UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

NOBERTO ANDREY RODRIGUES DE NEGREIROS

IMPLEMENTAÇÃO DE UM SISTEMA DINÂMICO DE FONTES PARA UM MOTOR DE JOGOS

MOSSORÓ
2021

NOBERTO ANDREY RODRIGUES DE NEGREIROS

IMPLEMENTAÇÃO DE UM SISTEMA DINÂMICO DE FONTES PARA UM MOTOR DE JOGOS

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Judson Santos Santiago

MOSSORÓ

2021

N385i Negreiros, Noberto Andrey Rodrigues de.
Implementação de um sistema dinâmico de fontes para um motor de jogos / Noberto Andrey Rodrigues de Negreiros. - 2021.
44 f. : il.

Orientador: Judson Santos Santiago.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Ciência da Computação, 2021.

1. Jogos eletrônicos. 2. Motor de jogos. 3. Fonte e texto. I. Santiago, Judson Santos, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automáto em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Mossoró / Setor de Informação e Referência
Bibliotecária: Keina Cristina Santos Sousa e Silva
CRB: 15/120

NOBERTO ANDREY RODRIGUES DE NEGREIROS

IMPLEMENTAÇÃO DE UM SISTEMA DINÂMICO DE FONTES PARA UM MOTOR DE JOGOS

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência da Computação.

Defendida em: 17 de novembro de 2021

BANCA EXAMINADORA

_____

Prof. Dr. Judson Santos Santiago

UFERSA

_____

Prof. Dr. Helcio Wagner da Silva

UFERSA

_____

Prof. Dr. Paulo Henrique Lopes Silva

UFERSA

Dedico este trabalho à minha família, que sempre deu suporte que foi essencial na concretização desta etapa.

AGRADECIMENTOS

Agradeço a minha família, Rita de Cássia, Paulo, Rômulo e Ivaneide, por estarem sempre presentes e me guiarem durante a trajetória deste curso. Sem eles, não há a menor dúvida de que eu não teria conseguido.

Agradeço a minha gata, Luna, pela companhia nas longas noites de estudo.

Agradeço aos amigos Arthur Pereira, Gustavo Audi e Matheus Augusto, por suportarem minhas lamentações sobre as dificuldades encontradas durante o curso, além dos valiosos conselhos.

Agradeço aos amigos Itágores Lira e Jhonathan Lima, por caminharem juntos comigo em boa parte das disciplinas do curso e tornarem elas mais interessantes.

Agradeço aos amigos do trabalho, em especial Anna Rachel, Bruna Larine e Raphaelle Gurgel, por entenderem e cobrirem algumas responsabilidades quando precisei focar nas atividades da faculdade.

Agradeço a Deus, por proporcionar saúde, sabedoria e paciência durante todo o curso.

Agradeço a todos os professores do curso pelos ensinamentos e aconselhamentos.

Agradeço ao meu orientador, Prof. Dr. Judson Santiago, pelo apoio no desenvolvimento deste projeto, pelas ótimas aulas e pela inspiração como professor.

"Understand well as I may, my comprehension can only be an infinitesimal fraction of all I want to understand." (Ada Lovelace)

RESUMO

O mercado global de jogos eletrônicos vem crescendo muito nos últimos anos. O Brasil é o país com o maior mercado para jogos eletrônicos da América Latina. Esse vasto mercado tem atraído cada vez mais desenvolvedores. Para o desenvolvimento de jogos, usualmente é utilizado um motor de jogos, que é um conjunto de ferramentas que auxiliam no planejamento e desenvolvimento dos jogos. Tomando como plataforma o motor de jogos desenvolvido na disciplina "Programação de Jogos" do curso de Ciência da Computação da UFERSA, percebeu-se que algo pode ser melhorado no sistema de fontes e desenho de textos implementado. Visando realizar essa melhoria, foi realizada uma pesquisa sobre as características que envolvem o carregamento de fontes e desenho de texto em motores de jogos. Um sistema que se integre ao motor em estudo foi implementado, respeitando o balanceamento entre a complexidade do sistema na utilização para o ensino e a quantidade de recursos oferecidos. Buscou-se, ainda, garantir uma performance adequada.

Palavras-chave: Jogos eletrônicos; Motor de jogos; Fonte e texto.

ABSTRACT

The global electronic games market has been growing a lot in the recent years. Brazil is the country with the biggest market for electronic games of all Latin America. This big market has been increasingly attracting more developers. For the development of games, usually a game engine, which is a set of tools that aids in planning and developing games, is used. Taking as a platform the game engine developed in the Game Development subject of the Computer Science degree at UFERSA, it was realized that an improvement can be done on the implemented font and text drawing systems. To implement this improvement, research was conducted with the subject of properties and characteristics of font loading and text drawing in game engines. A system that seamlessly integrates into the focused engine was implemented, respecting and balancing between the complexity of the system as a teaching subject and the feature set offered. An adequate performance was also the goal.

Keywords: Electronic games; Game engine; Text and font.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

2D	Duas dimensões
3D	Três dimensões
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
D2D	Direct2D
D3D	Direct3D
D3D10	Direct3D 10
D3D11	Direct3D 11
DLL	Dynamic-link Library
FW1	FW1FontWrapper
GDI	Graphics Device Interface
OTF	OpenType Font
TTF	TrueType Font
UFERSA	Universidade Federal Rural do Semi-árido
UML	Unified Modeling Language

SUMÁRIO

1 **INTRODUÇÃO**

De acordo com Witkowski (2021), o mercado global de vídeo games já é maior do que a soma do mercado global de filmes com o mercado esportivo norte-americano, com um crescimento de 20% em 2020, atingindo uma receita de 179,70 bilhões de dólares. Ainda segundo ele, a pandemia foi um fator importante para esse crescimento tão expressivo, porém não há perspectivas de redução significativa para os próximos anos.

No Brasil, o mercado de jogos faturou 2,19 bilhões de dólares em 2020 e terá uma receita estimada de 2,30 bilhões em 2021, um crescimento de 5,1%. Isto faz do país o maior mercado de jogos eletrônicos da América Latina e o 12º maior do mundo (HENRIQUE, 2021).

Essa valorização de mercado atrai os desenvolvedores e é aqui que os motores de jogos auxiliam. Eles são ferramentas de software que fornecem um meio para reduzir o custo e complexidade no desenvolvimento de jogos, pois são arquitetados de forma a garantir uma grande abstração sobre aspectos complexos do desenvolvimento de jogos (HALPERN, 2018). Existem inúmeros motores disponíveis e isso acontece devido às diferentes dificuldades que os desenvolvedores encontram ao arquitetar e implementar um jogo. Porém, alguns componentes são encontrados em quase todos eles (SALAMA; ELSAYED, 2018). São eles:

- Sistema gráfico, com suporte a 2D ou 3D;
- Sistema de física, com suporte a colisões;
- Entrada de usuário, com possível suporte a controladores de jogos;
- *Scripting* através de linguagem própria ou terceira;
- Sistema de sons;
- Sistema de rede;

Durante a disciplina “Programação de Jogos”, parte do currículo optativo do curso de Ciência da Computação da UFERSA, um motor de jogos 2D é implementado em C++, utilizando a API gráfica do DirectX 11. Entre os diversos sistemas implementados, existe o de fonte e desenho de texto, responsável pelo carregamento de fontes e desenho dos textos na tela.

1.1 **Justificativa**

Apesar de o sistema original desempenhar a tarefa a que se propõe, algumas dificuldades são percebidas ao desenvolver algo mais complexo, como uma aplicação que envolva o uso de múltiplas fontes ou estilos. A pesquisa se justifica na existência diversos métodos que tratam

com o carregamento de fontes e desenhos de textos nos motores gráficos de uso geral e em jogos, alguns deles oferecendo o carregamento de fontes sem precisar utilizar folhas de caracteres.

1.2 **Objetivos**

1.2.1 Geral

Implementar um sistema dinâmico de fontes e desenho de texto em C++ que se integre ao motor da disciplina Programação de Jogos, em sua última versão, respeitando sua arquitetura de projeto e fornecendo ao usuário alternativas na utilização de fontes, com opções para utilizar fontes externas fornecidas pelo usuário ou alguma fonte padrão do sistema.

1.2.2 Específicos

- Pesquisar sobre trabalhos relacionados ao desenho de texto em motores de jogos;
- Implementar um sistema de carregamento dinâmico de fontes que não dependa de um software externo;
- Garantir a integração com o motor em estudo sem perder a capacidade de utilização para o ensino;
- Implementar novas funcionalidades que venham a ser úteis.

1.3 **Metodologia**

Para desenvolver o trabalho, fez-se: (i) um levantamento da fundamentação teórica envolvendo o desenho de texto; (ii) uma análise da viabilidade técnica da integração das alternativas existentes; (iii) execução da implementação do sistema; (iv) refinamento da implementação com objetivo de melhoramento da performance.

