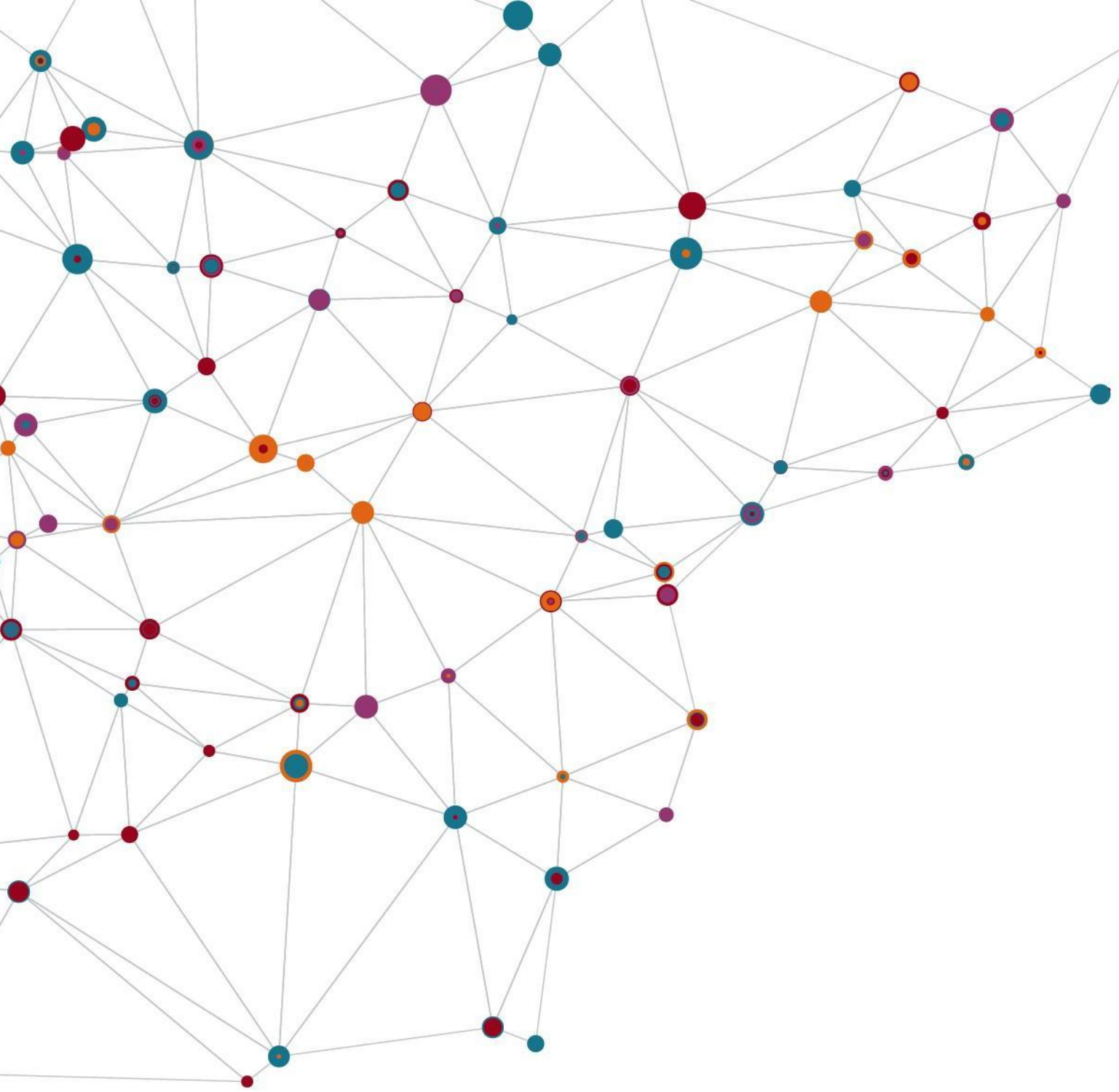


Formación Profesional

Unidad N°4

Clase 4: Introducción a Godot. Interfaz de trabajo. Escenas y nodos. Inspector. Prácticas en Godot - Captura la bandera. Spaceship.

Objetivos.

- Reconocer interfaz del programa.
- Reconocer comandos y operaciones básicas del programa.
- Apropiación y aplicación de conceptos básicos en relación al núcleo de videojuego.
- Iniciarse en el diseño de un videojuego.
- Iniciarse en la animación 2D por medio de Sprite Sheets.

Temas.

Introducción a Godot. Instalación. Interfaz de trabajo. Escenas y nodos. Inspector. Prácticas en Godot: Captura la bandera. Creando personajes. Creando enemigos. Núcleo de videojuego. Sprite sheets y animación.

