Movimentação e Input de teclado

Link Notion: https://cherry-client-b8f.notion.site/Movimenta-o-e-Input-de-teclado-262911d84e0d8062aa71e2cfbe6eb731

Para mover objetos na Unity, existem diferentes formas, cada uma indicada para uma necessidade específica.

- **Translate** → “anda para frente sempre, sem colisão”
- **MoveTowards** → “vai até o alvo, mas pode atravessar objetos”
- **Rigidbody.MovePosition** → “vai até o alvo/direção respeitando a física e colisores”
- **AddForce** → “empurra ou aplica impulso respeitando massa e física”
- **Lerp** → “se aproxima suavemente do alvo, mas depende de um fator de interpolação”

1. Alterando a posição diretamente

Transform.position

Usado quando não precisamos de Física realista, apenas mudar a posição.

```
transform.position = new Vector3(0, 0, 5);
```

Isso “teleporta” o objeto para a posição indicada. Os usos comuns são: reposicionar objetos (resetar player no spawn, colocar inimigo em ponto fixo, teletransporte).

Transform.translate

É um método que move o objeto relativamente à posição atual. Ele soma o deslocamento ao transform.position, fazendo um movimento contínuo.

```
// Move o objeto 1 unidade para frente
transform.Translate(Vector3.forward);
```

Exemplo 1 – Jogo de Plataforma 2D simples (ex: Super Mario clássico)

- O personagem se move para esquerda/direita sem interações físicas complexas.
- Translate garante movimento suave e previsível, sem precisar lidar com forças físicas.

```
using UnityEngine;

public class Player2D : MonoBehaviour
{
    public float speed = 5f;

    void Update()
    {
        float move = Input.GetAxis("Horizontal");
        transform.Translate(Vector3.right * move * speed * Time.deltaTime);
    }
}
```

Exemplo 2 – Movimento de Câmera em RPG (ex: Pokémon antigo)

- A câmera segue o personagem sem colisão, só mudando posição.
- Transform.position é suficiente e mais leve que Rigidbody.

```
using UnityEngine;
```

```
public class CameraFollow : MonoBehaviour
{
    public Transform player;
    public float smoothSpeed = 5f;

    void LateUpdate()
    {
        Vector3 newPos = new Vector3(player.position.x, player.position.y, transform.position.z);
        transform.position = Vector3.Lerp(transform.position, newPos, smoothSpeed * Time.deltaTime);
    }
}
```

Por que usar:

✅ Simples de implementar

✅ Ótimo para movimentações “controladas” sem física

❌ Não detecta colisões automaticamente

2. Rigidbody.MovePosition/Rigidbody.linearVelocity

É um método da Unity usado para **mover um objeto com física** de forma suave e previsível. Ele combina o controle de movimento (como **transform.position**) com as regras do sistema de física (como colisores e detecção de colisão).

Exemplo 1 – Top-Down Shooter (ex: Enter the Gungeon)

- O player se move em todas as direções, mas precisa colidir com paredes.
- MovePosition garante que o Rigidbody respeite a física e colisores.

```
using UnityEngine;

public class PlayerTopDown : MonoBehaviour
{
    public float speed = 5f;
    private Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");
        Vector3 movement = new Vector3(x, 0, z);

        rb.MovePosition(rb.position + movement * speed * Time.fixedDeltaTime);
    }
}
```

Exemplo 2 – NPC andando no cenário 3D (ex: Skyrim)

- NPCs precisam se mover mas também não atravessar paredes.
- MovePosition faz o NPC colidir com obstáculos sem ser empurrado por forças.

```
using UnityEngine;
```

```
public class NPCWalk : MonoBehaviour
{
    public float speed = 2f;
    private Rigidbody rb;
    private Vector3 direction = Vector3.forward;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        rb.MovePosition(rb.position + direction * speed * Time.fixedDeltaTime);
    }
}
```

Por que usar:

✅ Movimento direto com Física ligada

✅ Evita bugs de atravessar objetos

❌ Mais pesado que Transform

3. MoveTowards (ir de A até B)

Usado para movimentos automáticos entre pontos (sem input).

```
transform.position = Vector3.MoveTowards(
    transform.position,      // posição atual
    target.position,         // destino
    speed * Time.deltaTime   // velocidade
);
```

Exemplo 1 – Patrulha de Inimigo (ex: Castlevania)

- Um morcego vai de ponto A → B e volta.
- MoveTowards é perfeito porque define trajetória clara e suave.

```
using UnityEngine;

public class EnemyPatrol : MonoBehaviour
{
    public Transform pointA, pointB;
    public float speed = 2f;
    private Transform target;

    void Start()
    {
        target = pointB;
    }

    void Update()
    {
        transform.position = Vector3.MoveTowards(transform.position, target.position, speed * Time.deltaTime);

        if (Vector3.Distance(transform.position, target.position) < 0.1f)
        {
            target = target == pointA ? pointB : pointA;
        }
```

```
    }
}
```

Exemplo 2 – Plataformas móveis (ex:Celeste)

- Plataformas que deslizam para frente e para trás.
- Lerp cria o efeito suave de transição.

```
using UnityEngine;

public class MovingPlatform : MonoBehaviour
{
    public Transform pointA, pointB;
    public float speed = 2f;
    private float t;

    void Update()
    {
        t += Time.deltaTime * speed;
        transform.position = Vector3.Lerp(pointA.position, pointB.position, Mathf.PingPong(t, 1));
    }
}
```

Por que usar:

✅ Excelente para movimentos previsíveis

✅ Ideal para inimigos e elementos de cenário

❌ Não reage fisicamente a empurrões

4. Rigidbody.AddForce

Usado quando queremos física realista (empurrar, pular, explosões).

Exemplo 1 – Jogo de Esporte (ex: Rocket League)

- O carro bate na bola e ela é empurrada.
- AddForce cria movimento realista da bola, respeitando massa e atrito.

```
using UnityEngine;

public class BallKick : MonoBehaviour
{
    public float kickForce = 10f;
    private Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            rb.AddForce(Vector3.forward * kickForce, ForceMode.Impulse);
        }
    }
}
```

Exemplo 2 – Plataforma 2D com Pulo (ex: Hollow Knight)

- O personagem pula aplicando força para cima.
- AddForce gera impulso natural, diferente de só mudar a posição.

```
using UnityEngine;

public class PlayerJump : MonoBehaviour
{
    public float jumpForce = 7f;
    private Rigidbody2D rb;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
        }
    }
}
```

Por que usar:

✅ Cria sensação física realista

✅ Ótimo para objetos que devem ser “empurrados”

