DOCUMENTACIÓN VIDEOJUEGO GASH & CASH

Multimedia y Aplicaciones Móviles

JULIAN MARTÍNEZ PICAZO
IES Cañada de la Encina

Índice

1. Introducción

Motivación

"Gash & Cash" surge de la pasión por los juegos de carreras en 2D con físicas realistas, inspirándose en títulos como BikeMania y Hill Climb Racing. La idea principal es ofrecer una experiencia desafiante y entretenida, donde el jugador debe superar obstáculos, gestionar su combustible y aprender del terreno para completar cada nivel con éxito.

Descripción

Gas & Cash es un videojuego de carreras en 2D desarrollado con Godot Engine y diseñado para ser multiplataforma (Windows y Android). En el juego, los jugadores se ponen en la piel de un piloto de motocross cuyo objetivo es recoger el mayor número de monedas posible mientras evita quedarse sin combustible.

A diferencia de otros juegos de carreras tradicionales, Gash & Cash no se basa en un sistema de vidas, sino en la habilidad del jugador para aprender el recorrido y mejorar con cada intento. Si el personaje cae de cabeza, deberá reiniciar el nivel y aplicar lo aprendido para superarlo con mayor precisión.

El juego cuenta con dos niveles con las mismas mecánicas, pero con diferencias en dificultad, duración y temática del terreno, lo que ofrece mayor variedad y reusabilidad.

Objetivos

El desarrollo de Gash & Cash busca lograr una serie de objetivos:

- Lograr un sistema de movilidad fluido con físicas realistas.
- Creación de niveles (plataformas) 2D complejos que desafíen la destreza del jugador.
- Una mecánica de gestión de combustible, donde el jugador debe calcular su uso para evitar quedarse sin gasolina antes de completar el nivel.
- Compatibilidad con Windows y Android para maximizar la accesibilidad y distribución del juego.

2. Tecnologías y herramientas usadas

Para desarrollar Gash & Cash se ha utilizado una combinación de herramientas de edición, programación, diseño gráfico y sonido. A continuación, se detallan las tecnologías empleadas en cada área:

Motor de Juego

El videojuego fue desarrollado con Godot Engine 4 y programado en GDScript, el lenguaje nativo de Godot basado en Python. Su sistema basado en nodos permite una organización sencilla e intuitiva, facilitando la implementación de mecánicas como las físicas del vehículo, la movilidad del personaje y la generación del terreno.

Como herramienta especial dentro de Godot, se utilizó el plugin "SmartShape2D", obtenido del repositorio de Ryan Lloyd en GitHub. Gracias a este plugin, se logró crear los terrenos con una única

plataforma, permitiendo dar curvas a las colinas y ajustar la inclinación de las pendientes de forma dinámica.

Diseño Gráfico

El arte del juego fue desarrollado combinando herramientas de inteligencia artificial y edición digital:

- DALL·E se utilizó para la generación inicial de imágenes base.
- Canva y Paint 3D sirvieron para la edición y adaptación de los sprites del juego.

Inicialmente, se probaron assets de terceros para la creación del personaje principal. Sin embargo, debido a la falta de coherencia con el estilo visual del juego, se optó por desarrollar gráficos personalizados, asegurando una estética más alineada con la identidad del proyecto.

Sonido y Música

Para la edición y optimización del sonido, se emplearon las siguientes herramientas:

- PixaBay: Se obtuvieron efectos de sonido y música de licencia libre.
- Audacity: Se utilizaron filtros y ajustes para modificar los sonidos y adaptarlos a la ambientación del juego.

Control de Versiones

El código y los archivos del proyecto fueron gestionados a través de GitHub, dentro del repositorio "Juego-Motocross". Se llevaron a cabo commits regulares, permitiendo un control eficiente de los cambios y versiones del juego, asegurando la evolución del desarrollo.

3. Diseño del videojuego

La ambientación del juego se basa en circuitos al aire libre con terrenos irregulares, colinas y pendientes que desafían el control del vehículo y la estrategia del jugador para gestionar el combustible.