1.4 **Estrutura do Texto**

Ester documento está estruturado em cinco capítulos.

O Capítulo 1, Introdução, apresentou uma visão geral do trabalho, incluindo: problematização, objetivos do trabalho e metodologia utilizada.

O Capítulo 2, Sistema de Fontes Atual, possui uma apresentação do sistema em uso atualmente no motor em estudo, bem como vantagens e desvantagens.

No Capítulo 3, Estado da Arte e Alternativas, fez-se um levantamento dos motores mais utilizados atualmente no mercado e quais características estão presentes neles. Ainda foram levantadas as alternativas existentes considerando a utilização da API DirectX 11.

No Capítulo 4, Escolha e Implementação, foram descritos a escolha da biblioteca DirectWrite como base, bem como foi apresentada a implementação desenvolvida neste trabalho.

No Capítulo 5, Conclusão, são apresentadas as conclusões e possíveis trabalhos futuros.

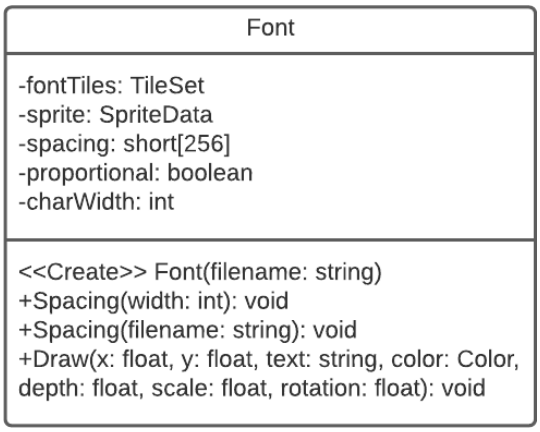
2 SISTEMA DE FONTES ATUAL

O motor gráfico possui diversos sistemas implementados que tratam das mais diversas funcionalidades necessárias para o funcionamento pleno de um jogo em duas dimensões. Entre eles, temos: sistemas de áudio, *sprites*, partículas, temporizadores, detecção de colisão, controladores de jogos, cenas, renderizador gráfico, textos e outros.

2.1 Descrição da classe Font

O sistema de desenho de texto é implementado diretamente através da classe Font e utiliza alguns mecanismos já implementados no próprio motor, principalmente os sistemas de *sprites* e de *tileset*. Sua representação na UML pode ser conferida na Figura 1.

Figura 1 — Modelagem UML da classe Font



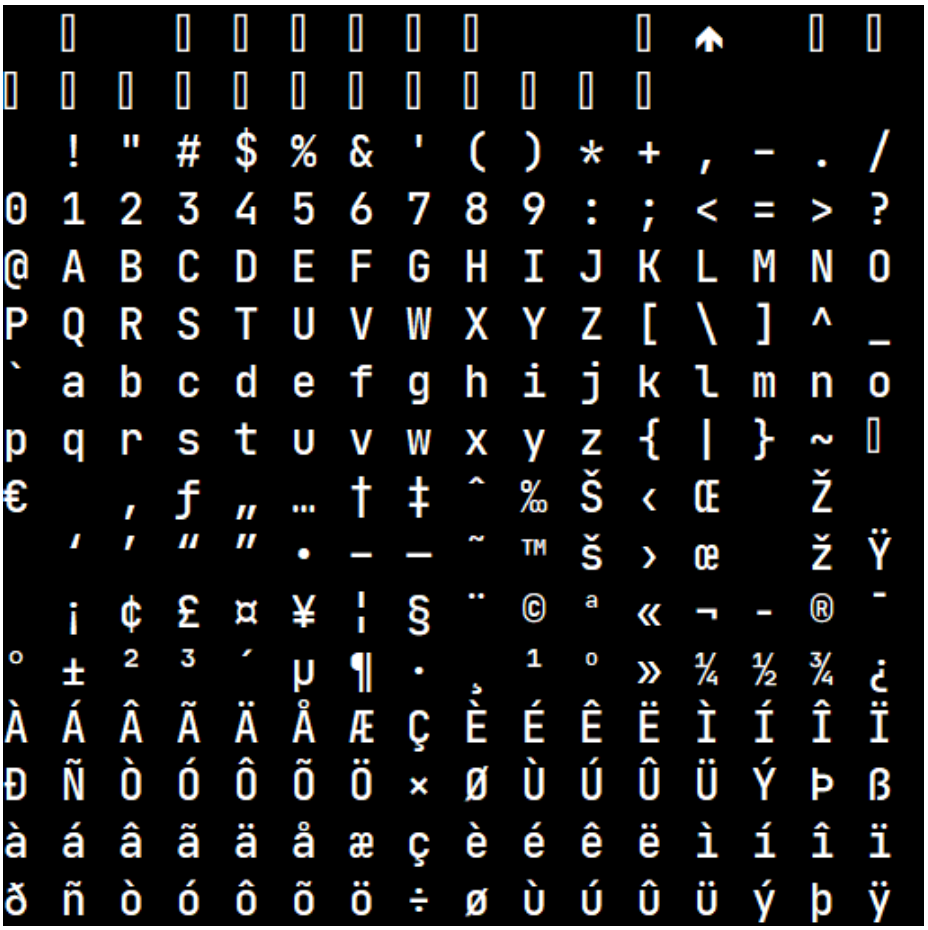
Fonte: O autor (2021)

Em sua atual implementação, o sistema de fontes é responsável pelo carregamento de uma folha de caracteres que representa uma fonte. Uma folha de caracteres é uma imagem que possui desenhada uma grade de caracteres que serão utilizados pela aplicação. Elas podem conter a quantidade de caracteres que o desenvolvedor desejar e não são restritas necessariamente aos caracteres do alfabeto.

No caso da classe Font, o mapeamento dos caracteres corresponde ao da tabela ASCII e a geração da folha de caracteres em si é feita através de um programa externo, como o *Bitmap Font Builder*. Utilizando o programa, deve-se escolher qual fonte será utilizada, além de opções como negrito, itálico, tamanho, cor dos caracteres e cor de fundo. O resultado é uma imagem

contendo a grade com os símbolos, como se pode ver na Figura 2, além de um arquivo opcional contendo informação sobre o espaçamento de cada um deles.

Figura 2 — Folha de caracteres da fonte Arial



Fonte: O autor (2021)

Além de carregar a folha de caracteres representando a fonte, o sistema implementa duas formas de exibição de texto: fixo, onde todos os caracteres possuem o mesmo espaçamento; e proporcional, onde cada caractere possui um espaçamento próprio. Dependendo de qual opção de espaçamento o usuário escolher, o sistema carrega ou não as informações de espaçamento no arquivo opcional gerado pelo programa externo.

2.2 **Desenhando o texto**

Após inicializada, a classe Font representa a folha de caracteres de uma determinada fonte, bem como o espaçamento entre os caracteres, sejam fixos ou proporcionais.

Para desenhar um determinado texto, chama-se o método Draw do objeto fornecendo como parâmetros a posição horizontal, a posição vertical e o texto a ser desenhado. Opcionalmente, podem ser fornecidos ainda uma cor para a fonte, uma profundidade, uma escala de tamanho e uma rotação. Abaixo está um exemplo de utilização da classe, com todo seu ciclo de vida:

```
#include "Font.h"

// criação do objeto Font
Font* fonte = new Font("Resources/jetbrains.png");
fonte->Spacing("Resources/jetbrains.dat");

// chamada para desenho do texto
Color cor = { 1,1,1,1 };
float profundidade = 0.0f;
float escala = 1.0f;
float rotacao = 0.0f;
fonte->Draw(100, 100, "Olá, mundo!", cor, profundidade, escala, rotacao);

// liberação da memória
delete fonte;
```

Com os parâmetros, o método Draw do objeto utiliza as mecânicas de cores, profundidade, escala de tamanho e rotação já implementadas nos sistemas de *sprite* e *tileset* para desenhar cada caractere do texto individualmente. Ou seja, cada caractere do texto é passado para o renderizador gráfico como um *sprite* derivado do *tileset.* A posição onde cada caractere será desenhado é fruto de um cálculo que leva em consideração o espaçamento do símbolo anterior, a rotação, a escala de tamanho e as posições horizontais e verticais fornecidas como parâmetros.

É importante notar que esse trabalho de desenhar o texto é feito a cada quadro e não leva em consideração se houve alguma alteração no conteúdo do texto.

2.3 **Vantagens e Desvantagens**

Como é de conhecimento comum no mundo da computação, toda arquitetura computacional trabalha em torno dos requisitos determinados no planejamento e naturalmente quase sempre existe um sacrifício em alguma característica para obter um benefício em outra (SOMMERVILE, 2011, p. 57-80).