1. Desarrollo.

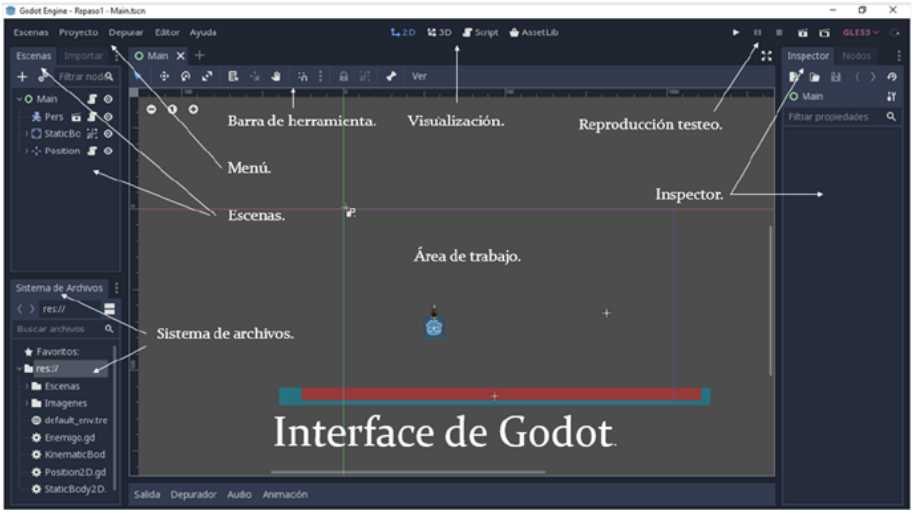
Introducción al motor de desarrollo de videojuegos “Godot”.

Godot es un motor de videojuegos 2D y 3D multiplataforma, de código abierto publicado bajo la Licencia MIT y desarrollado por la comunidad de Godot. El motor funciona en Windows, OS X, Linux . Soporta varios lenguajes de programación, entre ellos C#, Visual Script y GD Script, el cual está basado en la sintaxis y lógica de programación de Python.

Puede exportar los videojuegos creados a PC (Windows, OS X y Linux), teléfonos móviles (Android, iOS), y HTML5.

El desarrollador principal, Juan Linietsky, declaró en una presentación que el nombre Godot está relacionado con la obra teatral “Esperando a Godot” de Samuel Beckett, y representa el deseo de añadir continuamente características nuevas en el motor, acercándose cada vez más a un producto exhaustivo, pero no llegando nunca al final.

Interfaz y área de trabajo.



Sistemas de archivos: Es el sector en donde se podrá visualizar el total de archivos que contendrá nuestro proyecto.

Escenas: Es el sector en donde se podrá visualizar una escena y el total de nodos que la componen.

Menú: Es una barra situada en el extremo superior izquierdo de la pantalla donde se ubican las diferentes pestañas que contienen las herramientas principales de configuración del motor.

Barra de herramientas: Es una barra de rápido acceso a herramientas de uso frecuente.

Visualización: Es una barra situada en la parte superior central de la pantalla donde se ubican los botones que nos permitirán acceder a la vista 2D, 3D, el código y una librería de assets.

Reproducción y testeo: Es una barra que contiene los botones que nos permitirán jugar, pausar, detener, debugear y testear nuestro proyecto.

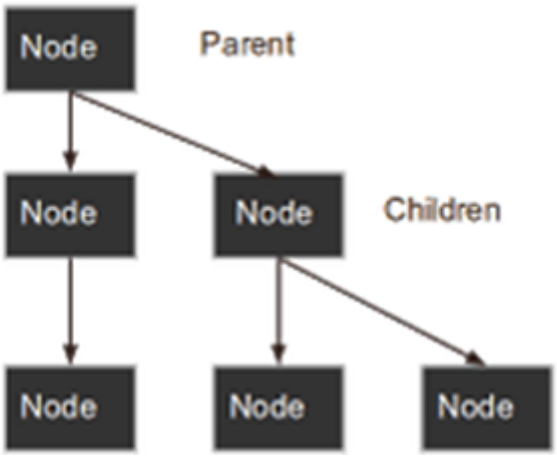
Inspector: Es el sector en donde se podrán visualizar y modificar una gran cantidad de características de un elemento seleccionado.

Nodos.

Todo dentro de Godot es un nodo y mediante ellos se crean objetos con características definidas. Antes de continuar conviene recalcar que la terminología Nodo hay que usarla con cuidado. Cuando nos referimos a los Nodos, nos referimos a pequeñas cajitas que conectan con líneas, y que son parte de un gráfico. Cuando nos referimos a Nodos de escena, esto implica que nos referimos a los elementos que configuran una escena, y que son parte del árbol de dicha escena. El nombre es similar pero su función es diferente.

Como se mencionó anteriormente, un nodo puede realizar una variedad de funciones especializadas. Sin embargo, cualquier nodo utilizado siempre tiene los siguientes atributos:

- Tiene un nombre.
- Tiene propiedades editables.
- Puede recibir una llamada a una función (callback) para procesar cada fotograma.
- Se puede extender (para tener más funciones).
- Se puede añadir a otro nodo como un hijo.



Lo último es importante. Los nodos pueden tener otros como hijos. Cuando se disponen de esta manera, los nodos se convierten en un árbol.

