1. Movimento do Personagem

Código:

Esse código trata do movimento do personagem Léo no eixo horizontal (X) e vertical (Y), além de aplicar física básica (gravidade e pulo).

Modelo Matemático:

Movimento Horizontal (Eixo X):

X(t) = v \cdot t

É a posição do personagem no eixo X.

É a velocidade (4 unidades por segundo, conforme especificado no código).

É o tempo decorrido.

Movimento Vertical (Eixo Y – Gravidade e Pulo):

Y(t) = y\_0 + v\_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2

É a posição vertical do personagem.

É a posição inicial (geralmente zero quando o personagem começa no chão).

É a velocidade inicial do pulo (10 unidades por segundo).

É a aceleração devido à gravidade (definida como -20).

É o tempo em segundos.

Fórmula de Força do Pulo:

\text{Força Y} = \text{Força do Pulo} - \text{Gravidade} \cdot t

1. Transição de Fase

Código:

Esse código trata de carregar a próxima fase quando o personagem entra em um gatilho.

Modelo Matemático:

Não há uma função matemática diretamente envolvida nesse código, pois a transição de fase é apenas a troca de cenas. A lógica é simplesmente:

Quando o personagem colide com o trigger (evento), a próxima fase é carregada.

1. Botão de Continuar

Código:

Esse código permite que o jogador continue para a próxima cena ao pressionar um botão.

Modelo Matemático:

Essa ação pode ser descrita como:

Carregamento de Cena:

Quando o botão é pressionado, a cena atual é identificada pelo índice da cena e a próxima cena é carregada.

\text{Cena atual} = \text{SceneManager.GetActiveScene().buildIndex}

\text{Próxima cena} = \text{Cena atual} + 1 ] Essa lógica não envolve uma operação matemática complexa, mas é um simples incremento do índice da cena.

1. Sistema de Vida do Personagem

Código:

Esse código controla a vida do personagem, permitindo que ele sofra dano e seja morto.

Modelo Matemático:

Vida Atual:

\text{Vida Atual} = \text{Vida Máxima} - \text{Dano Recebido}

A vida é decrementada de acordo com o dano recebido.

Função de Atualização da Barra de Vida: A barra de vida pode ser representada por uma função de valor relativo, com base na vida atual:

\text{Barra de Vida} = \frac{\text{Vida Atual}}{\text{Vida Máxima}}

Limitação de Vida (Clamping):

\text{Vida Atual} = \text{Clamp}(\text{Vida Atual}, 0, \text{Vida Máxima})

Morte do Personagem: A morte do personagem ocorre quando:

\text{Vida Atual} \leq 0

1. Câmera Seguindo o Personagem

Código:

Este código faz a câmera seguir o personagem de maneira suave.

Modelo Matemático:

Posição Desejada da Câmera: A posição da câmera é calculada somando um vetor de “offset” (distância) à posição do personagem:

P\_{\text{camera}}(t) = P\_{\text{personagem}}(t) + \text{offset}

É a posição da câmera no tempo .

É a posição do personagem.

A “offset” é uma constante que define a distância relativa entre o personagem e a câmera.

Movimento Suave: O movimento suave da câmera é calculado por uma interpolação linear (Lerp):

P\_{\text{suave}}(t) = \text{Lerp}(P\_{\text{atual}}(t), P\_{\text{desejada}}(t), \text{smoothSpeed})

É uma função que suaviza a transição entre a posição atual e a posição desejada da câmera.

É um fator que controla a suavidade do movimento (quanto mais baixo, mais lenta a transição).

Conclusão

Esses códigos aplicam conceitos matemáticos como:

Funções lineares e quadráticas para movimento.

Lógica booleana para condições de pulo e morte.

Interpolação linear para suavização de movimento.

Clamping para controlar os limites da vida do personagem.

Com isso, a matemática é uma ferramenta essencial para garantir que o jogo seja dinâmico, fluido e realista, controlando os movimentos, a física e as interações do jogador com o ambiente.