2.3.1 Vantagens

Entre os pontos fortes do atual sistema, a abstração do tratamento direto com um arquivo de fonte o permite ser completamente independente em relação ao sistema de fontes do sistema operacional. Dessa forma, o motor sequer precisa saber quais fontes estão instaladas no sistema.

Do ponto de vista de projeto do motor, um outro ponto forte é que ele se adequa perfeitamente ao sistema de exibição de *sprites* já implementado, agindo quase como um módulo *plug-and-play*.

Por último, temos o desempenho. Como é um sistema que já utiliza funcionalidades nativas do motor, para pequenos textos - em número de caracteres - dinâmicos, ou seja, que sofrem atualizações quadro a quadro, possui um desempenho excelente na execução do código.

2.3.2 Desvantagens

Entre os pontos fracos, o principal é a falta de flexibilidade em relação ao tratamento com a fonte. Como o sistema trabalha apenas com folhas de caracteres, qualquer necessidade de redesenhar um texto utilizando uma aparência diferente da fonte, como negrito ou itálico, resultará na obrigatoriedade de gerar uma nova folha de caracteres utilizando o programa externo.

Caso a necessidade seja apenas de aumentar ou diminuir o tamanho do texto, existe a possibilidade, através do parâmetro de escala, porém resulta numa perda de qualidade visual, exemplificado na Figura 3. A alternativa seria justamente gerar uma nova folha de caracteres com o tamanho de fonte maior.

Figura 3 — Textos desenhados no tamanho natural e amplificado a partir de um tamanho menor

Fonte: O autor (2021)

Sobre o desempenho, há considerações a se fazer, pois apesar de ter um excelente desempenho em pequenos textos dinâmicos, o sistema deixa a desejar em textos estáticos, que são os textos que sofrem nenhuma ou pouca alteração no conteúdo. Em textos com mais caracteres, o desempenho também sofre uma perda de velocidade na execução proporcional ao tamanho do texto em caracteres, independentemente de serem dinâmicos ou estáticos.

2.4 **Uso acadêmico e motivação para a melhoria**

Considerando que o motor em questão é utilizado como ferramenta acadêmica, algumas características devem levar em consideração a viabilidade do ensino. Para isso, o sistema de fontes atual é simples de entender e faz o trabalho necessário, com uma ressalva: flexibilidade.

Caso o estudante queira desenvolver algo que envolva um pouco mais de complexidade com o texto, certamente terá que gerar várias folhas de caracteres e testar manualmente cada uma delas para ter certeza de que atende a necessidade. Não é impossível e nem possui dificuldade elevada, mas é um trabalho manual e repetitivo, que tende a frustrar um pouco o usuário.

Ainda sobre a flexibilidade, devido à simplicidade do sistema atual, algumas outras características comuns no trabalho com texto não estão implementadas, como alinhamentos

horizontal e vertical, que decorre da impossibilidade de delimitar o texto em uma área determinada.

Um sistema que removesse o trabalho manual da geração das folhas de caracteres e implementasse algumas características comuns no tratamento de texto certamente melhoraria a usabilidade do motor gráfico. Como resultado dessa melhoria, o usuário poderia concentrar ainda mais na lógica do desenvolvimento de suas atividades, talvez melhorando seu aprendizado.

3 ESTADO DA ARTE E ALTERNATIVAS

3.1 Estado da Arte

Ao tratar com texto, algumas características estão sempre presentes e muitas delas são utilizadas com frequência: fonte a ser utilizada, tamanho, cor, estilo (negrito, itálico, sublinhado, riscado etc.), alinhamento, posicionamento, rotação, dentre outros.

Há também outras características menos utilizadas, mas que possuem muita importância para alguns casos de uso, especialmente considerando a enorme variedade de linguagens e tipos de escrita pelo mundo todo. Importante também notar a constante evolução das mídias digitais e das novas características de texto que estão surgindo e ganhando popularidade. Os *emojis*, por exemplo, já fazem parte do padrão Unicode e possuem 3633 entradas na tabela de códigos (UNICODE INCORPORATED, 2021).

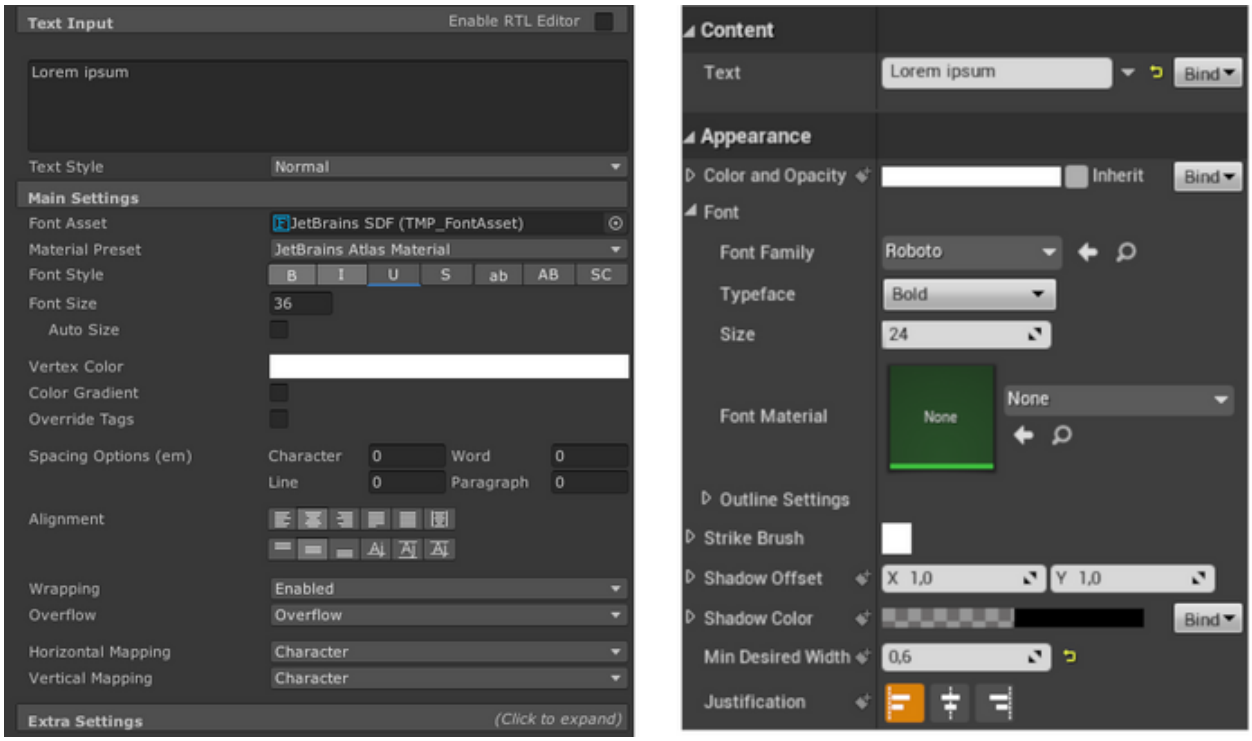
Há linguagens com sentidos de escrita diferentes, como algumas linguagens arábicas que possuem o sentido da escrita da direita para a esquerda. Existe ainda a língua japonesa, que possui variação na escrita, com textos mais formais e tradicionais sendo escritos no sentido vertical, de cima para baixo (ABE, 2019).

Com todas essas características a serem consideradas, renderizar textos é uma tarefa extremamente complexa e deve levar em consideração inúmeros fatores, portanto não existe um sistema perfeito para a tarefa. O que existem são sistemas que se esforçam para oferecer o melhor suporte possível, como disse Beingessner (2019).

Atualmente, existem diversos motores gráficos de jogos profissionais utilizados por grande parte da indústria de jogos comerciais, alguns deles de uso gratuito para fins não comerciais. Entre os mais populares, temos os consagrados *Unity Engine* e *Unreal Engine (BUCKLEY, 2021).*

Em suas implementações de desenho de texto em duas dimensões, é possível perceber a presença de diversas das características citadas anteriormente, inclusive em ambas as implementações. Na Figura 4, podemos ver diversas opções básicas de texto.

Figura 4 — Opções básicas de texto para os motores Unity e Unreal, respectivamente



Fonte: O autor (2021)

Com opções de compilação para computadores, consoles e até para celulares, ambos possuem uma implementação que abstrai completamente recursos dependentes do sistema operacional. Com essa diversidade de plataformas suportadas, as implementações garantem a paridade de funcionalidades em todas elas. Assim, um texto é exibido com as mesmas características visuais em uma aplicação desenvolvida para o computador, consoles ou qualquer outra plataforma suportada.