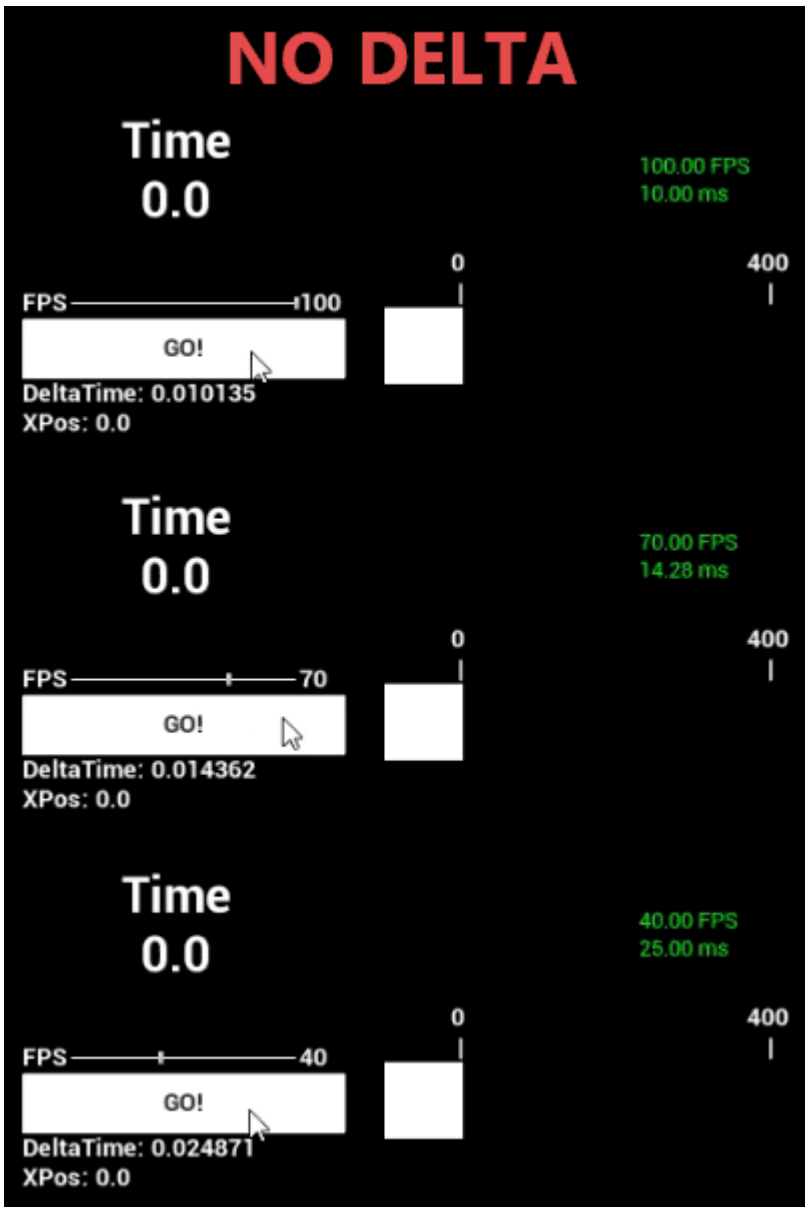
❌ Difícil de controlar com precisão milimétrica