En Godot, la capacidad de organizar nodos de esta manera crea una poderosa herramienta para organizar proyectos. Como los diferentes nodos tienen diferentes funciones, combinarlos permite la creación de funciones más complejas.

No te preocupes si esto aún no te hace clic. Continuaremos explorando esto en las siguientes secciones. El hecho más importante para recordar por ahora es que los nodos existen y se pueden organizar de esta manera.

Vamos a analizar un ejemplo: Pensemos en una roca.

La roca tiene las siguientes características: es un objeto sólido/rígido por lo tanto la física newtoniana influye sobre él, tiene una forma, esa forma delimita el área que ocupa y tiene color y textura, y esto le permite ser visible.

Para crear una roca en Godot lo haríamos de la siguiente forma:

- (Nodo) RigidBody2D
  - (Nodo) CollisionShape2D
  - (Nodo) Sprite.

El nodo **RigidBody2D** le indica al programa que queremos algo en 2 dimensiones, rígido que tiene masa, peso, rebote, fricción, es afectado por la gravedad, etc.

El nodo **CollisionShape2D** le dice al programa cuál será la figura 2D que delimitará el espacio que ocupa el cuerpo rígido.

Y el **Sprite** es básicamente la imagen o textura que representará el objeto. Normalmente entre el **CollisionShape2D** y el **Sprite** se define la forma del objeto que se crea.

Nótese que los nodos **CollisionShape2D** y **Sprite** están dentro del nodo RigidBody2D formando un árbol de nodos donde **RigidBody2D** es el padre y los nodos dentro suyo son los nodos hijos.

Escenas.

A su vez, este conjunto de nodos dispuesto en forma de árbol, es denominado una **Escena**. Cada objeto dentro de un juego puede ser una escena. Por ejemplo, si recreamos el juego ***Súper Mario Bros.***, Mario sería una escena que contendría un nodo principal llamado nodo padre y varios nodos hijos que aportarían una característica o cualidad especial a ese todo llamado Mario. Cada enemigo sería otra escena, el nivel otra, los honguitos otra y así sucesivamente.

Script.

Pero para hacer que Mario salte o camine, debemos utilizar una herramienta que Godot nos proporciona llamada **"Script"**. El script será la hoja donde le diremos al programa cómo deben comportarse los nodos mediante escritura de código y será creado y guardado dentro de su correspondiente nodo, es decir, el script que indica cuándo Mario debe saltar será a guardado dentro de Mario; de esta forma el programa sabrá que solamente Mario va a saltar y no todo el nivel.

Núcleo de videojuego.

Al hablar de núcleo de videojuego hacemos referencia al proceso de diseño, programación e implementación de la mecánica principal del juego, aquella acción que se repetirá una y otra vez continuamente.

Por ejemplo, en el videojuego Tetris, el núcleo estaría definido una vez que se programa la caída, movimiento lateral y rotación de una pieza. Una vez que el núcleo funciona, se comienza a agregar las diferentes funcionalidades, objetos y elementos que hacen a la composición final del videojuego, las restantes piezas, los fondos, la interfaz, sonidos, música.

Sprite sheets y animación.

¿Qué es un sprite?

Un Sprite, es básicamente un mapa de bits, imagen o ilustración que se utiliza en un videojuego. Normalmente cuenta con transparencia y es un asset reutilizable, lo que ayuda a conservar el buen rendimiento a la hora de administrar recursos.

Al realizar una animación dentro de un videojuego lo que hacemos es crear una secuencia de sprites de un objeto que tienen pequeñas variaciones entre sí. Al reproducir esa secuencia se logra la sensación de movimiento.

Si analizamos este proceso, fácilmente veremos que no es una forma óptima de administrar recursos ya que cada sprite se estará cargando por separado una y otra vez mientras dure la animación. Para darle solución a este tema existe el sprite sheet.

¿Qué es un sprite sheet?

Un sprite sheet, es una imagen conformada por sprites. Suele ser de un tamaño mayor que el de un sprite pero mantiene la relación de aspecto con él. En el sprite sheet se encuentran todos los sprites de un objeto en una sola imagen, esto hace que a la hora de animar el objeto el rendimiento sea mejor ya que su tiempo de carga es más rápido, pensemos que solo carga un asset.

Animación en un videojuego.

La animación es el resultado de la técnica y proceso que se lleva a cabo sobre una serie de imágenes para que estas den la sensación o noción de movimiento.

En videojuegos se puede utilizar una secuencia de sprites o un sprite sheet para realizar la animación.

Actualmente los motores de desarrollo de videojuegos tienen incorporado un animador en el cual se puede crear animaciones sin salir del motor, pero también

existe la posibilidad de crear animaciones con programas externos pero compatibles con el motor y luego introducirlas al juego como un asset.

Aplicación práctica de los conceptos.

Siguiendo con el espíritu de este curso, continuaremos con la aplicación práctica de los conceptos vistos creando un ejemplo de núcleo de videojuego.

Programaremos el núcleo de un videojuego que tiene por mecánica principal el movimiento en cuatro direcciones de un personaje.

1- Creación de un proyecto.