3.2 **Alternativas**

Como o motor gráfico da disciplina foi projetado para suportar apenas o sistema operacional Windows e utiliza a API gráfica DirectX 11, as opções de implementação devem seguir o mesmo requisito. Com isso, surgem algumas alternativas que são frequentemente utilizadas por outros motores: implementar um sistema próprio de renderização de caracteres, renderização utilizando folha de caracteres ou utilizar uma biblioteca externa compatível com o DirectX 11.

3.2.1 Sistema próprio de renderização de caracteres

Em situações ideais, desenvolver um sistema próprio de renderização de caracteres seria a opção mais recomendada, pois não existiria qualquer restrição no ponto de partida. Adicionalmente, o sistema poderia ter toda a arquitetura moldada para trabalhar em perfeita sintonia com o motor e possuir todas as funcionalidades desejadas pelo desenvolvedor. Isso pode ser facilmente observado nos motores gráficos profissionais, como Unity e Unreal Engine.

Como discutido, porém, renderização de texto não é uma tarefa fácil e inúmeros aspectos devem ser levados em conta e implementados individualmente. Definir essa opção como escopo de um trabalho de conclusão de curso seria desafiador, mas certamente consumiria todo o tempo disponível para a pesquisa sem produzir algum resultado satisfatório.

3.2.2 Renderização utilizando folha de caracteres

A renderização através de uma folha de caracteres pré-desenhados é uma técnica muito utilizada na computação gráfica, em especial em projetos de menor orçamento ou consoles antigos. Ela é simples de implementar, rápida para pequenos textos e de fácil compreensão, além de consumir poucos recursos e requerer pouco poder de processamento. É a opção adotada pelo sistema atual do motor.

Uma forma de eliminar o trabalho manual de gerar a folha de caracteres para cada fonte que se deseja trabalhar é implementar um sistema que gere as folhas de caracteres de forma dinâmica, durante a execução da aplicação. A desvantagem é que ao desenvolver um sistema como esse, nada impede de que o próprio sistema que gera as folhas de caracteres trate do desenho do texto em si, já que a parte complexa - o desenho dos caracteres - já estaria implementada. No fim das contas, o que estaria sendo desenvolvido seria um sistema próprio de renderização de caracteres, devendo lidar com toda a complexidade envolvida.

3.2.3 Utilizar biblioteca externa

Quando se fala sobre o uso de bibliotecas externas, é possível perceber diversos níveis de abstração. Existem bibliotecas que implementam praticamente todos os aspectos da solução para um determinado problema e fornecem uma API de alto nível. Por outro lado, existem também bibliotecas que facilitam o desenvolvimento de uma solução, mas ainda fornecem uma API de baixo nível, possibilitando ao desenvolvedor um maior controle sobre a implementação.

O problema de utilizar uma biblioteca que implementa quase todos os aspectos do problema é que, fatalmente, ela terá uma visão própria de como as coisas devem funcionar e dificilmente essa visão corresponde a do projeto que se pretende desenvolver. Isso pesa ainda mais em um projeto de cunho acadêmico, pois o motor gráfico segue uma filosofia própria de desenvolvimento em todos os seus aspectos. Ter um componente quebrando essa filosofia, por menor que seja a quebra, não é desejável. Um exemplo dessa situação, seria a biblioteca *FW1FontWrapper*, que implementa um sistema de fontes para desenho de texto no *Direct3D 11*. Ela implementa interfaces de uma outra biblioteca, a *DirectWrite*, para formatação, composição de texto e outras características (RUFELT, 2017, com adaptações).

Utilizar uma biblioteca mais flexível, que possui diferentes níveis de abstração, por outro lado, permite que a implementação se adapte à filosofia do projeto e seja tão complexa quanto a arquitetura demande. Uma biblioteca que se encaixa nessa descrição seria a *DirectWrite*.

4 ESCOLHA E IMPLEMENTAÇÃO

4.1 A escolha da biblioteca DirectWrite

4.1.1 Apresentando a biblioteca

A biblioteca DirectWrite faz parte da API gráfica do DirectX e fornece suporte completo ao texto e composição do padrão Unicode. Além disso, oferece um sistema de composição independente do dispositivo em uso; suporte para texto multiformato; texto acelerado por hardware, quando utilizado junto com o Direct2D; composição e desenho de texto em todas as linguagens suportadas. Ou seja, a biblioteca oferece toda a estrutura básica para o trabalho com texto,inclusive com opções próprias para desenho, através do Direct2D ou GDI (MICROSOFT, 2021, com adaptações).

Ela permite implementações que podem ser simples o bastante para desenhar uma única linha de texto ou complexas ao ponto de permitir que o desenvolvedor elabore seu próprio renderizador de texto, implementando as interfaces da biblioteca que achar necessário.

O método de desenho escolhido foi o da API da Direct2D, outra biblioteca que integra o DirectX, com o foco em desenhos em 2D, como o nome bem indica. Essa escolha se deu, pois como o motor em estudo já utiliza a Direct3D, irmã 3D da Direct2D, diversas estruturas seriam reaproveitadas e a filosofia se manteria. A integração se dá através de uma lógica de interoperabilidade entre a D3D e a D2D, com dois métodos de ligação: através da obtenção da *swapchain* – uma técnica de *buffer* da renderização que destina alguns *buffers* onde os quadros são acumulados antes de serem encaminhados para a tela, como forma de amenizar pequenos travamentos e estabilizar os quadros - do motor ou através da obtenção de uma superfície 2D gerada pelo motor.

4.1.1.1 Vantagens

Utilizar a DirectWrite na implementação traria alguns benefícios. Ser uma biblioteca do próprio DirectX é um fator de enorme importância, pois como o motor já o utiliza, a integração poderia acontecer de forma mais natural.

Outra vantagem advém da flexibilidade com a qual o desenvolvedor pode trabalhar com a DirectWrite. No escopo deste trabalho, foi utilizada apenas uma fração das funcionalidades

oferecidas pela biblioteca, garantindo o funcionamento desejado para o momento. Porém, a possibilidade de estender a funcionalidade do sistema, caso necessário, existe e pode ser feita aos poucos.

4.1.1.2 Desvantagens

Por outro lado, existem alguns pontos negativos. Do ponto de vista acadêmico, cria-se uma espécie de caixa preta na parte do desenho de texto em si, já que o sistema recorre à implementação própria da biblioteca. Apesar de ser uma desvantagem, não é tão grave, pois como já discutido previamente, a área da computação que estuda esse aspecto possui uma complexidade enorme e, por si só, exigiria horas de aulas para compreensão.

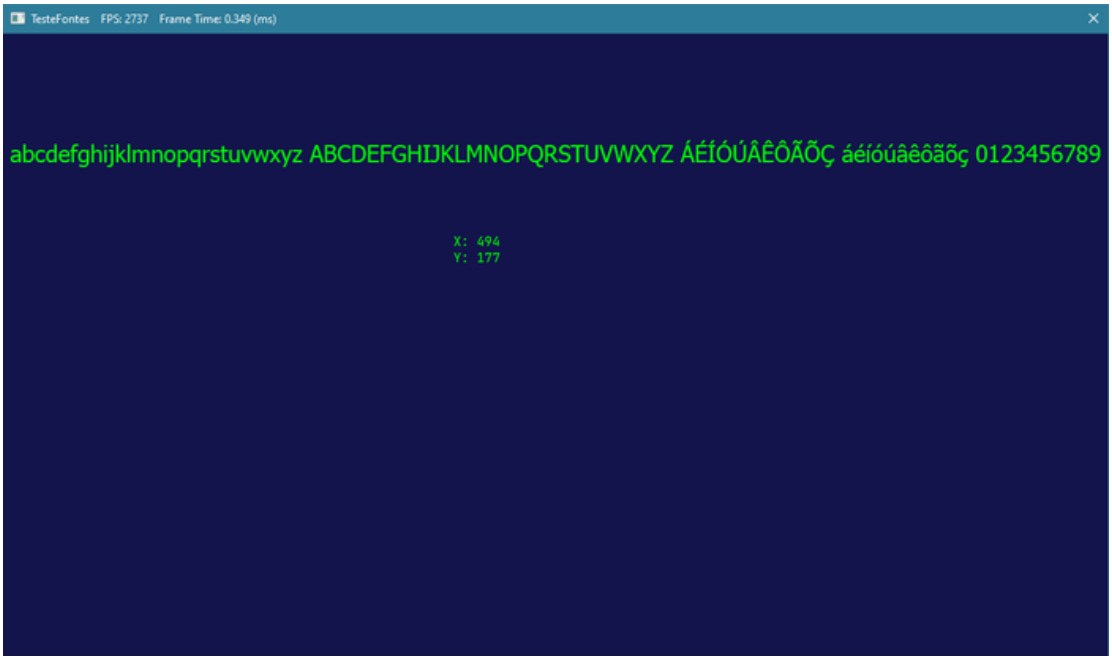
O ponto fraco que realmente traz uma consequência muito negativa e deve ser tratada se dá ao fato de a biblioteca não operar nativamente no D3D. Como sua operação se dá através de interoperabilidade do D2D com o D3D, sempre que um texto precisa ser desenhado, há uma mudança de contexto no dispositivo gráfico. Essa mudança de contexto possui um custo de processamento elevado e deve ser evitada sempre que possível (MICROSOFT, 2020, com adaptações).

