Gravitar - Docuemento Tecnico

Las funciones que utilizan en el juego son las siguientes, cada una se divide por su respectiva clase y objeto en el mundo de gravitar.

MECANICAS A DETALLE PARA EL DESARROLLO

Movimiento de la nave

Funcionamiento:

El funcionamiento primordial del movimiento es el ser atraído por la gravedad, en base a esto la nave tendrá más “momentum” por lo que puede ganar o perder mas velocidad conforme la gravedad de la zona en la que este.

Características:

- El movimiento del personaje consta de varios puntos a considerar:
- Gravedad: Esta es de 9.8
- Velocidad: Esta funciona en base a la gravedad.
- Rotación: Se gira en 360 grados a los dos lados.
- Aceleración: Con la fuerza generada con la gravedad esta puede aumentar o disminuir.
- Escudo: Este se activa con el botón de reversa y activa laser que destruyen bunkers.
- Gasolina: Se empieza con 1000 pts.

Mecánicas ocultas:

- Cuando el personaje esta avanzando y deja de presionar el botón de avanzar, la nave sigue con la inercia de la velocidad.
- La velocidad afecta en el movimiento, esta incrementa conforme la gravedad.
- El escudo solo funciona con el botón de reversa.

Main Camera

La cámara cuenta con dos scripts indispensables para su buen funcionamiento.

Camera Smooth

Este script es el que permite a la cámara seguir al jugador en todo momento en el juego, aparte cuenta con la habilidad de ajustarse a la velocidad a la que va la nave del personaje principal para hacer un efecto de transición.

```
// Variables
    public Transform target;            // Se asigna una posicion especifica del
jugador
    public float smoothSpeed = 0.125f;   // Se determina el tiempo de transicion
    public Vector3 offset;              // Se definen las variables en X,Y y Z
    // Update is called once per frame
    void Update () {
        Vector3 desiredPosition = target.position + offset; // Se asigna una posicion
deseada del jugador
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition,
smoothSpeed); // Se crea la transicion del seguimiento del jugador
        transform.position = smoothedPosition; // Se incluye a la posicion de la camara
    }
```

HUD

El HUD es la interfaz que tendrá el usuario para ver como van sus estadísticas y progreso en el juego, aquí se mandan a llamar datos sobre el jugador y el entorno de juego.

```
    // Variables
    public Sprite[] HeartSprites; // Se indican cuantos sprites tendra la animacion de
vidas
    public Image HeartUI;         // Se llama a la imagen de los sprite
    private PlayerMovement player; // Se llama a la clase del jugador
    public Text fuelText;         // Se crea una variable para indicar la gasolina
restante
    public Text scoreText;        // Se crea una variable para indicar la puntuacion
actual
    // Metodos
    void Start()
    {
        // Se llama al jugador
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerMovement>();
    }

    void Update()
    {
        // Hearths Text Info
        HeartUI.sprite = HeartSprites[player.lives];
        // Fuel Text Info
        fuelText.color = new Color32(11,255,40,255);
        fuelText.text = player.fuel + "";
        // Score Text Info
        scoreText.color = new Color32(11,255,40,255);
        scoreText.text = player.score + "";
    }
```

Player Movement

En este script se encuentran todas las variables y métodos para que la nave del jugador pueda moverse adecuadamente en el entorno de juego.

```
    // Use this for initialization
     BoxCollider2D col;
     private Rigidbody2D Target;
     #region MonoBehavor API
     // Region specifies a zone to work, code to be using
     // Variables
     public float maxVelocity = 3;
     public float rotationSpeed = 1;
     public int lives = 3;
     public int fuel = 5000;
     public int score;
     public float yAxis;  // Y Axis
     float xAxis;
     public AudioClip move;
     public AudioClip Die;

     // Metodos
     private void Start() {
         Target = GetComponent<Rigidbody2D>();
     }

     private void Update()
     {
         /* Defining the ways of movement */
         xAxis = Input.GetAxis("Horizontal"); // X Axis
         float velocidadmejor = xAxis;
         if (Input.GetKey(KeyCode.DownArrow))
         {
             yAxis = 0;
         }
         else
         {
             yAxis = Input.GetAxis("Vertical");   // Y Axis
             fuel -= 1;
         }
         // Condicion de muerte
         if (lives <= 0)
         {
             Destroy(gameObject);
             Application.Quit();
         }
         ThrustForward(yAxis);   // This is to get the force of the Y axis
         Rotate(transform, xAxis * -rotationSpeed);   // Getting the rotation of the
target
     }

     #endregion

     #region Movment API
     ////////////////////////////////////////////
```

```
    // ClampVelocity - Speed
    private void ClampVelocity()
    {
        // Making the target tu adjust to a new velocity
        float X = Mathf.Clamp(Target.velocity.x, -maxVelocity, maxVelocity);    // Min
and Max velocity
        float Y = Mathf.Clamp(Target.velocity.y, -maxVelocity, maxVelocity);

        Target.velocity = new Vector2(X*2, Y); // Set our new velocity
    }
    // Funcion para obtener una fuerza de resistencia
    private void ThrustForward(float amount)
    {
        Vector2 force = transform.up * amount; // This get a vector

        Target.AddForce(force);
    }
    // Funcion para poder rotar en su propio eje
    private void Rotate(Transform t, float amount)
    {
        t.Rotate(0, 0, amount);
    }
    #endregion
    // Funcion para reaparecer en el centro si el jugador sale de la pantalla
    void OnBecameInvisible()
    {
        transform.position = new Vector3(0, 8, 0);
    }
    // Funcion cuando hay una colision
    void OnTriggerEnter2D(Collider2D hitInfo)
    {
        // When is trigger
        bullet enemy = hitInfo.GetComponent<bullet>();
        shield escudo = hitInfo.GetComponent<shield>();
       // Si el personaje choca entonces
        if(gameObject)
        {
            AudioSource.PlayClipAtPoint(Die, Camera.main.transform.position); // Audio
de muerte
            transform.position = new Vector3(0, 8, 0); // Regresa al centro de la
pantalla
            lives -= 1; // Pierde una vida
        }
    }
```

