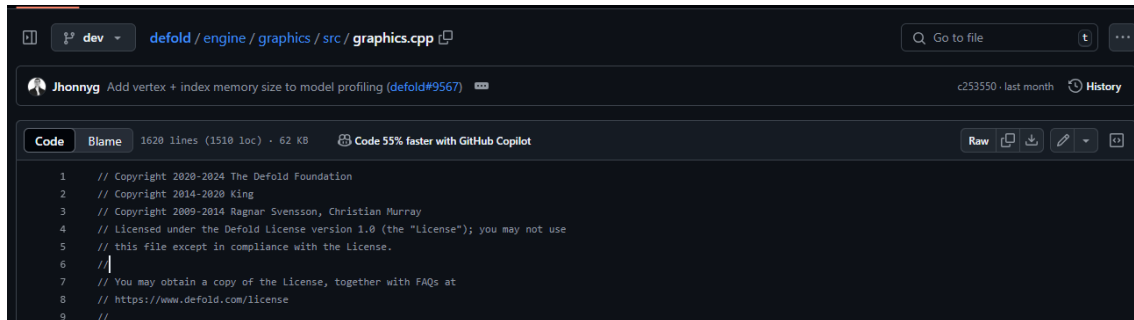


1 e 2 respostas estão nos arquivos das atividades em java nesta pasta

3 e 4

Código original está neste arquivo “graphics.cpp” linha 15 até 259

Caminho no git



```
1 // Copyright 2020-2024 The Defold Foundation
2 // Copyright 2014-2020 King
3 // Copyright 2009-2014 Ragnar Svensson, Christian Murray
4 // Licensed under the Defold License version 1.0 (the "License"); you may not use
5 // this file except in compliance with the License.
6 //
7 // You may obtain a copy of the License, together with FAQs at
8 // https://www.defold.com/license
9 //
```

Link do git: <https://github.com/defold/defold/blob/dev/engine/graphics/src/graphics.cpp>

Code Smells Identificados

1. Variáveis Globais:

```
static GraphicsAdapter* g_adapter_list = 0;

static GraphicsAdapter* g_adapter = 0;

static GraphicsAdapterFunctionTable g_functions;
```

Problema: O uso de variáveis globais pode causar problemas de manutenção, concorrência e testes.

Solução: Encapsular essas variáveis em uma classe ou estrutura para melhor modularidade e encapsulamento.

2. Funções Complexas:

Funções como RegisterGraphicsAdapter, SelectAdapterByFamily, e SelectAdapterByPriority têm lógica complexa que poderia ser refatorada em funções menores.

3. Repetição de Código:

As funções GetAdapterFamilyLiteral, GetTextureTypeLiteral, GetBufferTypeLiteral, GetGraphicsTypeLiteral, GetAssetTypeLiteral, e GetTextureFormatLiteral têm um padrão repetitivo que pode ser simplificado.

Sugestões de Melhorias

1. **Encapsular Variáveis Globais:** Encapsular as variáveis globais dentro de uma classe para melhor controle e modularidade.

cpp

```
class GraphicsManager {
public:
    static GraphicsManager& Instance() {
        static GraphicsManager instance;
        return instance;
    }

    void RegisterGraphicsAdapter(GraphicsAdapter* adapter,
                                GraphicsAdapterIsSupportedCb
is_supported_cb,
GraphicsAdapterRegisterFunctionsCb register_functions_cb,
                                GraphicsAdapterGetContextCb
get_context_cb,
                                int8_t priority);

    bool SelectAdapterByFamily(AdapterFamily family);
    bool SelectAdapterByPriority();

private:
    GraphicsManager() : g_adapter_list(nullptr),
g_adapter(nullptr) {}

    GraphicsAdapter* g_adapter_list;
    GraphicsAdapter* g_adapter;
    GraphicsAdapterFunctionTable g_functions;
};
```

2. **Refatorar Funções Complexas:** Dividir a lógica complexa em funções menores e mais gerenciáveis.

cpp

```
void GraphicsManager::RegisterGraphicsAdapter(GraphicsAdapter*
adapter,

GraphicsAdapterIsSupportedCb is_supported_cb,

GraphicsAdapterRegisterFunctionsCb register_functions_cb,

GraphicsAdapterGetContextCb get_context_cb,
                                int8_t priority) {
    adapter->m_Next = g_adapter_list;
    adapter->m_IsSupportedCb = is_supported_cb;
    adapter->m_RegisterCb = register_functions_cb;
    adapter->m_GetContextCb = get_context_cb;
    adapter->m_Priority = priority;
    g_adapter_list = adapter;
}
```

```

bool GraphicsManager::SelectAdapterByFamily(AdapterFamily
family) {
    if (family != ADAPTER_FAMILY_NONE) {
        return selectAdapter([family](GraphicsAdapter* adapter)
{
            return adapter->m_Family == family && adapter-
>m_IsSupportedCb();
        });
    }
    return false;
}

bool GraphicsManager::SelectAdapterByPriority() {
    return selectAdapter([](GraphicsAdapter* adapter) {
        return adapter->m_IsSupportedCb();
    });
}

template <typename Predicate>
bool GraphicsManager::selectAdapter(Predicate predicate) {
    GraphicsAdapter* next = g_adapter_list;
    GraphicsAdapter* selected = nullptr;
    while (next) {
        if (predicate(next) && (!selected || next->m_Priority <
selected->m_Priority)) {
            selected = next;
        }
        next = next->m_Next;
    }
    if (selected) {
        g_functions = selected->m_RegisterCb();
        g_adapter = selected;
        return true;
    }
    return false;
}

```

3. **Eliminar Repetição de Código:** Utilizar macros ou templates para reduzir a repetição nas funções de obtenção de literais.

cpp

```

#define GRAPHICS_ENUM_TO_STR_CASE(x) case x: return #x;

template <typename EnumType>
const char* GetEnumLiteral(EnumType value) {
    switch (value) {
        GRAPHICS_ENUM_TO_STR_CASE(ADAPTER_FAMILY_NONE);
        GRAPHICS_ENUM_TO_STR_CASE(ADAPTER_FAMILY_NULL);
        GRAPHICS_ENUM_TO_STR_CASE(ADAPTER_FAMILY_OPENGL);
        GRAPHICS_ENUM_TO_STR_CASE(ADAPTER_FAMILY_VULKAN);
        // ... outros casos
        default: break;
    }
    return "<unknown>";
}

const char* GetAdapterFamilyLiteral(AdapterFamily family) {
    return GetEnumLiteral(family);
}

```

Nota:

O código não apresentou nenhuma melhora desde a última atualização acredito que eles não vão melhorá-lo

5

Três ferramentas populares para detectar code smells, juntamente com os tipos de code smells que cada uma pode identificar

SonarQube	Duplicação de código, métodos longos, complexidade ciclomática alta, nomes de variáveis inadequados, etc.
Checkstyle	Métodos longos, nomes inconsistentes, classes grandes, falta de modularização
ESLint	Código repetido, funções/métodos grandes, estrutura incorreta, variáveis não utilizadas