**GUIA DE DESENVOLVIMENTO**



**BPM Game Engine**

**Versão 1.0**

Fabio Takeshi Ishikawa

Janeiro de 2020

Sumário

[**1. Introdução** 3](#_Toc35293138)

[**1.1 O que é BPM Game Engine** 3](#_Toc35293139)

[**2. Visão Geral da Arquitetura** 4](#_Toc35293140)

[**2.1 Camada de Hardware** 5](#_Toc35293141)

[**2.2 Camada de Sistema Operacionais** 6](#_Toc35293142)

[**2.3 Camada API de Terceiros** 6](#_Toc35293143)

[**2.4 Camada Multiplataforma** 6](#_Toc35293144)

[**3. Configuração do Ambiente de Desenvolvimento** 6](#_Toc35293145)

[4. Dependências 6](#_Toc35293146)

[5. Debugando seu código 6](#_Toc35293147)

[6. Classes e Objetos Globais 6](#_Toc35293148)

[6.1 Classe Game Engine 7](#_Toc35293149)

[6.1.1 Atributos 7](#_Toc35293150)

[6.1.2 Métodos 7](#_Toc35293151)

[6.2 Classe Window 7](#_Toc35293152)

[6.2.1 Atributos 7](#_Toc35293153)

[6.2.2 Métodos 8](#_Toc35293154)

[6.2 Classe ApiWrapper (classe abstrata) 8](#_Toc35293155)

[6.2.1 Atributos 9](#_Toc35293156)

[6.2.2 Métodos 9](#_Toc35293157)

[6.3 Classe WinApi Wrapper (herdade de ApiWrapper) 9](#_Toc35293158)

[6.3.1 Atributos 9](#_Toc35293159)

[6.3.2 Métodos 9](#_Toc35293160)

[6.4 Classe GameEngineEventHandler (classe abstrata) 9](#_Toc35293161)

[6.5 Classe Render 9](#_Toc35293162)

[6.6 Classe IO 9](#_Toc35293163)

[6.7 Classe Mouse 9](#_Toc35293164)

[6.8 Classe Teclado 9](#_Toc35293165)

[6.9 Classe AssetManager 9](#_Toc35293166)

[7. Personalização 10](#_Toc35293167)

[8. Convenção de Código 10](#_Toc35293168)

# **1. Introdução**

Este documento tem como objetivo descrever em detalhes a arquitetura de BPM Game Engine, todas as classes que fazem parte do software e os seus relacionamentos, gerenciamento de configuração, ferramentas e linguagens de programação utilizadas para o desenvolvimento, convenção e boas práticas de programação, depuração de código, descrição e planejamento de testes, especificação do fluxo de trabalho com a ferramenta de controle de versão e práticas de integração contínua.