Hacemos doble click en el ícono de Godot para ejecutar el programa. La primera pantalla que veremos es el gestor de proyectos. Hacemos click en el botón **Proyecto nuevo**, nombramos el proyecto, seleccionamos o creamos una carpeta para alojar el proyecto, seleccionamos el renderizador y hacemos click en el botón **Crear y editar**.

2- Creación de una escena principal.

Ahora estamos ubicados en el área de trabajo de Godot. Por defecto viene predeterminada la vista 3D. Haremos click en el botón 2D que se encuentra en la barra de visualización.

Luego haremos click en el botón **Nodo personalizado** que se encuentra en la pestaña de **Escena**, se abrirá la ventana de **Crear Nuevo Nodo** y en la barra de búsqueda escribiremos **KinematicBody2D**, luego haremos click en el botón **Crear**.

A continuación haremos click derecho sobre el nodo **KinematicBody2D** que ha aparecido en el área de **Escena**. Se desplegará una ventana de opciones en donde haremos click en la primera opción, **Agregar Nodo Hijo**. Se volverá a abrir la ventana **Crear Nuevo Nodo** y en la barra de búsqueda escribiremos **CollisionShape2D**, luego haremos click en el botón **Crear**.

Repetiremos estos pasos para agregar dos nodos más al **KinematicBody2D**, el nodo **Sprite** y el nodo **AnimationPlayer**.

El árbol de nodos debería quedar conformado de la siguiente manera:

KinematicBody2D

CollisionShape2D

Sprite

AnimationPlayer

3- Animación.

Después de esto seleccionaremos el nodo **Sprite** y ubicaremos en el **Inspector**, el apartado de **Texture** en donde cargaremos el sprite sheet de nuestro personaje. El sprite sheet que vamos a utilizar es de 3 sprites de ancho por 4 de alto, lo que significa que la animación de todo nuestro personaje será de 12 frames (cuadros).

Desplegamos la pestaña **Animation** en el inspector y seteamos el **Vframe** en 4 y el **Hframe** en 3.

Seleccionamos el nodo **AnimationPlayer** en el área de escenas y esto abrirá un sector de trabajo de animaciones en la parte inferior de la pantalla.

Hacemos click en el botón **Animación** y luego en el botón **Nuevo**. Nombramos a nuestra animación **Izquierda**.

Ubicamos la imagen de un reloj que indica la duración de la animación y lo seteamos en 0.6. (Esta configuración puede variar a gusto del desarrollador. Cuantos más frames tenga la animación, más fluido será el movimiento, por lo cual, el seteo de tiempos debería modificarse.)

Ubicamos el **snap** que indica el paso de la animación y lo seteamos en 0.2. (Esta configuración puede variar a gusto del desarrollador. Cuantos más frames tenga la animación, más fluido será el movimiento, por lo cual, el seteo de tiempos debería modificarse.)

Seleccionamos el nodo **Sprite** y en el inspector ubicamos la propiedad **frame**. Seteamos el valor de **frame** en 5 y luego hacemos click en la imagen de la llave que está al lado.

Ubicamos el contador de la animación y lo seteamos en 0.2, luego, seteamos el valor de **frame** en 4 y volvemos a hacer click en la imagen de la llave que está al lado.

Repetimos los pasos nuevamente, ubicamos el contador de la animación y lo seteamos en 0.4 seteamos el valor de **frame** en 3 y luego hacemos click en la imagen de la llave que está al lado.

En este punto ya hemos creado la animación con la cual nuestro personaje podrá caminar hacia la izquierda.

Del mismo modo podemos crear las animaciones para las tres direcciones restantes.

4- Programando el movimiento.

Hacemos click derecho en el nodo **KinematicBody2D** y luego en **Adjuntar script**. En la ventana que se ha abierto, ubicaremos la propiedad **Plantilla** y la setearemos en **No comentada**, luego hacemos click en el botón **Crear**.

Una vez abierta la hoja de script, entre el **extend KinematicBody2D** y el **func _ready():** agregaremos:

```
var movimiento = Vector2()
var velocidad = 100
```

Después del **func _ready():** agregaremos:

```
func _physics_process(delta):
    if Input.is_action_pressed("ui_right"):
        movimiento.x = velocidad
        $AnimationPlayer.play("Derecha")
    if Input.is_action_pressed("ui_left"):
        movimiento.x = -velocidad
        $AnimationPlayer.play("Izquierda")
    if Input.is_action_pressed("ui_up"):
        movimiento.y = -velocidad
        $AnimationPlayer.play("Arriba")
    if Input.is_action_pressed("ui_down"):
        movimiento.y = velocidad
        $AnimationPlayer.play("Abajo")
    if Input.is_action_just_released("ui_right") or
    Input.is_action_just_released("ui_left") or
    Input.is_action_just_released("ui_up") or
    Input.is_action_just_released("ui_down"):
```

```
            movimiento.x = 0
            movimiento.y = 0
            $AnimationPlayer.stop()
    move_and_slide(movimiento)
```

De este modo ya hemos diseñado y programado el movimiento de nuestro personaje, nuestro núcleo de juego.

