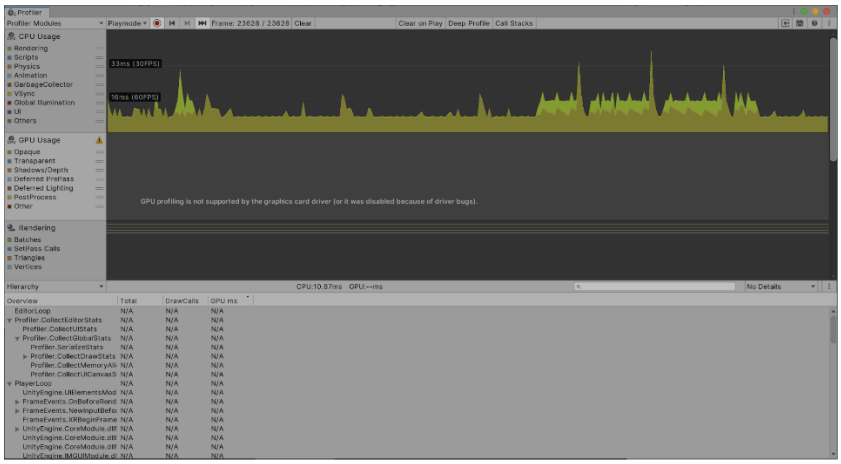
Otimização em Unity

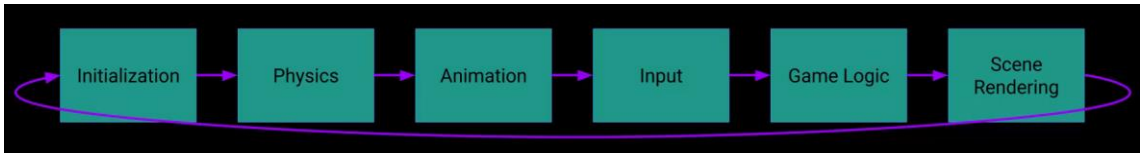
Profiler:



O Profiler é uma ferramenta valiosa para identificar e resolver problemas de desempenho, mas é mais eficaz quando o jogo está em execução e você precisa entender o que está causando uma queda de desempenho, como baixas taxas de quadros (FPS) ou uso excessivo de recursos da CPU ou GPU.

No entanto, para problemas como o jogo travar ou o código não ser executado corretamente, o Profiler pode não ser diretamente útil, pois esses problemas geralmente estão relacionados a erros de lógica, exceções não tratadas ou outros problemas de programação que não afetam diretamente o desempenho do jogo.

Basicamente, a engine funciona através de um ciclo, que fica se repetindo diversas vezes em um frame. Performance é quando falamos da velocidade para completar o ciclo, ou seja, a velocidade que roda meu jogo, e otimização é quando precisamos melhorar uma área do ciclo. Como Animations, Rendering, Physics etc.



Mas agora, como deixar o processo mais eficiente?

Bom, isso depende, afinal nem sempre podemos fazer isso da melhor forma possível e ao mesmo tempo funcional. Mas tem boas praticas que podemos seguir, como:

- Não deixar loops rodando o tempo todo sem uma verificação primeiro
- Somente chamar funções e métodos quando necessário
- Também podemos fazer o código executar a cada contagem X de quadros
- Usar o cache, pegar as informações no start/awake e deixar guardadas
- Diminuir a carga em cima do garbage collector (cuidado com tipo referência e valor)
- Usar métodos como Pool Objects
- Usar o LOD quando possível, para controlar o render pela distancia
- Usar Addressables para gerenciar os assets em projetos mobile

Aqui estão algumas áreas comuns de foco para otimização na Unity:

1. Desempenho da CPU

- Redução de cálculos desnecessários: Evitar loops desnecessários e operações redundantes.
- Uso eficiente de algoritmos: Escolher algoritmos mais eficientes para tarefas complexas.
- Pooling de objetos: Reutilizar objetos em vez de destruí-los e instanciá-los repetidamente.
- Cálculos em lote: Processar múltiplos itens em uma única operação.

