiene precio. * La interfaz de usuario es poco intuitiva. * El juego está en inglés. * La versión de Kindle Fire tiene traducción a ruso y francés. * Los niveles son desarrollados por diferentes usuarios del juego. |
| Daisy the Dinosaur | <https://apps.apple.com/es/app/daisy-the-dinosaur/id490514278> | * iOS | * Aplicación para aprender principios básicos de programación. * El objetivo de la aplicación es hacer que el dinosaurio baile y se mueva por la pantalla mediante instrucciones. | * La aplicación es gratuita en la tienda de Apple. * La interfaz es muy amigable y sencilla. * Las instrucciones se introducen en forma de bloques con una etiqueta de lo que hará el dinosaurio. * Tiene un tipo de juego libre en el que el niño hace lo que quiera y un modo reto donde se plantean objetivos a cumplir. * La aplicación está únicamente en inglés. |
| Kodable | <https://apps.apple.com/es/app/kodable/id577673067> | * iOS | * Videojuego para aprender a programar mediante la resolución de retos. * El objetivo del juego es resolver los retos ambientados en el recorrido de unos alienígenas en su planeta. | * El videojuego es gratuito en la tienda de Apple. * Tiene un modo para aprender JavaScript desde cero, pero requiere de un pago monetario en el juego. * El juego es muy amigable gráficamente, muy colorido y atractivo para niños. * No es necesario saber leer para jugar, ya que el juego explica todo mediante gráficos y voz. * Disponible únicamente en inglés. |

De los videojuegos relacionados mencionados y descritos con anterioridad se pueden observar que algunos fueron desarrollados para dispositivos móviles con iOS como sistema operativo, cabe destacar que estos dispositivos son costosos de adquirir y por lo tanto, no toda la población puede contar con uno; también hay juegos jugables en línea, lo que quiere decir que requieren de un navegador web y de una conexión permanente a internet para poder jugar, estos navegadores están disponibles en cualquier equipo de cómputo y dispositivo móvil, aunque es más común y cómodo su uso en equipos de cómputo, que de igual manera que con los dispositivos móviles con iOS, estos equipos pueden ser costosos y no todas las familias pueden darse el privilegio de contar con uno en casa. (revisar datos de la INEGI sobre familias con equipos de cómputo y celulares)

## Metodología

Para la ejecución de este proyecto se recurrió a una metodología de desarrollo de videojuegos y un modelo de conceptualización de videojuegos educativos. Es de mencionar, que al momento no se ha identificado en la literatura una metodología estandarizada para el desarrollo de videojuegos educativos. Se propuso el uso de una metodología en particular y un modelo de conceptualización en el desarrollo del videojuego considerándolos como un diferenciador de este trabajo de tesis.

Esta propuesta metodológica está basada principalmente en la metodología SUM para el desarrollo de videojuegos; descartando las etapas de planificación y gestión de riesgos, las razones son que la documentación del proyecto ya incorpora una planificación de tiempo de todo el proyecto donde cada etapa de la metodología ya está contemplada y debido a que al ser un proyecto de desarrollo pequeño y contar con un equipo conformado por una persona, los problemas surgidos se pueden gestionar de manera inmediata y tomar la mejor decisión dependiendo de la situación actual del proyecto.

También, al ser SUM una metodología enfocada al desarrollo de videojuegos en general, no contempla características principales a tomar en cuenta al momento de conceptualizar videojuegos educativos, por esta razón la etapa de concepto de la metodología SUM se reemplazó por el modelo conceptual para el diseño de videojuegos educativos planteado por Ruíz (2014), el cual es un modelo orientado a la conceptualización de videojuegos con valor educativo.

Del modelo conceptual de Ruíz se descartaron las entidades de socialización y servicios, debido a que, el prototipo no cuenta con algún tipo de interacción entre jugadores y solamente se desarrolló el apartado jugable del videojuego y no incluye ningún tipo de medio cuyo propósito no sea jugar. De esta manera, en la Figura 2 se puede observar el proceso metodológico que se llevó a cabo en el proyecto.

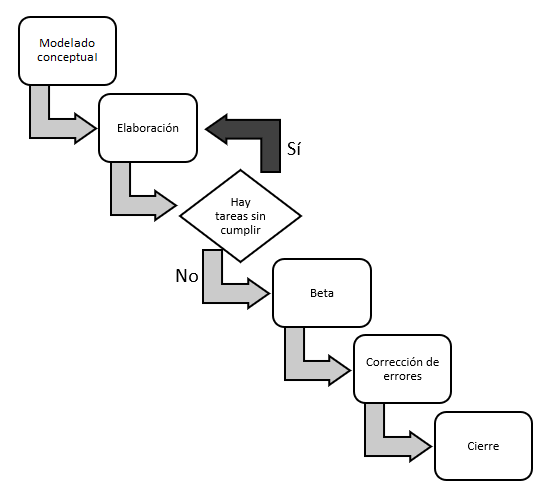


Figura 2. Etapas de la metodología de trabajo del proyecto.

A continuación, se describen cada una de las etapas de la metodología que se llevaron a cabo en el transcurso del proyecto.

* Modelado conceptual: aquí se plantearon las mecánicas que regirán la jugabilidad del videojuego, los retos que el jugador afronta durante el juego, los escenarios que adornan cada nivel del juego, el diseño y caracterización de los personajes, el *feedback* que el juego brinda al jugador con cada acción realizada mientras está jugando, la historia de fondo del videojuego, las recompensas que el jugador obtiene al cumplir ciertos requisitos, los métodos de persistencia para continuar jugando en cualquier momento y la definición de la interfaz de usuario.
* Elaboración: se llevó a cabo la programación del videojuego tomando como base todo lo definido en la etapa anterior. Todo el concepto se dividido en tareas que se enlistaron para su desarrollo individual en esta etapa. Este proceso de desarrollo es iterativo e incremental tal como se puede observar en la Figura 2. Cuando ya no hubo tareas por desarrollar es cuando esta etapa llegó a su fin y se pasó a la siguiente.
* Beta: es una versión temprana del videojuego, en esta versión se llevaron a cabo pruebas de jugabilidad para detectar errores y *bugs* en la programación. También permitió comprobar que las mecánicas y el comportamiento del juego funcionaran de manera correcta.
* Corrección de errores: los errores y *bugs* detectados en la beta, se enlistaron y se corrigieron permitiendo refinar el resultado final.
* Cierre: se obtuvo la versión final del prototipo después de haber corregido los errores detectados en la beta, se libera el juego para realizar pruebas con los usuarios finales.

Por último, a pesar de que se menciona la liberación del juego dentro de la metodología, el resultado de este proyecto es un prototipo de videojuego educativo que no fue lanzado al mercado y está diseñado para niños de quinto y sexto de primaria principalmente.

# Marco teórico

## Videojuego

Un videojuego es un juego digital interactivo, creado con fines de entretenimiento en general y se basa en la interacción con personas o personajes ficticios (Ticante *et al*., 2019). De una manera más amplia Frasca (2001), define a los videojuegos como cualquier forma de software de entretenimiento por computadora, ya sea textual o basado en imágenes, usando cualquier plataforma electrónica como computadoras personales o consolas e involucrando a uno más jugadores en un entorno físico o de red.

Así como cada persona tiene un diferente punto de vista de un mismo tema, diferentes autores tienen una concepción distinta de lo que es un videojuego, sin embargo, Wolf y Perron (2003), identifican cuatro elementos fundamentales que comparten todos los videojuegos, estos son gráficos, interfaz, actividad del jugador y algoritmo. Los gráficos son un conjunto de visualizaciones cambiantes en una pantalla que produce una imagen basada en pixeles; la interfaz marca la frontera entre el jugador y el videojuego, es el medio por el cual el mundo real se comunica con el videojuego, cuantifica las acciones del jugador y las convierte en datos; las actividades del jugador consisten en la entrada de datos por medio de la interfaz, así una acción en el videojuego será el resultado de una acción realizada físicamente por el jugador; y el algoritmo es el que define el videojuego, las reglas, las respuestas de este y principalmente controla todo.

Dicho de una manera más sencilla, los videojuegos son juegos que se juegan en un medio electrónico. Estos utilizan distintos dispositivos de interacción para capturar las acciones del jugador, como teclados de computadora, *mouse*, *joysticks* de consola, pantallas táctiles, entre otros.

### Tipos de videojuegos

Los videojuegos se pueden clasificar de distintas maneras. De acuerdo a su género o tipo se clasifican en juegos de acción, juegos de aventura, juegos *arcade*, juegos deportivos, juegos de estrategia, juegos de simulación o juegos de mesa, tal y como se muestra en la Tabla 2.

Tabla 2 Clasificación de videojuegos por su género.

|  |  |
| --- | --- |
| Tipo | Descripción |
| Acción | Se basan principalmente en eliminar enemigos y su contenido suele ser violento. Implican una inmersión grande, concentración y por lo general se relacionan con exposiciones cortas de tiempo. |
| Aventura | El protagonista del juego atraviesa niveles, lucha contra enemigos y recoge objetos de valor. Casi siempre son juegos de larga duración con argumentos extensos y dependen completamente de la trama para su valoración. |
| Arcade | Son juegos de plataformas, laberintos y aventuras en donde el usuario debe superar pantallas para seguir jugando. Imponen un ritmo rápido de juego y requieren tiempo mínimos de reacción por parte del jugador. |
| Deportivo | Recrean diversos deportes y requieren de habilidad, rapidez y precisión. No suelen contener tramas argumentales de ningún tipo y contienen un alto grado de socialización. |
| Estrategia | Consisten en trazar una estrategia para superar al contrincante. Exigen concentración, saber administrar recursos, pensar y definir estrategias de juego. Se caracterizan por tener una curva de aprendizaje de algún tipo, ya que se pueden perder las primeras partidas, pero el jugador va aprendiendo de esto. Pueden ser útiles para desarrollar la inteligencia y pueden llegar a ser complicados de dominar. |
| Simulación | Permiten experimentar el funcionamiento de máquinas, fenómenos, situaciones y asumir el mando en estas. Representan acciones que se pueden vivir fuera de un contexto de peligro real, como volar un avión o conducir un tanque de la segunda guerra mundial. |
| Juegos de mesa | La tecnología informática sustituye al material tradicional del juego e incluso al adversario por un personaje controlado por inteligencia artificial. |

Cabe resaltar que existen más géneros de videojuegos con sus propias características diferenciadoras, también un género no es excluyente de otro, al contrario, se pueden combinar distintos géneros para generar un videojuego más variado en mecánicas.

### Videojuego educativo

Según López (2016), los videojuegos educativos tienen un propósito más allá del entretenimiento del usuario, cuyo objetivo es crear entornos de aprendizaje virtuales que permitan experimentar con problemas del mundo real, con la finalidad de hallar soluciones y obtener conocimiento sin temor a equivocarse ni tener repercusiones sobre sus decisiones. De esta manera se puede intentar un objetivo cuantas veces el jugador quiera sin importar cuantas veces falle, ya que sus errores en el videojuego no afectan al mundo real.

### Motor de videojuego

Vallejo & Martín (2019), describen un motor de videojuegos como una herramienta de software la cual separa los componentes fundamentales de un videojuego (sistema de renderizado gráfico, sistema de detección de colisiones, sistema de audio, sistema de físicas) en módulos, y mediante una interfaz de usuario permite modificar y enlazar los módulos para desarrollar un videojuego.

En la actualidad existen varios motores de videojuegos, algunos más complejos que otros, algunos para uso exclusivo de empresas y otros de uso libre para los entusiastas del desarrollo de videojuegos; cada uno con sus ventajas y desventajas. En la Tabla 3, se listan algunos motores tomados en cuenta para el desarrollo del proyecto, todos con una curva de aprendizaje baja, la tabla contiene las características principales de cada motor.

Tabla 3. Motores de videojuegos.

|  |  |
| --- | --- |
| Motor | Características |
| Godot | * Es de código abierto con licencia del MIT. * Cuenta con un sistema de programación gráfica por nodos. * Permite programación escrita para mayor control del juego mediante GDScript (lenguaje exclusivo de Godot y basado en Python), C# y C++. * Permite el desarrollo de videojuegos en 2D y 3D, aunque está más enfocado al 2D. * Exporta juegos para Windows, macOS, Linux, UWP, BSD, iOS, Android y HTML5 de manera nativa. * Exporta juegos a consolas Nintendo Switch, PS4, Xbox One mediante aplicaciones de terceros. * Cuenta con un sistema para crear animaciones. |
| GameMaker Studio 2 | * Es un motor de pago y cuenta con una prueba gratuita de 30 días o una licencia gratuita para hacer pruebas. * Cuenta con un sistema de programación gráfica *Drag and Drop* (arrastrar y soltar) muy intuitivo. * Permite la programación escrita para mayor control del juego mediante GML (*GameMaker Language*) basado en C. * Especializado en el desarrollo de videojuegos en 2D. * Exporta juegos para Windows, macOS, Linux, HTML5, Android, iOS, PS4 y Xbox One. |
| GameSalad | * Es un motor de pago, pero cuenta con una prueba gratis. * Especializado en el desarrollo de videojuegos en 2D. * Cuenta únicamente con un sistema de programación gráfica *Drag and Drop* (arrastrar y soltar). * Exporta juegos para Windows, iOS, Android y HTML5. |
| Construct 2 | * Especializado en el desarrollo de videojuegos 2D. * Exporta juegos para Windows, macOS, Linux, HTML5, Android y Nintendo Wii U. * Cuenta con una interfaz gráfica que permite la programación sin códigos. * Permite la programación escrita para mayor control del juego mediante HTML5 y JavaScript. |

Se excluyeron los motores más populares y completos como *UnrealEngine, Unity* y *CryEngine* debido a su alta curva de aprendizaje y por la naturaleza del proyecto que estará enfocado al desarrollo de un videojuego 2D. Así, tomando en cuenta las características del motor y el precio por el uso del mismo; se usó Godot para el desarrollo del proyecto, esto debido a su libertad de uso y personalización del videojuego mediante código de ser necesario.

## Estructuras de control

Existen tres estructuras de control básicas que guían el orden de ejecución de un programa de computadora: las estructuras secuenciales, selectivas e iterativas. Estas son esenciales para cualquier lenguaje de programación, ya que son las que dan lógica al programa desarrollado. Según Ristić *et al*. (2016), el aprender estos conceptos básicos de programación facilitará el aprendizaje de cualquier lenguaje de programación.

### Estructura secuencial

Es una estructura de control en la que las instrucciones se ejecutan una después de otra en el orden en el que se encuentran escritas en un programa (Márquez *et al*., 2011).

Una estructura secuencial se puede ver de la siguiente manera:

|  |
| --- |
| *Instrucción 1* |
| *Instrucción 2* |
| *.* |
| *.* |
| *.* |
| *Instrucción n* |

La *n* representa el infinito y las instrucciones se ejecutan una por una de arriba hacia abajo hasta la *n*-ésima instrucción.

### Estructura selectiva

Son aquellas que evalúan una expresión, usualmente una condición booleana, y a partir del resultado permiten tomar decisiones en una, dos o más opciones. Una expresión booleana no es más que una expresión que puede tener como valor verdadero o falso.

### Estructura iterativa

Una estructura iterativa permite repetir una acción; la repetición es controlada por una expresión que es una condición booleana.

## Pensamiento computacional

El pensamiento computacional implica resolver problemas, diseñar sistemas y comprender el comportamiento humano, haciendo uso de los conceptos fundamentales de la informática (Wing, 2006). La Sociedad Internacional de Tecnología en educación y la Asociación de Docentes en Ciencias de la Computación (ISTE & CSTA) (2011), consideran al pensamiento computacional como un proceso de resolución de problemas que incluyen (pero no están limitadas a) las siguientes características:

* Formulación de problemas de manera que se puedan utilizar computadoras y otras herramientas para su resolución.
* Organización lógica y analítica de datos.
* Representación de datos de manera abstracta, como modelados y simulaciones.
* Automatización de soluciones a través de pensamiento algorítmico (una serie de pasos ordenados).
* Identificación, análisis, e implementación de posibles soluciones con la finalidad de obtener la más eficiente y efectiva combinación de pasos y recursos.
* Generalización y transferencia de este proceso de resolución de problemas a un conjunto variado de problemas.

Otro concepto surgido a través de investigaciones científicas es definido por The Royal Society (2012), quienes consideran al pensamiento computacional como el proceso de reconocimiento de aspectos de la informática en el mundo que nos rodea, y la aplicación de herramientas y técnicas de la informática para comprender y razonar sobre los sistemas y procesos tanto naturales como artificiales.

