MEMORIA

Índice

Introducción

En este proyecto realizaremos el desarrollo de un videojuego, explicaremos paso a paso el desarrollo del mismo, estos están establecidos en el índice, ordenados cronológicamente para poder tener un orden de los acontecimientos.

Contexto

Hemos emprendido este proyecto, ya que los videojuegos siempre han estado presentes en nuestras vidas, por eso mismo, uno de nuestros objetivos es desarrollar uno.Sin embargo, no solo buscamos cumplir este objetivo, ya que tenemos más objetivos detrás de este, que comentaremos más adelante.

Justificación

Hemos elegido este proyecto, ya que tenemos mucho que trabajar en todos los ámbitos que envuelven al proyecto, como por ejemplo: conocimientos informáticos, como programación y diseño gráfico, también buscamos mejorar en el ámbito personal.

Objetivos

Los objetivos que tenemos previstos cumplir para el desarrollo del videojuego, los clasificaremos y desarrollaremos en dos partes:

Objetivo general: en el objetivo general, nos centramos en la finalización exitosa del videojuego, incluyendo el desarrollo de este como un objetivo, ya que es lo mínimo que queremos conseguir con este.

Objetivo específico: estos objetivos son más personales, ya que buscamos mejorar nuestro conocimiento sobre la programación y diseño, pero sobre todo, buscamos mejorar nuestros ámbitos personales, como la constancia, superación, criterio, trabajo en equipo, liderazgo… Estos serían unos pocos ámbitos en los que queremos mejorar, pero no solo buscamos esto, ya que también buscamos superar la calidad de nuestro proyecto, consiguiendo aplicar todos los conocimientos adquiridos y así realizar un videojuego acorde a nuestros ideales.

Título del proyecto

Nuestro proyecto se llama Jynza, ya que al juntar nuestros nombres “Juan e Izan”, nos dimos cuenta de que el nombre nos gustaba, entonces le hicimos unos retoques así como usar la “y”, esa es la procedencia del nombre, simple pero original.

Tipo de proyecto

Juego plataforma 2D

Integrantes del equipo

Juan Gonzalez Fernandez e Izan Algar Ruiz.

Licencia del proyecto (CC)

Estamos utilizando una licencia Creative Commons que permite a otros usar nuestro trabajo, pero no con fines comerciales (No Comercial, NC). También deben darnos crédito por la obra original (Atribución, BY). Además, si alguien realiza cambios o mods basados en nuestro trabajo, esos cambios también deben compartirse bajo las mismas condiciones (Compartir Igual, SA). Esto significa que, si alguien crea una modificación de nuestra obra, esa modificación debe permitir el uso no comercial, dar atribución y permitir obras derivadas.

Presentación de promotores:

Somos Juan e Izan, no tenemos nociones sobre cómo desarrollar un videojuego, pero gracias a este proyecto vamos a conseguir las bases para conseguirlo.
En el apartado artístico se encargará Izan, ya que tiene un gran afán sobre el diseño de personajes, niveles etc…

En el apartado jugable, es decir la programación de personajes se encargará Juan, ya que quiere realizar estudios sobre programación y le servirá para tener unas nociones en un futuro.Por último, añadir nuestras habilidades ya que aunque no tengamos nociones sobre cómo desarrollar videojuegos la

experiencia de pasar gran parte de nuestra vida jugándolos, nos hará ser más autocríticos y objetivos.

Resumen del proyecto

En este proyecto vamos a desarrollar un juego plataforma 2D, en el cual nos hemos basado en nuestros animales favoritos como representación de nuestro videojuego.En este juego los protagonistas son un Pingüino y un Panda, los cuales deben de abatir o esquivar a los enemigos para llegar a la meta y conseguir superar el nivel.

Los niveles están inspirados en el hábitat natural de estos animales, en el caso del pingüino su nivel estará basado en el clima nevado con montañas nevadas, en el caso del Panda en un clima tropical que está basado en una jungla.Para realizar este proyecto el cual siempre ha sido uno de nuestros sueños “desarrollar nuestro propio videojuego”, vamos a utilizar el programa Godot, este programa lo hemos estudiado para poder exprimir al máximo sus posibilidades dentro de nuestras capacidades.

En conclusión, hemos elegido este proyecto porque siempre hemos soñado con crear un videojuego, sobre todo porque es nuestro mayor hobby y queremos dar un paso adelante y lograr que llegue al mayor número de personas posible, sabemos que no serán muchas pero le pondremos mucho empeño.

Palabras clave

Plataformas 2D, videojuego, pixel art, nodes, Godot.

Resumen en inglés

In this project, we are going to develop a 2D platformer game inspired by our favorite animals. The main characters in the game are a penguin and a panda, and their objective is to defeat or dodge enemies to reach the goal and successfully complete each level.

The levels are designed based on the natural habitat of these animals. For the penguin, the level will be set in a snowy climate with snowy mountains, while for the panda, it will be in a tropical climate inspired by a jungle. We have always dreamt of developing our own video game, and to bring this dream to life, we

have chosen to use the Godot engine. We have extensively studied this program to make the most of its capabilities within our skill set.

In conclusion, we have chosen this project because we have always dreamt of creating a video game, especially since it is our greatest hobby. We want to take a step forward and make it reach as many people as possible. We are aware that it may not be many, but we will put a lot of effort into it.

Keywords (palabras clave en inglés)

2D platformer, video game, pixel art, nodes, Godot.

Tecnologías

Las tecnologías usadas para el desarrollo son, Godot y Pixilart, cada una usada para un ámbito distinto:

Godot: es un motor gráfico de desarrollo de videojuegos con código libre, que usaremos para programar el videojuego, así como las animaciones, sistemas etc….

Pixilart: es una herramienta de dibujo en píxeles gratuita y online, que es perfecta para el diseño que queremos emplear, la usaremos para el diseño de personajes, niveles etc…

Aseprite: Es una herramienta de dibujo y animación de pixel art. Era la que queríamos utilizar para nuestros diseños, pero al ver que era de pago, la descartamos. Aun así, es una herramienta muy buena.

Descripción de los componentes

Lo que queremos conseguir es tener un juego disfrutable, bien hecho y que pueda divertir a mucha gente. Por ello, nos documentamos exhaustivamente sobre Godot y el diseño en estilo pixel art. Para organizarnos, asignamos tareas específicas.

Tarea 1 por parte de los dos:

- Nos pusimos de acuerdo para comenzar a documentarnos en los primeros días, ya que no teníamos mucha experiencia en diseño y programación. Durante este tiempo, hemos revisado numerosos vídeos e investigado en foros para obtener información y orientación.

Tarea 2 por parte de Juan:

- Ir probando todo lo aprendido para hacer un simulacro del juego y así adquirir facilidad para el desarrollo del juego principal.

Tarea 3 por parte de Izan:

- Ir buscando inspiración para los diseños y empezar a aplicar todo lo aprendido sobre el estilo pixel art. Probar diseños diferentes para ver si nos gustan, y si no, poder cambiarlos.

Estrategia y planificación del proyecto

Nuestra estrategia y planificación, se basa en adaptar un producto ya existente, como pueden ser los juegos plataforma 2D, es decir nuestro caso.Es la más viable, ya que entraremos en un mercado que lleva tiempo en desarrollo, la industria de los videojuegos, entonces al saber sobre esto, nuestro plan es realizar una sobreexposición de nuestro videojuego, ya que al tener conocimientos de este mercado sabemos lo que los usuarios buscan, entonces crearemos diversos medios de comunicación como pueden ser las redes sociales, para potenciar la expectación de nuestro proyecto.

Metodología de trabajo

Después de evaluar que metodología de trabajo usar, nos hemos decantado por la **“Ágil”**, ya que priorizamos la flexibilidad, a lo qué nos referimos con flexibilidad, es a realizar cambios necesarios en cualquier momento para la mejora continua del proyecto.Entonces, al usar esta metodología podemos moldear los objetivos del proyecto, y su gestión, siempre y cuando este cambio sea para velar por la mejora continua.