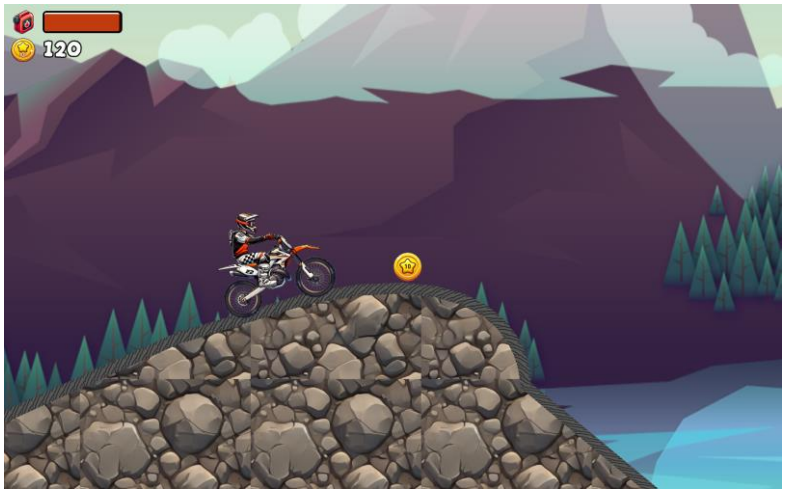
Niveles

El juego cuenta con dos niveles que comparten la misma mecánica, pero varían en dificultad, duración y temática del terreno. La mecánica básica del juego se basa en un tutorial de Lucylavend. Gracias a este tutorial he aprendido a crear el personaje, las acciones y su dinámica con el entorno.

Además también he aprendido a hacer el terreno curvo con el plugin de SmartShape2D, con otro tutorial de Lucy.

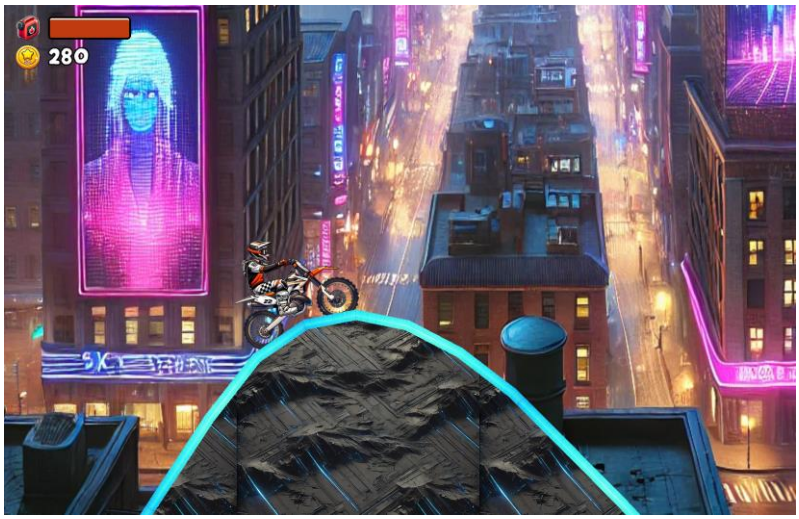
Nivel 1

Diseñado para introducir al jugador a las mecánicas básicas del juego, con pendientes suaves y una duración moderada. Se presenta al jugador en una montaña donde deberá transcurrir por colinas suaves y pendiente moderadas, a modo de tutorial.