4.1.2 Teste de viabilidade

Baseando-se nos fatores apresentados acima, a biblioteca foi escolhida como a principal candidata para a implementação. Inicialmente, utilizando a última versão disponível do motor gráfico em estudo, foi desenvolvido um protótipo para demonstrar a viabilidade técnica da biblioteca para trabalhar em conjunto com os atuais sistemas já implementados no motor.

Na Figura 5, é exibido o resultado do primeiro protótipo. Ainda sem nenhuma correspondência entre as funcionalidades já existentes no sistema, apenas fornecendo a opção de desenho de texto básico.

Figura 5 — Protótipo demonstrando viabilidade técnica da utilização em conjunto com o motor da biblioteca DirectWrite



Fonte: O autor (2021)

Estava provada a viabilidade técnica da utilização da biblioteca DirectWrite, em interoperabilidade através do D2D e, portanto, foi promovida de candidata a escolhida.

4.2 **Ajuste à filosofia do motor**

Na implementação inicial, o texto foi desenhado, porém ainda apresentava alguns detalhes dissonantes da filosofia do motor. Um deles, o tratamento de desenho em camadas distintas, estava completamente ausente e tornava inviável a utilização nesse estado. Todos os textos desenhados em tela apareciam na profundidade correspondente ao "fundo" da tela, ou seja, qualquer coisa desenhada ficaria acima deles, cobrindo-os completamente. Isso foi resultado do método de integração inicialmente escolhido entre o motor e o D2D.

A lógica que o motor em estudo utiliza determina que toda a atualização e o desenho da tela são realizados seguindo uma ordem:

1. Todos os objetos de uma determinada cena são atualizados;
2. O sistema gráfico limpa a tela para o próximo quadro;
3. Os objetos são processados e encaminhados para o renderizador;
4. O desenho é, de fato, realizado na memória gráfica;
5. Finalmente, o sistema apresenta os desenhos na tela.

No protótipo inicial, foi utilizado o método de interoperabilidade através da obtenção da *swapchain. O* desenho era realizado de forma direta, sem passar pelo renderizador do motor. Isso fez com que não existisse a possibilidade de se determinar a camada na qual o texto seria desenhado, pois o desenho na memória gráfica acontecia na etapa onde os objetos eram processados, item 3 da lista acima.

Era necessária uma nova estratégia. O sistema deveria ser repensado para levar em consideração a sequência lógica acima ou utilizaria-se o método de interoperabilidade utilizando uma superfície 2D, que teria como efeito colateral nos permitir desenhar o texto em uma textura 2D de forma mais simples. Essa última opção certamente era a mais atrativa, pois permitiria utilizar vários mecanismos já implementados no motor e garantiria seu desenho no momento exato, com suporte à camada incluso.

Com a estratégia determinada, foi feita uma nova implementação para se adequar completamente à arquitetura do motor, utilizando seus mecanismos de renderização para desenhar as texturas contendo os textos desenhados. Essa nova tentativa teve como resultado um sistema visualmente e funcionalmente correto, como pode ser visto na Figura 6.

Figura 6 — Nova implementação com suporte ao desenho de texto em diversas camadas, controladas pelo desenvolvedor

Fonte: O autor (2021)

Seu desempenho, porém, não era dos melhores. O desempenho foi medido utilizando o contador interno de quadros por segundos (QPS) do motor e considerou-se a média de quadros reportada em 30 segundos, exibindo o mesmo conteúdo em texto para as implementações. O resultado pode ser visto na Tabela 1.

Tabela 1 — Medição de quadros por segundos da segunda implementação

SITUAÇÃO	SEGUNDA IMPLEMENTAÇÃO	SISTEMA ORIGINAL
Textos estáticos	694 QPS	1478 QPS
Textos dinâmicos	521 QPS	1264 QPS

Fonte: O autor (2021)

4.3 **Refinamento**

Com um desempenho insatisfatório, foi preciso investigar o que estava causando tanto processamento desnecessário. Após investigação através de profiladores do Visual Studio Community 2019, percebeu-se que o sistema estava consumindo muito processamento em uma chamada de função do próprio D2D. Era o preço da interoperabilidade entre D2D e D3D.

Nas duas implementações anteriores, o sistema processava o texto a ser desenhado e chamava pela função de desenho individualmente em todos os quadros, para todos os textos presentes na cena, independentemente de o texto ter ou não sofrido alguma alteração. Isso gerava uma textura para cada texto que era posteriormente enviada ao renderizador gráfico do motor.

Após muita consideração, uma nova estratégia foi pensada: dividir o desenho dos textos por camadas, de forma a diminuir as chamadas de desenho do D2D, reduzindo assim o preço pago pela interoperabilidade. Outra característica repensada foi a de redesenhar os textos mesmo quando eles não sofriam nenhuma alteração.

Na nova e última implementação, o sistema adotou um esquema de desenho de textos divididos pelas camadas em que estão alocados e também leva em consideração se o texto realmente precisa ser redesenhado, ou seja, se sofreu alguma alteração em seu conteúdo ou posição desde o último quadro.

Com isso, foi alcançado um desempenho excelente para textos estáticos. É importante destacar que o ganho de performance para este tipo de texto foi expressivo, pois o desenho do

texto pelo D2D é realizado apenas uma vez, no primeiro quadro da cena. Em todos os quadros subsequentes, é realizado apenas o desenho das texturas contendo os textos. Essa operação requer um processamento muito menor.

Para textos dinâmicos, onde há a necessidade de atualizar o texto em todos os quadros, o desempenho evoluiu e está satisfatório graças ao mecanismo de desenhar textos agrupados por camadas.

Na Tabela 2, pode-se perceber uma enorme melhora (713%) nos textos estáticos em relação à segunda implementação, bem como uma melhora significante também nos textos dinâmicos (103%).

Tabela 2 — Medição de quadros por segundos da implementação final

SITUAÇÃO	**IMPLEMENTAÇÃO FINAL**	**SISTEMA ORIGINAL**
Textos estáticos	5644 QPS	1478 QPS
Textos dinâmicos	1061 QPS	1264 QPS

Fonte: O autor (2021)

4.4 **O novo sistema**

4.4.1 Apresentação

O novo sistema de fontes e texto foi arquitetado seguindo a filosofia de projeto do motor gráfico em estudo, buscando sempre o equilíbrio entre o funcionamento e o nível de complexidade para sua utilização no ensino de programação de jogos.

O sistema foi dividido em seis classes que trabalham em conjunto para desenhar o texto. São elas: *NewFont*, responsável pelo carregamento e representação de fontes; *FontEnumerator* e *FontLoader*, responsáveis pelo carregamento de arquivos de fontes fornecidos pelo desenvolvedor; *TextRenderer*, responsável pelo processamento dos textos que serão desenhados em um quadro; *Text*, representa um texto e suas características; *TextBox*, herda características da classe *Text* e representa um quadro com área limitada que possui um texto.