Estudio económico y presupuestario

Diseño de Juego

Diseñadores de juegos: Salarios mensuales multiplicados por la duración del proyecto.
Herramientas de diseño: Licencias de software como Unity, Unreal Engine, etc.
Desarrollo de Software

Desarrollo de software

Desarrolladores y programadores: Salarios mensuales por desarrollador.
Herramientas de desarrollo: Licencias de software de desarrollo y plataformas de colaboración.
Testing y QA: Costos de personal adicional o contratación de empresas de testing.
Arte y Diseño Gráfico

Arte y diseño gráfico

Artistas gráficos y animadores: Salarios mensuales por artista/animador.
Herramientas de arte: Licencias de software de diseño gráfico como Adobe Creative Suite, Blender, etc.
Freelancers: Costos por contratación de trabajos específicos.

Marketing y Promoción

Equipo de marketing: Salarios mensuales por persona.
Campañas publicitarias: Costos de anuncios en diferentes plataformas.
Participación en eventos: Inscripción y costos de viaje.Soporte Técnico

Soporte técnico

Personal de soporte: Salarios mensuales por persona.
Infraestructura: Costos de servidores y mantenimiento.

Análisis de requisitos

Los resultados obtenidos, deben de tener estos requisitos mínimos:

-Desarrollar 2 niveles, cada uno con una temática personalizada para cada uno de los personajes.(panda y pingüino)

-Tener 1 personaje jugable

-Diferentes niveles con diseño original

-Diseño de menú acorde con la temática del videojuego

En conclusión, esto sería lo mínimo que contendrá el proyecto, ya que el desarrollo del mismo está planteado con estas bases.

Descripción de los componentes

En esta parte explicamos los puntos de la programación que se ha utilizado para cada parte.

- Node2D: Lugar donde se unen todos los elementos del videojuego para crear el escenario final.

- Area2D: Las habilidades tienen un gran número de nodos, por ejemplo, el CharacterBody2D, se elegirá este nodo ya que su característica es realizar acciones cuando otro objeto entra en contacto, y no se desea que el objeto sea empujado o movido, sino solo que reaccione.

- CollisionShape2D: Define la hitbox* de los personajes, la función de una hitbox es otorgar al jugador una colisión con el resto de elementos del juego que también tengan una hitbox; sin ella, el personaje atravesaría el resto de elementos.

- TileMap: Un nodo que permite seleccionar arte de una imagen y agregarlo en forma de bloques al escenario.

- Sprite2D: Lugar donde se colocan todos los diseños.

- AudioStreamPlayer: Lugar donde se colocan y organizan todos los elementos de audio del videojuego, como efectos especiales y música.

- Camera2D: Como el nombre indica es la cámara que sigue al personaje para que no se salga del plano.

- AnimatedPlayer2D: Lugar donde se establecen los frames para la animación.

- Label: Lugar donde proyectar textos.

- BoxContainer: Lugar donde se establece un container para marcar.

- Slider: Lugar donde es una barra deslizadora para manejar el volumen por ejemplo.

- Button: Como el propio nombre dice un botón.

Escenario:
- Se a utilizado el Node2D, Area2D, CollisionShape2D, TileMap, Sprite2D y AudioStreamPlayer.

Personaje:
- Se ha utilizado el Sprite2D, CharacterBody2D, AnimatedPlayer2D, CollisionShape2D, Camera2D.

Pantalla de inicio:
- Se ha utilizado Node2D, Label, BoxContainer, Button, Slider.

Objetos:
- Se ha utilizado CollisionShape2D, AnimatedSprite2D.

Definición de les funcionalidades

1. Jugabilidad: Jugabilidad medio difícil y cómodo, accesible para todo tipo de jugador.
2. Obstáculos: Fáciles al principio pero cuanto más avanzas más difícil se vuelve.
3. Niveles: Niveles diferenciados con dificultad propia.

Producto y servicio

Nuestro servicio es el desarrollo y dirección total de nuestro videojuego, ya que en este apartado no podemos ofrecer más.En cuanto al producto podemos ofrecer nuestro videojuego completo.

Promoción y distribución

Promoción

Este apartado es el que tenemos más manejo, ya que no dependemos de nadie para hacer crecer nuestro proyecto, sería de ayuda alguien de renombre, pero no lo necesitamos.

En la promoción tenemos muy claro que vamos a realizar una serie de redes sociales que promocionen “teasers”, de nuestro juego y nos hagamos escuchar con avances de este.Las redes sociales que más usuarios mueven, son las siguientes:

- **Instagram**
- **TIkTok**
- **YouTube**
- **Discord**

Con estás redes sociales, podríamos hacer una promoción a gran escala, ya que son plataformas que manejan muchos usuarios, los cuales tarde o temprano llegarán a nuestros videos.Si fuera necesario desarrollariamos una página web para más información.

Distribución

Para distribuir nuestro videojuego, tenemos claro que la página principal que usaremos para la distribución y venta de este, será “Steam”, ya que es una plataforma de renombre en la cúal hay cientos de miles de videojuegos.

Nuestra idea es sacar el juego en un valor de **1$**, ya que al ser principiantes es prácticamente imposible que alguien se fije en nuestro juego si no es un valor mínimo.Si vemos que ni aún así nadie se fija en este, lo pondremos gratis.Sin embargo, tenemos otra idea, la cual es subir una parte del juego corta, como **“BETA”**, en la cual pueden probar el juego, y si quieren continuar deberían de pagar su valor.

Ubicación de “stock”:

Nuestro “stock”, estará ubicado en páginas digitales de distribución de videojuegos, ya que hacer una stock físico no nos es accesible al necesitar una gran economía o un proveedor con poder adquisitivo.Tenemos pensado ubicar nuestro producto en **Steam**, una tienda digital de distribución de videojuegos con mucho renombre, nos parece perfecta al ser muy famosa y accesible, ya que podemos subir nuestro producto sin problema, simplemente se llevarían un porcentaje de las ventas al utilizar su página como distribución, su precio base es de 100$, después de los 1000$ en ventas esta suma se devuelve.

Facturación

Al ser un juego de desarrolladores primerizos, tenemos una baja estimación en cuanto a facturación hablamos, ya que nuestro videojuego saldrá al mercado a **2$**.Puede parecer un precio bajo, pero tenemos que tener en cuenta que al ser un juego **indie**, debe tener un precio acorde a su trabajo y desarrollo.En cuanto a la facturación total, tenemos en cuenta el vender **100** copias de nuestro juego, ya que con esto conseguimos un buen porcentaje de facturación, al no gastar prácticamente nada durante el desarrollo del videojuego, ya que gran parte de lo que necesitamos ya lo tenemos.No nos podemos olvidar de las redes sociales, ya que estas también nos proporcionarán un incremento de la facturación en

caso de conseguir una buena suma de seguidores, estimamos conseguir **100$**, en cuanto a facturación de redes sociales.

Facturación estimada.Total = 150$

El porqué de este total, es básicamente porque un porcentaje de las copias de nuestro juego, se lo llevará la página que usaremos de distribución, entonces por ese apartado tendremos un ingreso menor.

Creación del videojuego (pasos)

Para la creación de videojuegos, los pasos suelen ser los mismos en la mayoría de ocasiones, la diferencia es la distribución de estos pasos:

1.Lluvia de ideas sobre el juego: en este paso nos centramos en orientar nuestras ideas hacia el videojuego, ya que necesitamos una base para saber hacía dónde orientar a este.En este paso, estamos constantemente ideando sobre el videojuego, para que nuestra creatividad sea la que predomine y nuestras ideas sean innovadoras.

2.Aplicación de las ideas: este paso se complementa con el anterior ya que todas las ideas que estuvimos desarrollando, empiezan a surgir en forma de bocetos de personajes, paisajes, diseño de niveles, tipos de armas…Este paso es muy importante ya que aplicamos todo lo aprendido en el anterior, para así tener una idea todavía más clara sobre donde irá nuestro proyecto.