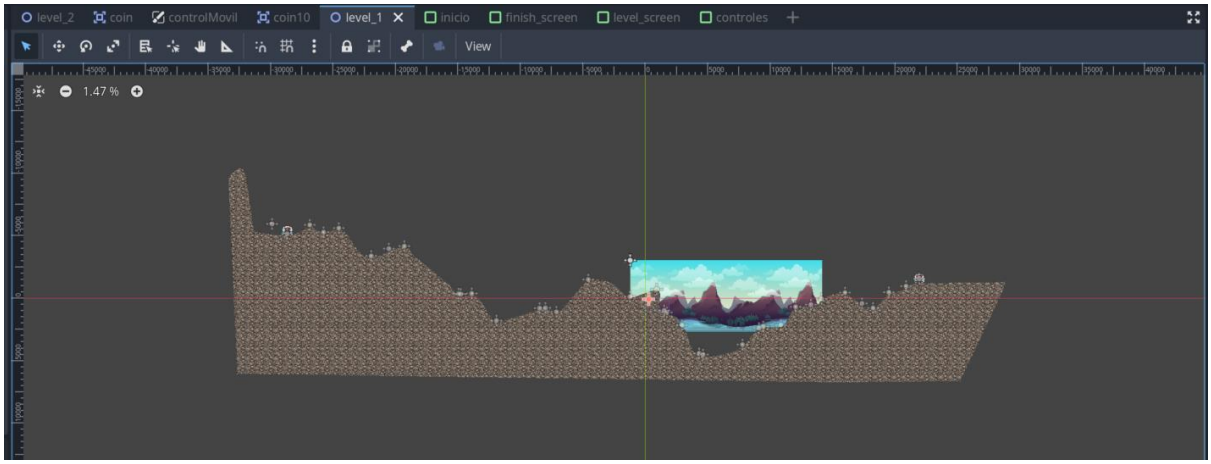


Nivel 2

Aumenta la dificultad con terrenos más complejos, obstáculos más pronunciados y una mayor exigencia en la gestión del combustible. La temática de este nivel cambia, en lugar de una montaña se nos presenta en una ciudad futurística estilo “CyberPunk”. Cambiando las texturas y parallax del terreno.



Ambos niveles fueron creados con el plugin SmartShape2D en Godot, lo que permitió diseñar circuitos dinámicos con curvas suaves y variaciones en la inclinación del terreno. Gracias a esto todo el nivel se compone de una sola plataforma optimizada.



Plataforma creada con smartShape2D

Personaje

El personaje principal es un piloto de motocross creado con inteligencia artificial, en un principio se optó por un coche todoterreno, pero debido a que la zona de colisión se resumiría al techo de vehículo, decidí cambiarlo por una moto y poner la zona de colisión sobre la cabeza del piloto. De este modo conseguimos un mayor realismo y junto con la implantación de control aéreo del vehículo ganamos mayor maniobrabilidad y espectacularidad, pidiendo hacer back-flips y front-flips.

Sprites para conformar el piloto.

La dinámica de muerte se consigue utilizando un “pin-join2D” que une los rigidBody de la cabeza y del cuerpo del piloto. Gracias a las físicas y ajustando los pesos de cabeza y cuerpo del piloto, podemos hacer que, en caso de sufrir una gran caída, este “pin-join” puede llegar a rotar si la caída es muy fuerte. Si esto se produce y el giro pasa de 90 grados, se produce la muerte del piloto y por tanto se pierde el progreso del nivel, comenzando de nuevo tras cinco segundos.

En caso de colisionar con el terreno, la dinámica es la misma, si golpeamos con la cabeza en el suelo, el cuello rotará más de 90 grados y el piloto morirá, reiniciando automáticamente el nivel tras cinco segundos.