2. Desempenho da GPU

- Redução de Draw Calls: Minimizar o número de chamadas de desenho, usando técnicas como batching.
- Uso eficiente de shaders: Utilizar shaders otimizados e adequados para o hardware alvo.
- LOD (Level of Detail): Usar diferentes níveis de detalhe para objetos baseados na distância da câmera.
- Occlusion Culling: Evitar renderizar objetos que estão fora da visão da câmera.

3. Uso de Memória

- Gerenciamento de Assets: Carregar apenas os assets necessários no momento e liberar quando não forem mais necessários, utilizando sistemas como Addressables.
- Redução de Garbage Collection: Minimizar alocações de memória que resultam em coleta de lixo frequente.
- Uso de tipos de valor e referência: Entender e usar eficientemente structs e classes para reduzir o overhead de memória.

4. Tempo de Carregamento

- Carregamento assíncrono: Carregar assets de forma assíncrona para não bloquear o jogo.
- Divisão de cenas: Dividir grandes cenas em partes menores que podem ser carregadas sob demanda.
- Uso de Asset Bundles/Addressables: Gerenciar e carregar assets sob demanda de forma mais eficiente.

5. Desempenho de Física

- Simplificação de colisionadores: Usar colisionadores mais simples para objetos complexos.
- Desativação de física desnecessária: Desativar simulações físicas em objetos que não requerem física em certos momentos.
- Uso eficiente de Rigidbodies: Usar rigidbodies apenas quando necessário.

6. Desempenho de Animação

- Redução de Blend Trees: Simplificar árvores de blend de animação para reduzir cálculos.
- Otimização de rigs: Usar rigs de animação simplificados para personagens.

Ferramentas de Otimização na Unity

- Unity Profiler: Para identificar gargalos de performance.
- Frame Debugger: Para analisar as chamadas de desenho.
- Memory Profiler: Para analisar o uso de memória e encontrar leaks.

Exemplos Práticos de Otimização em Projetos 2D e 3D

1. Pooling de Objetos

Pooling de objetos é uma técnica para reutilizar instâncias de objetos em vez de criar e destruir constantemente. Isso é especialmente útil em jogos com muitos objetos temporários, como balas ou inimigos.

Exemplo em 2D:

Em um jogo de tiro espacial, use um pool para gerenciar balas, reutilizando-as em vez de destruí-las e criar novas a cada tiro.

Exemplo em 3D:

Em um jogo de ação em terceira pessoa, use um pool para gerenciar inimigos, melhorando a performance ao reutilizar instâncias de inimigos.

2. Addressables

Addressables permitem carregar e descarregar assets dinamicamente, gerenciando a memória de forma eficiente.

Exemplo em 2D:

Carregar e descarregar sprites para diferentes níveis em um jogo de plataforma.

Exemplo em 3D:

Carregar e descarregar modelos 3D para diferentes ambientes em um jogo de exploração.

3. Level of Detail (LOD)

LOD é uma técnica para reduzir a complexidade dos modelos 3D conforme eles se afastam da câmera, melhorando a performance sem comprometer a qualidade visual de perto.

Exemplo em 3D:

Configurar LODs para objetos distantes em um jogo de mundo aberto, usando modelos de baixa, média e alta complexidade dependendo da distância do objeto à câmera.

4. Occlusion Culling

Occlusion Culling evita a renderização de objetos que estão fora do campo de visão da câmera, economizando recursos de processamento.

Exemplo em 3D:

Habilitar occlusion culling em um jogo de tiro em primeira pessoa para melhorar a performance em ambientes complexos.

5. Uso Eficiente de Física

Reduzir o custo da simulação física desativando simulações desnecessárias ou simplificando coliders.

Exemplo em 2D:

Desativar simulações físicas em objetos que não precisam ser atualizados a cada frame, como objetos estáticos ou de baixa importância.

Exemplo em 3D:

Usar coliders simples para objetos complexos, como coliders convexos para formas complexas, reduzindo o custo de processamento físico.