Por su parte, Manquilef (2021), define que “el pensamiento computacional es más bien una idea de cómo pensar para dar solución a problemas, más que simplemente enseñar a programar o la interacción humano-computador”. De igual manera identifica tres componentes fundamentales que forman parte de esta forma de pensar:

* Abstracción: separar el problema y aislar cada parte de esta para su análisis individual para después hallar la relación entre todas las partes.
* Descomposición: complementa la abstracción, y es la característica de dividir el problema en partes más pequeñas, donde cada parte es un subproblema del principal.
* Reconocimiento de patrones: reducir el problema hasta encontrar un patrón de comportamiento, tanto en la entrada como en la salida, de la información relacionada al problema, para así hallar una solución general.

## Metodología de desarrollo de videojuegos SUM

Existen gran cantidad de metodologías para desarrollar software, cada una con un diferente enfoque. No existe una metodología que sea mejor que otra, ya que; dependiendo de los objetivos del desarrollo, el tiempo, la cantidad de gente involucrada, el presupuesto, entre otros factores; se elige una metodología con la cual trabajar. Incluso puede darse el caso de que se combinen metodologías si el proyecto así lo amerita. Para este proyecto en específico, el desarrollo del videojuego se llevó a cabo mediante la implementación de la metodología SUM, exceptuando la etapa de *Concepto* que fue reemplazada por el modelo conceptual para el diseño de videojuegos educativos.

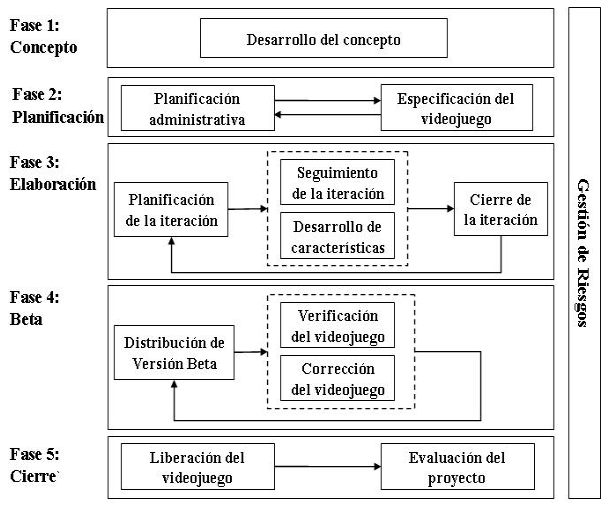


Figura 3. Estructura de la metodología SUM. Extraída de http://www.gemserk.com/sum/

La metodología SUM es una metodología basada en SCRUM, cuyo objetivo al desarrollar videojuegos es “la obtención de un software de calidad en tiempo y costo, así como la mejora continua del proceso para aumentar su eficiencia y eficacia” (Arenas, 2018, p. 24). Se compone de cinco fases, *Concepto*, *Planificación*, *Elaboración*, *Beta* y *Cierre*, pero hay una fase que se lleva a cabo durante todo el desarrollo del proyecto y que se ejecuta a la par de la fase actual del desarrollo, esta es la *Gestión de riesgos*. La Figura 3 muestra la estructura y el flujo que la metodología sigue durante todo el proceso de desarrollo.

### Concepto

Esta primera fase contempla la idea principal del videojuego a desarrollar, que tipo de juego será, la historia, la ambientación y principalmente las mecánicas de juego, esto último permitirá definir un rumbo concreto al momento de programar el videojuego; aspectos de negocio del videojuego, a quién va dirigido, modelo de negocio; y por último contempla el aspecto técnico del videojuego, el lenguaje de programación, el motor y las herramientas para su desarrollo (Acerenza *et al*., 2009; Arenas, 2018; Murillo *et al*., 2018).

### Planificación

En esta fase lo principal es medir los tiempos que lleva cada parte del desarrollo, desde la fase de concepto hasta la fase de cierre y elaborar un cronograma de actividades.

### Elaboración

Es la parte donde se lleva a cabo la programación, el desarrollo gráfico, y, por ende, el resultado funcional del videojuego. Esta fase se lleva a cabo de manera iterativa e incremental, dando como resultado en cada iteración una parte funcional del juego. Esta misma fase se divide en las siguientes actividades que se llevan a cabo igual en cada iteración:

1. Planificación de la iteración: se plantean los objetivos y características que se obtendrán al finalizar la iteración, estos se descomponen en tareas que se llevarán a cabo a lo largo de la iteración y también se establecen las validaciones que analizan que la parte realizada está cubierta.
2. Seguimiento de la iteración: controla el avance de la iteración en base a los objetivos establecidos.
3. Desarrollo de características: se llevan a cabo las tareas (véase Figura 4) establecidas para cubrir la característica planteada, una vez que una tarea está terminada se realizan pruebas para encontrar errores, en caso de haberlos, buscar maneras de corregirlos.
4. Cierre de la iteración: evalúa el estado actual del videojuego respecto a lo ocurrido en la iteración y así adaptar el plan de desarrollo con lo obtenido. Se llevan a cabo pruebas de usuario sobre la característica desarrollada y basado en eso se eliminan, agregan o modifican características para la siguiente iteración.

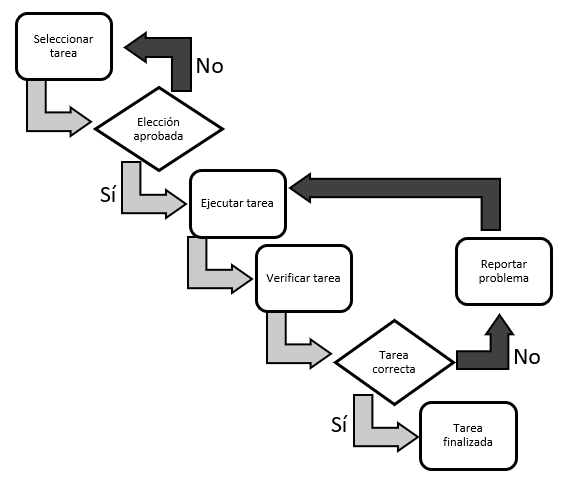


Figura 4. Proceso de desarrollo de tareas.

### Beta

Es una versión de prueba del videojuego completo, que permite probar la funcionalidad del videojuego para encontrar errores y *bugs* de programación para después corregirlos, también tiene como propósito verificar que las mecánicas y el comportamiento del videojuego sean los correctos.

### Cierre

Se llega al cierre después de haber refinado la versión beta, en esta etapa el videojuego se presenta al público objetivo para evaluar el rendimiento del videojuego y tomar consideraciones a futuro.

### Gestión de riesgos

Su objetivo es reducir la ocurrencia y el impacto de problemas en todo momento mediante el análisis de problemas que podrían ocurrir durante el proceso y establecer métodos de solución para estos casos.

## Modelo conceptual para el diseño de videojuegos educativos

En 2014, Mario Rafael Ruiz Vargas presentó en su investigación doctoral “Un Modelo Conceptual Para El Diseño de Videojuegos Educativos” (Ruiz, 2014), una propuesta para el diseño conceptual de videojuegos educativos, que en palabras del propio Ruiz, no se centra en los retos y problemas específicos asociados con un género de juego; más bien reúne y organiza las características que la literatura de videojuegos considera a menudo como significativas dentro de la producción de una experiencia de videojuego atractiva, divertida y educativa. No está orientado como tal al desarrollo funcional del videojuego, más bien, es una guía para concebir la idea inicial y las entidades más importantes para un videojuego educativo.

Dicha propuesta identifica 12 aspectos fundamentales en un videojuego educativo: mecánica, retos, escenario, caracterización de personajes, *feedback*, socialización, *debriefing, storyline*, recompensa, persistencia, servicios, interfaz del juego e interacción. Descritas por Ruiz de la siguiente manera:

### Mecánica

“Define los elementos o entidades principales del juego y las acciones e interacciones entre ellas que pueden tener lugar a lo largo de la partida. Determina qué es lo que puede suceder durante una partida” (Ruiz, 2014, p. 43).

### Retos

Ruiz (2014), define los retos como “las metas que las distintas entidades deberán tratar de lograr durante el juego, junto con las situaciones o eventos que deberán evitarse”. A partir de las mecánicas definidas, se podrán plantear los objetivos que el jugador deberá lograr en el juego, esto a su vez se podrá asociar con el objetivo de aprendizaje. Por ejemplo, separar perros y gatos de una manada de animales, este objetivo podría asociarse con “aprender a identificar”. Además de las metas, se pueden definir objetivos o eventos negativos que el jugador debe tratar de evitar para enriquecer la experiencia de juego del usuario.

### Escenario

Definido por Ruiz (2014) como “el espacio virtual y las representaciones que serán utilizadas para encarnar las entidades definidas en la mecánica”. Cuando en un mismo juego se haya definido más de una escena (lugares donde el juego puede tener lugar) puede ser necesario establecer enlaces entre ellas que determinen la forma en que los jugadores pueden navegar de unas a otras.

La descripción de una escena puede cubrir diversos aspectos incluyendo el tipo de dimensiones de la misma, el tipo de perspectiva, el área de visión de la escena, posiciones de la cámara, imágenes de fondo y los objetos o entidades con los que el jugador podría interactuar.

### Caracterización de personajes

Ruiz (2014) recomienda que “el diseño de los personajes de juegos dirigidos a niños de corta edad suele utilizar representaciones gráficas cercanas a los dibujos animados con los que están acostumbrados a ver” (p. 51).

En relación con la descripción de las habilidades y comportamientos, es necesario tener en cuenta que cada personaje deberá estar asociado a alguna entidad de juego de las previamente definidas en la mecánica básica. El control de la entidad podrá estar o bien en manos de uno de los jugadores PCC (*Player Controlled Character*) o bien será llevado a cabo automáticamente por el propio ordenador NPC (*Non-Player Character*). En este último, dependiendo del grado de sofisticación del juego y de la necesidad de simular un comportamiento inteligente, estas reglas pueden ser implementadas en forma de un conjunto de eventos que determinen cuándo se dispara, o bien a través de algún algoritmo o técnica propia del área de inteligencia artificial.

### *Feedback*

Lo define Ruiz (2014) como “los elementos y procedimientos que se utilizarán en cada momento para dar a conocer al jugador las consecuencias de sus acciones, su grado de progreso hacia los objetivos y en general, el estado y situación actual de la partida” (p. 44).

Además del *feedback* asociado a los objetivos del juego, suele ser habitual en los juegos educativos encontrarse con otro tipo de *feedback* que tiene por finalidad principal dar soporte al principio educativo del juego, suministrando información adicional que contribuya a la adquisición de conocimiento y al logro de los objetivos educativos.

### Socialización

“Establece los mecanismos de comunicación e interacción entre jugadores que el juego soportará” (Ruiz, 2014, p.44). Incluir algún tipo de componente que dé soporte a la socialización entre jugadores, es sumamente importante para el caso de los videojuegos educativos, ya que no sólo puede contribuir a incrementar la motivación de los jugadores, sino que además proporcionaría la base para el desarrollo del aprendizaje como ejercicio socio cultural.

### *Debriefing*

El término hace referencia al proceso durante el cual los participantes de un proceso educativo reflexionan y analizan sobre la actividad desarrollada, asociando lo aprendido con conceptos previamente conocidos, y formando hipótesis que puedan ser verificadas en experiencias futuras. En otras palabras, “son actividades que no forman parte del juego en sí y cuyo fin es hacer recapacitar al jugador sobre lo que ocurre en el juego y relacionarlo con la actividad en el mundo real” (Ruiz, 2014, p. 44).

### *Storyline*

El *storyline* “define el contexto y el argumento del juego, ordenando los objetivos que se propondrán al jugador en distintas secuencias” (Ruiz, 2014, p.44). Contextualizar el desarrollo del juego dentro de una historia, permite añadir a los factores intrínsecamente motivantes asociados al propio juego, el de la curiosidad del jugador por conocer la evolución de la historia.

### Recompensa

La recompensa se define como “conjunto de premios que se activan en función de los logros y acciones del jugador y que tienen por fin aumentar su motivación de progresar en el juego” (Ruiz, 2014, p. 44). Es frecuente que los videojuegos hoy en día incluyan mecanismos muy explícitos para premiar al jugador, algunos ejemplos de esto pueden ser, la acumulación de puntos, la posibilidad de explorar áreas secretas, la activación de juegos extra o de bonificación, o la capacidad de personalizar el avatar con ciertos objetos o permitir hacer uso de habilidades especiales.

### Persistencia

La persistencia “son un conjunto de mecanismos que el juego implementará para permitir o no al jugador afrontar los desafíos en varias sesiones de juego, y que establecerán la totalidad de la información que se mantendrá entre ellas” (Ruiz, 2014, p. 44).

En general, la elección de la información a persistir suele verse condicionada por el género al que el videojuego pertenece. De esta forma, en juegos de aventura o de rol, donde el mantenimiento de la continuidad de la historia es fundamental, es necesario almacenar el estado del juego prácticamente al completo. Por el contrario, en juegos de acción o plataforma es suficiente con almacenar el punto aproximado en el que se encuentra la historia y parte de la información relativa al estado actual del personaje.

### Servicios

Los servicios se definen como “herramientas y aplicaciones incluidas en el juego, con el fin de dar soporte a otros elementos de diseño o de enriquecer y aumentar las posibilidades ofrecidas por el juego” (Ruiz, 2014, p. 44). Por ejemplo, un videojuego puede incluir servicios de mensajería o de *chat* que den soporte a los distintos tipos de comunicación entre participantes especificados en el componente de socialización, servicios de foro que sirvan para llevar a cabo ciertas actividades de *debriefing*, o servicios de ranking que den apoyo a mecanismos de recompensa previamente definidos.

### Interfaz del juego e interacción

Este elemento es definido como “el conjunto de medios, tanto de entrada como de salida, de los que el jugador dispondrá para comunicarse con el juego” (Ruiz, 2014, p. 44). El diseño de los controles de un juego educativo plantea problemas específicos, debido a que esta característica no sólo tiene que ver con el impacto en la capacidad de juego, sino también si estas características se pueden utilizar para mejorar las capacidades de aprendizaje y habilidades específicas.

En conclusión, con el presente trabajo de investigación se desarrolló un prototipo didáctico de un videojuego para el aprendizaje de estructuras secuenciales, selectivas e interactivas orientado a niños, dando especial importancia al modelo conceptual para el desarrollo de videojuegos.

## Pruebas de usabilidad

Para Sánchez (2011), la usabilidad es un atributo que define la facilidad de uso de un sistema que requiera de la interacción con un usuario, generalmente aplicado a un software, aparato o sistema creado con algún objetivo particular, aunque también se refiere a la usabilidad como un factor para la aplicación de métodos con el objetivo de mejorar la facilidad de uso de estos sistemas durante el proceso de diseño.

Una definición más formal para este concepto está dada por la Organización Internacional para la Estandarización (ISO por sus siglas en inglés) donde dice que “la usabilidad es el grado en el que un sistema, producto o servicio puede ser utilizado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un determinado contexto de uso” (ISO 9241-11, 2018, página pendiente).

La usabilidad no es una propiedad individual de un sistema, sino está compuesta por cinco características (Nielsen, 1994; Sánchez, 2011).

* Facilidad de aprendizaje: indica el tiempo y la facilidad para aprender a utilizar el sistema.
* Eficiencia de uso: una vez aprendido a usar el sistema determina la rapidez para realizar las tareas del sistema.
* Retención sobre el tiempo: el uso del sistema debe ser fácil de recordar, reduciendo la curva de aprendizaje al volver a interactuar con el sistema después de un periodo de tiempo.
* Tasas de error: el sistema debe ofrecer una baja tasa de errores y en caso de ocurrir alguno ayudar al usuario a resolverlos de manera fácil.
* Satisfacción: debe ser placentero de utilizar, muestra de manera subjetiva la impresión del usuario respecto al sistema.

La evaluación de la usabilidad de un sistema permite identificar fallos en el diseño y así actuar para generar mejoras y correcciones en iteraciones futuras del prototipo. Para esto se pueden utilizar diferentes métodos, aunque habrá que analizarlos y elegir el que más se acople a las necesidades del sistema y al grupo de usuarios finales.

Agregar métodos de evaluación, revisar con la maestra Silvia

# Cronograma de actividades



Tabla 4. Cronograma de actividades.

# Desarrollo

## 4.1. Conceptualización y diseño

Para la conceptualización del videojuego se utilizó el modelo conceptual para el diseño de videojuegos educativos de Ruiz (2014); esta propuesta consta de 12 aspectos fundamentales a tomar en cuenta al momento de bosquejar el concepto de un videojuego serio, sin embargo, para la conceptualización del presente trabajo se descartaron los conceptos de *socialización* y *servicios*, empleando solamente diez, esto debido al alcance del videojuego que no requiere de dichos aspectos.

Esta última decisión se tomó puesto que el juego no cuenta con ningún componente de interacción para la socialización entre jugadores y, por último, el juego requiere de enfocar la atención del jugador solamente en jugar, además ambos aspectos requieren de más tiempo de desarrollo y no son estrictamente necesarios para completar el juego y llegar al objetivo principal, que es desarrollar el pensamiento computacional.