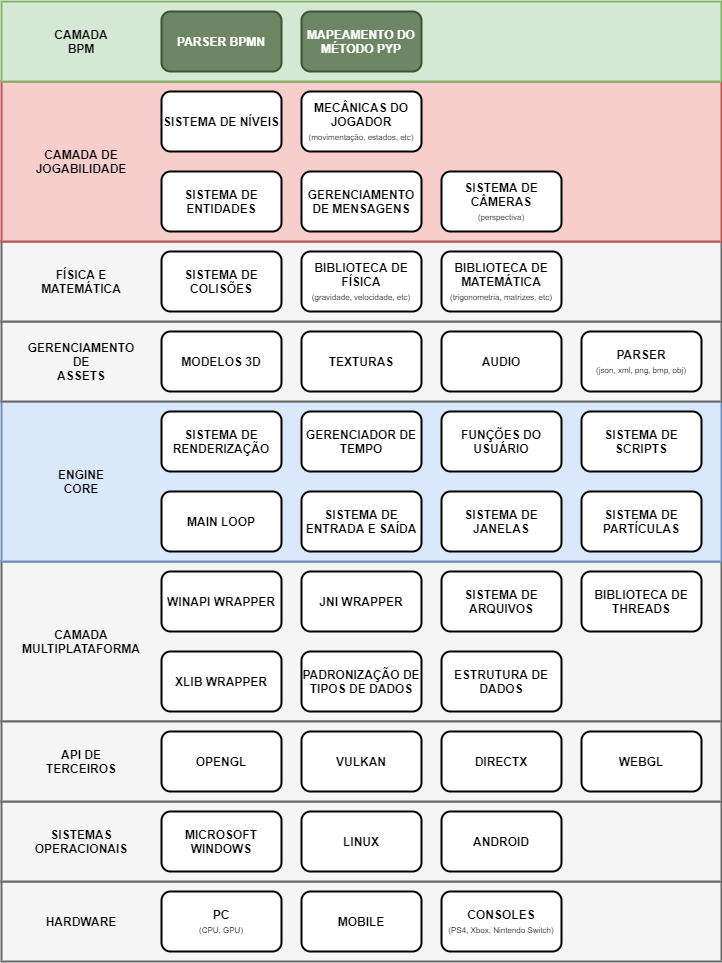
## **1.1 O que é BPM Game Engine**

BPM Game Engine é um motor de jogo que surgiu para apoiar a criação e desenvolvimento de jogos digitais baseados em modelos de processos de negócio. Este novo gênero de jogo surgiu a partir da pesquisa de doutorado “Play Your Process (PYP) – Um método de Design de Jogos Digitais Baseados em Modelos de Processos de Negócio”, onde é apresentado um método que permite transformar elementos de modelos de processos de negócio em elementos de design de jogos.

Apesar deste motor de jogo oferecer suporte completo para o desenvolvimento de jogos baseados em modelos de processos de negócio, ele também pode ser utilizado para criar jogos de diferentes gêneros.

# **2. Visão Geral da Arquitetura**

A seguir, segue a visão geral dos componentes que compõe BPM Game Engine. Os componentes estão divididos em diversas camadas que representam o relacionamento e a dependências entre os componentes. Todos os componentes da figura são descritos nesta seção.



*Figura 01. Visão geral dos componentes de BPM Game Engine dividido em camadas.*

## **2.1 Camada de Hardware**

Este projeto nasceu com a ideia de possibilitar o desenvolvimento de jogos digitais para múltiplas plataformas. Portanto toda a sua arquitetura foi pensada considerando as questões que envolve portabilidade para diversos tipos de hardware como PC (sistemas *multicores* combinados com *GPU*s – *Graphical Processing Unit*), dispositivos móveis e consoles (Playstation 4, Xbox One e Nintendo Switch).

Neste projeto há também diversas considerações relacionadas aos tipos de hardware, arquiteturas de memória e computação paralela para permitir o desenvolvimento de jogos de alto desempenho. Estas considerações são essenciais pois a evolução do hardware chegou em um nível onde praticamente todos os sistemas possuem processadores que possuem diversos núcleos e a utilização de múltiplos GPUs. Por exemplo, sistemas como Playstation 4 possui oito núcleos divididos em dois clusters onde cada cluster de quatro núcleos compartilham a mesma memória cache (L2) e todos os núcleos compartilham a mesma memória RAM formando uma arquitetura de memória NUMA (*non uniform memory access*). A utilização de paralelismo é fundamental para a criação de jogos de alto desempenho e está intimamente relacionada com o sistema de hardware. (GREGORY, 2018).

## **2.2 Camada de Sistema Operacionais**

Para suportar diferentes tipos de hardware é necessário lidar também com diferentes tipos de sistemas operacionais. Além disso, diversos elementos de um jogo digital são dependentes do sistema operacional como o gerenciamento de janelas da interface gráfica de usuário, biblioteca de threads, sistema de arquivos e, assim por diante. O foco deste projeto é permitir a construção de jogos para os seguintes sistemas operacionais: Microsoft Windows 10, Linux, Android e, futuramente, para iOS e MacOS.

## **2.3 Camada API de Terceiros**

O projeto utiliza API de terceiros para acessar o componente gráfico (GPU) do sistema. OpenGL é uma API (Application Program Interface) para o hardware gráfico que permite a renderização de modelos tridimensionais em tempo real. Ela é extremamente popular e, além de tudo, é portável. Existe também a possibilidade de integrar DirectX e Vulkan em versões futuras.