Objetos

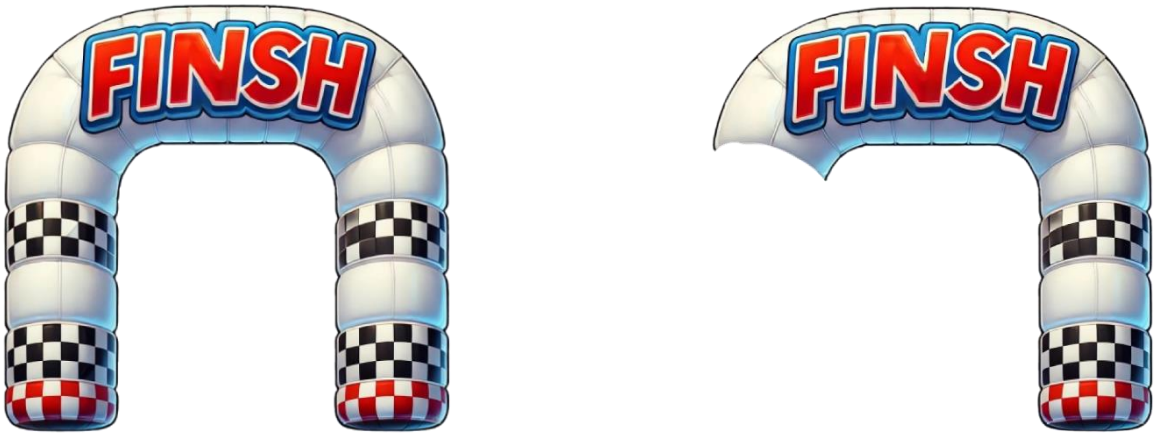
Para dar una mejor perspectiva y mayor dinamismo al juego hemos creado una serie de elementos de recolección, monedas y garrafas de fuel. Al coger colisionar con el rigidBody2D de una moneda, activamos una animación por la cual suba hacia arriba y desaparece y sumará sus puntos al marcador del personaje. Existen monedas de 50 y 10 puntos que marcarán la puntuación final del nivel.

Sprite de los Coleccionables

En el caso de la gasolina ocurre de forma similar, la animación es la misma, solo cambia el sonido. pero en este caso la gasolina recargará nuestra barra de combustible, si esta barra llega a cero, la moto se parará y moriremos, y tras 5 segundos se reiniciará en nivel.

Barra de combustible y contador de monedas

La meta del nivel se ha creado con un área2D que lanzará la escena de nivel completado, donde aparecerá el marcador final y la opción de salir o cambiar de nivel. La meta se realiza usando dos sprites, uno de fondo y otro con solo con una parte del dibujo posicionado delante, de esto modo conseguimos la apariencia de que nuestro nodo personaje pasa por debajo de la meta.



Sprites utilizados en la línea de meta

Menús

Para llevar a cabo la gestión del videojuego se han creado una serie de menús similares entre sí. Utilizamos un sprite de fondo que se ajusta al tamaño de la pantalla y sobre el insertamos el título del videojuego con la tipografía de Duramadrop obtenida desde Google Fonts, y diversos sprites usados en el juego.

Sobre este fondo se han insertado los botones con un fondo naranja que cambian de color y emiten un sonido al posicionar el ratón sobre ellos.

Menú principal de Gas & Cash

Para mantener la integridad de apariencia en los menús, el resto se han creado con la plantilla del menú principal. Creando un menú de inicio, otro para elegir nivel y otro de completado, todos conectados entre ellos.

Escena para elegir nivel y escena final

Para hacer el juego más intuitivo y dar crédito a los desarrolladores que han ayudado, hemos creado dos escenas con información útil. Una para explicar los controles del juego en la escena de nivel y otra para los créditos, que permite acceder a los sitios web de los desarrolladores y artistas si clicamos en su nombre, por medio de un enlace.

Escena de controles y créditos

Además las escenas de menú principal y la escena final cuentan con una AudioStream con las canciones del usuario Audiogreen en pixabay, utilizando las tituladas: Punk Rock, Phonk y Sport Fitness.

4. Funcionalidad

La funcionalidad de Gas & Cash es muy simple, el objetivo es llegar al final del nivel sin que el cuello del piloto gire más de 90 grados, o que el personaje se quede sin combustible.

Movilidad y física del personaje

El elemento principal del videojuego recae sobre el personaje. El personaje está compuesto de 4 rigidBody2D, la moto (moto y cuerpo piloto), las ruedas (unidas por un pinJoint2D con la moto), el cuello y cabeza (unidos a la moto con otro pinJoint2D).