A continuación, se describen cada uno de los aspectos de conceptualización y diseño del prototipo didáctico de un videojuego para el aprendizaje de estructuras secuenciales, selectivas e iterativas orientado a niños.

### 4.1.1. Mecánica

Como mecánicas del juego se tomaron en cuenta las acciones que el jugador puede realizar en todo momento, las acciones de los enemigos, las acciones de los objetos del entorno, así como las interacciones que el jugador puede tener con estos. Se separaron por entidad y se trabajaron de manera individual para luego juntar las interacciones entre ellas.

#### 4.1.1.1. Personaje principal

Se definieron diferentes estados y acciones que el jugador puede realizar con el personaje principal, estas mecánicas se dividieron en dos tipos, mecánicas de movimiento y mecánicas de acción. Las de movimiento componen los estados que el personaje principal puede tener a lo largo del juego, los cuales son el estado *quieto* y *en movimiento*; el primero se ejecuta cuando el jugador no está tocando los controles haciendo que el personaje principal se quede quieto esperando, el segundo se ejecuta cuando el jugador utiliza los controles de movimiento y hacen que el personaje principal se mueva en la dirección que el jugador indicó, estas direcciones pueden ser arriba, abajo, adelante y atrás.

Las mecánicas de acción constan de las acciones que el jugador puede realizar diferentes a las mecánicas de movimiento, las acciones son *golpear* y *recoger.* La primera se ejecuta en el momento en el que el jugador presiona el botón designado para golpear, y hace que el personaje principal lance un golpe hacia la dirección que esté mirando en ese momento, esta acción se puede realizar mientras se está *quieto* o *en* *movimiento*; la segunda acción ocurre cuando el personaje principal pasa por encima de un objeto del mapa que pueda recogerse, esta es una acción que se realiza de manera automática pero también implica que el jugador se mueva hacia el objeto.

#### 4.1.1.2. Enemigo

Las mecánicas del enemigo se realizan de manera automática y se basan en el cumplimiento de ciertas condiciones. Contrario al personaje principal, los enemigos sólo tienen mecánicas de movimiento los cuales son *quieto* y *atacando*, el primero es similar al del personaje principal con la diferencia de que sale del estado de manera automática; la segunda es similar a la mecánica de *en movimiento* en el aspecto de que el enemigo se mueve con la diferencia de que sigue al jugador en un área definida alrededor del lugar donde se encuentra. Estas dos están relacionadas ya que una lleva a la otra, cuando el jugador entra al área definida el enemigo sale del estado de espera y comienza a seguir al jugador de manera automática.

#### 4.1.1.3. Objetos

Algunos objetos del juego tienen sus propias mecánicas las cuales son activadas por el jugador, aquellos objetos con los que el jugador puede interactuar son *ítems* para pasar el nivel e *ítems* valiosos (llave de colores y diamantes respectivamente), árboles con *ítems* en la copa, botones y puertas.

Los *ítems* tienen la mecánica de ser recogidos del suelo por el jugador. Los tienen árboles la mecánica de tirar el *ítem* que se encuentra la copa, lo cual hace posible que el jugador pueda recoger este y utilizarlo para pasar el nivel. Los botones activan un mecanismo, específicamente la activación de un puente desaparecido el cual aparece cuando se utiliza el botón. El último objeto es la puerta, esta tiene la mecánica de abrirse solamente si el jugador tiene la llave correcta para abrirlo.

El personaje utiliza el *golpear* para enfrentar enemigos, tirar objetos de los árboles y presionar botones para activar el mecanismo de puente; utiliza *en movimiento* para recoger objetos tirados en el mapa, así como abrir puertas cuando se tiene la llave correspondiente; los enemigos usan *atacando* para herir al jugador si estos llegan a tocarlo. Las mecánicas de las diferentes entidades se resumen en la Tabla 5.

Tabla 5. Mecánicas de entidades

|  |  |
| --- | --- |
| Entidad | Mecánica |
| Personaje principal | * Movimiento: se divide en estar quieto en un lugar del mapa y moverse en una dirección especificada. * Golpear: acción que permite herir enemigos y derrotarlos, también se utiliza para tirar objetos de los árboles y activar botones. * Recoger: se usa para obtener objetos tirados en el mapa. |
| Enemigo | * Movimiento: se divide en estar quieto en un lugar del mapa y moverse para atacar y herir al jugador. |
| Objetos | Varía dependiendo del objeto.   * *Ítem*   + Recogible: puede ser recogido por el jugador. * Árbol   + Tirar *ítem*: tirar algún objeto que se encuentre en la copa del árbol. * Botón   + Presionable: puede ser presionado por el jugador y activa un mecanismo de puente. * Puerta   + Abrir: se abre dependiendo de si el usuario cuenta con la llave necesaria para dicha acción. |

La interacción de las mecánicas del jugador con los enemigos y los diferentes objetos genera la composición de la jugabilidad de todo el juego, que es recorrer el mapa derrotando enemigos y obteniendo los objetos necesarios para pasar el nivel actual.

Esta jugabilidad fue la que permitió que el juego pueda ser utilizado como un medio para el aprendizaje de estructuras secuenciales, selectivas e iterativas; los aprendizajes fueron embebidos dentro de cada nivel y su respectiva solución, sin ser evidente a simple vista para el jugador, esto con la idea de que se centrara en completar el juego y que desarrollar el pensamiento computacional fuera una consecuencia de este objetivo.

### 4.1.2. Retos

Se definieron retos principales que el jugador deberá enfrentar, estos son conseguir todas las llaves del mapa para abrir las puertas y así pasar al siguiente nivel, derrotar a los enemigos que se encuentren en el mapa para no recibir daño, evitar ser tocado por los enemigos para no perder vida, tratar de terminar el nivel lo más deprisa que se pueda y recoger objetos valiosos del mapa. Este último es opcional y no es necesario para pasar al siguiente nivel; pero de realizarlo permitirá que el jugador complete totalmente el nivel, sirve como un incentivo para explorar el mapa y completar el juego en un 100%; y en caso de que el jugador reciba mucho daño de los enemigos se retornará al punto de último punto de guardado del jugador.

El jugador deberá tener cuidado de los enemigos ya que en caso de recibir mucho daño de los enemigos se retornará al punto de último punto de guardado del jugador., se optó por recargar una porción de la vida total del jugador cada cierto tiempo, este se recarga cada 3 minutos. También deberá tener cuidado con el tiempo para completar el nivel debido a que los enemigos reaparecerán de manera automática después de 10 minutos de haberlos derrotado.

### 4.1.3. Escenario

Los escenarios definidos son de tipo 2D con un enfoque permanente en el personaje principal, esto quiere decir que el escenario se moverá con los movimientos del jugador, como si una cámara siempre estuviera siguiendo al jugador.

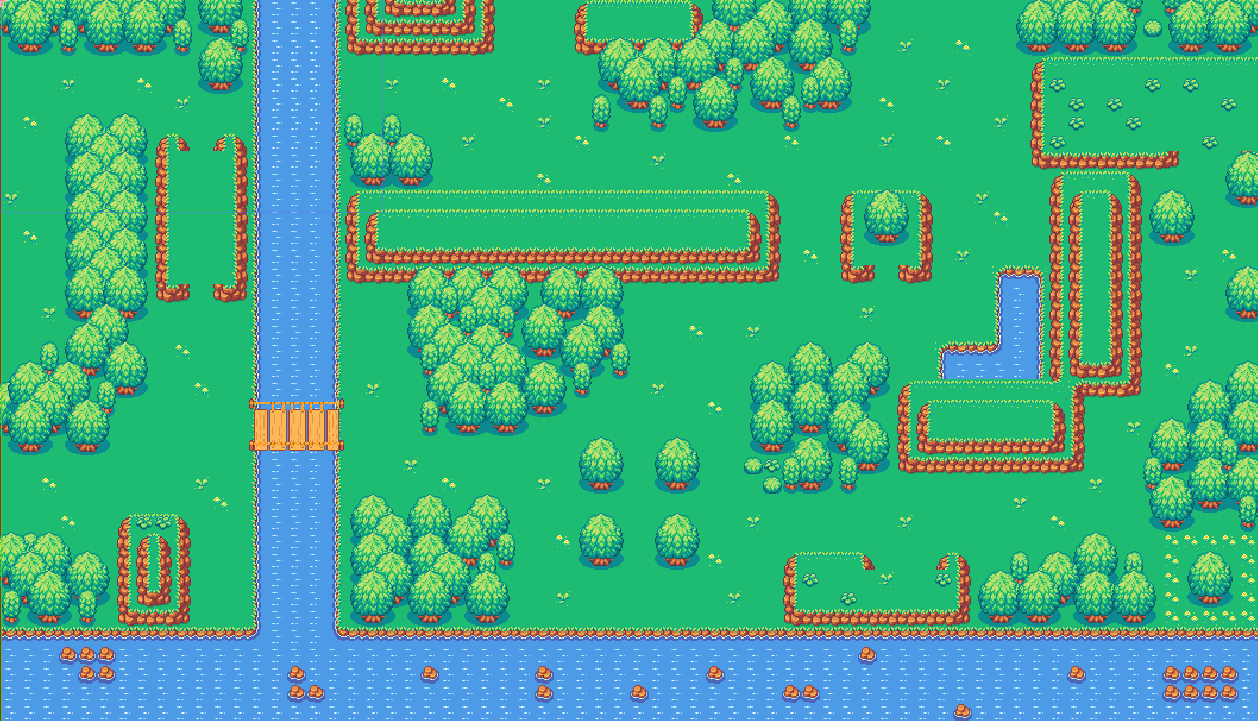
 

Figura 5. Ambientación del escenario Figura 6. Ambientación del escenario

Todos los escenarios creados tienen la misma ambientación de tipo bosque, con montañas repartidas en el mapa y ríos que evitarán el paso del jugador; en las Figuras 5 y 6 se puede observar el tipo de ambientación de los escenarios. Estos están conectados de manera que forman un mapa más grande que representa el mundo del juego.

### 4.1.4. Caracterización de personajes

#### 4.1.4.1. Personaje principal

Para el diseño del personaje principal se tomaron en cuenta características necesarias para ser visualmente agradables para el jugador, ya que es un juego dirigido a niños; se buscó un diseño que les gustara y les pareciera bonito, de esa manera estarían interesados en jugar con él.

Se indagaron diferentes opciones hasta que se encontró un diseño que cumplía con las características buscadas (véase la Figura 7). Este personaje se puede describir como un perrito vestido con playera, zapatos y con una capa; el cual fue nombrado como *Shushu*.



Figura 7. Diseño del personaje principal

Tiene la particularidad de ser un animal con el que todos los niños están familiarizados, además de ser un diseño muy bien logrado; el perrito representa un amigo fiel con el que siempre podrán contar, que siempre los defenderá y siempre hará todo para hacerlos felices, además la capa hace que se vea más llamativo y simule ser un héroe para los niños.

#### 4.1.4.2. Enemigos

Se decidió que el diseño del enemigo fuera sencillo, pero con un diseño intimidante y que no quedara fuera de lugar con el estilo del escenario y con el diseño del jugador; después de buscar se decidió por utilizar el diseño de la Figura 8. Como se puede observar, el diseño es parecido a un monstruo de color rojo, esta característica también permite que resalte de los fondos del nivel y sea más sencillo localizarlo, también se eligió debido a que es un poco más grande que el jugador haciendo que sea más intimidante para él. Además, debido a la ambientación de la historia, es parte de los peligros que habitan el bosque donde nuestro personaje principal se pierde y tendrá que enfrentar para poder regresar con su familia.



Figura 8. Diseño del enemigo

### 4.1.5. *Feedback*

Se optaron como mecanismos de *feedback* los diferentes elementos de la interfaz de usuario; estos avisan al usuario sobre el estado de salud del personaje, así como los objetos recogidos a lo largo del nivel. Estos elementos se pueden observar en la **figura 5**.

**Insertar imagen de la interfaz de usuario**

### 4.1.6. *Debriefing*

Este elemento como tal no forma parte del juego, pero se decidió mostrar un mensaje para el usuario cuando finalice cierta cantidad de niveles (véase **figura 6**), esto para explicarle de manera resumida el conocimiento que acaba de aplicar para lograr pasar el nivel.

**Insertar imagen de mensaje**

### 4.1.7. *Storyline*

Se creó una historia corta que explicara y fundamentara los objetivos del juego, y permitiera que el jugador tuviera una motivación para terminar el juego. La historia se desarrolló en torno al personaje principal. La premisa de la historia es la siguiente: *un día Shushu salió con su familia a un picnic en el bosque, sin embargo, Shushu se distrajo y se separó de su familia por lo que olfateo su aroma para poder encontrarlos; lo que no esperaba es que en el camino tendría que cruzar la parte más profunda del bosque y que se encontraría con varios restos y dificultades que tendría que superar para poder regresar con ellos.*

Como se puede observar la premisa es sencilla y fácil de comprender, de esa manera se puede lograr una fácil comprensión de la situación, además de que le da al jugador una razón y explicación del juego y porqué se debería de completar.

### 4.1.8. Recompensa

Como se mencionó anteriormente en la sección de retos, el jugador tiene la opción de recoger todos los objetos valiosos del nivel, al hacer esto se le premiará con mejoras para su personaje, dependiendo de la cantidad de objetos que recoja entonces se le asignará una recompensa; se puede ver la lista de recompensas basado en el número de objetos en la **tabla 2**.

|  |  |
| --- | --- |
| Cantidad de objetos valiosos | Recompensa |
| 6 | Aumento de velocidad del personaje al moverse por el mapa. |
| 9 | Armadura que reduce el daño de los enemigos. |
| 15 | Espada de metal que hace más daño y no se rompe. |

Tabla 7. Recompensas

Como se pudo ver en la tabla anterior, las recompensas ayudan al jugador a pasar más fácil los niveles del juego, además de que tendrá la posibilidad de terminar el juego con el 100% de progreso.

### 4.1.9. Persistencia

La persistencia en el juego se acató mediante un punto de guardado automático que se habilita en diferentes momentos de la partida, estos son al momento de que el jugador recoge una llave o un objeto valioso, o cuando el jugador termina el nivel en el que se encuentra. Esto le permitirá retomar el juego en un punto avanzado de la partida sin perder los objetos ya recogidos con anterioridad. Este sistema de guardado será almacenado en la memoria del teléfono celular del jugador.

### 4.1.10. Interfaz del juego e interacción

La interfaz del juego se definió con elementos que permiten que el usuario pueda revisar su estado actual (cuántos objetos ha recogido, cuánta vida le queda y cuánto durará su arma actual). La interacción se realiza mediante botones que se visualizan en la pantalla del teléfono, este se compone de una cruceta de direcciones y un botón para atacar, estos son los que el jugador utilizará para realizar cualquier acción dentro del juego. Se puede ver la interfaz definida en la **figura 7**.

**Insertar imagen de la interfaz**

## 4.2. Planificación

Esta fase del desarrollo se utilizó para medir la cantidad de tiempo dedicada a cada parte del juego, los tiempos de inicio y de finalización de cada etapa. Siguiendo la metodología SUM para el desarrollo de videojuegos, se tomaron como fases: concepto, elaboración, beta, cierre y gestión de riesgos. Los tiempos dedicados a cada etapa se pueden apreciar en la **figura 8**.

**Insertar diagrama de gantt**

## 4.3. Elaboración

En esta fase del proyecto se llevó a cabo todo el bosquejado y programación del juego, esta última se hizo siguiendo una planificación, seguimiento, desarrollo y cierre por iteraciones presente en la metodología SUM para el desarrollo de videojuegos. Cada iteración tenía una duración de dos semanas, pero también tenían cierta dependencia con los objetivos planteados en la planificación por lo que algunas veces duraron menos tiempo.

### 4.3.1. Planificación de la iteración

Este punto se ejecutó mediante el cumplimiento de objetivos las cuales representaban una parte funcional del resultado final del videojuego y que se descomponían en tareas específicas que juntas completaban el objetivo, todo esto mientras que de manera paralela y siempre presente se llevaba a cabo la fase de gestión de riesgos, con la cual se detectaban los problemas que surgieran en el momento, se evaluaban y se planteaban diferentes alternativas de solución.

El objetivo planteado en la planeación de la iteración se descomponía en tareas que se desarrollaban de manera individual y que al final se unían para generar una parte funcional del juego. Los objetivos requerían cierta cantidad de tiempo y esfuerzo dependiendo de la complejidad que implicaba desarrollarlas. La ejecución de un objetivo se iba decidiendo conforme se terminaba otra, se revisaba el estado actual del juego y se evaluaba que partes del juego seguían faltando para terminar el desarrollo, de igual manera se evaluaba cada objetivo para definir su complejidad y prioridad, si un objetivo era muy complejo entonces este se descomponía en sub objetivos que implicaban tener tareas aisladas entre sí. Los objetivos de toda la fase de desarrollo se ejecutaron en el orden que se puede observar en el diagrama de la **figura 9**.