3.Diseño final: en este punto los diseños que antes eran bocetos son terminados, y terminan siendo el diseño final de nuestros personajes y niveles, al tener estos diseños decididos toca plasmarlos en digital, para poder darles movimiento y vida a través de nuestro videojuego.
4.Programación: en este paso los diseños ya han sido digitalizados, entonces comenzamos con la programación de las animaciones, colisiones … Al tener esto a cabo podemos pasar al siguiente paso.

5.Sonidos y música: en este paso aplicamos los sonidos y música a nuestro videojuego, así conseguimos una mayor inmersión y que nuestro juego tenga más ritmo.En este paso debemos de introducir la música y los sonidos en la base de nuestro juego, para que cuando detecte un movimiento se ejecute un sonido acorde a este.

6.Prueba del juego "Test": en este paso tenemos nuestro juego totalmente programado, pero debemos probarlo para conseguir encontrar todos los fallos posibles para tenerlo listo, y conseguir lanzarlo al mercado con muy pocos fallos.

7.Lanzamiento al mercado: nuestro juego entra en el mercado, es muy importante ya que debemos de observar cómo es recibido, y sobre todo es cuando comienzan a salir todos los errores de estos, ya que habrán varios jugadores jugando simultáneamente y pueden encontrar errores.

8.Mantenimiento del juego: último paso y el más complejo, este es el que debemos de mantener más tiempo, ya que al recibir las sugerencias y críticas de los usuarios, tenemos que estar abiertos a realizar actualizaciones en nuestro juego, para solventar los fallos e implementar cambios en el juego.

Mantenimiento del juego

Para mantener el juego tenemos pensadas varias opciones, entre las cuales se encuentran, actualizaciones, parches , DLC y paquetes con productos adicionales del juego.

Actualizaciones: esta es una de las mejores opciones para mantener nuestro videojuego, ya que realizar actualizaciones frecuentemente, hace que el juego permanezca vivo más tiempo, ya que sabemos de casos en los que al no realizar actualizaciones frecuentes, el juego en cuestión "muere" más rápido que si las realizara.

Parches: són muy parecidos a las actualizaciones, ya que se deben realizar cambios en el juego para realizarlos, pero en este caso no es para conseguir contenido nuevo o añadir alguna implementación, en este caso los parches son para solucionar fallos en el juego, que se encontraron al inicio de este, o a raíz de una actualización.

DLCs: a lo que hacemos referencia como **"DLCs",** es a un tipo de uso comercial que realizan muchas desarrolladoras de videojuegos, para conseguir vender contenido adicional, pero del mismo juego.Esto lo realizaremos de la siguiente forma, lanzaremos el juego con un precio base, después, añadiremos contenido adicional del juego, por la mitad de precio aproximadamente del precio base..

Productos adicionales: nos referimos a productos adicionales, como nuevos aspectos para los personajes, enemigos distintos, modalidades nuevas…. Esto se encontrará dentro del juego, pero solo será accesible para quien pague el precio del contenido.

Recursos humanos tareas y funciones:

Primero, hablaremos sobre las **tareas y funciones**, las cuales tendremos cada uno en el proyecto para elaborar el videojuego.En el apartado artístico, tenemos a **Izan Algar**, el cual se encargará de los diseños de personajes, diseño de niveles e interfaz, así como el menú del juego, por último animación.Por otra parte, en el apartado de programación ,**Juan Gonzalez,** será el encargado de esta función, el cual se encargará de programar el movimiento, colisiones de estructuras, sistema de monedas, sistema de vidas y sistema de daño.Teniendo estos puntos en cuenta, podemos pasar con los recursos humanos, que hemos encontrado acertados en esta situación, hablando desde cada perspectiva:

1. **Desarrollo y Formación:** en el caso de ambos, hablando del ámbito de desarrollo de videojuegos, no tenemos nada de experiencia, así que decidimos formarnos desde páginas de internet y videos.En este tipo de formación, conseguimos desarrollar nuestros conocimientos más rápido de lo esperado, ya que al tener unas bases de informática, podemos tener una idea sobre el tema de desarrollo.También, basándonos en nuestra larga experiencia con los videojuegos, nos hacemos una idea de como debería de ser este, en el final de su desarrollo.
2. **Gestión del Desempeño:** al ser un proyecto de 2 personas, hemos gestionado equitativamente el trabajo de cada uno, como he comentado anteriormente, tenemos una función específica que tiene un desempeño igual o similar, entonces al punto que quiero llegar, es que tenemos el mismo desempeño, pero en diferentes ámbitos.
3. **Compensación y Beneficios:** en este caso como comentamos al principio de este proyecto, no buscamos una compensación o un beneficio económico, buscamos unas compensaciones y beneficios más personales.En nuestro caso de lo que hablamos es de lo siguiente, mejora personal, incrementar nuestros conocimientos, cumplir nuestro objetivos…Todos estos beneficios, son los que nos llevan con ilusión a finalizar este proyecto.

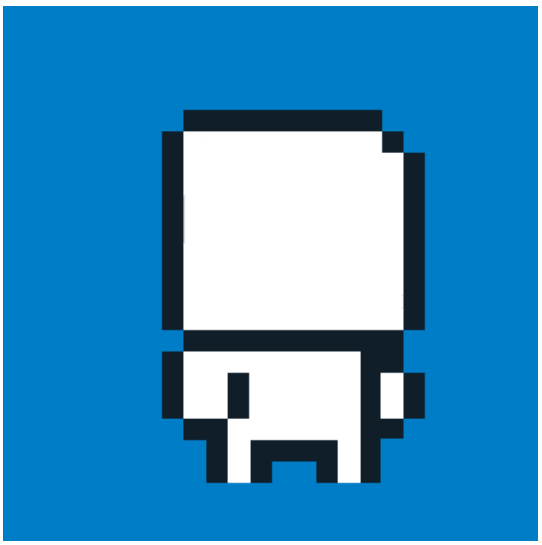
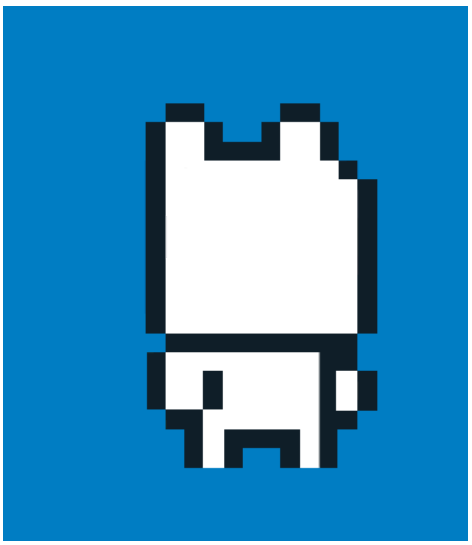
4. **Clima Laboral y Cultura Organizacional:** nuestra organización es muy simple, cuando tenemos tiempo libre para invertir en este proyecto, lo aprovechamos al máximo para exprimir sus posibilidades, entonces la organización es esta, trabajar cuando tenemos suficiente tiempo a nuestra disposición, y no desaprovechar ni un segundo.Por último, hablando del clima laboral, solemos trabajar desde el mismo lugar, que es parecido a una oficina, como puede ser nuestra institución o nuestras viviendas, el clima no es un problema para ninguno de estos casos, pero en nuestra institución suele haber exceso de ruido, y muchas veces cuesta concentrarse, pero por lo demás consideramos que es un buen lugar de trabajo.
5. **Salud y bienestar:** no tenemos ningún problema en este ámbito, ya que si tenemos algún problema de salud puntual, no tenemos problema en aprovechar el tiempo para tener una buena recuperación, y estar activos lo antes posible.En el apartado de bienestar, las mayores complicaciones que nos podremos encontrar son malestares menores, a estos nos referimos a los siguientes, mala concentración, indecisión , inconformismo, falta de ideas … Hemos nombrado estos malestares, para tenerlos en cuenta, ya que no creemos que se ocasionen con frecuencia.