Por medio de un Script hacemos que los rigidbodys de las ruedas giren hasta un tope de velocidad al pulsar sobre la tecla “arriba” del teclado. Y exactamente lo mismo, pero girando al contrario si pulsamos la tecla “abajo”.

```
    # Movimiento hacia adelante (acelerar)
    if Input.is_action_pressed("ui_up"):
        driving += 1
        use_fuel(delta)
        for wheel in wheels:
            if wheel.angular_velocity < max_speed:
                wheel.apply_torque_impulse(speed * delta * 110)

    # Movimiento hacia atrás (marcha atrás)
    if Input.is_action_pressed("ui_down"):
        driving += 1
        use_fuel(delta)
        for wheel in wheels:
            if wheel.angular_velocity > -max_speed:
                wheel.apply_torque_impulse(-speed * delta * 110)
```

Para estabilizar el personaje se realiza una acción similar al pulsar la tecla “derecha” e “izquierda”, utilizando el factor delta y el método apply_torque_impulse(). De esta forma el manejo de la moto se hace más efectivo y divertido.

```
    # Control aereo (Rotación en el aire)
    if Input.is_action_pressed("ui_right"):
        apply_torque_impulse(7500 * delta * 110)  # Rotación en sentido horario (derecha)

    if Input.is_action_pressed("ui_left"):
        apply_torque_impulse(-7500 * delta * 110)  # Rotación en sentido antihorario (izquierda)
```

Para controlar la muerte del personaje, analizamos la rotación del rigidBody de la cabeza del personaje. Lo que activará el sonido de caída y cambiará el marcador de muerdo como true, lo que activará un contador de cuenta atrás de 5 segundo y reiniciará el nivel.

```
    if $Casco.rotation_degrees > 90 || $Casco.rotation_degrees < -90 && !dead:
        $Caida.play()
        dead = true
```

```
else:
    if $GameOverTimer.is_stopped():
        $GameOverTimer.start()
```

Para que la moto suene al acelerar, tan solo cambiaremos el multiplicador del pitch (velocidad de reproducción) del audioStreamPlayer del motor, lo que asemeja una aceleración:

```
    if driving == 1:
        $EngineSFX.pitch_scale = lerp($EngineSFX.pitch_scale, 2.0, 2.0 * delta)
        $EngineSFX.stream.loop = true
    else:
        $EngineSFX.pitch_scale = lerp($EngineSFX.pitch_scale, 1.0, 2.0 * delta)
        $EngineSFX.stream.loop = true
```

Para el tema del consumo de combustible creamos dos funciones, refuel(), que se activará al colisionar con la escena de fuel, y el método use_fuel() que irá gastando el contador de fuel cuando se pulsa en acelerar.

```
func refuel():
    fuel = 100
    get_parent().update_fuel_UI(fuel)

func use_fuel(delta):
    fuel -= 15 * delta
    fuel = clamp(fuel, 0, 1000)
    get_parent().update_fuel_UI(fuel)
```

Si el fuel llega a cero, o la cabeza rota más de 90 grados, se activará el contador de cuenta atrás de 5 segundos, y si llega al final se activa el método para reiniciar la escena del nivel.

```
func _on_game_over_timer_timeout() -> void:
    get_tree().reload_current_scene()
```

Colisiones y animaciones

Para implementar la opción de poder recoger objetos, lo hemos hecho por medio de animaciones, y grupos de colisiones. Cuando el personaje entra dentro de la zona de colisión de la moneda, se activa la animación de “pickup” y se activa el audioStreamPlayer de la moneda, creando la sensación de recogida.

Además, activará el método add_coins() que sumará el valor de la moneda a contador global de monedas.

```
1   extends Area2D
2
3   @export var value = 50
4
5
6   func _on_body_entered(body: Node2D) -> void:
7       if body.is_in_group("player1"):
8           get_tree().get_current_scene().add_coins(value)
9           $AnimationPlayer.play("pickup")
10          $CollisionShape2D.set_deferred("disabled", true)
11
12
```

Para la recogida de gasolina, la dinámica es similar, pero en este caso llamaremos al método refuel(), que recargará nuestra barra de fuel. La animación utilizada es la misma que para las monedas, con un sonido diferente.

