Curso: Ciência da Computação

Integrantes: Enzo Henrique Neves Sena, Enzo Yuichi Sakita, Katiê Prado

Disciplina: Cálculo

A parte matemática do nosso jogo “Beefed Up!”, aparece na parte de pontuação onde através da coleta de moedas. Em uma de nossas quests do game, o objetivo é coletar o máximo de moedas possíveis (1600) para que o nosso protagonista possa melhorar e reconstruir a cidade.

Cada moeda tem o valor de 200.

**Análise Matemática no Código:**

**1. Operações Algébricas:**

* A operação totalCoins += coinValue; representa **uma soma repetida**, ou seja, um modelo de **progressão aritmética**.
  + Cada moeda coletada adiciona +200 ao total.
  + Isso pode ser modelado pela função:

f(x)=200xf(x) = 200xf(x)=200x

onde xxx é o número de moedas coletadas e f(x)f(x)f(x) é o total de pontos acumulados.

**2. Equações e Inequações:**

* + if (totalCoins >= limiteParaTrocarCena)Resolvendo:

x≥8x \geq 8x≥8

Isso significa que o jogador precisa **coletar no mínimo 8 moedas** para avançar de fase.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto, Aplicativo, Email

O conteúdo gerado por IA pode estar incorreto.

Uma imagem contendo mulher, estrada, em pé, segurando

O conteúdo gerado por IA pode estar incorreto.

Jogador ao recolher 4 moedas.

Exemplificando a progressão aritmética

Uma imagem contendo mulher, mesa

O conteúdo gerado por IA pode estar incorreto.

No menu de configurações o mecanismo de volume para equalizar o som ambiente do jogo.

Interface gráfica do usuário, Site

O conteúdo gerado por IA pode estar incorreto.

**📐 Parte 1: Cálculo – Onde há uso de lógica matemática ou física**

**⚙️ 1. Gravidade e pulo (Movimento.cs)**

if (Input.GetKeyDown(KeyCode.Space) && estaNoChao)

{

ForcaY = 7f; // impulso para cima

}

if (ForcaY > -9.81f)

{

ForcaY += -9.81f \* Time.deltaTime; // aceleração para baixo (gravidade)

}

controller.Move(new Vector3(0, ForcaY, 0) \* Time.deltaTime);

* Usa fórmula da **aceleração constante** (gravidade).
* A gravidade é simulada manualmente com ForcaY.
* **Força inicial de salto** e depois queda com aceleração negativa.

**🧠 2. Verificação de colisão com o chão**

estaNoChao = Physics.CheckSphere(peDoPersonagem.position, 0.3f, colisaoLayer);

* Usa uma **esfera imaginária (raio = 0.3)** para detectar se está tocando o chão.
* Isso é cálculo espacial com **interseção de formas geométricas**.

**📏 3. Distância entre jogador e inimigo (Perseguindo.cs)**

float distancia = Vector3.Distance(transform.position, jogador.position);

* Calcula a **distância euclidiana** entre dois pontos no espaço (fórmula √(x² + y² + z²)).
* Se a distância for menor que o limite, o inimigo **para e ataca**.

**🧲 4. Raio de interação (Interação.cs)**

Ray playerAim = new Ray(interactionPosition, InteractionSource.forward);

if (Physics.Raycast(playerAim, out RaycastHit hitInfo, InteractionRange))

* Cálculo de **vetores direcional e distância** para lançar o raio.
* Usa **física 3D** com Raycasting (cálculo de interseção entre uma linha e objetos no mundo).