En conclusión, estos son los puntos que queríamos tratar, en recursos humanos y funciones, los consideramos bastante simples pero no por ello poco importantes.

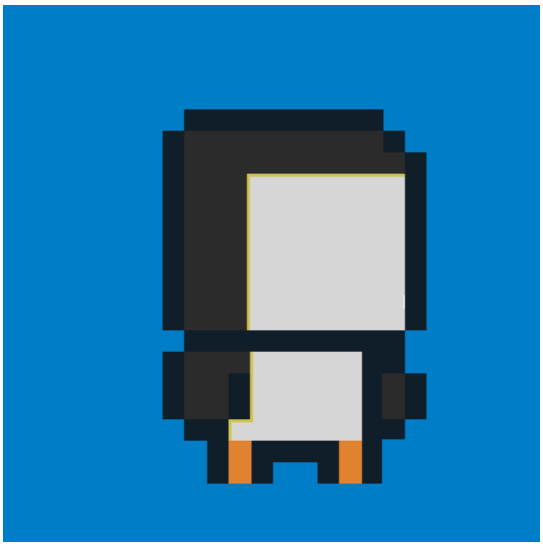
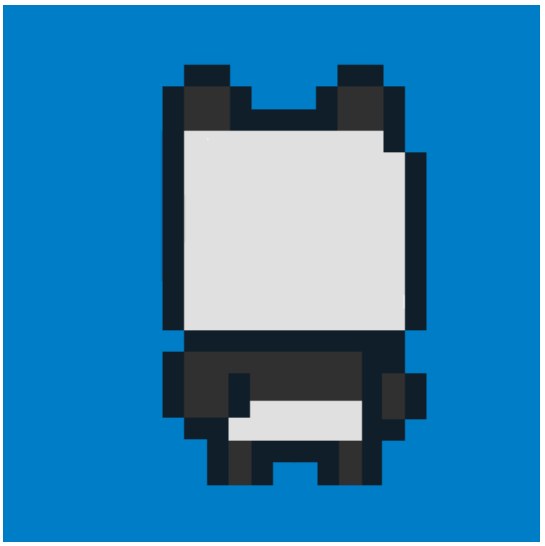
Diseños

Diseños de los personajes

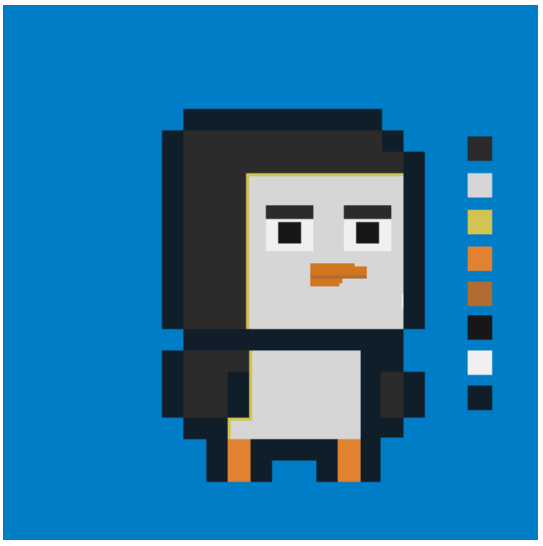
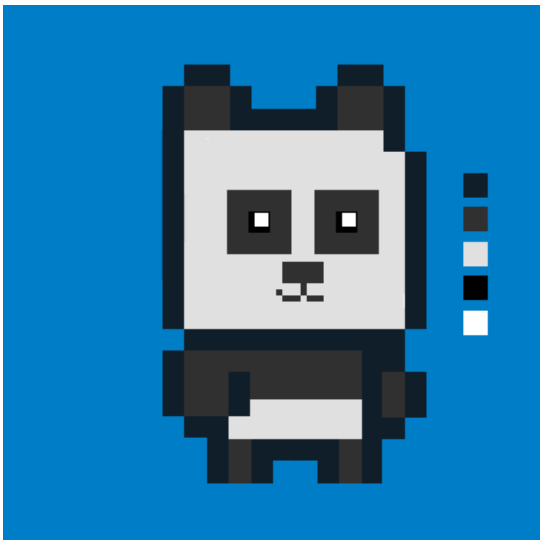
Para los diseños, tuvimos la idea de representar a los personajes principales como nuestros animales favoritos: un panda y un pingüino. Para crear estos personajes, primero establecimos una base para cada uno y, a partir de ahí, realizamos las modificaciones necesarias antes de proceder con la animación.

Una vez que teníamos una base definida para cada personaje, pasamos a aplicar los colores correspondientes para darles el aspecto de un panda y un pingüino.

Con los colores listos, el siguiente paso fue diseñar las caras, lo cual les daría más vida y personalidad.

Con los personajes completamente finalizados, comenzamos a trabajar en los movimientos de las extremidades y las expresiones faciales, asegurándonos de tener todo lo necesario para lograr una animación de calidad.

Animación

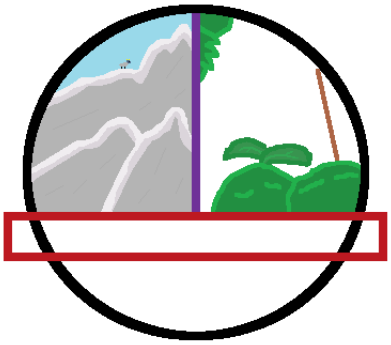
Para la animación de estar quieto, colocamos el modelo del personaje y utilizamos la herramienta de selección para bajar la cabeza y modificar los brazos, logrando así la animación deseada.

Utilizamos este ejemplo para crear la animación de nuestro personaje principal, "Juan el Pingüino".

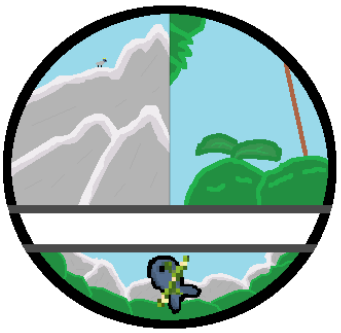
Con estos dos "frames", obtuvimos la animación de estar quieto. Todas las animaciones se han hecho de la misma manera, "frame" por "frame".

Logo

Para crear el logo, tuvimos la idea de representar el hábitat de cada personaje en una forma redonda y colocar un rectángulo para el nombre del juego.



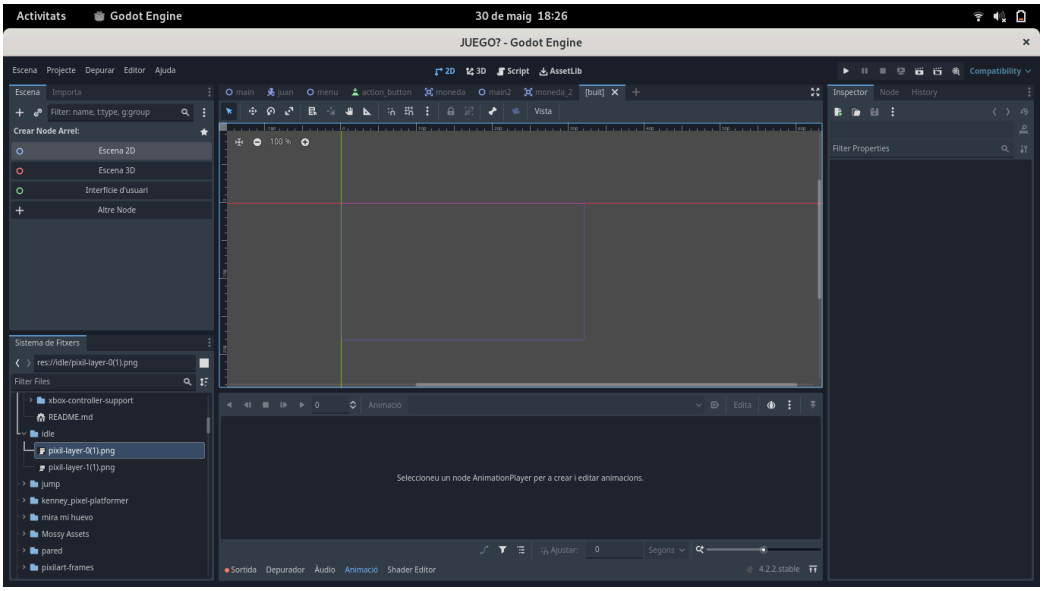
Luego, añadimos la comida favorita de cada uno: bambú para el panda y pescado para el pingüino. Se nos ocurrió poner un mar detrás de la comida, con nuestros personajes ofreciendo su comida favorita al otro.