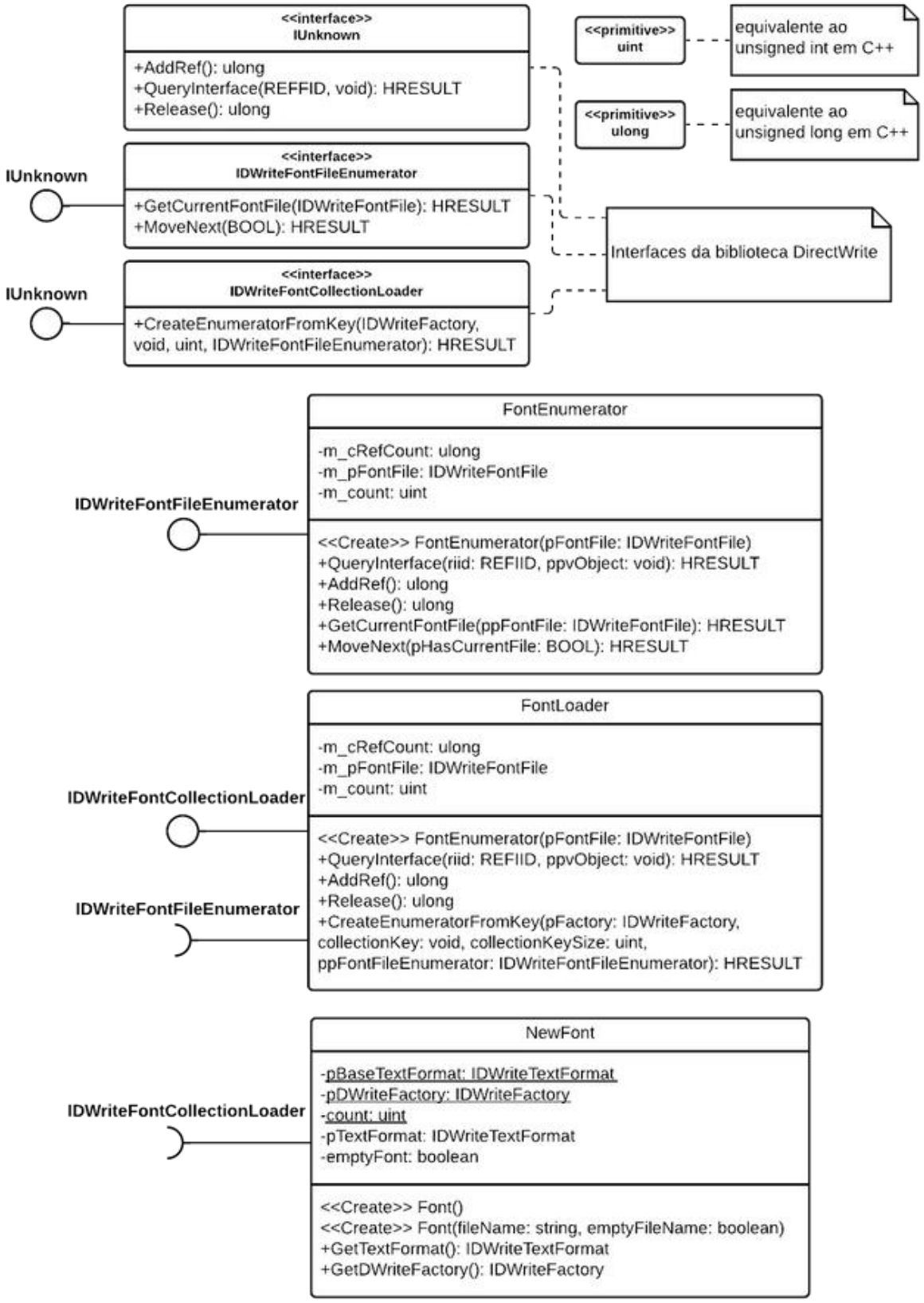
4.4.1.1 NewFont, FontEnumerator e FontLoader

A classe *NewFont* é o ponto de entrada para o desenho de um texto utilizando o novo sistema. Ela representa uma fonte carregada de um arquivo de fonte nos formatos TTF (*True Type Font*) ou OTF (*Open Type Font*), além de permitir a utilização de uma fonte padrão já instalada no sistema. No código fonte abaixo, segue exemplo de utilização da classe NewFont com inicialização através de fonte específica ou com a fonte padrão do sistema:

```
#include "NewFont.h"

// cria o objeto NewFont com uma fonte específica fornecida
NewFont* fonte = new NewFont("Resources/JetBrains.ttf");

// cria o objeto NewFont com uma fonte padrão do sistema
// Arial, se disponível, ou outra da mesma família
NewFont* fonte2 = new NewFont();

// libera a memória
delete fonte;
delete fonte2;
```

Para conseguir carregar arquivos de fontes fornecidas pelo usuário, duas classes de ajuda foram implementadas, *FontEnumerator* e *FontLoader*, que implementam as interfaces básicas da DirectWrite para o manuseamento dos arquivos. Respectivamente, foram implementadas as interfaces *IDWriteFontFileEnumerator* e *IDWriteFontCollectionLoader*, que são utilizadas internamente pela classe *NewFont*. A modelagem UML dessas três classes podem ser vistas na Figura 7.

Figura 7 — Modelagem UML das classes NewFont, FontEnumerator e FontLoader



Fonte: O autor (2021)

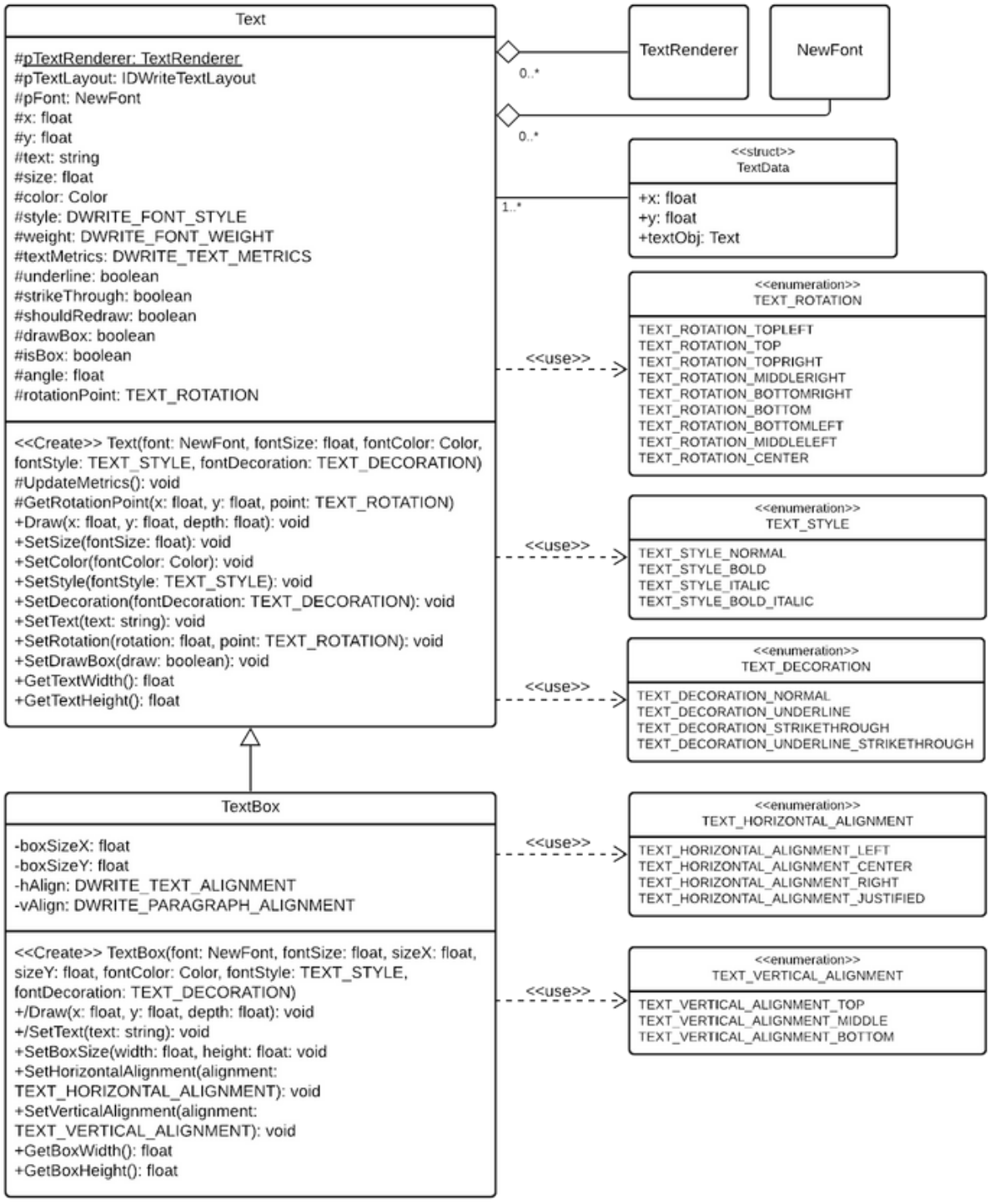
4.4.1.2 Text e TextBox

A classe *Text* implementa e representa todas as características básicas de um texto e, para isso, depende de uma representação de um arquivo de fonte, que é fornecida pela classe *NewFont*. *Text* é responsável pela formatação, processamento do texto e inscrição para desenho através do renderizador de texto, fornecido através da classe *TextRenderer*. Além disso, *Text* também é uma classe base para outra, a *TextBox*.

TextBox é uma classe que herda todas as funcionalidades de *Text* e adiciona a capacidade de delimitar uma área onde o texto será disposto. Essa nova classe não possui correspondente no sistema original.

Na Figura 8, é possível ver a modelagem UML da implementação de ambas as classes.