**Adjuntar diagrama de Gantt o algún otro que liste los objetivos para el desarrollo**

### 4.3.2. Seguimiento de la iteración

Esta parte se llevó a cabo cada que se finalizaba un objetivo, esto con la finalidad de verificar como avanzar con la iteración actual; dependiendo de la finalización de los objetivos planteados para la iteración, esta podía terminar antes de completarse las dos semanas definidas para cada una, en estos casos entonces se pasaba a la siguiente planeación de iteración, para así definir los siguientes objetivos. Si era el caso contrario, y quedaban objetivos por cumplir, entonces se definían las tareas para el siguiente objetivo de la iteración y así proceder normalmente con el desarrollo de ese objetivo.

### 4.3.3. Desarrollo de características

Fue la parte más laboriosa de todo el desarrollo, ya que fue donde se llevó a cabo la programación del juego en cada iteración. En esta parte fue que se desarrolló cada objetivo planteado; a continuación, se describirá el trabajo realizado con cada objetivo.

#### 4.3.3.1. Diseño de niveles

El diseño de niveles se realizó mediante un bosquejo rápido de cada nivel que conforma el juego. Se diseñó de tal manera que al pasar un nivel se incentivara el desarrollo del pensamiento computacional. Esta etapa no requirió de ningún tipo de codificación ni la creación de un diseño refinado, solamente se requería de una idea inicial para tener un punto de partida, y esta se fue refinando en la etapa de programación de nivel.

Se tomaron en cuenta la posición inicial del jugador, la posición de la meta o salida del nivel, la posición de los *ítems* del juego, la posición de árboles normales y árboles con *ítems*, y los elementos que conforman la naturaleza que ambienta el juego. La combinación de la posición de los elementos y los árboles definen el camino que el jugador puede seguir en el nivel. Ejemplos de bosquejos del diseño de niveles se pueden observar las **figuras 10 y 11**.

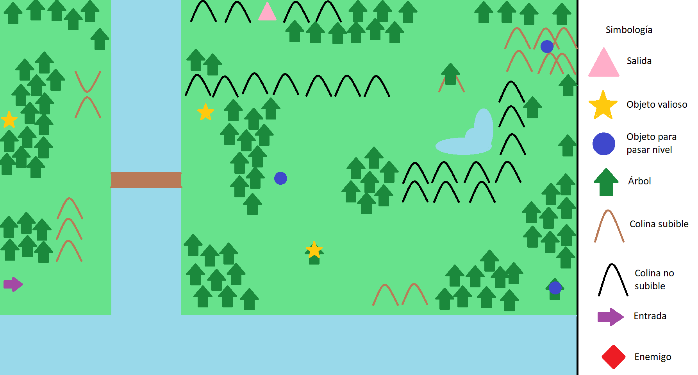
 

Figura 9. Borrador del nivel 1 Figura 10. Borrador del nivel 2

Para el desarrollo del juego se diseñaron ocho niveles que el jugador tendrá que superar para terminarlo en su totalidad. Como se mencionó al principio, cada nivel fue diseñado con la intención de enseñar un tipo de estructura que fomenta el aprendizaje del pensamiento computacional.

Los niveles 1, 2 y 3 se diseñaron como niveles de introducción a la jugabilidad y mecánicas del juego, así como en la enseñanza de estructuras secuenciales; esta enseñanza se puede notar en uno de los retos del juego, que es buscar las llaves para abrir la puerta que le permitirán pasar al siguiente nivel. La enseñanza se encuentra en los pasos a seguir del jugador, explorar el mapa, derrotar enemigos, recoger la llave y abrir la puerta. Estos niveles se pueden observar en las **figuras 12, 13 y 14**. Los enemigos aparecen desde el nivel 3 para introducir la mecánica de golpe al jugador, hasta el nivel final del juego donde cada vez habrán más enemigos.

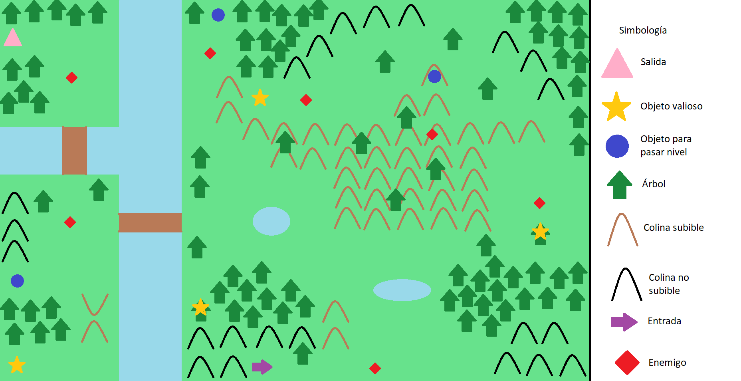


Figura 11. Borrador del Nivel 3

Los niveles 4 y 5 se enfocan en la enseñanza de estructuras selectivas, y se mantuvieron presentes las estructuras secuenciales dentro de estos; se añade una situación que en los niveles anteriores no había, y es la introducción de un puente incompleto y que no se puede cruzar, esta situación se resuelve activando el mecanismo al presionar el botón cercano al puente que hace aparecer este y permite cruzarlo. La enseñanza de estructuras selectivas se encuentra presente en la decisión tomada por el jugador frente a esta situación donde se pregunta qué hacer si el puente está incompleto y qué hacer cuando está completo. Estos niveles se pueden observar en las **figuras 15 y 16**.



Figura 12. Borrador del Nivel 4

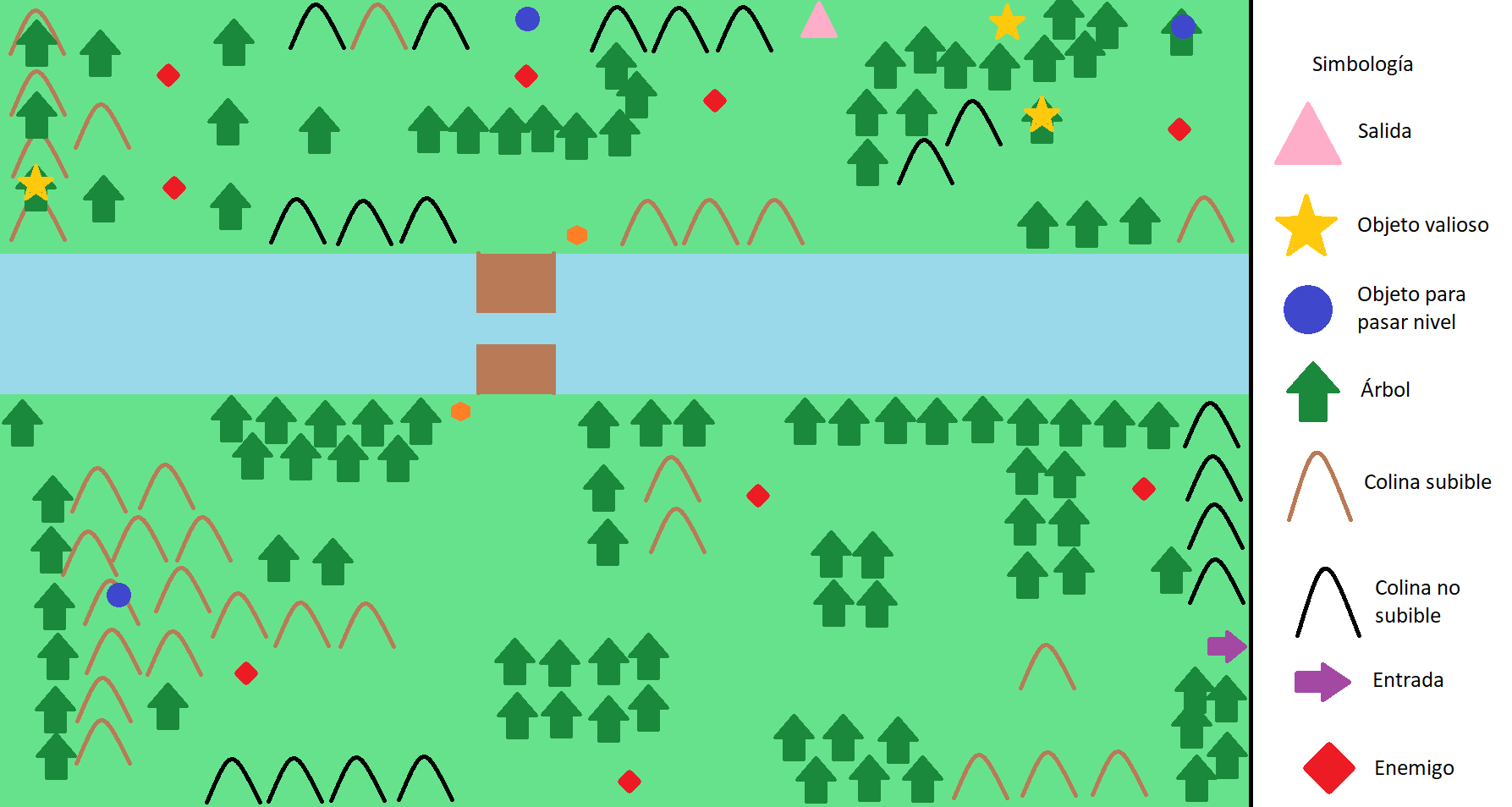


Figura 13. Borrador del Nivel 5

Los niveles 6, 7 y 8 introducen la enseñanza de estructuras cíclicas, siempre manteniendo lo aprendido en niveles anteriores, como las estructuras secuenciales y selectivas. En estos niveles hay más de una puerta, las cuales están dispuestas en un orden específico por lo que el jugador tendrá que buscar las llaves en ese mismo orden; para asegurar que el jugador recoja las llaves en el mismo orden, se bloquearon las posibles entradas que el jugador puede tomar para recoger una llave. Este camino se desbloquea cada que se abre una puerta. Las estructuras cíclicas se encuentran en la actividad de volver a buscar la llave que abre la siguiente puerta hasta que todas las puertas estén abiertas. Estos niveles se pueden observar en las **figuras 17, 18 y 19**.

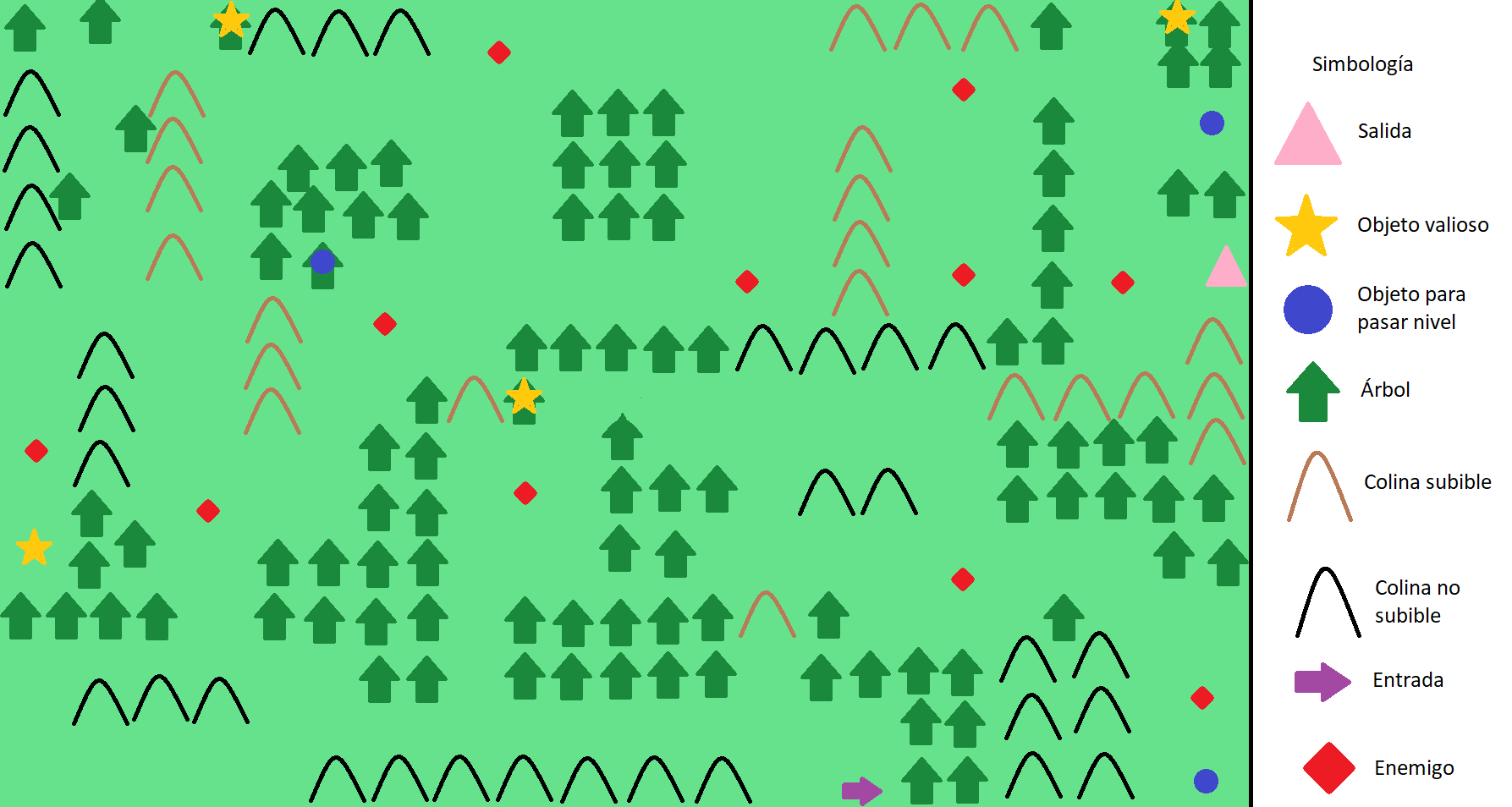


Figura 14. Borrador del Nivel 6



Figura 15. Borrador del Nivel 7

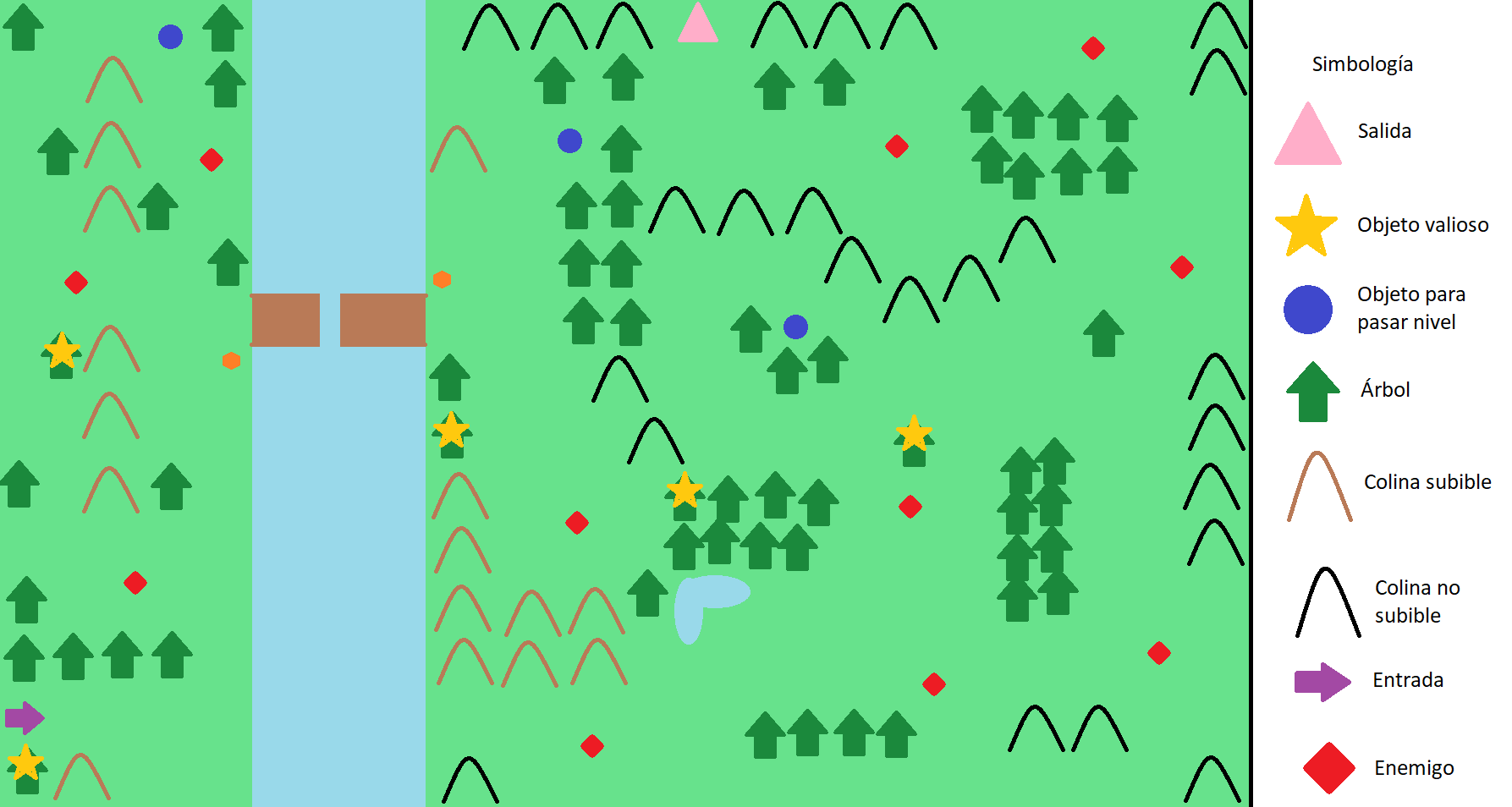


Figura 16. Diseño final Nivel 8

#### 4.3.3.2. Diseño de interfaz de usuario.

En esta etapa, al igual que el diseño de niveles, se realizó un bosquejo de la parte que el jugador vería para seguir su estado actual en el nivel, este se conforma por los siguientes elementos visuales: la cantidad de vida restante del jugador, los *ítems* recogidos, la vida restante de su arma y los botones en pantalla que el jugador requiere para darle instrucciones al personaje principal.