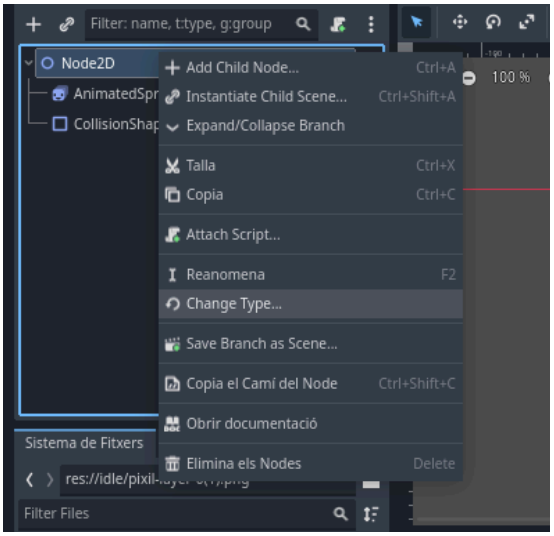


Para el logo final, añadimos detalles y correcciones. Colocamos el título en un estilo que lo hace parecer un bambú congelado.

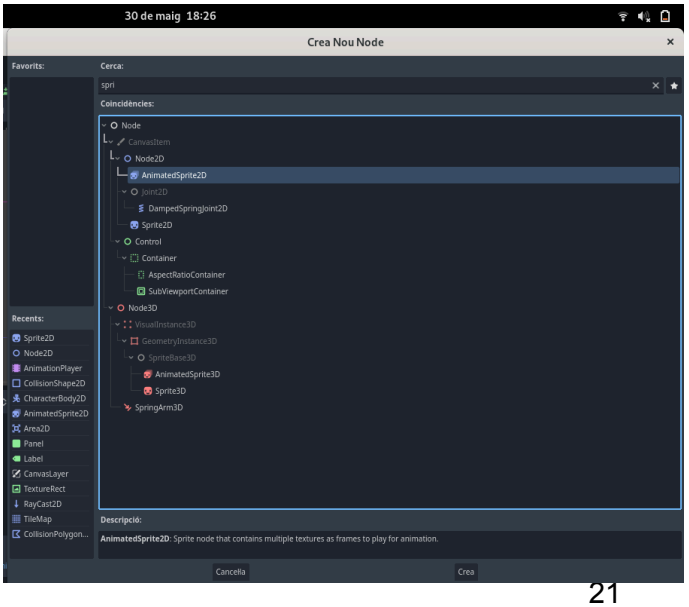
Como poner la animación en Godot

Para la implementación de la animación, debemos aplicarla a una escena 2D.

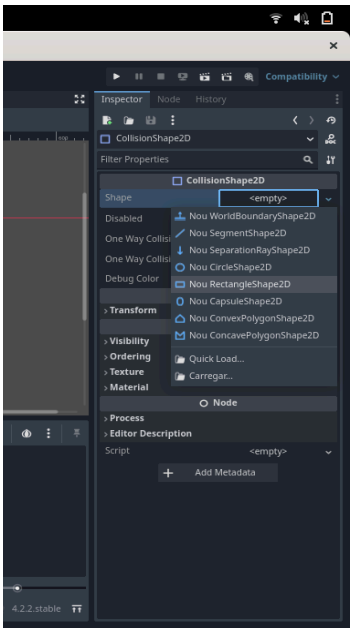


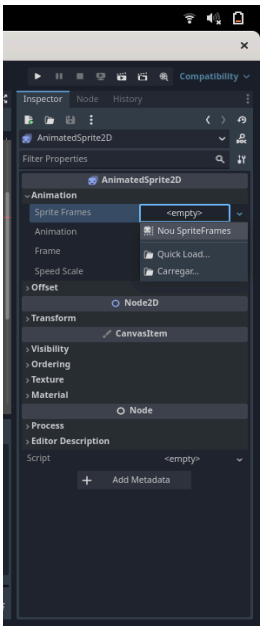


Una vez completado el paso anterior, hacemos clic derecho en Node2D y seleccionamos Agregar nodo hijo.

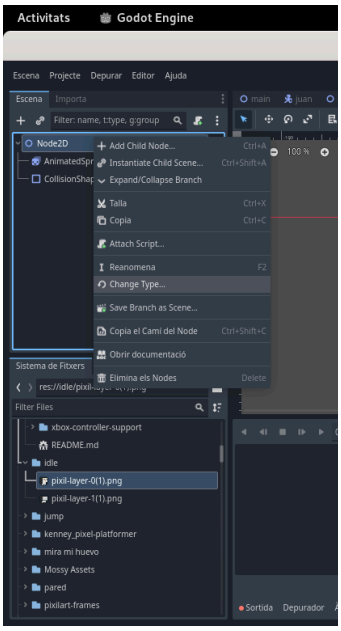


Agregamos AnimatedSprite2D y CollisionShape2D. El nodo AnimatedSprite2D se usará para insertar los sprites y darles animación, mientras que el nodo CollisionShape2D permitirá que el personaje interactúe con el terreno y no caiga del mapa.

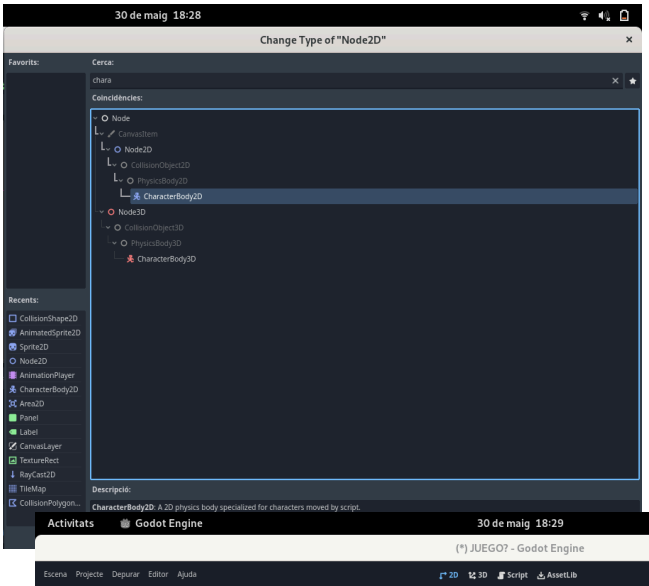




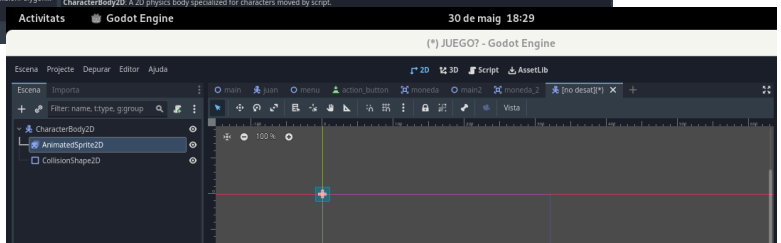
Una vez añadidos los nodos ponemos en la parte derecha la forma de la collision en nuestro caso será la rectangular, en el nodo de la animación ponemos en sprite frame uno nuevo así ya tenemos configurado todo para poder añadir la animación.