Figura 8 — Modelagem UML das classes Text e TextBox



Fonte: O autor (2021)

4.4.1.3 TextRenderer

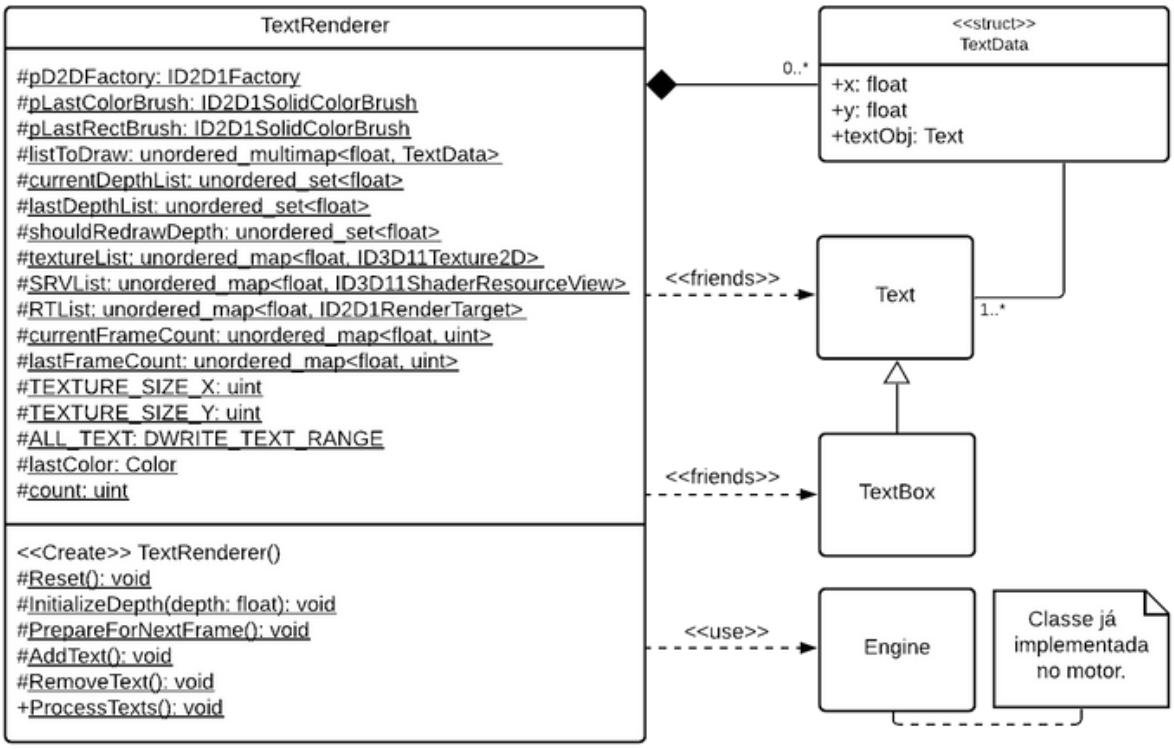
A classe *TextRenderer* é responsável pelo processamento dos textos que se inscreveram - através da função *Text::Draw* - para serem desenhados em um determinado quadro da aplicação. Esse processamento leva em consideração em qual camada cada objeto de texto deseja ser desenhado e realiza o desenho da textura contendo os textos. Cada camada possui sua própria textura, reduzindo as chamadas de desenho de texto do D2D de uma por texto para uma por camada.

Após desenhadas, as texturas com os textos são encaminhadas para o renderizador do motor gráfico, aproveitando toda a estrutura de dados já implementada. Dessa forma, a implementação dessa classe seguiu toda a lógica preexistente e se tornou quase um módulo, onde só é preciso uma chamada da função *ProcessTexts()* pelo motor, como podemos ver no trecho de código abaixo, retirado da função Engine::Loop(), implementada no motor:

```
...

if (!paused)
{
     // calcula o tempo do quadro
     frameTime = FrameTime();

     // atualização do jogo
     game->Update();

     // limpa a tela para o próximo quadro
     graphics->Clear();

     // desenha o jogo
     game->Draw();

     // processa textos
     textRenderer->ProcessTexts();

     // renderiza sprites
     renderer->Render();

     // apresenta o jogo na tela (troca backbuffer/frontbuffer)
     graphics->Present();
}

...
```

Na Figura 9, podemos ver a modelagem UML para a classe.

Figura 9 — Modelagem UML da classe TextRenderer



Fonte: O autor (2021)

4.4.2 Funcionalidades

4.4.2.1 NewFont

O novo sistema pode representar uma fonte de duas formas: através de um arquivo externo fornecido pelo usuário nos formatos TTF e OTF ou, caso o usuário não forneça nenhum parâmetro na chamada de criação do objeto, utilizar uma fonte padrão do sistema. Na implementação, optou-se pela fonte Arial, porém se ela não estiver instalada no sistema, a própria biblioteca DirectWrite busca uma fonte padrão.

4.4.2.2 Text

Cada objeto *Text* representa um texto a ser desenhado e, para isso, possui algumas características obrigatórias: fonte, cor, estilo e decoração. Esses parâmetros são fornecidos na criação do objeto e fazem parte da estrutura interna da classe. Como foi implementado em

conjunto com a classe *NewFont*, a fonte fornecida na criação do objeto *Text* deve ser naquele formato.

Entre os métodos implementados, temos alguns que modificam os parâmetros iniciais bem como outros que permitem alterar outras características não-obrigatórias ou encaminhar o texto para o renderizador de textos.

O método *SetText* permite a edição do texto contido no objeto e recebe como parâmetro uma *string*; *SetSize* altera o tamanho da fonte do texto e recebe como parâmetro um número *float; SetColor* modifica a cor do texto e recebe como parâmetro um objeto do tipo *Color*, já previamente implementado no motor; *SetStyle* e *SetDecoration* permitem a alteração do estilo (normal, negrito e itálico) e decoração (normal, sublinhado ou riscado), respectivamente, e recebem como parâmetro um dos valores das enumerações *TEXT_STYLE* e *TEXT_DECORATION*, também respectivamente; *SetRotation* possibilita a rotação do texto e recebe um ângulo - número *float* - em graus e um ponto de rotação sobre o qual o texto será rotacionado, esse ponto é um valor da enumeração *TEXT_ROTATION*; *Draw* é o método responsável por encaminhar o texto para o renderizador de texto e recebe como parâmetros números *float* que representam as posições x e y, bem como uma profundidade que determina em qual camada o texto será desenhado, também um número *float*; finalmente, *GetTextWidth* e *GetTextHeight* retornam a largura e altura do texto, respectivamente.

Todas as enumerações citadas no parágrafo anterior podem ser observadas na Figura 8.

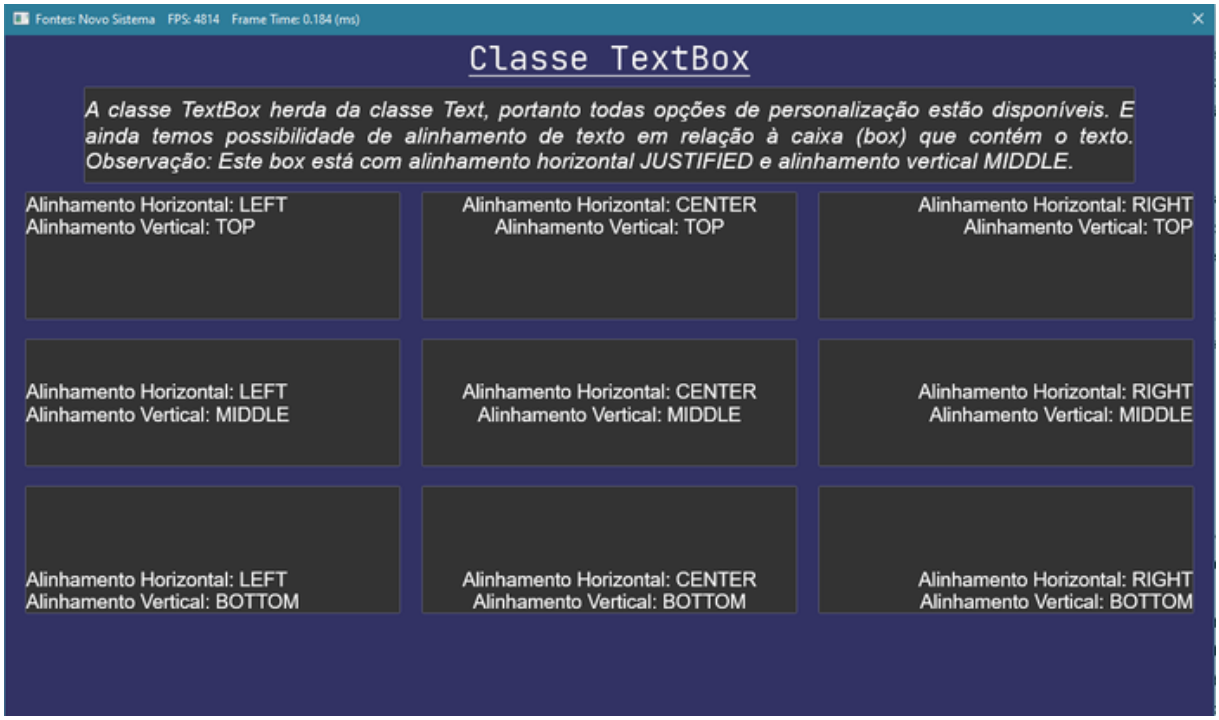
No trecho de código seguinte, um exemplo contemplando todo o ciclo de vida de um objeto *Text* é exibido:

```
#include "NewFont.h"
#include "Text.h"
// cria o objeto NewFont com uma fonte específica fornecida
NewFont* fonte = new NewFont("Resources/JetBrains.ttf");

float tamanho = 18.0f;
Color cor = { 1, 1, 1, 1 };
TEXT_STYLE estilo = TEXT_STYLE_BOLD;
TEXT_DECORATION decoracao = TEXT_DECORATION_UNDERLINE;
// cria o objeto Text com opções padrões
// de cor, estilo e decoração
Text* texto = new Text(fonte, tamanho);

// cria outro objeto Text com opções personalizadas
Text* texto2 = new Text(fonte, tamanho, cor, estilo, decoracao);

// modifica alguns parâmetros
texto->SetText("Olá, mundo!");
texto2->SetText("Olá, universo!");
texto->SetColor({ 1,0,0,1 }); //vermelho
texto->SetStyle(TEXT_STYLE_ITALIC);
texto->SetRotation(15.0f); // 15 graus de rotação

// chamada para desenho
// x, y e profundidade
texto->Draw(100, 100, 0.1f);
texto2->Draw(200, 200, 0.1f);

// libera a memória
delete texto;
delete texto2;
delete fonte;
```

