**Documentação do Jogo**

Mario Jump

Autores:

Vinicius Rezende

Documentação:

Lucas Fraga

Belo Horizonte

01/07/2023.

# Documentação do Sistema

[Documentação do Software 2](#_Toc75023787)

[Introdução 3](#_Toc75023788)

[Escopo do software 3](#_Toc75023789)

[Nome do sistema e de seus componentes principais 3](#_Toc75023790)

[Missão ou objetivo do software 3](#_Toc75023791)

[Arquitetura do Sistema 3](#_Toc75023792)

[Funcionalidades do produto 3](#_Toc75023793)

[Usuários e sistemas externos 3](#_Toc75023794)

[Descrição 3](#_Toc75023795)

[Documentação do código 4](#_Toc75023796)

[Documentação da Estrutura de dados geral do software 4](#_Toc75023797)

[Função <nome da função> 4](#_Toc75023798)

[Função <nome da função> 4](#_Toc75023799)

[Testes do software 4](#_Toc75023800)

[Casos de testes do software: função <nome da função> 4](#_Toc75023801)

[Casos de testes do software: função <nome da função> 4](#_Toc75023802)

## Introdução

### Escopo do Jogo

#### Nome do jogo e de seus componentes principais

|  |
| --- |
| Mario jump |

#### Análise do público-alvo

|  |
| --- |
| Crianças na idade de alfabetização |

#### Pré-requisitos

|  |
| --- |
| Ter algum compilador e a biblioteca Allegro instalados na máquina |

#### Objetivos educacionais e específicos do jogo

|  |
| --- |
| Alfabetização de crianças |

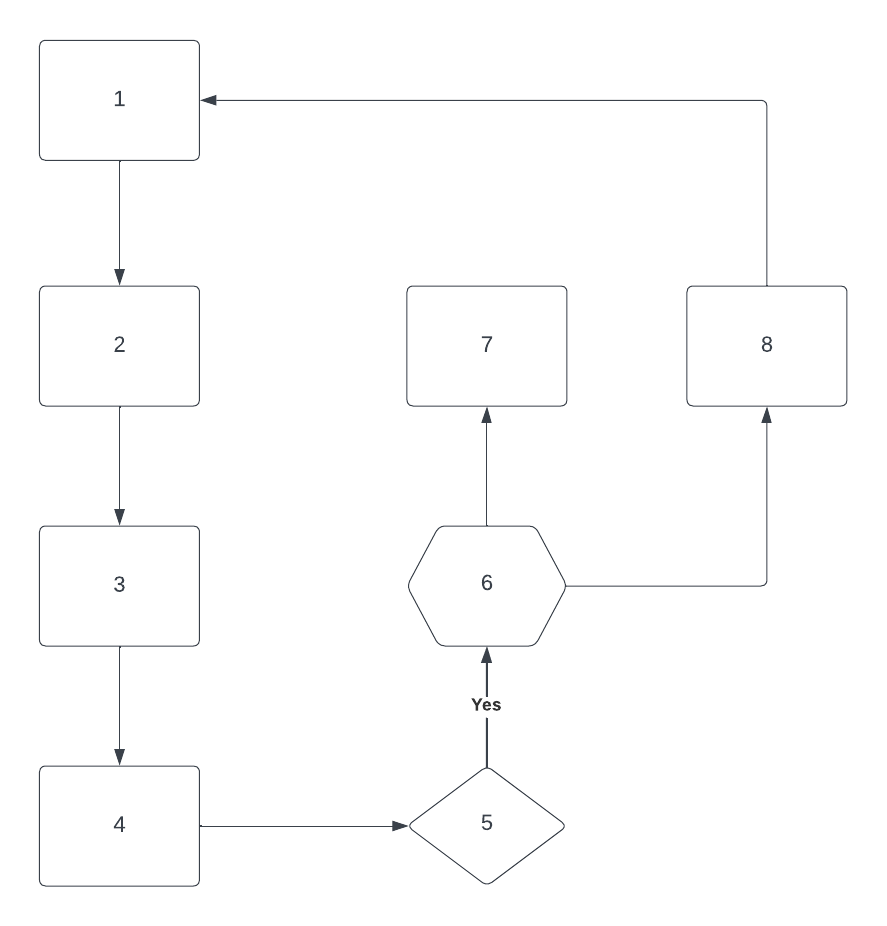
#### Itens de Avaliação

|  |
| --- |
| Estrutura do software, Formato de integração dos componentes do código |

#### Estratégia de apresentação

|  |
| --- |
| Rápida e objetiva, buscando demonstrar o produto e suas funcionalidades |

#### Arquitetura do Sistema



1. Inicialização do Allegro e geração do bitmap do menu principal

2. Os bitmap’s do fundo são gerados

3. Ao clicar no botão “start” o bitmap do player é inicializado ao mesmo tempo a função que habilita as letras que o player deve digitar se inicia.