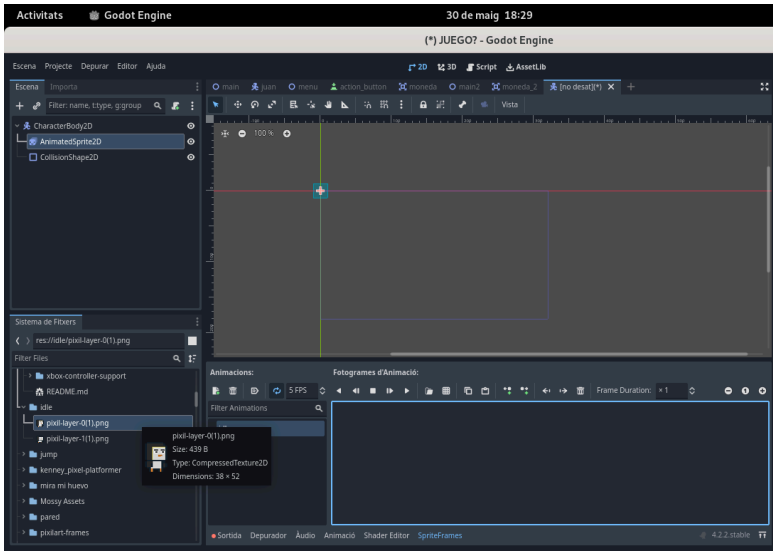
Antes de continuar, debemos cambiar el tipo de Node2D. Para hacerlo, hacemos clic derecho en Node2D y seleccionamos “Change Type”.



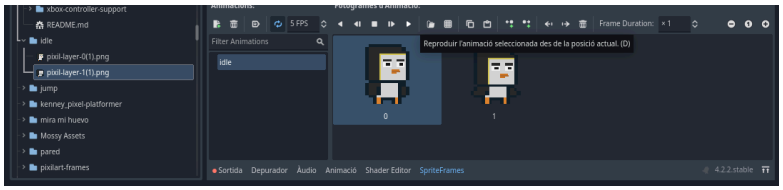
En “Change Type”, buscamos CharacterBody2D y lo seleccionamos. Una vez modificado, todo estará configurado.



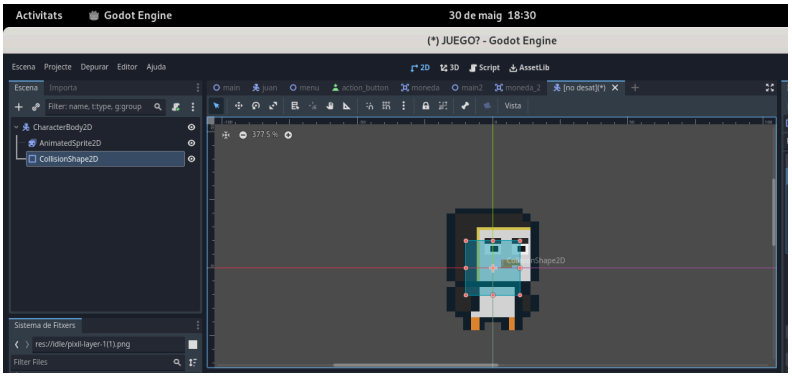
Para añadir una animación, hacemos clic en AnimatedSprite2D y luego en 'Default' en la parte inferior, donde podemos cambiar el nombre de nuestra animación.



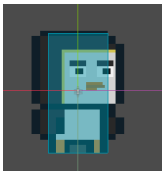
Una vez que hayamos establecido el nombre, buscamos en la parte izquierda los Sprite Frames que deseamos agregar. Cuando los encontremos, los arrastramos al recuadro marcado en azul que se encuentra a la derecha.



Una vez que los Sprite Frames estén agregados, se mostrarán de la siguiente forma.



Para ajustar la colisión, arrastramos desde el punto naranja hasta colocarlo según sea necesario, como se muestra en la siguiente imagen.



Diseños descartados

Descartamos los dos estilos de personajes porque, al finalizarlos, notamos que no se ajustaban al diseño que teníamos en mente. Por lo tanto, decidimos descartarlos y crear nuevos diseños. Así, llegamos a los diseños que ya tenemos.

Objetos

Para los objetos en cada nivel, hemos implementado una animación sencilla:

1. En el primer nivel, animamos un huevo con un pingüino, mostrando cómo se rompe la cáscara y sale el pingüino.

2. En el segundo nivel, animamos una seta como si alguien la estuviera comiendo.

3. En el último nivel, colocamos una pequeña estatua de un pingüino hecha de piedra, con una animación que muestra cómo se rompe y se le cae la cabeza.

Programación en Godot

Moviemiento del personaje

Salto, Salto coyote y Doble salto:

```
func _physics_process(delta):
	
	if is_on_floor():
		if velocity.y < 0:
			$AnimatedSprite2D.play("jump")
		
	
	if is_on_floor():
		leaved_floor = false
		had_jump = false
		count_jumps = 0
	if not is_on_floor():
		$coyote_timer.start()
		leaved_floor = true
	velocity.y += gravity * delta
	
```

En este código tenemos la función **"_physics_process"** en el momento en el que se realiza un proceso físico se establecen los condicionales dentro de la función.

if velocity.y < 0:
$AnimatedSprite2D.play("jump")

- Si la velocidad es mayor que 0 se activa la animación de salto.

if is_on_floor():
leaved_floor = false
had_jump = false
count_jumps = 0

- Si está en el suelo, no ha salido del suelo es decir "false", y no ha saltado el contador de saltos se queda en 0 ya que no ha realizado la acción.

if not is_on_floor():
$coyote_timer.start()
leaved_floor = true
velocity.y += gravity * delta

- Si no está en el suelo, se activa el “coyote timer”, que es un timer que hemos implementado, para que en el momento de que el personaje no esté en el suelo, pueda saltar.

```
func right_to_jump():
    if had_jump:
        if count_jumps < max_jumps: return true
        else: return false
    if is_on_floor():
        had_jump = true
        return true
    elif not $coyote_timer.is_stopped():
        had_jump = true
        return true
```

En esta función, estamos implementando en el momento en el que realice el salto, tenemos los condicionales para determinar qué acciones ocurren.

if had_jump:
if count_jumps < max_jumps: return true
else: return false

- Si ha saltado, el contador de saltos es menor al número máximo de saltos se vuelve verdadero, es decir, que al no llegar al número máximo puede realizar otro salto.”Else”,lo demás se convierte en falso.

if is_on_floor():
had_jump = true
return true
elif not $coyote_timer.is_stopped():
had_jump = true
return true

- Si el personaje está en el suelo (is_on_floor()), establece had_jump en verdadero y permite el salto. Si el personaje no está en el suelo pero el temporizador "coyote" ($coyote_timer) aún está activo, también permite el salto. El temporizador "coyote" permite un breve período de tiempo para saltar después de haber dejado el suelo, simulando un salto "en el aire".

```
if double_jump:
    double_jump = false
    $AnimatedSprite2D.play()
    velocity.y = JUMP_VELOCITY
```

if double_jump:
double_jump = false
$AnimatedSprite2D.play()
velocity.y = JUMP_VELOCITY

- Este fragmento permite que el personaje realice un doble salto y actualiza el estado del personaje para reflejar la acción

```
if Input.is_action_just_pressed("ui_up") and right_to_jump():
    if count_jumps <= 2:
        double_jump = true
    count_jumps +=1
    velocity.y = JUMP_VELOCITY
```

- Aquí verifica si el jugador ha presionado la tecla de salto y si se cumplen las condiciones para saltar. Si el contador de saltos es menor o igual a 2, permite el doble salto, incrementa el contador de saltos y aplica la velocidad de salto al personaje.

Variables utilizadas para el salto

```
var jump = false
```

```
var had_jump: bool = false
var max_jumps: int = 2
var count_jumps: int = 0
var double_jump: bool = false
```

Estas variables, son las que hemos utilizado en el código anterior, para poder especificar el conteo de saltos, el doble salto y el máximo de saltos.