La interfaz también contiene elementos necesarios durante el tiempo no jugable, como lo son la pantalla inicial, los menús de juego que permiten al jugador iniciar el juego y el menú de pausa. Estos se pueden ver en las **figuras 20, 21, 22, 23, 24, 25 y 26 (una imagen por cada elemento de la interfaz)**.









#### 4.3.3.3. Desarrollo de niveles.

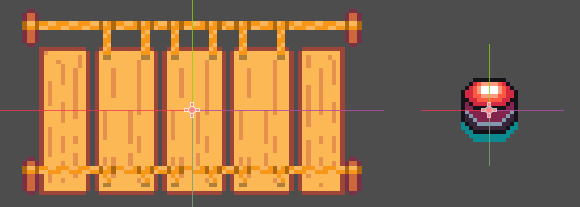
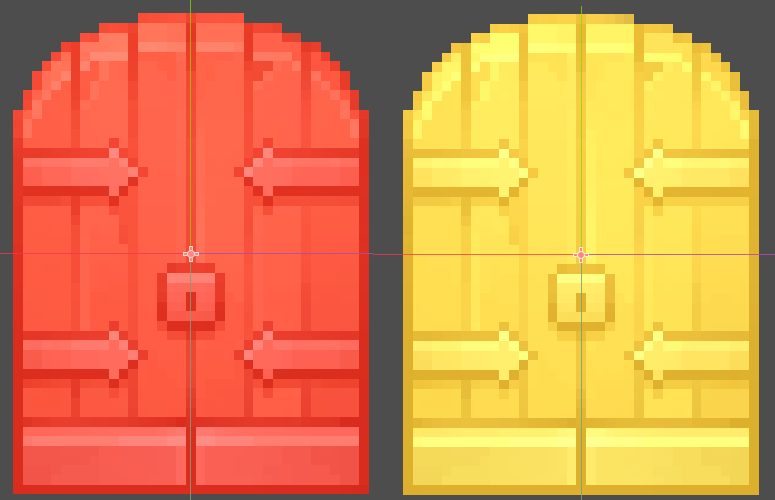
Teniendo los bocetos de niveles definidos, se buscaron diferentes *game assets* que se adaptaran al boceto inicial. Dhule (2022) describe a los *assets* como los elementos artísticos que conforman cualquier videojuego, estos son la apariencia gráfica de los personajes, enemigos, objetos, escenarios, letras, botones, música, efectos de sonido, entre otros componentes visuales que pueden conformar el juego.

Para la programación de niveles se buscaron *assets* gráficos para representar los escenarios plasmados en el boceto anterior de diseño de niveles. Se optó por elegir un conjunto de imágenes ambientado en un bosque diseñado para juegos de tipo RPG (véase la **figura 27**). El conjunto incluye más elementos gráficos, pero sólo se utilizaron los que representan la naturaleza para el escenario (árboles, hierba, agua, ríos, rocas). Así mismo se utilizaron más paquetes, **figura N**, que incluían otro tipo de elementos gráficos, los cuales se combinaron para generar un conjunto de elementos que se utilizaron en todos los niveles del juego.





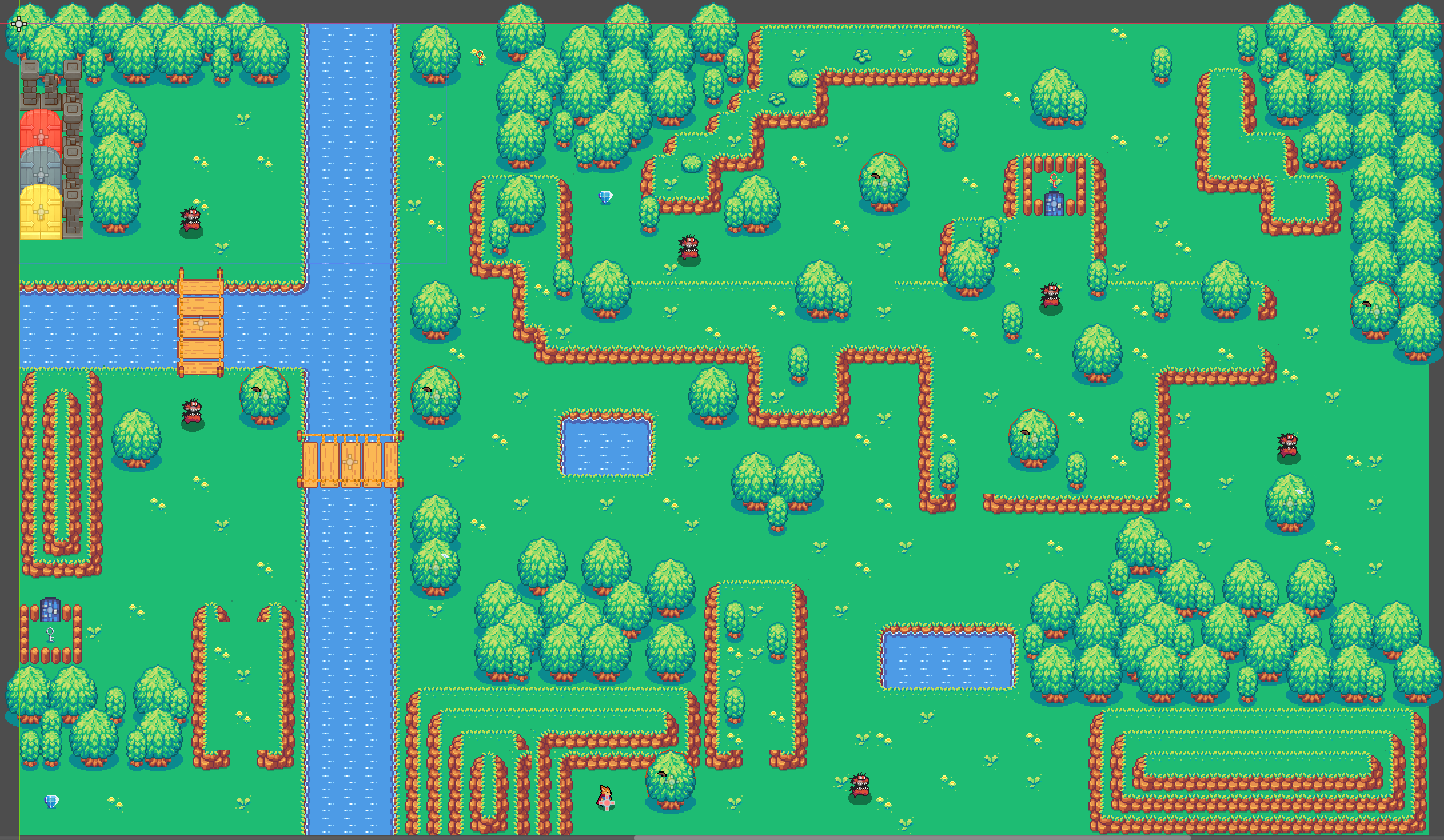
También debido a la necesidad específica de *assets* para situaciones más puntuales se decidió crear nuevos elementos gráficos y modificar algunos otros (puentes, botones, puertas, árboles y otros objetos), algunos ejemplos se pueden ver en las **figuras 28, 29 y 30**.

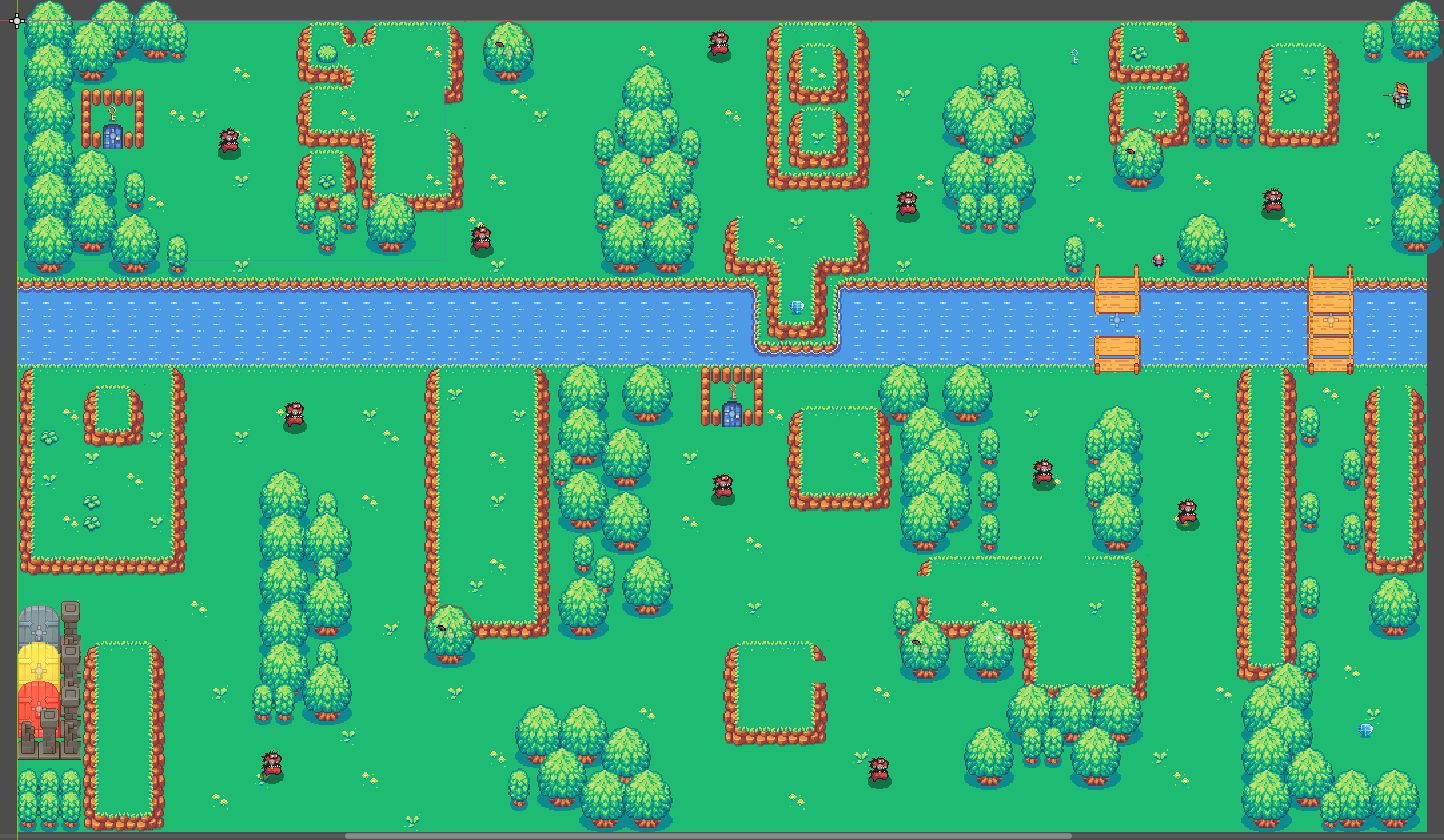
  

Habiendo elegido los *assets* que el juego utilizaría, estos se cargaron dentro de *Godot Engine* para su uso en el entorno de programación. Se utilizaron los elementos del escenario (árboles, montañas y hierba) y junto a una funcionalidad de *Godot Engine* se creó un *TileMap*, esta permite crear escenarios de manera rápida clonando los *assets* utilizados en el *tilemap*, utiliza un sistema de *clickear* en un punto del lienzo y arrastrar un *asset* seleccionado, con esto se pueden crear escenarios grandes en muy poco tiempo; y habiendo definido un bosquejo de todos los niveles, la tarea de crearlos en *Godot Engine* fue rápida. El resultado de esto se puede observar en la **figura 31**.

