Relatório Técnico - Debug e Gerenciamento de Memória na RagX Engine

Data de geração: 24/06/2025 00:00

Resumo da Sessão:

Hoje foi realizada uma análise aprofundada do sistema de destruição de recursos e vazamento de memória na RagX Engine.

Problema Inicial:

- Vazamentos de memória eram reportados no final da execução.
- Sistema de logging (_CrtDumpMemoryLeaks) acusava blocos de 8~48 bytes alocados.

Diagnóstico:

- Todos os objetos (como IWindow) estavam sendo corretamente destruídos por SafeDestroy.
- A ordem de chamada de _CrtDumpMemoryLeaks estava incorreta.
- Isso causava falsos positivos, pois a engine ainda estava viva no momento do dump manual.

Solução:

- Foi removida a chamada direta a _CrtDumpMemoryLeaks().
- Foi configurado corretamente:

_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);

- Com isso, o Visual Studio passou a reportar apenas vazamentos reais (após o encerramento total da engine).

Explicações Técnicas:

- SafeDestroy: executa Unregister + Destroy + reset.

- Collector: gerencia subsistemas que implementam IDestroyable.

- IWindow herda de IDestroyable, logo o Window também.

- Destroy é chamado de forma polimórfica.

- std::make_unique<Window>() estava sendo corretamente rastreado.

Observações Importantes:

- _CrtDumpMemoryLeaks() manual deve ser evitado.

- _CrtSetDbgFlag com as flags certas faz o dump automático no fim da aplicação.

- Chamadas a Shutdown() devem ocorrer antes do retorno do main.

Confirmações:

- Breakpoints mostraram que Destroy e SafeDestroy foram executados corretamente.

- Logs exibiram a destruição da janela.

- Última execução finalizou com código 0 e sem nenhum bloco de memória pendente.

Conclusão:

A estrutura de coleta e destruição da RagX Engine está funcionando corretamente.

O problema era apenas de ordem de finalização. Agora está 100% resolvido.

Assinado,

Assistente Técnico - OpenAI