Prototipo de juego.

Hemos programado el núcleo de un juego. Ahora iremos añadiendo funcionalidades hasta lograr un prototipo de juego tipo **"captura la bandera"**.

1- Nuevo Proyecto.

Crearemos un nuevo proyecto en Godot. Una vez abierto crearemos un nodo de control **Node2D**, lo renombramos **Main** y adjuntamos un script. Luego guardaremos el proyecto con Ctrl+S.

Dentro de esta escena crearemos al personaje principal al igual que lo hicimos en la unidad anterior y lo renombramos **Personaje**.

2- Creando una bandera.

Agregar un **Sprite** arrastrando la bandera desde el área de sistemas de archivos hasta el área de trabajo. De ser necesario la renombramos **Bandera**.

Le agregaremos un **Area2D**, luego un **CollisionShape2D** y un **shape**. Ajustamos el shape al Sprite.

3- Creando un enemigo.

Los pasos a seguir son similares a los de la creación de un personaje.

Crear un **KinematicBody2D**, renombramos **Enemigo**, agregar al **Enemigo CollisionShape2D** y un **Sprite**. Agregar una textura al **Sprite,** utilizaremos para esto el ícono de Godot que viene por defecto en el sistema de archivos. Luego agregamos un shape al **CollisionShape2D**. Ajustar el **shape** al **Sprite**.

Agregar un script al **Enemigo**, con el siguiente código:

Entre el **extendKinematicBody2D** y el **func _ready():** agregar:

```
var movimiento = Vector2()
var velocidad = 100
```

Después del **func _ready():** agregar:

```
func _physics_process(delta):
    move_and_slide(movimiento)
    set_vector(get_node("../Bandera").global_position - global_position)
func set_vector(vector):
    movimiento = vector.normalized() * velocidad
```

Con todo lo hecho hasta ahora tenemos un personaje que podemos mover en cuatro direcciones y un enemigo que se mueve automáticamente. Pero aún no funciona ya que faltan las condiciones de victoria y derrota.

4- Creando una condición de victoria/derrota

Seleccionaremos el nodo **Area2D** que pertenece a la bandera y haremos click en la pestaña **Nodo** que se encuentra al lado de la pestaña del Inspector, ubicamos la señal **body_entered(body: Node)**, haremos click en ella y luego click en el **Main** para conectar la señal con el script.

Agregamos el siguiente script al **Main** dentro de la función que se ha creado:

```
func _on_Area2D_body_entered(body):
    if body.name == "Personaje":
        $Enemigo.queue_free()
        $Bandera.queue_free()
    if body.name == "Enemigo":
        $Personaje.queue_free()
    pass
```

Ya hemos creado un prototipo.

Agregando Splash Screen y un Menú Principal.

¿En qué consisten las Splash Screens? Es un recurso tan sencillo como un fondo simple, generalmente blanco o del color principal de la aplicación, junto con el o los logotipos de nuestra aplicación, videojuego, empresas, etc, y un breve texto como el nombre de la app o las empresas por ejemplo.

En videojuegos que utilizan muchos recursos se usa de pantalla de carga. Es muy importante ya que junto al menú principal son la carta de presentación del videojuego.

El Menú Principal se compone de un conjunto de submenús, en videojuegos son botones, desplegables, links, etc, que nos permite realizar las primeras interacciones con el videojuego. Normalmente está dispuesto en una pantalla con fondo y diseño definido. Desde aquí se accede al resto de sistemas que componen el videojuego.

1- Agregando un splash screen.

- Crear una nueva escena.
- **Proyecto→ Ajustes de Proyecto→ Rendering→ Environment.**
- En **Environment** setear el **Default Clear Color** en el color de fondo que deseamos. Luego **Cerrar**.
- Creamos una escena **Main** con un nodo de control.
- Al nodo **Main** le agregamos un **Label**, un **Sprite** y un **AnimationPlayer**.
- Descargamos una fuente de la web.
- Seleccionamos el **Label** y en el **Inspector** agregamos el texto, luego nos dirigimos a **Custom Fonts**, hacemos click en **vacío** y seleccionamos **Nuevo DynamicFont**.
- Hacemos click en **Nuevo DynamicFont** y luego en **Font**.
- Agregamos la fuente descargada al **Sistema de Archivos** y la arrastramos hasta el desplegable **vacío**.
- Desplegamos **Settings** y seteamos el tamaño de la fuente y la ubicación centrada dentro de la pantalla.
- Seleccionamos el **Sprite** y cargamos el **Texture** con el ícono de Godot. Lo centramos en la pantalla.
- Seleccionamos el **AnimationPlayer** y creamos una nueva animación llamada **Splash_Screen**.
- Seleccionamos el **Sprite**, en el **Inspector** seleccionamos **Visibility** y desde allí hacemos click en **Modulate** y modificamos el canal alfa a 0.
- Hacemos click en la llave a un costado de **Modulate** y le damos **crear** en la ventana emergente.
- Seteamos la duración de la animación en 8.
- Del segundo 0 al 2 seteamos el pasaje del alfa de 0 a 100.
- Del 2 al 4 se mantiene.
- Del 4 al 6 vuelve a 0.
- Hacemos lo mismo para el **Label** pero seteamos el **Modulate** con un retraso de 2 segundos.
- Del 2 al 4 aparece.
- Del 4 al 6 se mantiene,
- Del 6 al 8 desaparece.
- Marcamos **Autoreproducir al Cargar**.
- Seleccionamos el **Nodo2D** y agregamos un script.
- Seleccionamos el **AnimationPlayer** y cambiamos de **Inspector** a **Nodo**.
- Conectamos la señal **animation_finished(anim_name: String)** al script.