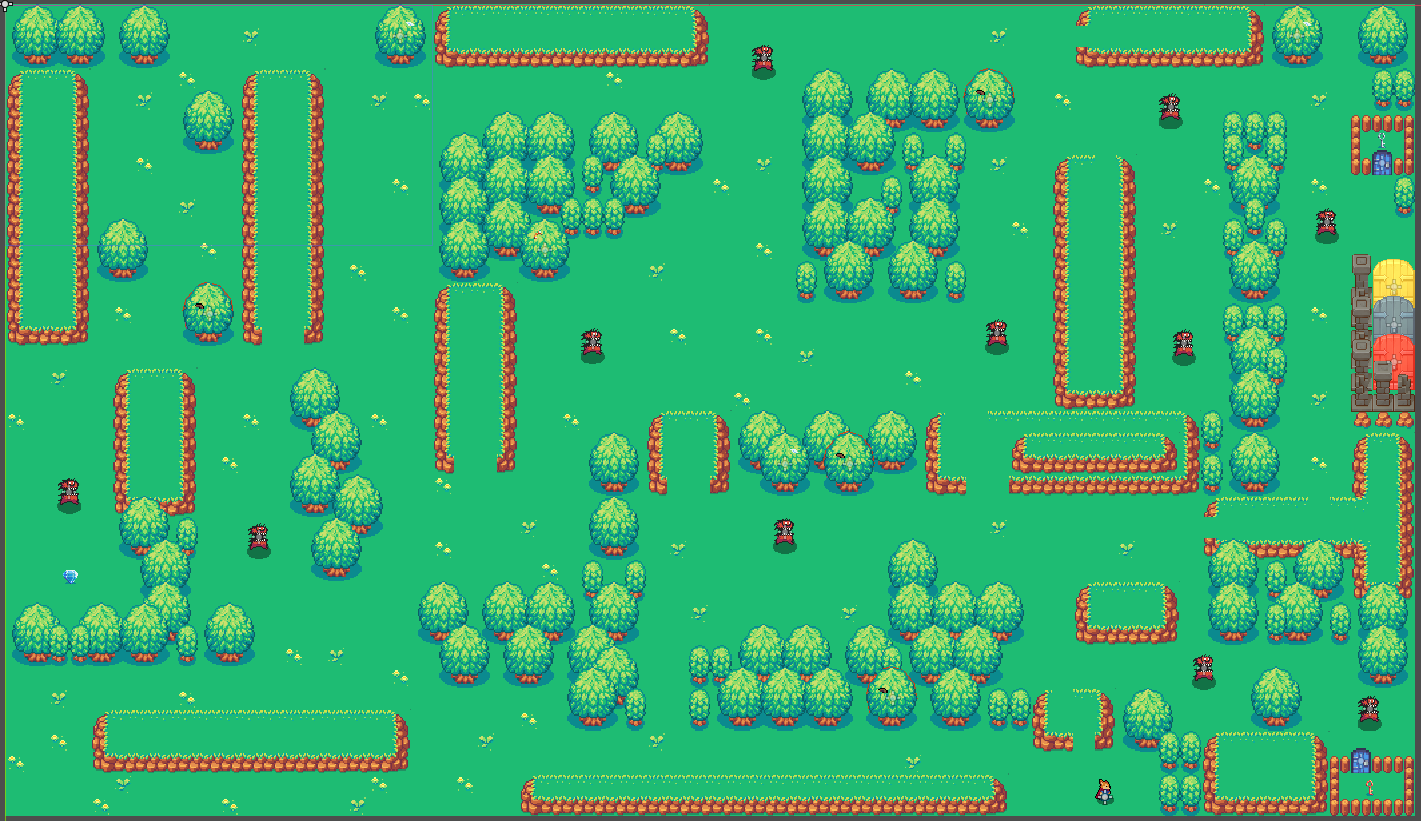


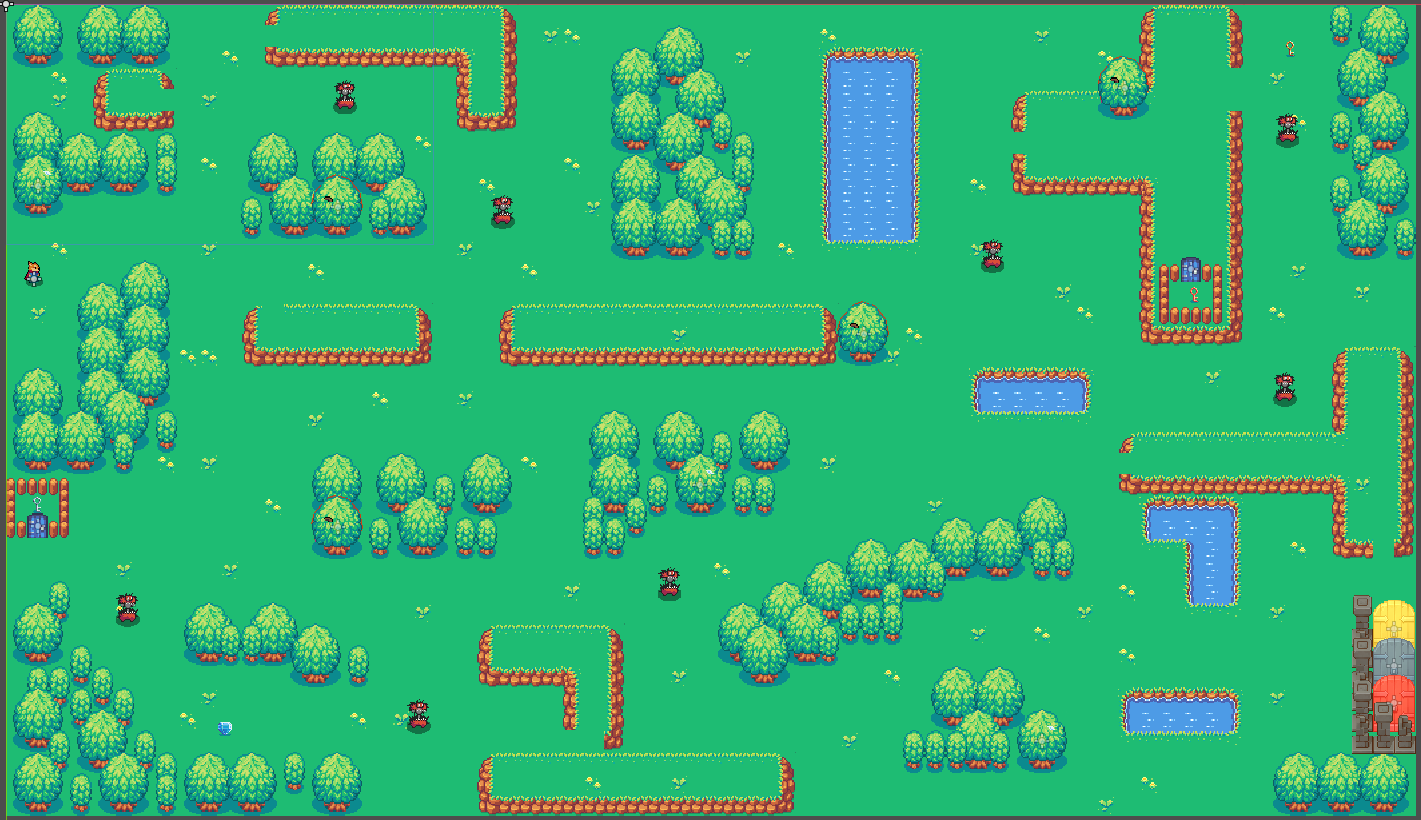


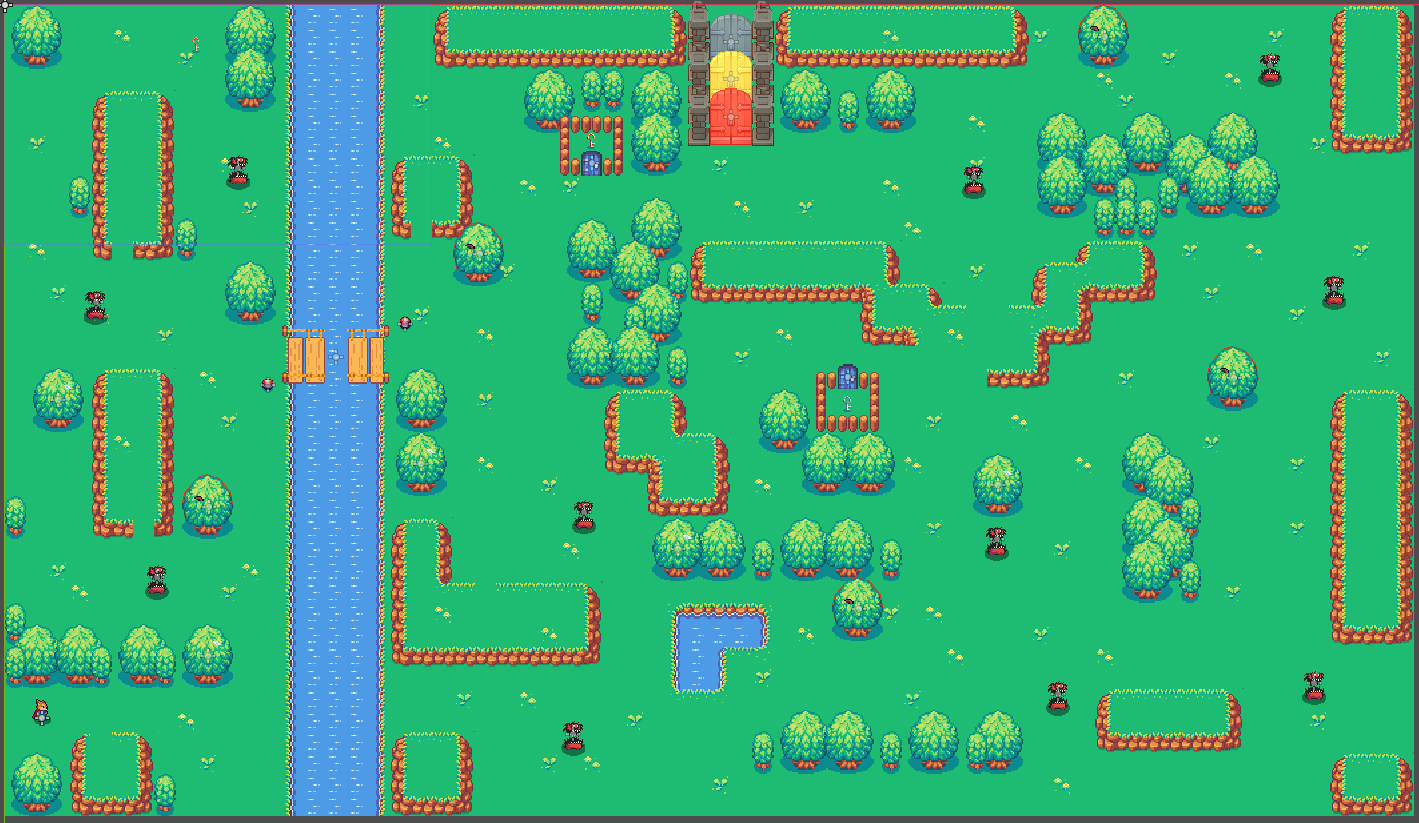












#### 4.3.3.4. Programación de personajes y objetos

Al igual que los objetos del escenario, los *assets* de los personajes, enemigos y objetos con los que el jugador puede interactuar se cargaron dentro del entorno de programación. Cabe resaltar una gran diferencia entre *assets* de personajes u objetos (como lo son los personajes principales, enemigos o cualquier otro elemento que tenga movimiento o cualquier otro tipo de animación) de los *assets* para los escenarios, ya que estos requieren de un conjunto de imágenesdonde el personaje u objeto se encuentre en diferentes posiciones para que así, con el uso de técnicas de animación típica de dibujos animados, se pueda simular el movimiento de dicho elemento. Esta particularidad hizo más complicada la tarea de encontrar *assets* que cumplieran con los criterios escogidos para los personajes y enemigos. En la **figura 32** se puede apreciar el *asset* completo del personaje principal, este se compone de muchas imágenes que juntas representan todas las animaciones que el personaje principal tiene. Cada animación requiere de una cantidad de imágenes por lo que, si el personaje tiene muchas animaciones diferentes, se traduce en más imágenes que el *asset* debe de contener. El *asset* completo del jugador se compone de 64 posiciones diferentes que representan una parte de cada animación del jugador.



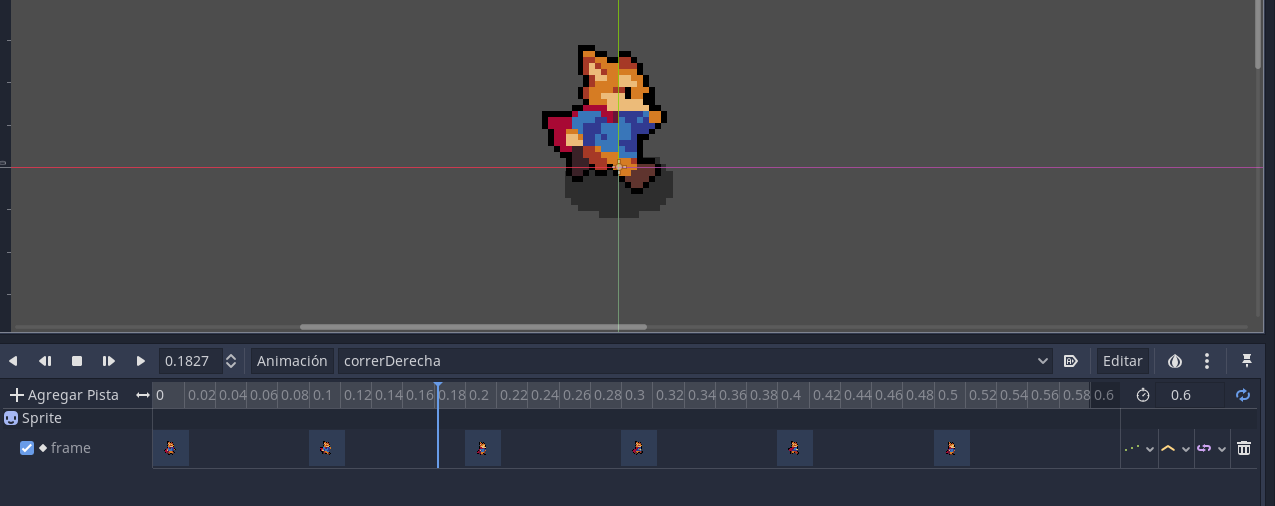
En la **Figura N** se pueden observar las posiciones usadas en la animación de atacar hacia la derecha.



*Godot Engine* tiene dos tipos de desarrollo de juegos, el primero es mediante el método de arrastrado de bloques que se juntan con otros para hacer funcionar el juego, el segundo es mediante código de programación, el motor cuenta con su propio lenguaje llamado *GD Script* el cual guarda un gran parecido al lenguaje de programación *Python*; para toda la programación del juego se utilizó *GD Script* con un paradigma orientado a objetos, esto trata a todos los *assets* del juego como objetos con propiedades individuales e independientes, lo cual permite que cuando se modifique un objeto, todos los demás del mismo tipo tendrán las mismas propiedades y modificaciones.

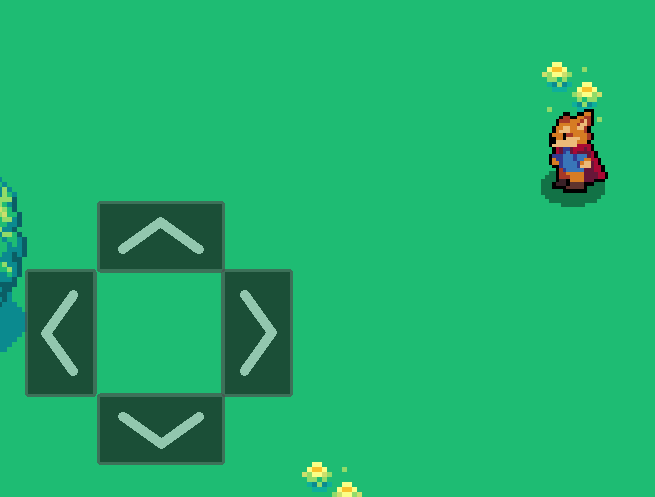
##### 4.3.3.4.1. Programación de personaje principal

Primero se programaron las animaciones del personaje principal; *Godot Engine* cuenta con una herramienta para realizar animaciones con *assets* cargados en el entorno de programación (véase la **Figura N**). Las diferentes animaciones del personaje principal son: correr hacia adelante, hacia atrás, hacia arriba, hacia abajo; atacar desarmado hacia adelante, hacia atrás, hacia arriba, hacia abajo; atacar con un arma hacia adelante, hacia abajo, hacia arriba y hacia abajo; las posiciones de estar quieto también cuentan como animaciones dentro de *Godot Engine* por lo que se suman las animaciones de mirar hacia adelante, hacia atrás, hacia arriba y hacia abajo.



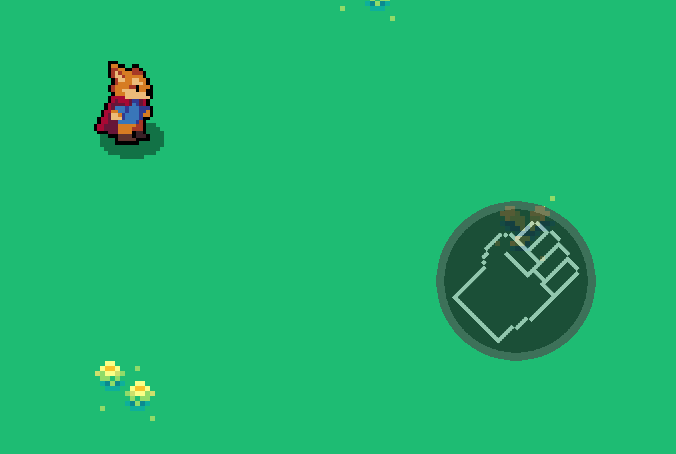
Una vez hechas las animaciones, era necesario especificar en qué momento debía reflejarse cierta animación en el juego; por ejemplo, cuando se dé la instrucción de caminar hacia adelante se debe mostrar la animación de caminar hacia adelante. Se separaron las animaciones en dos tipos (en animaciones de movimiento y animaciones de golpeo) para trabajar de manera ordenada; se procedió con las de movimiento que se activan con los botones de dirección adelante, atrás, arriba y abajo. Esto se hizo mediante la programación de una instrucción activada por la pulsación de una tecla en la interfaz del juego.

Cabe recalcar que las animaciones por sí solas no hacen que el jugador se pueda mover, por lo que también se agregó la instrucción en el mismo bloque de código al ejecutarse la animación. Este mismo proceso se realizó para todas las animaciones de movimiento con sus botones de dirección correspondiente. Cuando el jugador deja de presionar el botón de movimiento, el personaje regresa a su posición de quieto en la dirección de movimiento anterior, el resultado de todo esto se puede observar en las siguientes **figuras 33, 34 y 35**.



Después de programar el movimiento del jugador, se procedió con la programación de la mecánica de golpeo. Este proceso se hizo de manera similar al de movimientos, con la diferencia de que se utiliza un botón de la interfaz de usuarios dedicado para esta acción y se ejecutan las animaciones de golpeo, al finalizar el golpe la animación de quieto se ejecuta al instante; la dirección del golpeo y la dirección quieta del personaje principal dependen de la dirección de movimiento hecha antes de realizar el golpe. Esta acción se puede observar en las **figuras 36, 37 y 38**.

###### Gestión de riesgos

Terminada la programación de las animaciones principales, se notó un problema y es que cuando el jugador chocaba con un objeto del escenario ya sea este un árbol, una llave, una roca, el río o cualquier otro elemento dentro del escenario, el jugador podía atravesar estos como si no existieran (véase la **Figura N**). Esto se debió a la falta de *objetos de colisión* en el juego. Dhule (2022), explica las colisiones como cuerpos con una forma definida cuyo comportamiento se rige por la física del mundo real. Por ejemplo, si corremos hacia una pared sin parar, al entrar en contacto con la pared sentimos este contacto por ser ambos objetos con un cuerpo y una masa, al entrar en contacto podrían suceder diferentes cosas como por ejemplo parar de manera abrupta o podría ser posible tirar la pared; al entrar en contacto con la pared podemos decir que colisionamos o chocamos con la pared; el mismo comportamiento se aplica a los videojuegos, donde las colisiones siguen este mismo principio de sentir el contacto entre dos objetos y que algo suceda al chocarse.