Conceitos adicionais

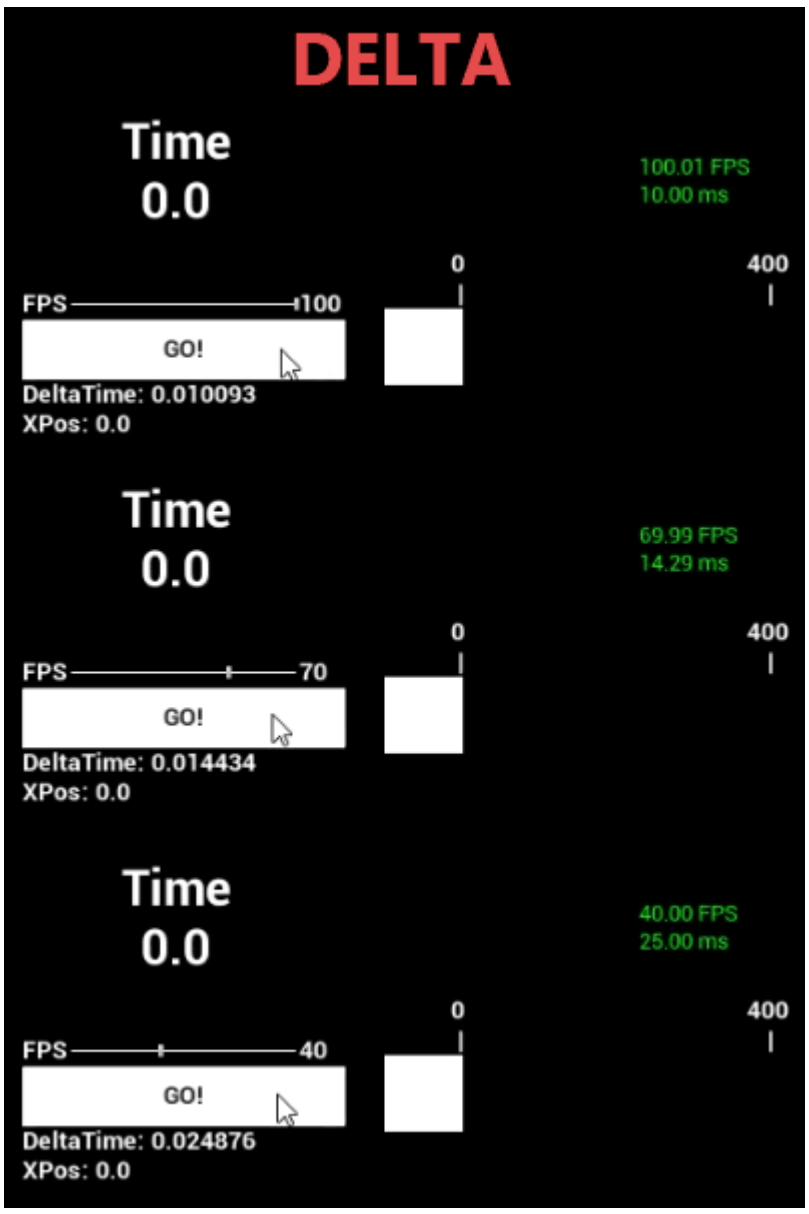
Time.deltaTime

Imagina um jogo rodando a 30 FPS e outro a 60 FPS. Sem deltaTime, no de 60 FPS o objeto se moveria o dobro da distância.

Sem Deltatime



Com Deltatime



Exemplo correto:

```
transform.Translate(Vector3.forward * speed * Time.deltaTime);
```

Assim, o movimento será o mesmo, independentemente do FPS.

Entrada do Usuário (Input)

Para controlar o jogador, precisamos capturar teclas ou botões.

Métodos principais

- **Input.GetAxis("Horizontal")** → retorna -1 a 1 (setas ou A/D).
- **Input.GetKey(KeyCode.Space)** → verdadeiro enquanto a tecla estiver pressionada.
- **Input.GetKeyDown(KeyCode.Space)** → verdadeiro apenas no frame do clique.

Alguns KeyCodes úteis para jogos:

- KeyCode.Space → espaço (pular/dash)
- KeyCode.Escape → esc (pausar/abrir menu)
- KeyCode.Mouse0 → clique esquerdo
- KeyCode.Mouse1 → clique direito
- KeyCode.LeftShift / KeyCode.RightShift → shift esquerdo/direito
- KeyCode.Alpha0 ... KeyCode.Alpha9 → teclas numéricas 0–9 do teclado principal
- KeyCode.UpArrow, KeyCode.DownArrow, KeyCode.LeftArrow, KeyCode.RightArrow → setas

Exemplo de script de movimento:

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    public float speed = 5f;

    void Update()
    {
        float horizontal = Input.GetAxis("Horizontal"); // A/D ou ←/→
        float vertical = Input.GetAxis("Vertical");     // W/S ou ↑/↓

        Vector3 direction = new Vector3(horizontal, 0, vertical);

        transform.Translate(direction * speed * Time.deltaTime);
    }
}
```

Tabela comparativa

Método	Movimento	Parada automática	Física / Colisão	Uso típico / Exemplo
Transform.position	Define posição absoluta no mundo	✅ sim (vai direto ao ponto)	❌ ignora colisão	Teletransporte, spawn, reposicionar player/inimigo
Transform.Translate	Contínuo em uma direção (soma à posição atual)	❌ manual	❌ ignora colisão	Player simples, câmera, scroll infinito
Vector3.MoveTowards	Vai até um ponto alvo específico	✅ sim	❌ ignora colisão	Patrulha de inimigo, projétil indo a um alvo
Rigidbody.MovePosition	Move em direção a um ponto ou direção	✅ se combinado	✅ respeita colisores	Player top-down, NPCs, objetos que não devem atravessar paredes
Rigidbody.AddForce	Aplica força ou impulso, movimento baseado em física	❌ depende da física	✅ respeita massa, gravidade e colisores	Pulo de personagens, bolas, explosões, empurrar objetos

Método	Movimento	Parada automática	Física / Colisão	Uso típico / Exemplo
Vector3.Lerp	Interpola suavemente entre dois pontos	✅ se t = 1 ou acumulado	❌ ignora colisão	Suavização de câmera, transições de UI, plataformas móveis