Caminar

```
if direction:
    $AnimatedSprite2D.play("walk")
    velocity.x = direction * SPEED
```

```
if direction > 0:
    $AnimatedSprite2D.scale.x= 1
if direction < 0:
    $AnimatedSprite2D.scale.x= -1
```

Este apartado verifica si el personaje tiene una dirección de movimiento, inicia la animación de caminar y actualiza la velocidad horizontal del personaje según la dirección de movimiento y la velocidad definida.

if direction > 0:
$AnimatedSprite2D.scale.x= 1
if direction < 0:
$AnimatedSprite2D.scale.x= -1

- Ajusta la dirección en la que mira el sprite del personaje en función de su movimiento, asegurándose de que el personaje mire hacia la derecha cuando se mueve en esa dirección y hacia la izquierda cuando se mueve en la dirección opuesta.

Dash

if Input.is_action_just_pressed("dash") and can_dash:
dashing = true
can_dash = false
$dash_timer.start()
$dash_again_timer.start()

- Permite que el personaje realice un dash cuando el jugador presiona la tecla de dash, establece los estados necesarios para controlar el dash y usamos el temporizador para manejar la duración del dash y el tiempo de espera antes de permitir otro dash.

```
if dashing:
        velocity.x = direction * DASH_SPEED
        $AnimatedSprite2D.play("dash")
else:
        velocity.x = direction * SPEED
```

- Ajusta la velocidad y la animación del personaje dependiendo de si está en el estado de dash o no, permitiendo que se mueva más rápido y con una animación diferente cuando realiza un dash

```
func _on_dash_timer_timeout():
        dashing = false
```

- Esta función se llama automáticamente cuando el temporizador de dash termina, y su propósito es cambiar el estado del personaje para indicar que el dash ha finalizado.

```
func _on_dash_again_timer_timeout():
        can_dash = true
```

- Esta función se llama automáticamente cuando el temporizador que permite volver a hacer dash termina, y su propósito es permitir que el personaje pueda realizar otro dash.

Variables

```
var dashing = false
var can_dash = true
```

dashing: Es una variable booleana que indica si el personaje está actualmente en el estado de dash (desplazamiento rápido). Se establece en verdadero cuando el personaje está realizando un dash y en falso cuando no lo está.

can_dash: También es una variable booleana que indica si el personaje puede realizar un dash en ese momento. Se establece en verdadero cuando el personaje puede realizar un dash y en falso cuando no puede, como cuando ya está en el medio de un dash o cuando se está esperando un período de tiempo antes de permitir otro dash.

Menú principal

```
func _on_start_pressed():
    get_tree().change_scene_to_file("res://scenes/main.tscn")
```

- Al presionar el botón de inicio, el juego cambiará de la escena actual a la escena "main.tscn", que es la escena principal del juego.

Opciones

```
func _on_pressed():
    settings.show()
    get_parent().hide()
```

- Se muestra el menú de configuración y se oculta su contenedor padre.Para mostrar el menú de configuración mientras se oculta el menú principal del juego.

Back

```
4 func _on_pressed():
5     get_tree().quit()
6
```

func _on_pressed():
get_tree().quit()

- Al presionar el objeto al que se le asignó la función _on_pressed(), se cerrará el menú por completo.

Video

```
func _fullscreen(button_pressed):
	
	if button_pressed == true:
		DisplayServer.window_set_mode(DisplayServer.WINDOW_MODE_FULLSCREEN, 3)
		%FullScreen.button_pressed = true
	if button_pressed == false:
		DisplayServer.window_set_mode(DisplayServer.WINDOW_MODE_WINDOWED, 0)
		%Borderless.button_pressed = true
	print("test")
```

- Esta función cambia entre el modo de pantalla completa y el modo de ventana en función del estado del botón que la activa, y actualiza las propiedades de los botones relacionados en consecuencia. Luego, imprime "test" en la consola para verificar su funcionamiento.

```
func _on_full_screen_toggled(toggled_on):
	if toggled_on:
		DisplayServer.window_set_mode(DisplayServer.WINDOW_MODE_FULLSCREEN)
		Persistence.config.set_value("Video", "fullscreen", DisplayServer.WINDOW_MODE_FULLSCREEN)
	else:
		DisplayServer.window_set_mode(DisplayServer.WINDOW_MODE_WINDOWED)
		Persistence.config.set_value("Video", "fullscreen", DisplayServer.WINDOW_MODE_WINDOWED)

	Persistence.save_data()
```

- Esta función maneja el cambio entre el modo de pantalla completa y el modo de ventana cuando se activa o desactiva el botón, y también actualiza la configuración de persistencia del juego para recordar la preferencia del jugador.

```
func _on_borderless_toggled(toggled_on):
	DisplayServer.window_set_flag(DisplayServer.WINDOW_FLAG_BORDERLESS, toggled_on)
	Persistence.config.set_value("Video", "borderless", toggled_on)
	Persistence.save_data()
```

- Esta función maneja el cambio entre el modo de ventana con y sin bordes cuando se activa o desactiva el botón, y actualiza la configuración de persistencia del juego para recordar la preferencia del jugador.

```
func _on_vsync_item_selected(index):
	DisplayServer.window_set_vsync_mode(index)
	Persistence.config.set_value("Video", "vsync", index)
	Persistence.save_data()
```

- Esta función maneja la selección de opciones de sincronización vertical en un menú desplegable, ajusta el modo de sincronización vertical en función de la opción seleccionada y actualiza la configuración de persistencia del juego para recordar la preferencia del jugador.

Audio

```
func _ready():
	%Master.value = Persistence.config.get_value("Audio", '0')
	AudioServer.set_bus_volume_db(0,linear_to_db(%Master.value))

	%Music.value = Persistence.config.get_value("Audio", '1')
	AudioServer.set_bus_volume_db(1,linear_to_db(%Music.value))

	%SFX.value = Persistence.config.get_value("Audio", '2')
	AudioServer.set_bus_volume_db(0,linear_to_db(%SFX.value))
```

- Este código carga los valores de volumen guardados en la configuración persistente para los diferentes tipos de audio del juego (maestro, música, efectos de sonido) y ajusta los niveles de volumen correspondientes utilizando las funciones proporcionadas por AudioServer.

```
13  func _on_master_value_changed(value):
14      pass # Replace with function body.
15      set_volume(0,value)
```

- En resumen, esta función _on_master_value_changed ajusta el volumen cuando el valor del control deslizante de volumen maestro cambie.

```
func _on_music_value_changed(value):
    set_volume(1,value)
```

- En resumen, esta función asegura que cuando el valor del control deslizante de volumen de la música cambie, el volumen del canal de audio de la música se ajuste de acuerdo con ese valor.

```
func _on_sfx_value_changed(value):
    set_volume(2,value)
```

- Esta función asegura que cuando el valor del control deslizante de volumen de los efectos de sonido cambie, el volumen del canal de audio de los efectos de sonido se ajuste de acuerdo con ese valor

```
func set_volume(idx, value):
    AudioServer.set_bus_volume_db( idx, linear_to_db(value) )
    Persistence.config.set_value("Audio",str(idx),value)
    Persistence.save_data()
```

- Esta función set_volume ajusta el volumen de un canal de audio específico, guarda este ajuste en la configuración persistente del juego y luego guarda los datos actualizados en la configuración persistente para que los cambios sean permanentes.