4.Os bitmap’s dos canos são iniciados.

5. Seis objetos canos são criados em loop em posições aleatórias do eixo y para serem obstáculos do player

6. Casos possíveis: O player não colide (7)

O player colide (8).

7. O player continua o jogo até ele colidir com algum cano.

8.O player colide com algum cano, aparece o botão “ Restart ” e ao aperta-lo

### Funcionalidades do produto

|  |  |
| --- | --- |
| Número | Funcionalidade do sistema |
|  | Pulo do player com alfabeto |
|  | Pulo do player com espaço |
|  | Sistema de colisão |
|  | Sistema de pontuação |
|  | Sistema de restart |

### Documentação do código

#### Documentação da Estrutura de dados geral do software

Classe: Backgraund

Descrição: Responsável pela criação do Background.

Variáveis utilizadas:

float x;

float y;

float velX;

float velY;

int dirX;

int dirY;

int width;

int height;

*ALLEGRO\_BITMAP* \*image;

Classe: GameObject.

Descrição: Obter o caminho dos objetos.

Variáveis: *Position* \_position = *Position*(0, 0);

const char \*path;

Classe: Imputs.

Descrição: Responsável por analisar os imputs.

Classe:Player.

Discrição: Responsável pela criação do player.

Variáveis: *ALLEGRO\_BITMAP* \*image;

float velocidadeAtual = 0;

float gravidade = 9.8;

float velocidade;

*KeyCode* jumpKeyCode = 1;

Classe: Pointer.

Descrição: Responsável pela criação do Ponteiro

Variáveis: int qPointer;

*ALLEGRO\_COLOR* letterColor;

*ALLEGRO\_FONT* \*font = al\_create\_builtin\_font();

const char \*text;

Classe: StartMenu.

Descrição: Responsável pela criação dos menus.

Variáveis: float x;

float y;

int width;

int height;

bool pressed = false;

bool iniciar = false;

*ALLEGRO\_BITMAP* \*image;

Classe: Tube.

Descrição: Responsável pela criação do obstáculo.

Variáveis;

*ALLEGRO\_BITMAP* \*image;

bool passou = false;

Struct: Position.

Descrição: Settar as posições X e Y dos objetos.

Variáveis: float eixoX;

float eixoY;

#### Função <void update()>

Descrição: Função para atualizar a posição da imagem de acordo com a velocidade e a direção.

Parâmetros: nenhum

Entrada: nenhum

Saída: nenhum.

***Função < void draw()>***

Descrição: Responsável pela exibição do Background

Parâmetros: nenhum.

Entradas: nenhum.

Saídas: nenhum.

***Função <getMousePosition() >***

Descrição: Responsável por pegar a posição do mouse.

Parâmetros: nenhum.

Entradas: nenhum.

Saídas: Position () posição do mouse.

#### ***Função <*** ***void setMousePosition()>***

Descrição: Função para definir a posição do mause.

Entradas: nenhuma.

Parâmetros: int x: valor do eixo X do mouse

int y: valor do eixo Y do mouse

Saida: nenhuma

***Função <*** checkLeftClick() ***>***

Descrição: Função para conferir se o botão esquerdo do mouse foi clicado

Entradas: nenhum

Parâmetros:nenhum

Saídas: true: o botão esquerdo foi clicado

false: o botão esquerdo não foi clicado

#### ***Função <checkRightclick>***

Descrição: Função para conferir se o botão direito do mouse foi clicado

Entradas:

Parâmetros:

Saídas: true: o botão direito foi clicado

false: o botão direito não foi clicado

#### ***Função <checkMidlleClick>***

Descrição: Função para conferir se o botão do meio do mouse foi clicado

Entradas: nenhum.

Parâmetros: nenhum.

Saídas: true: o botão do meio foi clicado

false: o botão do meio não foi clicado

***Função <*** static bool checkKeyDown(KeyCode keyCode) ***>***

Descrição: Função para conferir se uma determinada tecla foi apertada

Entradas: nenhum.