- En el cuerpo de la función agregamos la siguiente línea de código:
    - get_tree().quit()

2- Agregando un Menú Principal.

- Creamos una escena **Menu** con un nodo de control.
- Al nodo **Menu** le agregamos un **Sprite** y dos **Button**.
- Descargamos un fondo de la web.
- Añadimos el fondo al **Sprite** y lo ubicamos en la pantalla.
- Ubicamos ambos botones en la pantalla modificando el texto y el tamaño.
- Uno de los botones será el **Play** y el otro será el **Exit**.
- Seleccionamos el **Menu** y agregamos un script.
- Seleccionamos el **Button 1** y cambiamos de **Inspector** a **Nodo**.
- Conectamos la señal **pressed()** al script.
- En el cuerpo de la función agregamos la siguiente línea de código:
    - **get_tree().change_scene("res://Node2D.tscn")**
- Seleccionamos el **Button 2** y cambiamos de **Inspector** a **Nodo**.
- Conectamos la señal **pressed()** al script.
- En el cuerpo de la función agregamos la siguiente línea de código:
    - **get_tree().quit()**
- Volvemos al script del **Nodo2D** y reemplazamos:
    - **get_tree().quit()** → **get_tree().change_scene("res://Menu.tscn")**

2. Autoevaluación

Descripción.

Te invitamos a que evalúes tu comprensión y dominio del tema tratado. Al superar el desafío se te habilitará la siguiente unidad y podrás continuar.

Instrucciones.

A continuación hallarás algunas preguntas que pondrán a prueba tus saberes. Para pasar esta etapa es importante que hayas leído la unidad y practicado con los ejercicios que propuso la misma.

Preguntas y respuestas.

1. Cuando hablamos de Nodo, nos referimos a:

Rtas:

a) Un punto de una imagen.
b) Un elemento de la escena configurable.
c) Una escena.

2. El sistema de Script de Godot me permite:

Rtas:

a) Programar el comportamiento de los nodos individualmente.
b) Programar el comportamiento del árbol de nodos.
c) Programar el comportamiento de toda la escena.

3. El CollitionShape2d define:

Rtas:

a) La hoja donde le diremos al programa cómo deben comportarse los nodos mediante escritura de código.
b) La imagen o textura que representará el objeto.
c) La figura 2D que delimitará el espacio que ocupa el cuerpo.

4. ¿Qué es un núcleo de videojuego?

Rtas:

a) Es el proceso de diseño, programación e implementación de la mecánica principal del juego.
b) El renderizado de una mecánica con su respectiva animación.
c) Un documento que define un prototipo.

5. ¿Qué es un sprite sheet?

Rtas:

a) Es un mapa de bits.
b) Es una imagen.
c) Es una imagen conformada por sprites.

6. ¿Qué es un prototipo de videojuego?

Rtas:

d) Es el núcleo con el agregado de las diferentes funcionalidades, objetos y elementos que permiten el testeo del flujo completo del videojuego.
e) Es el agregado de las diferentes funcionalidades, objetos y elementos que hacen a la composición final del videojuego.
f) Es un videojuego que no está terminado.

3. Tarea.

1. Crear una pelota en Godot.

2. Modificar el prototipo realizado en clase agregando más enemigos, estos deben perseguir al jugador.

3. Seguir el siguiente tutorial para el desarrollo de un videojuego tipo space shooter:

Replicar prototipos de videojuegos: Spaceship.

Siguiendo con el espíritu de este curso, continuaremos con la aplicación práctica de los conceptos vistos replicando un videojuego hasta que obtengamos un prototipo.

Primero programaremos el núcleo de un videojuego estilo Spaceship que tiene por mecánica principal el movimiento en dos direcciones y disparos de una nave. Luego crearemos un enemigo y una condición de victoria y derrota. Así habremos hecho un prototipo de videojuego más.

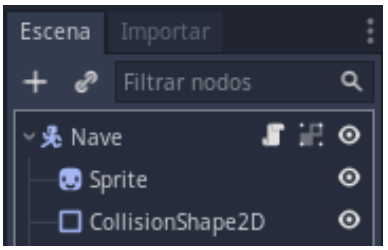
1- Creación de un proyecto y una escena principal.

Crearemos un proyecto y una escena principal de igual forma que en la clase pasada, llamaremos a la escena principal **Main**. Luego sumaremos todos los recursos necesarios al sistema de archivos:

- Nave.png
- Rayo.png
- Roca.png
- Explosion.png
- Laser_sound.ogg

2- Creación de Nave.