## **2.4 Camada Multiplataforma**

# **3. Configuração do Ambiente de Desenvolvimento**

# 4. Dependências

# 5. Debugando seu código

# 6. Classes e Objetos Globais

## 6.1 Classe Game Engine

### 6.1.1 Atributos

Objeto APIWrapper (classe abstrata)

Este objeto é responsável por executar métodos específicos do sistema operacional. Para cada sistema operacional, é necessário criar uma classe derivada de APIWrapper e definir os métodos virtuais. O tipo do objeto é definido no momento da compilação através de uma constante de compilação.

Objeto GameEngineEventHandler (classe abstrata)

Este objeto especifica as funções de callback da game engine como, por exemplo, as funções de inicialização, renderização e de finalização. Como padrão, a game engine cria as suas próprias funções de callback vazias. Elas devem ser configuradas com funções de callback definidas pelo usuário.

String Name

Especifica o nome do aplicação ou jogo e será utilizada como texto na barra de títulos de uma janela.

### 6.1.2 Métodos

SetGameEngineEvents

Método utilizado para configurar as funções de callback.

mainLoop

Função de execução da game engine. Assim que esse método é chamado, a game engine obtem o controle total da aplicação. Obviamente, ela respeitará as funções de callback do usuário para responder aos eventos conforme o desejo do usuário.

InitGameEngine

Método para inicializar a game engine.

## 6.2 Classe Window

### 6.2.1 Atributos

String titulo

Especifica o texto que será exibido na barra de títulos da janela de aplicação.

Int largura

Especifica a largura da janela em pixels

Int altura

Especifica a altura da janela em pixels

Int estilo

Especifica o estilo da janela. Os estilos possívels para uma janela são: com barra de título, sem barras de titilo, diferentes tipos de botões do sistema, todas as opções, janela redimensionavel, e não redimensionavel, janela maximizada, minimzada)

Int posicaoInicialX

Especifica a posição inicial da janela no eixo X.

Int posicaçãoInicialY

Especifica a posição inicial da janela no eixo Y.

### 6.2.2 Métodos

CreateWindow

Método para criar uma janela de aplicação.

DestroyWindow

Método para destruir uma janela de aplicação.

ResizeWindow

Método para redimensionar o tamanho de uma janela de aplicação.

setWindowPosition

Método para posicionar a janela

## 6.2 Classe ApiWrapper (classe abstrata)

Esta classe é responsável por tornar portável os atributos e métodos específicos de um sistema operacional. Ela é uma classe abstrata e os seus métodos devem ser definidos pelas classes derivadas.

### 6.2.1 Atributos

### 6.2.2 Métodos

## 6.3 Classe WinApi Wrapper (herdade de ApiWrapper)

Esta classe define os atributos e os métodos específicos do sistema operacional Windows.

### 6.3.1 Atributos

WGLOpenGL

WindowClass

### 6.3.2 Métodos

Window Procedure

Método responsável por tratar os eventos do sistema operacional.

## 6.4 Classe GameEngineEventHandler (classe abstrata)

Esta classe é responsável por definir funções de callback que serão chamadas pela game engine implicitamente. O usuário deverá criar uma classe derivada e definir as suas próprias funções de callback e configurar a game engine.

## 6.5 Classe Render

Esta classe é responsável por armazernar diversas configurações relacionadas renderização.

## 6.6 Classe IO

Esta classe é responsável por armazenar diversas informações a respeito de entrada e saída de dados.

## 6.7 Classe Mouse

Esta classe é responsável por receber os dados de entrada do mouse e disponibilizar para a game engine.

## 6.8 Classe Teclado

Esta classe é responsável por receber os dados de entrada do teclado e disponibilizar para a game engine.

## 6.9 Classe AssetManager

Esta classe é responsável por gerenciar diversos recursos como assets.

# 7. Personalização

# 8. Convenção de Código