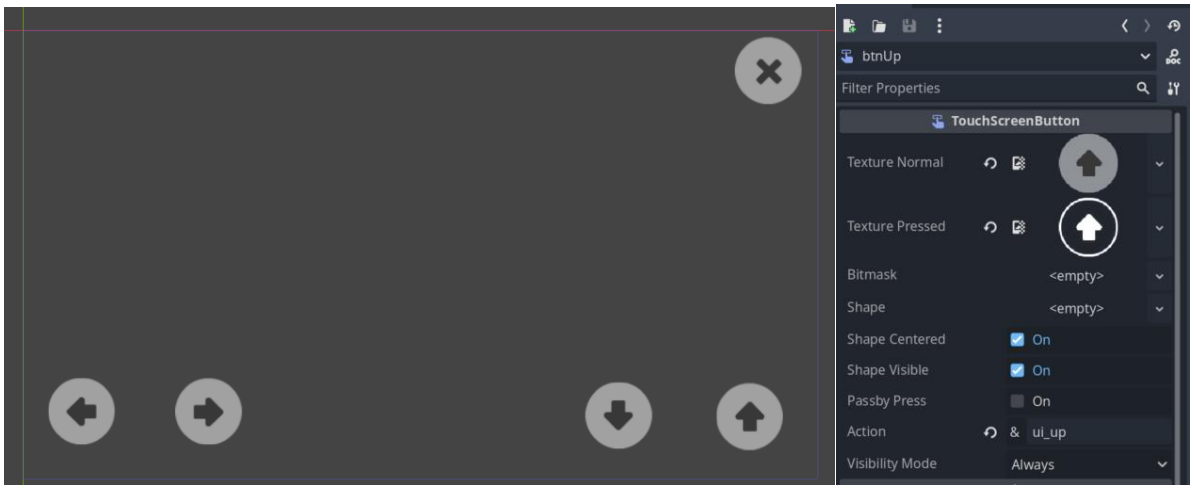
```gdscript
extends Area2D


func _on_body_entered(body: Node2D) -> void:

	if body.is_in_group("player1"):
		get_tree().get_current_scene().get_node("Player1").refuel()
		$AnimationPlayer.play("pickup")
		$CollisionShape2D.set_deferred("disabled", true)
```

Para las colisiones con la meta, que marcan el final del nivel, hemos creado un área2D con una zona de colisión qué al detectar la entrada del personaje, emite un sonido de victoria y lanza una escena final con la puntuación obtenida.


```gdscript
extends Area2D

func _on_body_entered(body: Node2D) -> void:
	$sonidoWin.play()
	if body.is_in_group("player1"):
		print("¡Meta alcanzada!")
		Global.total_coins = Global.level_coins #guardo monedas en la v
		call_deferred("change_scene")

func change_scene():
	await get_tree().create_timer(2).timeout #Termina al curzar meta
	get_tree().change_scene_to_file("res://scenes/finish_screen.tscn")

```

Controles SmartPhone

Para la versión de Android hemos implementado un canvasLayer con TouchScreenButton que recojan la acción y se han asociado cada botón a una acción del teclado.



Gestión del Nivel

Por último, la funcionalidad del nivel es similar en ambos. Se encargan de gestionar la interfaz del juego, es decir la barra de gasolina, que se está conectada al personaje, el contador de monedas y el final de la cuenta atrás para el reinicio del nivel.

Cada una de estas acciones se lleva a cabo con una función, al igual que le contador global de monedas que sirve para pasar la cantidad de monedas entre escenas.

Funcionalidades de Menús

Para mantener la integridad del videojuego, todos los menús son similares, el mismo fondo y la misma paleta de colores.