4.4.2.3 TextBox

TextBox representa uma caixa que delimita a área onde o texto será contido. É um recurso adicional que não possui equivalente no sistema original e fornece novas opções de desenho de texto, especialmente o alinhamento horizontal e vertical do texto, além de possibilitar o desenho de texto justificado. Na Figura 10, é demonstrado o funcionamento da classe, com as diversas opções de alinhamento, bem como exibição da própria área que delimita o texto.

Figura 10 — A classe TextBox em funcionamento, demonstrando as funcionalidades de alinhamento dentro da caixa



Fonte: O autor (2021)

Como uma especialização da classe *Text*, *TextBox* possui todas as características descritas naquela, com algumas informações adicionais. Na criação do objeto, além de fonte, cor, estilo e decoração, devem ser fornecidos também a altura e largura da caixa que conterá o texto.

Além dos métodos disponíveis através da classe mãe, *TextBox* implementa alguns métodos específicos para tratar do alinhamento do texto e do tamanho da caixa. *SetBoxSize* permite alterar o tamanho da caixa que contém o texto, recebendo como parâmetros dos números *float* que representam altura e largura; *SetHorizontalAlignment* e *SetVerticalAlignment* alteram a formatação do texto e recebem como parâmetros valores das enumerações *TEXT_HORIZONTAL_ALIGNMENT* e *TEXT_VERTICAL_ALIGNMENT*, respectivamente; por último, *GetBoxWidth* e *GetBoxHeight* retornam a largura e altura, respectivamente, da caixa.

Todas as enumerações citadas no parágrafo anterior podem ser observadas na Figura 8.

No trecho de código abaixo, segue um exemplo de utilização da classe *TextBox*, da inicialização ao fim do uso:

```
#include "NewFont.h"
#include "TextBox.h"
// cria o objeto NewFont com uma fonte específica fornecida
NewFont* fonte = new NewFont("Resources / JetBrains.ttf");

float tamanhoFonte = 18.0f;
float tamanhoX = 200.0f;
float tamanhoY = 200.0f;

// cria o objeto TextBox com opções padrões
// de cor, estilo e decoração
TextBox* caixa = new TextBox(fonte, tamanhoFonte, tamanhoX, tamanhoY);

// modifica alguns parâmetros
caixa->SetText("Olá, caixa de texto!");
// define alinhamento horizontal como justificado
caixa->SetHorizontalAlignment(TEXT_HORIZONTAL_ALIGNMENT_JUSTIFIED);
// define alinhamento vertical como centralizado
caixa->SetVerticalAlignment(TEXT_VERTICAL_ALIGNMENT_MIDDLE);
caixa->SetDrawBox(true); // habilita desenho da caixa no fundo

// chamada para desenho
// x, y e profundidade
caixa->Draw(100, 100, 0.1f);

// libera a memória
delete caixa;
delete fonte;
```

5 CONCLUSÃO

Este trabalho realizou pesquisas teóricas sobre as principais técnicas de desenho de texto utilizadas em motores gráficos, em especial para motores utilizados no desenvolvimento de jogos. Com base nestas pesquisas e considerando o escopo do motor em estudo, ficou claro que a alternativa mais viável seria a utilização de uma biblioteca externa. Porém, a biblioteca deveria fornecer o maior controle possível para possibilitar uma integração sem causar maiores alterações na lógica preexistente do motor.

Este trabalho implementou um sistema de fontes e desenho de textos usando a biblioteca DirectWrite como base. Buscou-se, ainda, oferecer o maior número de funcionalidades sem comprometer seu uso para o ensino, balanceando desempenho, complexidade e respeito à filosofia do projeto. Como resultado, a implementação final conseguiu uma ótima performance para textos estáticos e um desempenho satisfatório para textos dinâmicos, além de fornecer uma funcionalidade inexistente no sistema original, que são as caixas de texto.

Como trabalhos futuros, podemos listar:

- aprofundamento no trabalho de otimização, em especial para textos dinâmicos.
- implementar outras interfaces da biblioteca DirectWrite, garantindo ainda mais controle e integração com o motor.
- experimentar a implementação de um sistema híbrido, onde seria utilizada a DirectWrite para desenho da folha de caracteres e o sistema atual para o desenho do texto em si.

REFERÊNCIAS

ABE, Namiko. **Should Japanese Writing Be Horizontal or Vertical?**. ThoughtCo. 2019. Disponível em: https://www.thoughtco.com/should-japanese-writing-be-horizontal-or-vertical-4070872. Acesso em: 6 out. 2021.

BEINGESSNER, Alexis. **Text Rendering Hates You**. 2019. Disponível em: https://gankra.github.io/blah/text-hates-you. Acesso em: 6 out. 2021.

BUCKLEY, Daniel. **Unity vs. Unreal - Choosing a Game Engine**. GameDev Academy. 2021. Disponível em: https://gamedevacademy.org/unity-vs-unreal/. Acesso em: 6 out. 2021.

HALPERN, Jared. **The What and Why of Game Engines**. Medium. 2018. Disponível em: https://medium.com/@jaredehalpern/the-what-and-why-of-game-engines-f2b89a46d01f. Acesso em: 19 out. 2021.

HENRIQUE, Arthur. **Mercado de jogos no Brasil deve atingir US$ 2,3 bilhões em 2021**. Olhar Digital. 2021. Disponível em: https://olhardigital.com.br/2021/05/05/games-e-consoles/mercado-de-jogos-no-brasil-2021-pesquisa/. Acesso em: 19 out. 2021.

MICROSOFT. **DirectWrite (DWrite)**. docs.microsoft.com. 2021. Disponível em: https://docs.microsoft.com/en-us/windows/win32/directwrite/direct-write-portal. Acesso em: 12 out. 2021.

MICROSOFT. **Improving the performance of Direct2D apps**. docs.microsoft.com. 2020. Disponível em: https://docs.microsoft.com/en-us/windows/win32/direct2d/improving-direct2d-performance. Acesso em: 12 out. 2021.

RUFELT, Erik. **FW1FontWrapper**. CodePlex Archive. 2017. Disponível em: https://archive.codeplex.com/?p=fw1. Acesso em: 6 out. 2021.

SALAMA, Ramiz; ELSAYED, Mohamed. Basic elements of game engine. **Global Journal of Computer Science: Theory and Research**, v. 8, n. 3, p. 126-131, dez. 2018. Disponível em: https://www.researchgate.net/publication/331103137_Basic_elements_and_characteristics_of_game_engine. Acesso em: 19 out. 2021.

SOMMERVILE, Ian. **Engenharia de Software**. Tradução Kalinka Oliveira e Ivan Bosnic. 9ª ed. São Paulo: Pearson Prentice Hall, 2011. 529 p. Tradução de: Software Engineering.

UNICODE INCORPORATED. **Emoji Counts, v14.0**. Unicode. 2021. Disponível em: https://www.unicode.org/emoji/charts-14.0/emoji-counts.html. Acesso em: 6 out. 2021.

WITKOWSKI, Wallace. **Videogames are a bigger industry than movies and North American sports combined, thanks to the pandemic**. Marketwatch. 2021. Disponível em: https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990. Acesso em: 19 out. 2021.