Asignar teclas

```
func _ready():
    set_process_unhandled_key_input(false)
    display_key()
```

- La función _ready() desactiva el manejo de entradas de teclado no procesadas y luego llama a otra función llamada display_key(), que probablemente se encarga de mostrar las teclas en la interfaz de usuario o realizar alguna otra acción relacionada con las teclas.

```
func display_key():
    text = InputMap.action_get_events(action)[0].as_text()
```

- Esta función devuelve el texto de la tecla asociada a una acción específica en el InputMap. Por ejemplo, si la acción "jump" está asociada a la tecla "W", entonces esta función devolverá "W" cuando se llame con "jump" como argumento.

```
func remap_action_to(event):
    InputMap.action_erase_events(action)
    InputMap.action_add_event(action, event)

    Persistence.config.set_value("Controls", action, event)
    Persistence.save_data()

    text = event.as_text()
```

- Esta función permite al jugador volver a asignar una acción a un nuevo evento de entrada, actualiza la configuración persistente del juego para reflejar este cambio y devuelve el texto asociado al nuevo evento de entrada asignado.

```
func _on_pressed():
    set_process_unhandled_key_input(true)
    text = "press any key"
```

- Cuando se llama a esta función _on_pressed(), activa el manejo de entradas de teclado no procesadas y establece un mensaje en la interfaz de usuario para indicar que el jugador debe presionar una tecla.

```
func _unhandled_key_input(event):
    remap_action_to(event)
    set_process_unhandled_key_input(false)
    release_focus()
```

- Esta función maneja eventos de teclado no procesados en una acción en el InputMap a un nuevo evento de teclado, luego desactiva el manejo de eventos de teclado no procesados y libera el enfoque del nodo.

Sistema de monedas

```
func _on_moneda_recolectar_monedas():
    monedas += 1
    get_node("CanvasLayer/TextoMonedas").text = ": "  + str(monedas)

    if monedas == 5:
        get_tree().change_scene_to_file("res://scenes/main2.tscn")
```

- Esta función maneja la recolección de monedas en el juego. Incrementa el contador de monedas, actualiza el texto mostrando la cantidad de monedas recolectadas y cambia la escena si el jugador ha recolectado un número específico de monedas (en este caso, 5).

```gdscript
func _on_body_entered(body):
    if body.is_in_group("jugador"):
        $AudioStreamPlayer2.playing = true
        emit_signal("recolectar_monedas")
        queue_free()
```

- Esta función maneja el evento de un cuerpo físico que entra en contacto con el área a la que está conectado este script. Si el cuerpo es del grupo "jugador", reproduce un sonido, emite una señal indicando que se ha recolectado una moneda y luego elimina el objeto de la escena.

Sistema de daño

```gdscript
func _on_body_entered(body):
    if body.get_name() == "Juan":
        print("Has muerto")
        get_tree().reload_current_scene()
```

- Esta función maneja el evento de un cuerpo físico que entra en contacto con el área a la que está conectado este script. Si el cuerpo tiene el nombre "Juan", imprime un mensaje de muerte en la consola y recarga la escena actual.

Música

```gdscript
@onready var player = $AudioStreamPlayer
```

- Declara una variable llamada player y la inicializa con una referencia al nodo $AudioStreamPlayer en el árbol de nodos. El prefijo @onready indica que esta variable se inicializa una vez que el nodo al que está asociado este script esté listo. Esto significa que player contendrá una referencia válida al nodo $AudioStreamPlayer que se encuentra en la escena.

```
func _ready():
    player.play()
```

- Este fragmento de código se ejecuta cuando el nodo asociado al script está listo para ser utilizado. Aquí, se llama al método play() en el objeto player. Esto indica que el audio asociado a ese reproductor comenzará a reproducirse tan pronto como el nodo esté listo.

Visión a futuro

- Implementar el segundo personaje para crear una segunda parte del videojuego.
- Crear un videojuego nuevo llamado Paqui&Juan.
- Arreglar posibles problemas.
- Mejorar el videojuego con los comentarios y propuestas de los usuarios.
- Añadir más contenido.

Glosario

A
AnimationPlayer: Nodo utilizado para crear y reproducir animaciones. Permite animar propiedades de otros nodos.
Area2D/Area3D: Nodos que permiten detectar la presencia de objetos dentro de un área definida, útil para colisiones y detección de eventos.
B
BoxContainer: Lugar donde se establece un container para marcar.
C
Camera2D/Camera3D: Nodos que representan la vista del jugador. La cámara puede seguir al jugador o tener movimientos programados.
CollisionShape2D/CollisionShape3D: Nodos que definen la forma de colisión de un objeto físico.
D
Delta: Tiempo transcurrido desde el último frame, utilizado para movimientos y actualizaciones dependientes del tiempo.
E
Editor: Entorno de desarrollo integrado (IDE) de Godot donde se crean y editan proyectos, escenas y scripts.
Export: Proceso de compilar y empaquetar el juego para diferentes plataformas como Windows, Linux, Android, iOS, etc.

G

GDScript: Lenguaje de programación propio de Godot, diseñado para ser fácil de aprender y utilizar dentro del motor.

GUI (Graphical User Interface): Interfaz gráfica de usuario. En Godot se puede crear usando nodos de tipo Control.

I

Inspector: Panel del editor donde se pueden modificar las propiedades de los nodos seleccionados.

Instance: Creación de una copia de una escena o nodo que se puede reutilizar en diferentes partes del juego.

L

Label: Lugar donde se escribe el texto para proyectar.

N

Node: Elemento básico de Godot. Todos los objetos dentro de una escena son nodos, que pueden tener funcionalidades específicas dependiendo de su tipo.

R

RayCast2D/RayCast3D: Nodos que proyectan una línea invisible en el espacio para detectar objetos en su camino, útil para detección de colisiones y línea de visión.

S

Scene: Unidad básica de organización en Godot, compuesta por un árbol de nodos que define una parte del juego, como un nivel o un personaje.

Script: Código escrito en GDScript, C#, o VisualScript que añade lógica a los nodos.

T

TileMap: Nodo utilizado para crear y gestionar mapas de tiles, permitiendo construir niveles de forma eficiente.

V

Viewport: Nodo que puede renderizar una escena completa. Puede ser utilizado para crear efectos como cámaras secundarias o mini-mapas.

Consecución de los objetivos

Todos los objetivos que nos pusimos como meta se han cumplido en nuestro videojuego. A lo largo del desarrollo, fuimos incorporando nuevas ideas, pero las descartábamos a medida que avanzábamos. Por ejemplo, nos propusimos incluir ataques y enemigos, pero al final decidimos no hacerlo y nos centramos en la idea principal de un juego de plataformas, donde el jugador debe pasar niveles y superar obstáculos.

Valoración de la metodología y planificación

Nuestra metodología y planificación han surtido efecto, y hemos conseguido todos los objetivos que teníamos propuestos.Al dividir el trabajo entre nosotros hemos conseguido poner todo lo que teníamos en nuestra mano, ya que hemos aplicado lo aprendido después de documentarnos.

Conclusiones

En este proyecto no hemos embarcado en el mundo del desarrollo de videojuegos, nos ha servido para incrementar nuestros conocimientos generales sobre el desarrollo de estos.Sin embargo, no solo nos quedamos ahí ya que hemos conseguido llegar a nuestros objetivos, esto nos ha conllevado a mejorar como personas y ser más responsables, ya que para desarrollar un videojuego tenemos que tener disciplina y sobretodo trabajo en equipo.Hemos conseguido cumplir los objetivos que teníamos en mente para nuestro proyecto, incluso añadiendo objetivos mientras pasaba el tiempo.

Bibliografía de la creación del videojuego y animación:

https://www.youtube.com/playlist?list=PLO93miOntN603vPv4kI7Cj_AUshj5-oEm
https://www.youtube.com/playlist?list=PLD8DG6aXaGSupfJk9DI-CXI33bVL6mA4_
https://www.youtube.com/playlist?list=PL5PTqiCIVoiVQtSQxgfEhKBUbZBooY-Uv
https://www.youtube.com/watch?v=rgJn2gA8exY
https://www.youtube.com/watch?v=Z8jcjy_jZyk
https://www.youtube.com/watch?v=7qYxnY_ghw8&list=PLNEAWvYbJJ9nNOpe6fun7m6L_M8xslYnT&index=9
https://www.youtube.com/watch?v=PFfCBTK1uEc
https://www.youtube.com/watch?v=2zZRQOzcw7Y&list=PLaFm85QGi-mGIf5VJBggxdTdaqQq7gR8J&index=5
https://www.youtube.com/watch?v=5sUWuSXyv8o
https://www.youtube.com/watch?v=r_lGkg7-01A&list=PL-oJEh-N3A3SOPWuMuulbnJv0BFgvBnVG&index=17