Habiendo identificado el problema, se procedió primero a crear objetos de colisión para todos los objetos del mapa, *Godot Engine* cuenta con su propio sistema de creación y manipulación de colisiones el cual se configura por medio de parámetros en el objeto y cuyo comportamiento al colisionar con otros objetos se define con código *GD Script*. Las colisiones de los objetos que componen los escenarios se pueden observar en la **figura 39**.



Las colisiones en *Godot* *Engine* son representadas con contornos o figuras de color azul, estos pueden tener la forma que el programador requiera, desde polígonos sencillos como cuadrados, rectángulos o circunferencias, hasta figuras que se acoplen a algún *asset* en particular.

Al personaje principal también se le asignó una colisión para que este pueda tener contacto con los objetos del entorno. De esta manera, con ambos elementos del juego con sus respectivos objetos de colisión y ahora contrario a la prueba anterior, el jugador ya no podía traspasar los árboles, rocas, el río y otros elementos del escenario. Los resultados se pueden observar en la **figura 40**.



Otro problema surgió con este cambio y es que el jugador ahora podía salir del escenario en las partes donde no había elementos con los que el jugador pudiera colisionar (véase **figura 40**). Para resolver esto se creó un objeto de colisión que delimitó el escenario, entonces cada que se intentaba pasar los límites definidos del escenario, era detenido por una pared invisible con la que chocaba (véase **figura 41**).

##### 4.3.3.4.2. Programación de cámara

Uno de los objetivos definidos en la etapa de concepto era recorrer el mapa para encontrar objetos por lo que el jugador no debería poder ver el mapa completo sino una sola parte del total de este. Para lograr eso se utilizó un recurso con el que *Godot Engine* también cuenta y es la creación de cámaras que enfocan ciertos lugares; en este caso se utilizó una cámara que seguía constantemente al personaje principal, en la **figura 42 y 43** se puede ver la cámara empleada y cómo esta sigue al jugador a todos lados.

##### 4.3.3.4.3. Programación de llaves

Habiendo finalizado con los objetos del escenario y por ahora con las mecánicas del personaje principal, se procedió a programar las mecánicas de los objetos del escenario con los que el usuario podrá interactuar, estos son llaves, objetos valiosos, árboles y puertas.

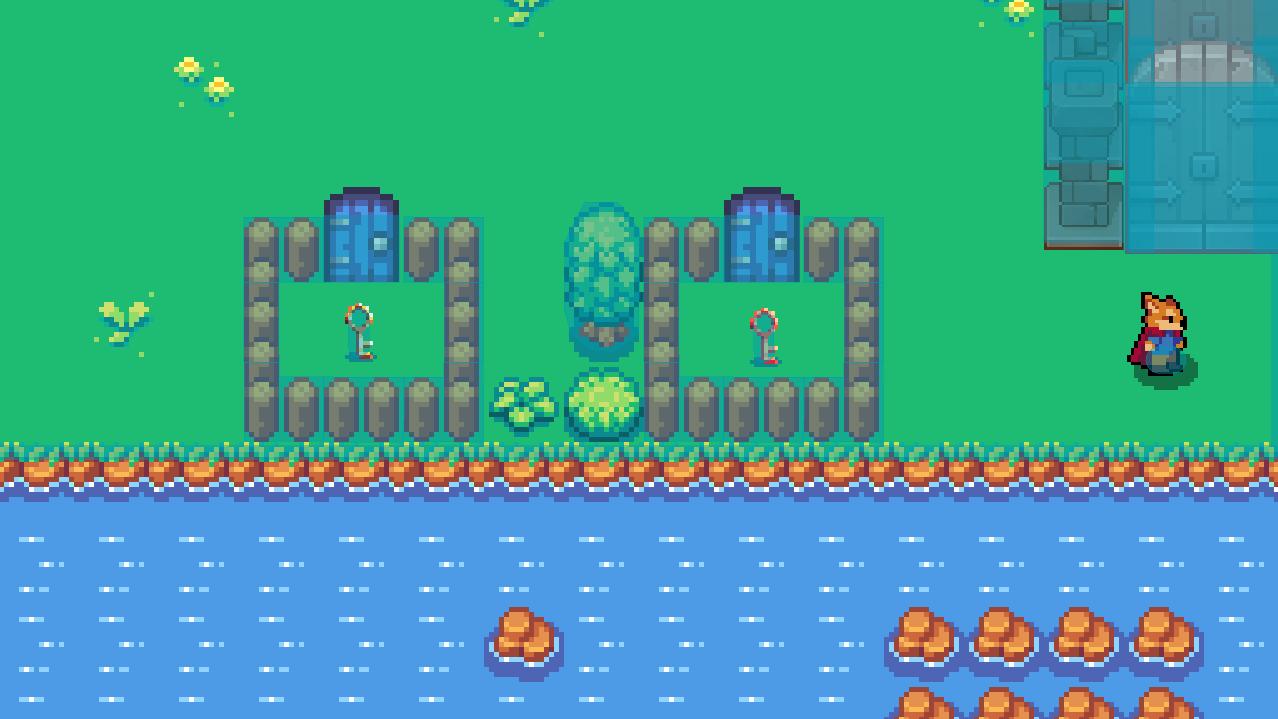
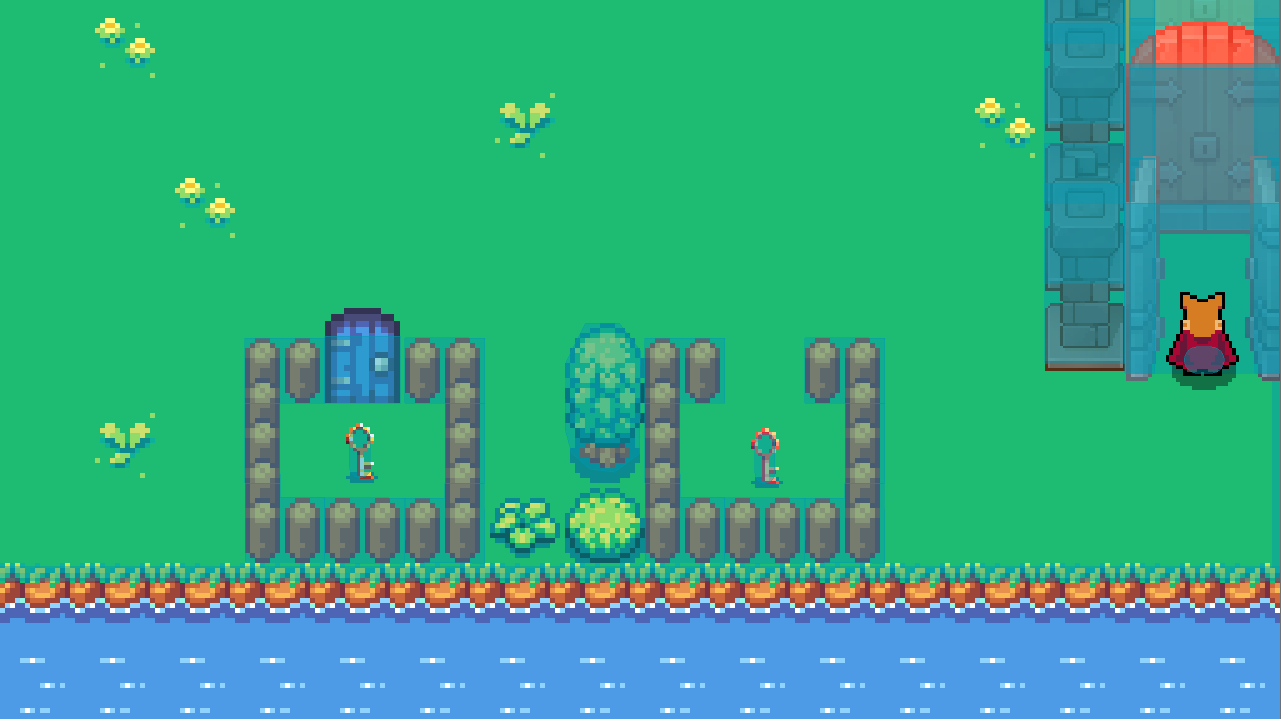
Se inició con la programación de las llaves las cuales tienen la mecánica de ser recogidos por el jugador al pasar encima. Para esta mecánica se utilizó un engaño visual para el jugador, él vería como si el personaje principal aparentemente guardara la llave, pero internamente en el juego la llave deja de existir en su totalidad, es decir es eliminado del nivel actual; a pesar de esto en las propiedades del objeto del jugador se deja constancia de haber recogido la llave. Esto se ejecuta en el momento en el que el objeto de colisión del jugador hace contacto con el objeto de colisión de la llave, el mismo comportamiento aplica para todas las llaves del juego y también para los objetos valiosos y otros como las ramas y las puertas, en las **figuras 44, 45, 46 y 47** se puede observar este acontecimiento en las llaves y los objetos valiosos

##### 4.3.3.4.4. Programación de puertas

La diferencia de esta mecánica en las puertas es que cuando la colisión entre el jugador y la puerta se da, esta última no desaparece, sino que se reemplaza la imagen de puerta cerrada con una puerta abierta, haciendo parecer que se abre. Para que esto suceda es necesario que el jugador cumpla con la condición de haber recogido anteriormente la llave del mismo color que la puerta. Solamente entonces se abre cuando ambos objetos colisionan como se puede ver en las **figuras 48, 49 y 50**.

###### Gestión de riesgos

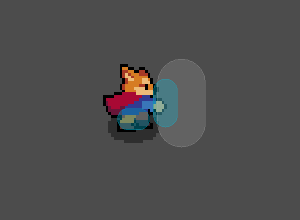
Otro problema se detectó al momento de haber terminado con la programación de las mecánicas de las puertas y que no estaba siendo tomado en cuenta al momento de planear este objetivo; y es que se suponía que cuando el jugador lograra abrir todas las puertas automáticamente debería de pasar al siguiente nivel, pero esto no pasaba.

*GD Script* de *Godot Engine* funciona con métodos ya definidos en el lenguaje de programación y hacen que la tarea de cambiar de niveles a otros se puede hacer sin mayor problema, es necesario ejecutar el método *get\_tree().change\_scene("res://Niveles/Nivel\_2.tscn")* donde el texto entre comillas indica donde se encuentra el archivo del nivel al que se pasará.

##### 4.3.3.4.5. Programación de árboles

###### Gestión de riesgos

En el caso de los árboles algunos tienen la mecánica de poder ser golpeados y tirar un objeto que tengan encima, esto con la finalidad de que el jugador pueda recoger este objeto y utilizarlo en el juego. Esta mecánica requirió de un elemento extra para que pudiera funcionar de la manera que se deseaba y es que era necesario que el golpe que el personaje da en el juego también tuviera una colisión para que esta pudiera reaccionar con la colisión del árbol y así reconocer que hubo un contacto con el golpe del jugador y el objeto que sufrió el golpe. La particularidad de la colisión del golpe es que solamente se activa cuando se da la orden de golpeo (**véase las figuras 51 y 52**), contrario a la colisión normal del usuario que siempre está activa debido a que es necesario para que se pueda reaccionar a todos los choques que tenga con todos los objetos del juego.

Ya definida la colisión del golpe del jugador, se procedió normalmente con el desarrollo de la mecánica del árbol de tirar un objeto. El cual guardaba cierto parecido con la técnica empleada en la puerta donde en vez de eliminar el objeto, como ocurre con las llaves, se utilizaron dos imágenes uno donde el árbol tiene el objeto en la copa y otra donde el objeto desaparece del árbol y aparece en el suelo, esto para simular la caída (**véase las figuras 53 y 54**).

##### 4.3.3.4.6. Programación de puentes

Siguiendo el desarrollo de mecánicas de objetos, se procedió con la mecánica de los puentes del juego. Al analizarlo se pudo observar como este guardaba un gran parecido con la mecánica de los árboles, ya que requiere de un golpe que active una acción que afectará al objeto; la diferencia era el lugar donde el golpe iba dirigido, la primera iba dirigido al árbol y en la segunda el golpe iba dirigido a un botón cercano al puente, al dar este golpe se emplean recursos para hacer creer al usuario que la acción se está realizando después de golpear. En el caso del puente lo que sucede después es que la parte rota del puente vuelve a aparecer lo que permite que el jugador pueda cruzarlo de manera normal (**véase las figuras 55, 56 y 57**).

##### 4.3.3.4.7. Programación de enemigos

En el juego sólo existe un tipo de enemigo y este es duplicado para generar muchos enemigos iguales durante todo el juego. Los enemigos, a diferencia de los demás objetos dentro del juego, tienen la capacidad de moverse, similar al jugador principal, pero estos lo hacen de manera automática y lo hacen únicamente para seguir al jugador. Debido a esta pequeña similitud se inició la programación de los enemigos de manera similar a como se hizo con el personaje principal. Cabe recalcar que no se utilizaron varias animaciones como se hizo con el personaje principal, ya que no se vio necesario profundizar en una complejidad parecida al personaje principal en este aspecto; una de las razones se debió a que los enemigos no siempre estarán enfocados en la cámara del jugador y estos serán eliminados cuando se encuentren con el jugador.

Se procedió programando las animaciones de los enemigos, estos solamente tienen un tipo de animación que es estar moviéndose en su lugar (véase la **figura 58**); esta animación se ejecuta en dos direcciones, atrás y adelante, y esta dependerá de la dirección en que el jugador es perseguido, esto se puede observar en la **figura 59**.

**Insertar imágenes de enemigo parado y en diferente dirección**

Para poder hacer que el enemigo persiguiera al jugador, se utilizó una técnica basada en un área de acción que detecta cuando el jugador ha entrado dentro del área (véase la **figura 60**), cuando él entra, se activa el movimiento del enemigo haciendo que se sienta atraído hacía la posición del jugador y lo sigue mientras se mueva o se quede quieto, de esta manera se simula la persecución del jugador (véase la **figura 61**).

Al igual que todos los objetos del juego, los enemigos también requieren de un objeto de colisión para que cuando el jugador choque con el enemigo, se pueda producir daño hacia el jugador disminuyendo así su vida. De manera contraria cuando el objeto de colisión del golpe choque con el enemigo este último recibirá daño, disminuirá su vida y retrocederá un poco hacia el lado contrario que se estaba moviendo (véase **figura 62**).



**Insertar imágenes de las colisiones entre enemigo y jugador**

#### 4.3.3.5. Programación de interfaz de usuario y menús

##### 4.3.3.5.1. Programación de la interfaz de usuario

Habiendo concluido con la programación de mecánicas principales del juego, se procedió con la parte no jugable del juego, pero aún así muy importante para brindarle a jugador un *feedback* de lo que ocurría mientras jugaba. Esta parte fue definida en un borrador en la parte de diseño de interfaz de usuario, y de la misma manera que sucedió con el diseño de niveles se buscaron los *assets* necesarios para pasar del borrador a la programación en *Godot Engine*, los *assets* elegidos se pueden observar en las **figuras 63, 64 y 65**.

**Insertar imágenes de los assets de la interfaz de usuario**

Primero se colocó cada *asset* en su lugar dentro de la pantalla de juego (véase la **figura 66**) para después sincronizar las acciones ocurridas durante del juego con los elementos de la interfaz. Un ejemplo de esto es cuando el jugador recibe daño, el elemento que representa la vida del jugador tiene que reflejar esta pérdida de vida (ver **figura 67 y 68**).

**Insertar imágenes de feedback de daño y cambios en la interfaz**

Esto se pudo realizar gracias a las propiedades que se asignaron al objeto del personaje principal; como la vida restante, los objetos recogidos y la durabilidad del arma que tienen en el momento; estas propiedades fueron enlazadas con el elemento que correspondían en la interfaz de usuario y cuando estas propiedades sufrían algún cambio dentro del juego entonces el elemento también se actualizaba para reflejar este cambio y que pudiera ser visto por el jugador.

##### 4.3.3.5.2. Programación de menús del juego

Los menús como tal son elementos no jugables del juego, pero estos son los que le permiten al usuario que pueda tomar decisiones que afectarán al juego como programa en general. Al iniciar el programa permiten al jugador empezar la partida, cambiar la configuración del juego o salir del juego. Mientras se está jugando permiten al usuario realizar las acciones anteriores además de pausar el juego para evitar que la partida se siga ejecutando y así el jugador pueda navegar en el menú sin ser atacado por enemigos en la partida.

Al inicio se construyó un borrador del cómo se verían los menús en el juego, y siguiendo ese mismo borrador como ejemplo se generaron *assets* para utilizar en el diseño y la programación. En la **figura 69** se puede observar el borrador realizado junto con el *asset* diseñado en base a este.