La funcionalidad del menú se reduce a cambiar de escena al pulsar un botón, por ello colocamos dos señales, una para captar cuando el ratón se posa sobre el botón, para que active un sonido de clic y cambie el color, y otra señal para que cambie la escena si es botón es pulsado.


```gdscript
extends PanelContainer

var level_scene = "res://scenes/level_screen.tscn"
var creditos_screen = "res://scenes/opciones_screen.tscn"



func _on_btn_start_pressed() -> void:
    get_tree().change_scene_to_file(level_scene)

func _on_btn_opcion_pressed() -> void:
    get_tree().change_scene_to_file(creditos_screen)

func _on_btn_salir_pressed():
    get_tree().quit()


func _on_btn_start_mouse_entered() -> void:
    $clikBoton.play()


func _on_btn_opcion_mouse_entered() -> void:
    $clikBoton.play()


func _on_btn_salir_mouse_entered() -> void:
    $clikBoton.play()
```

5. Pruebas y optimización

Durante el desarrollo del juego hemos tenidos ciertos problemas que hemos solucionado de la siguiente forma:

1. La moto se hacía muy difícil de maniobrar si solo utilizábamos acelerador y freno y el juego no era entretenido.
   Solución: Por ello implementamos el control de la misma mediante el torque con los botones de izquierda y derecha.

2. El personaje no moría al caer sobre su cabeza, y si lo hacía al caer la moto desde alto.
   Solución: Hacer al piloto más pesado, y reducir el peso de la cabeza. De esta forma al caer el peso de la moto sobre la cabeza, se produce la muerte del personaje, y al pesar la cabeza menos, no muere cuando cae desde una gran altura.

3. En Smartphone no funcionaban los botones del touchscreen del menú.
   Solución: al no ser el menú un elemento que necesite de doble pulsación, eliminamos los botonesTouch por botones normales y activamos la opción de detectar clics en pantallas táctiles.

6. Publicación y Mejoras

Publicación

Para hacer accesible Gash & Cash al público, hemos decidido publicarlo en Itch.io, una plataforma popular para la distribución de videojuegos indie. Esta plataforma nos permite permite descargar y probar el juego de manera sencilla, además de recibir retroalimentación de la comunidad. Desde Itch.io está disponible para Windows y Android.

Enlace de Gas & Cash

Además, al utilizar nosotros mismos recursos ofrecidos en Itch.io por la comunidad, nos sentimos con el deber de hacer lo mismo, y ofrecer nuestro trabajo a coste cero, para que el videojuego y sus recursos pueden ser utilizados por otros desarrolladores y fomentar está dinámica de aprendizaje y colaboración que nos beneficia a todos.

Todos los recursos están disponibles desde el repositorio de github.

Futuras Mejoras

En futuras versiones de Gash & Cash, se planean mejoras y expansiones como:

1. Creación de nuevos niveles con temáticas de desierto, o incluso la luna, cambiando pesos y fuerzas para alterar las físicas.
2. Ofrecer puntos por maniobras temerarias como backflips o frontflips.
3. Introducción de nuevos personajes como coches o diferentes motos.
4. Nuevos modos como contrarreloj (hacer el nivel en un tiempo limitado) o Supervivencia (nivel sin fin aumentando dificultad).

7. Fuentes Consultadas

- En primer lugar consultamos tutoriales como el de Findemor para aprender a usar Godot y ver sus características: Cómo usar Godot y Aprender desde CERO a hacer juegos.

- Para crear la mecánica principal del video juego utilizamos un tutorial de LucyLavend, una desarrolladora indie que no ensenó a crear un juego estilo hill climb: How to make Hill Climb Racing in Godot. Gracias a esta creadora aprendimos a utilizar el plugin de smartShape y crear plataformas con relieves curvos: How To Make 2D Curved Terrain In Godot.

- Todos los audios del juego son libres de uso y se han conseguido a través de Pixabay. Debemos dar especial relevancia al creador AudioGreen, pues suyas son las tres canciones principales del videojuego: Perfil de pixabay de AudioGreen.

- Para hacer muchos de los elementos secundarios del juego he revisado tutoriales de otros desarrolladores, para hacer un menú de inicio, para crear los botones de Android, para poner un fondo móvil y también para aprender a exportar el proyecto en Android.

- Todas las imágenes del video juego han sido creadas con IA, DALL-E y retocadas con Canva y Paint3D.