Para la creación de la nave utilizaremos el siguiente árbol de nodos:



Al nodo **Nave** le añadiremos además un script con el siguiente código:

```
extends KinematicBody2D

var movimiento = Vector2()

var velocidad = 200

func _physics_process(delta):

    if Input.is_action_pressed("ui_left"):
```

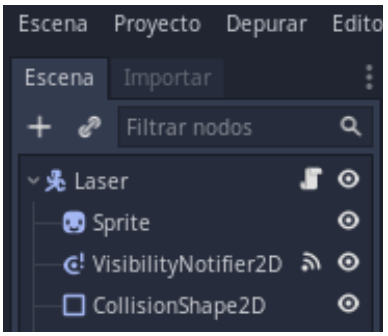
```
            movimiento.x = -velocidad
        if Input.is_action_pressed("ui_right"):
            movimiento.x = velocidad
        if Input.is_action_just_released("ui_left") or
Input.is_action_just_released("ui_right"):
            movimiento.x = 0
            movimiento.y = 0
        move_and_slide(movimiento)
```

No nos olvidemos de guardar la nave como una escena que llamaremos igual que el nodo principal, **Nave**.

Dentro del **Main** debemos crear dos nodos **StaticBody2D**, añadirles un **CollisionShape2D** y colocarlos en ambos laterales de la pantalla de modo que cuando nuestra nave se acerque, colisione y no salga de la pantalla. Renombramos a los **StaticBody2D Pared 1 y Pared 2.**

Ahora vamos a hacer que nuestra nave dispare un láser, para esto vamos a crear otra escena cuyo nodo principal será un **KinematicBody2D** llamado Laser al igual que la escena.

El árbol de nodos debe quedar de la siguiente manera:



Agregaremos un script al nodo **Laser** con el siguiente código:

```
extends KinematicBody2D
func _physics_process(delta):
    position.y -= 10
    pass
```

Ahora seleccionemos el **VisibilityNotifier2D** y luego despleguemos la pestaña **Nodo** que está junto a la del **Inspector**. Esto nos permitirá ver el panel de señales y grupos. En el panel de señales seleccionamos la función **screen_exited()**, damos click al botón **conectar** que está debajo de todo y en la ventana que se nos aparece vamos a seleccionar nuestro nodo **Laser** y nuevamente hacemos click en el botón **conectar**.

Esto habrá añadido una función a nuestro código que modificaremos y debe quedar de la siguiente manera:

```
extends KinematicBody2D
func _physics_process(delta):
    position.y -= 10
    pass
func _on_VisibilityNotifier2D_screen_exited():
    queue_free()
    pass
```

La función **_on_VisibilityNotifier2D_screen_exited()** detecta si nuestra escena, el laser, ha salido fuera de la pantalla; si esto es así, mediante el método **queue_free()** le decimos que libere la escena. De esta forma no tendremos elementos vivos dentro de la escena pero que no podamos ver por estar fuera de la pantalla, no cumplen ninguna función y que ocupan lugar en la RAM.

Ya tenemos la nave y el láser, ahora debemos hacer que la nave dispare a este láser. Para esto vamos nuevamente a la escena **Nave**.

Agregaremos a nuestro código lo siguiente:

Dos líneas que son, una variable de control y una línea que precarga la escena **Laser** dentro de la escena **Nave** y la guarda en una variable para reconocerla y podamos trabajar con ella desde donde estamos.

```
var disparo = true
var pre_laser = preload("res://Laser.tscn")
```

Un **if** que agregaremos dentro del **_physics_process(delta)** y que al detectar que se presiona la flecha arriba del teclado llama a la función **Disparo()**.

```
if Input.is_action_pressed("ui_up"):
    Disparo()
    pass
```

Por último creamos la función Disparo() que instancia al láser como hijo de la nave y lo posiciona en la posición de la nave. con la variable de control y un cronómetro controlamos la cadencia de los disparos.

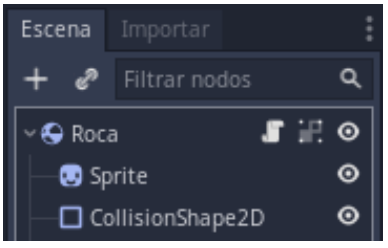
```
func Disparo():
    if disparo:
        var laser = pre_laser.instance()
        get_parent().call_deferred("add_child", laser)
        laser.position.x = position.x
        laser.position.y = position.y
        disparo = false
        yield(get_tree().create_timer(.1),"timeout")
        disparo = true
        pass
```

Es importante que la nave y las paredes que no permiten que salga de la pantalla estén en la segunda capa de colisión, de lo contrario el laser colisionará con la nave. Esto lo podemos modificar desde el **Inspector** en el panel Collision.

Con esto terminamos nuestro núcleo, ahora proseguiremos con el prototipo final.

3- Creación de un enemigo.