**Insertar imágenes de borrador de menú y asset diseñado**

El juego cuenta con dos menús diferentes, el menú de la pantalla inicial y el menú de pausa del juego. Como se pudo ver en la **figura 69** estas acciones son *Iniciar una partida nueva*, *Cargar la última partida* y *Salir del juego*. El menú diseñado no cuenta con una opción de configuración ya que no hubo ningún elemento configurable por el usuario. Además, la principal diferencia entre ambos es que el menú de pausa sólo tiene la opción de salir del juego.

A cada botón del menú se le asignó su acción correspondiente, estos se activan en el momento en el que el jugador toca el botón desde su pantalla haciendo que se ejecute un bloque de código dedicado para esta.

La primera permite al jugador iniciar una partida nueva cada vez que lo utilice, antes de realizar la acción se pedirá al usuario que confirme esto (véase la **figura 70**). La segunda permitirá al jugador cargar la partida en el último momento de guardado que se alcanzó en el juego, de esa manera no perderá el progreso que ya lleva acumulado en el juego. La última hace que el juego se cierre del celular del usuario, para esto también se le solicitará al jugador que confirme nuevamente que realmente quiere salir del juego.

**Insertar imágenes de la confirmación del usuario**

#### 4.3.3.6. Programación de guardado del juego

El guardado del juego permite que el jugador pueda iniciar la partida en el mismo lugar donde se haya quedado, además de conservar todos los objetos que haya conseguido anteriormente. Esta característica del juego requirió de la manipulación de archivos dentro del dispositivo celular del jugador.

Se definió un archivo donde todos los objetos y estados del jugador fueran guardados. De manera interna en el archivo esto se hizo con una estructura *JSON* debido a que *Godot Engine* tiene métodos para manipular este tipo de estructura de datos, esta estructura se puede visualizar en la **figura 71**. El archivo se puede visualizar como si fuera una caja que almacena todos los objetos y estados del jugador en un momento específico del juego.

**Insertar ejemplo JSON del archivo**

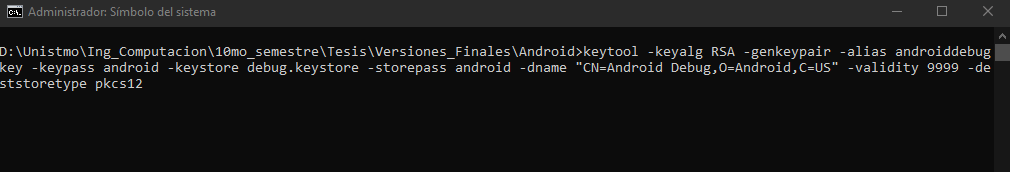
Debido a que leer y escribir sobre el archivo no era lo más eficiente se optó por un sistema de guardado automático que ejecutaba el código dedicado al guardado del juego cuando el jugador consiguiera una llave, un objeto valioso, abriera una puerta o consiguiera pasar el nivel. En estos momentos específicos se escribía sobre el archivo y se almacenaba la información del jugador.

Para iniciar el juego en este punto donde se guardó por última vez, también se utilizó un bloque de código que permitía la lectura del archivo con los datos almacenados anteriormente. Este también se ejecutaba en momentos específicos del juego cuando el jugador seleccionara la opción en el menú de juego y cuando la vida del jugador se terminaba. Entonces se lee el archivo y se procede a cargar los datos devolviendo al jugador al último punto de guardado.

#### 4.3.3.7. Exportación de Godot a Android

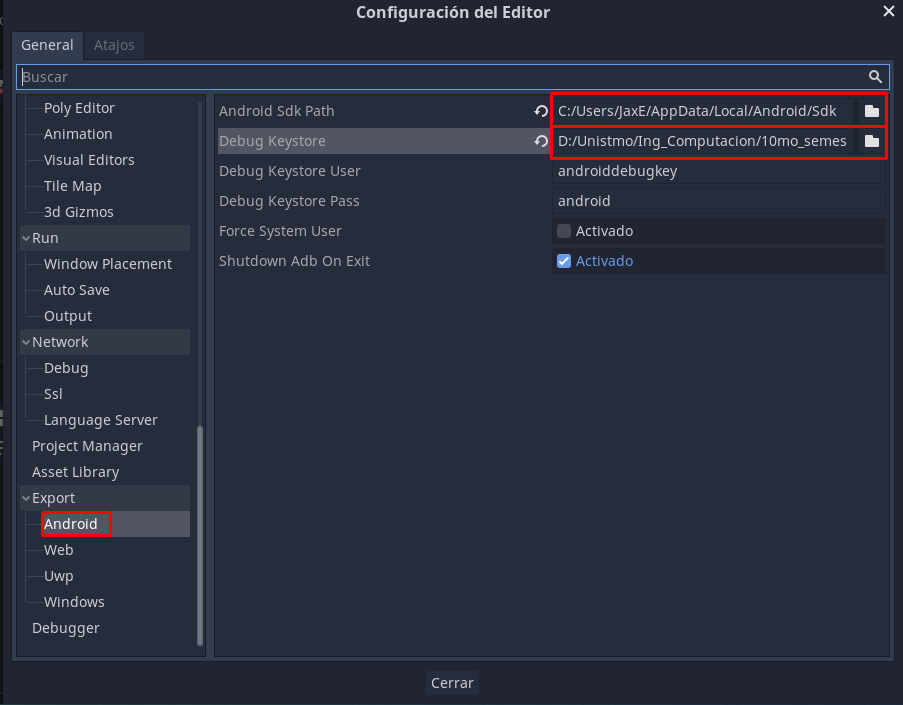
Habiendo terminado la programación del videojuego era necesario un último paso, el cual consistía en que el juego pudiera ser ejecutado en dispositivos móviles con *Android*; ya que, así como estaba, este solamente podía ser ejecutado directamente en *Godot Engine* y sin el programa no se podía jugar.

La documentación oficial en línea de *Godot* explica cómo realizar la exportación a *Android*. Fue necesario crear un archivo *debug.keystore* para poder realizar la instalación de la aplicación exportada a un dispositivo móvil. Esto se realizó ejecutando, dentro de la carpeta que contendrá la aplicación, el siguiente comando en la consola de comandos de *Windows*: *keytool -keyalg RSA -genkeypair -alias androiddebugkey -keypass android -keystore debug.keystore -storepass android -dname "CN=Android Debug,O=Android,C=US" -validity 9999 -deststoretype pkcs12*. Este paso se puede observar en la **figura 72**.

Después de crear el archivo se configuró el entorno de desarrollo *Godot* para poder ejecutar el archivo. Se descargaron las plantillas de exportación necesarias para que *Godot Engine* pueda realizar la exportación. Para ello se accede a la opción Editor, se selecciona la opción *Administrar plantillas de exportación* (véase **figura 73a**) y se da clic en *Descargar e instalar* (véase **figura 73b**), automáticamente empezará a descargar las plantillas requeridas para la exportación del juego.

Ahora se establecen las rutas del *SDK* así como el archivo *debug.keystore*. Para ello se accedió a la configuración del editor (véase **figura 74a**), se seleccionó la opción Androidy se especificó la ruta donde se instaló el *SDK* de *Android* y elarchivo *debug.keystore* (véase **figura 73b**).

Con esta configuración realizada, ahora se pudo exportar el juego a la plataforma de *Android*. Para ello se siguieron los siguientes pasos; desde la opción Proyecto, se seleccionó la opción Exportar (véase **figura 75a**), se le asignó un nombre a la aplicación, se seleccionó una ruta donde el archivo exportado estaría almacenado y se seleccionó la opción Exportar Proyecto (véase **figura 75b**).

Estos pasos finales generaron un archivo con extensión *APK* nativa de aplicaciones para dispositivos con *Android*. Este archivo se transfirió al dispositivo móvil, se instaló para así poder realizar pruebas de jugabilidad y detectar errores para su corrección.

##### Gestión de riesgos

Para la exportación del juego se encontró con un problema que no se había previsto, y es que era necesario descargar e instalar *Android Studio* para poder realizar la exportación a la plataforma de *Android*, además de descargar e instalar un paquete *SKD* de *Java* (se utilizó la versión 11 de *Open JDK*). El programa se instaló en un equipo con un procesador Intel Core i3-9100F, 32 GB de memoria RAM, 1 TB de disco duro SSD y una tarjeta gráfica dedicada NVIDIA GeForce GXT 1050 Ti, el cual pudo ejecutar el programa de manera normal y se pudo proceder con la exportación del juego.

## 4.4. Beta

### Pruebas

En esta fase el principal punto evaluado fue la jugabilidad y las mecánicas del juego; esto con la intención de detectar errores que pudieran afectar o parar el progreso del jugador mientras va avanzando en los niveles. Las pruebas fueron realizadas por dos terceros ajenos al proyecto, con el conocimiento suficiente sobre programación, videojuegos y detección de errores en programas de software.

Cada uno obtuvo una copia jugable del videojuego, así como un formato de reporte de error (véase **anexo 1**) para que llenaran con información referente al error; este formato sirvió para reportar un error encontrado en la jugabilidad o mecánicas que afectaran al funcionamiento de la versión beta del juego. El reporte contiene una captura de pantalla del error, una descripción del error y los pasos para replicar el error.

### Errores encontrados

### Resolución de errores

# Referencias

Abt, C. (1970). *Serious* *Games.* (1ra. ed.). Viking Press.

Acerenza, N, Coppes, A, Mesa, G, Viera, A, Fernández, E, Laurenzo, T & Vallespir, D. (2009). Una Metodología para desarrollo de videojuegos :versión extendida*. Reportes Técnicos,* 09-13. <https://hdl.handle.net/20.500.12008/3420>

Álvarez, M. (2017). Desarrollo del pensamiento computacional en educación primaria: una experiencia educativa con Scratch. *Revista de Ciències de l’Educació*,(2). 45-64. <https://doi.org/10.17345/ute.2017.2.1820>

App Annie & IDC. (2020). Consumo de videojuegos por plataforma [Gráfico]. App Annie. <https://www.appannie.com/en/insights/market-data/gaming-spotlight-2020-review/>

Arenas, K. M. (2018). *Desarrollo de un Serious Gaming para estudiantes de Primer Año de Primaria aplicando la Metodología SUM*. [Tesis de ingeniería, Universidad Peruana Unión]. <http://hdl.handle.net/20.500.12840/2973>

Basogain, X., Olabe, M. A., & Olabe, J. C. (2015). Pensamiento Computacional a través de la Programación: Paradigma de Aprendizaje. *Revista de Educación a Distancia*, (46). 1-33. <https://doi.org/10.6018/red/46/6>

Compañ, P., Satorre, R., Llorens, F., & Molina, R. (2015). Enseñando a programar: un camino directo para desarrollar el pensamiento computacional. *Revista de Educación a Distancia*, (46). 1-15. <https://doi.org/10.6018/red/45/11>

Dhule, M. (2022). Adding Game Graphics. In: Beginning Game Development with Godot. Apress, Berkeley, CA. <https://doi.org/10.1007/978-1-4842-7455-2_2>

Empantallados.com (s.f.). Tipos de videojuegos, ¿cómo se clasifican?. <https://empantallados.com/tipos-de-videojuegos/>

Frasca, G. (2001). Videogames of the oppressed: videogames as a means for critical thinking and debate. [Tesis de maestría, Georgia Institute of Technology]. <https://ludology.typepad.com/weblog/articles/thesis/FrascaThesisVideogames.pdf>

International Society for Technology in Education, & Computer Science Teachers Association (2011). Operational Definition of Computational Thinking for K-12 Education. <https://cdn.iste.org/www-root/Computational_Thinking_Operational_Definition_ISTE.pdf>

López, C. (2016). El videojuego como herramienta educativa. Posibilidades y problemáticas acerca de los serious games. *Apertura*, *8*(1). 1-15. <http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1665-61802016000200010>

Manquilef, G. F. (2021). *Implementación de ciclos y condicionales para el desarrollo del pensamiento computacional en videojuegos*. [Tesis de ingeniería, Universidad Andrés Bello]. <http://repositorio.unab.cl/xmlui/bitstream/handle/ria/18313/a131787_Manquilef_G_Implementacion_de_ciclos_y_condicionales_2021_Tesis.pdf>

Márquez, G., Osorio, S., & Olvera, N. (2011). *Introducción a la programación estructurada en C* (1ra. ed.). México: Pearson.

Montás, G. (2020). *Adicción a los videojuegos en tiempos de pandemia*. Insight. <https://insight.com.do/adiccion-a-los-videojuegos-en-tiempos-de-pandemia/>

Murillo, X. A., Gutierrez, A., Ibañez, A. W., Quiroz, J. A., Sahonero, G., & Diaz, F. (2018). *Implementacion de la metodologia SUM modificada para el desarrollo de videojuegos orientados al aprendizaje en Bolivia*. 15th Ibero-American Symposium on Education, Cybernetics and Informatics, SIECI 2018. Orlando, Estados Unidos. <https://www.researchgate.net/publication/329525865_Implementacion_de_la_metodologia_SUM_modificada_para_el_desarrollo_de_videojuegos_orientados_al_aprendizaje_en_Bolivia>

Rahman, H. G. (2017). *Videojuego educativo en 3D para dispositivos móviles Android, enfocado al aprendizaje de la Lógica de Programación para usuarios entre los 5 a 18 años de edad*. [Tesis de ingeniería. Universidad Técnica de Ambato]. <https://repositorio.uta.edu.ec/jspui/handle/123456789/25705>

Ristić, O., Milošević, D., & Urošević, V. (2016). *The importance of programming languages in education*. TECHNICS AND INFORMATICS IN EDUCATION, 6th International Conference, Faculty of Technical Sciences. Čačak, Serbia. <http://www.ftn.kg.ac.rs/konferencije/tio2016/Radovi%20TIO%202016/EN/3)%20Information%20and%20Educational%20Technologies/319_046_Ristic%20i%20sar_EN.pdf>

Ruíz, M. R. (2014). *Un modelo conceptual para el diseño de videojuegos educativos*. Tesis doctoral. Universidad Carlos III de Madrid. <http://hdl.handle.net/10016/20595>

Sada, C. (2020). La pandemia y el auge en el uso del celular en México. *MediaLab*. <https://medialab.up.edu.mx/noticias/la-pandemia-y-el-auge-en-el-uso-del-celular-en-mexico/>

Téllez, M. (2019). Pensamiento computacional: una competencia del siglo XXI. *Educación superior*, *6*(1), 23-32. <http://www.scielo.org.bo/pdf/escepies/v6n1/v6n1_a07.pdf>

The Royal Society. (2012). *Shut down or restart? The way forward for computing in UK schools*. (informe nro. 1). Londrés, Reino Unido: Autor. <https://royalsociety.org/-/media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf>

Ticante, A. C., Herrera, C. M., Arguijo, P., Meléndez, R. Á., & Vázquez, A. H. (2019). Videojuego educativo para ayudar a comprender los principios básicos de la programación y desarrollar la habilidad lógica en niños de educación básica. *Research in Computing Science*, *148*(7), 127–139. <https://doi.org/10.13053/rcs-148-7-10>

Vallejo, D. & Martín, C. (2019). *Desarrollo de Videojuegos: Arquitectura del Motor de Videojuegos*. Bubok.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. <http://dx.doi.org/10.1145/1118178.1118215>

Wolf, M. & Perron, B. (2005). Introducción a la teoría del videojuego. *Formats: revista de comunicación audiovisual.* (4). <https://raco.cat/index.php/Formats/article/view/257329>

Zapata, M. (2015). Pensamiento computacional: Una nueva alfabetización digital. *Revista de Educación a Distancia*, (46). 1-47. <https://revistas.um.es/red/article/view/240321>

Zatarain, R. (2018). Reconocimiento afectivo y gamificación aplicados al aprendizaje de Lógica algorítmica y programación. *Revista electrónica de investigación educativa,* *20*(3), 115-125. <https://doi.org/10.24320/redie.2018.20.3.1636>

Zhao, Z. (2017). *Videojuegos, educación y desarrollo.* [Tesis de doctorado, Universidad Autónoma de Madrid]. <http://hdl.handle.net/10486/680698>

Zyda, M. (2005). From visual simulation to virtual reality to games. *Computer*, *38*(9), 25-32. <https://doi.org/10.1109/MC.2005.297>

<https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en> ISO 9241-11

# Nielsen Usability Engineering 1994 pag. 26

Sánchez 2011: <http://www.redicces.org.sv/jspui/handle/10972/1937>

Grau, X. F. (2000, November). Principios Básicos de Usabilidad para Ingenieros Software. In *JISBD* (pp. 39-46). : https://www.researchgate.net/profile/Xavier-Ferre/publication/221595210\_Principios\_Basicos\_de\_Usabilidad\_para\_Ingenieros\_Software/links/0deec53988e26512b6000000/Principios-Basicos-de-Usabilidad-para-Ingenieros-Software.pdf