Enemy

El enemigo no funciona independientemente pero enlaza las diferentes clases que se mostraran ac continuación.

```
    // Variables
    public int healt = 1;
```

```
    public int Turretlife = 1;
    public GameObject deadthEffect;
    public AudioClip Morir;
    private PlayerMovement player;
    // Use this for initialization

    void Update()
    {
        // Buscar al jugador
        if (Turretlife <= 0)
        {
            Destroy(gameObject);
        }
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<PlayerMovement>();
    }
    public void TakeDamege(int damage)
    {
        healt -= damage;

        if (healt <= 0)
        {
            Die();
        }
    }
    // Condicion de muerte
    void Die()
    {
        AudioSource.PlayClipAtPoint(Morir, Camera.main.transform.position);
        Instantiate(deadthEffect, transform.position, Quaternion.identity);
    }
    // Funcion Cuando lo atacan
    void OnTriggerEnter2D(Collider2D hitInfo)
    {
        Turretlife -= 1;
        player.score += 1000;
        Destroy(gameObject);
    }
```

Turret

La torreta es la que crea todo el funcionamiento de del enemigo indicando la distancia de ataque junto con el margen de error y recarga de las torretas.

```
    public float DistansFromPlayer, CoolDown;   // Variable puublica para mostrar la
distancia y la recarga
    public GameObject enemy;        // Se llama a la clase general de enemigo
    public GameObject Bullet;       // Se llama a la bala de la torreta
    public int protectionRadius,bulletSpeed;    // Se crea un radio de proyeccion y
velocidad de la bala
    private const int SPAWN_DISTANCE = 5;   // Se hace una condicion de creacion de
balas
```

```
    // Use this for initialization
    void Start()
    {
        protectionRadius = 35;
        bulletSpeed = 10;
        CoolDown = 5;
    }
    // Update is called once per frame
    void Update()
    {
        // Se llama a la clase general del enemigo
        enemy = GameObject.FindGameObjectWithTag("Enemy");
        // Si el jugador se encuentra cerca de la torreta entonces
        if (enemy != null)
        {
            // Calcula la distancia del jugador con la torreta
            DistansFromPlayer =
Vector3.Distance(GameObject.FindGameObjectWithTag("Enemy").transform.position,
GameObject.FindGameObjectWithTag("Player").transform.position);
            //print (DistansFromPlayer);
            if (DistansFromPlayer <= protectionRadius)
            {
                // Se ataca al jugador
                attackEnemy();
            }
        }
    }
    void attackEnemy()
    {
        // Se calcula cuando se este cerca del rango de vision de la torreta
        transform.LookAt(enemy.transform);
        CoolDown -= Time.deltaTime;
        if (CoolDown <= 0)
        {
            // Se recarga y se vuelve a activar
            Debug.DrawLine(transform.position, enemy.transform.position, Color.red);
            Instantiate(Bullet, transform.position + SPAWN_DISTANCE *
transform.forward, transform.rotation);

            print("attack Enemy");
            CoolDown = 5;
        }
    }
```

Bullet Controller

Este script es el encargado de crear las fuerzas de la bala al igual que su inercia y tiro, ya que sin estas la bala se dispararía sin un sentido lógico, de este modo se creo una función que crea una inicialización aleatoria de los parámetros para tener una funcionamiento mas errático.

```
    // Variables
```

```
    public float bulletSpeedhigh;
    public float bulletSpeedLow;
    public float bulletAngle;
    public float bulletTorqueAngle;
    Rigidbody2D bulletRB;
    // Use this for initialization

    void Start () {
        bulletRB = GetComponent<Rigidbody2D>();
        bulletRB.AddForce(new Vector2(Random.Range(-bulletAngle, bulletAngle),
Random.Range(bulletSpeedLow, bulletSpeedhigh)), ForceMode2D.Impulse); // Adding force
plus some random of the angles
        bulletRB.AddTorque((Random.Range(-bulletTorqueAngle, bulletTorqueAngle)));
        Destroy(gameObject, .6f);
    }
```