Nuestro enemigo será una roca y para esto utilizaremos lo aprendido en la creación de roca de la clase 8. La crearemos como una escena aparte, el árbol de nodos sería el siguiente:



Añadiremos también un script a la roca para usar más adelante.

Ahora haremos que la roca aparezca cada 1 segundo en una posición comprendida sobre el margen superior de la pantalla, es decir fuera de la pantalla, y del ancho de la pantalla. Para esto vamos a añadir en el **Main** un nodo llamado **Position2D.** A este nodo vamos a llamarlo **Spawn 1** y le agregamos un **AnimationPlayer** como hijo.

Posicionamos al **Spawn 1** en las coordenadas X=0,Y=-32. Luego creamos una animación que dure 6 segundos. En el segundo 0 las coordenadas son las iniciales, X=0,Y=-32. En el segundo 3 las coordenadas son X=720,Y=-32. Y en el segundo 6 las coordenadas son nuevamente las iniciales, X=0,Y=-32. Debemos activar los botones **Autorreproducir al Cargar** y **Loop**.

Añadiremos también un script al **Spawn 1** con el que estamos precargando la roca dentro de **Spawn 1,** instanciándola en la posición del nodo y controlando la cadencia.

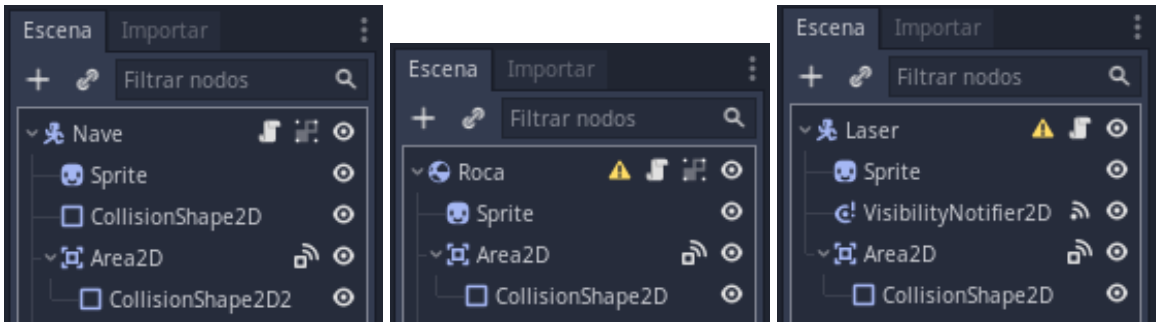
```
extends Position2D
var pre_roca = preload("res://Roca.tscn")
var cae = true
func _physics_process(delta):
    if cae:
        var roca = pre_roca.instance()
        get_parent().call_deferred("add_child", roca)
        roca.position.x = position.x
        roca.position.y = position.y
        cae = false
        yield(get_tree().create_timer(1),"timeout")
        cae = true
    pass
```

Nos falta ahora crear las condiciones del videojuego, que el láser desaparezca cuando toca una roca, que la roca desaparezca cuando toca el láser y que la nave desaparezca cuando toca una roca

4- Programando condiciones.

Lo que debemos lograr es que cada escena reconozca cuándo y con quién está colisionando. En este caso vamos a hacerlo mediante detección de áreas y no de colisiones. Para esto vamos a añadir un **Area2D** con un **CollisionShape2D** a la nave, a la roca y al láser.

Los árboles de nodos deben quedar de la siguiente forma:



Al **Area2D** de la roca y del láser le vamos a transferir el **CollisionShape2D** del nodo padre, es por eso que haremos caso omiso a la advertencia de los nodos que acusan haber quedado sin un **CollisionShape2D**. Para la detección de objetos utilizaremos el que ahora le pertenece como hijo a los nodos **Area2D**.

Para el caso de la nave si crearemos un nuevo **CollisionShape2D** para el **Area2D**, ya que el primer **CollisionShape2D** detectará las paredes y el **CollisionShape2D2** detectará a la roca.

Lo que también haremos con estas tres escenas es añadir o etiquetar sus áreas dentro de un grupo. Para esto seleccionamos el **Area2D** de cada escena y vamos a la pestaña **Nodo** que está junto a la del **Inspector**. Hacemos click en **Grupos** y en este panel creamos un nuevo grupo con el nombre de cada escena.

Ahora vamos a **Nave**, seleccionamos el **Area2D** desplegamos la pestaña **Nodo** y en el panel de **Señales** conectamos la señal **area_entered(area: Area2D)** con el script de la nave.

En el script de la nave modificamos la función que se creó para que nos quede de la siguiente manera:

```
func _on_Area2D_area_entered(area):
    if area.is_in_group("Roca"):
        queue_free()
    pass
```

Haremos lo mismo para la roca y el láser:

Script roca:

```
func _on_Area2D_area_entered(area):
    if area.is_in_group("Laser") or area.is_in_group("Nave"):
        queue_free()
    pass
```

Script láser:

```
func _on_Area2D_area_entered(area):
	if area.is_in_group("Roca"):
		queue_free()
	pass
```

Y de esta forma hemos terminado nuestro prototipo.