Parâmetros: KeyCode keyCode: valor da tecla que deseja conferir

Saídas: nenhum.

***Função <*** static bool checkKeyUp(KeyCode keyCode) ***>***

Descrição: Função para conferir se uma determinada tecla foi solta

Entradas:

Parâmetros: KeyCode keyCode: valor da tecla que deseja conferir

Saídas: true: a tecla foi solta

false: a tecla não foi solta

***Função <*** static bool checkKeyIsPressing(KeyCode keyCode) ***>***

Descrição: Função para conferir se uma determinada tecla esta pressionada

Entradas:

Parâmetros: KeyCode keyCode: valor da tecla que deseja conferir

Saídas: true: a tecla está pressionada

false: a tecla não está pressionada

***Função <*** static void init() ***>***

Descrição: Função para preencher todo o vetor keys

Entradas: nenhuma.

Parâmetros: nenhuma.

Saídas: nenhuma.

***Função <*** static void update(ALLEGRO\_EVENT event) ***>***

Descrição: Função para atualizar o valor do input de acordo com os eventos pree definidos

Entradas: nenhuma.

Parâmetros: ALLEGRO\_EVENT event evento a ser verificado

Saídas: nenhuma.

***Função <***void draw() ***>***

Descrição: responsavel pela exibição do player na tela

Entradas: nenhuma.

Parâmetros: nenhuma.

Saídas: nenhuma

***Função <*** void move() ***>***

Descrição: responsavel por aplicar a fisica e simular a movimentação do player

Entradas: nenhum.

Parâmetros: nenhum

Saídas: nenhum.

***Função <*** void update() ***>***

Descrição: responsavel pela atualização da posição do player

Entradas: nenhum.

Parâmetros: nenhum.

Saídas: nenhum.

***Função <*** Player() ***>***

Descrição: responsavel pela destruição do bitmap

Entradas: nenhuma.

Parâmetros: nenhuma.

Saídas: nenhuma.

***Função <*** void setQPointer(int q) ***>***

Descrição: responsavel por setar o valor dos pontos

Entradas: nenhuma

Parâmetros: nenhuma.

Saídas: nenhuma.

***Função <*** void draw() ***>***

Descrição: responsavel pela exibição da pontuação

Entradas: nenhuma,

Parâmetros: nenhum.

Saídas: nenhuma.

***Função <*** ~Pointer() ***>***

Descrição: responsavel pela destruição do bitmap

Entradas: nenhuma,

Parâmetros: nenhum.

Saídas: nenhuma.

***Função <*** changePressState(bool *pressed*) ***>***

Descrição: responsavel pela alteração dos valores de pressed e iniciar

Entradas: nenhuma.

Parâmetros: bool pressed :valor de pressed e de iniciar

Saídas: nenhuma.

***Função <*** checkBtnIsPress() ***>***

Descrição: responsavel por coletar o valor de pressed

Entradas: nenhuma

Parâmetros: nenhum.

Saídas: this->pressed;

***Função <*** getStart() ***>***

Descrição: responsavel por coletar o valor de iniciar

Entradas: nenhuma.

Parâmetros: nenhum.

Saídas: this->iniciar;

***Função <*** void update() ***>***

Descrição: responsavel pela atualização da exibição do menu

Entradas: nehuma.

Parâmetros: nenhu.

Saídas: nenhuma.

***Função <*** void draw() ***>***

Descrição: responsavel pela exibição do menu na tela

Entradas: nenhuma.

Parâmetros: nenhum.

Saídas: nenhuma.

***Função <*** ~StartMenu() ***>***

Descrição: responsavel pela destruição do bitmap

Entradas: nenhuma.

Parâmetros: nenhum.

Saídas: nenhuma.

***Função <*** void move() ***>***

Descrição: responsavel por aplicar a fisica e simular a movimentação do tubo

Entradas: nenhuma.

Parâmetros: nenhum.

Saídas: nenhuma.

***Função <*** collision(Player \*InitialPlayer) ***>***

Descrição: responsável por conferir se o jogador colidiu com o tubo

Entradas: nenhuma.

Parâmetros: Player \*InitialPlayer

Saídas: boleam

***Função <*** pointer(Player \*InitialPlayer) ***>***

Descrição: responsavel por conferir se o jogador marcou ponto

Entradas: nenhuma.

Parâmetros: Player

Saídas: boleam

***Função <***~Tube()***>***

Descrição: responsavel por destroir o bitmap do tubo

Entradas: nenhuma.

Parâmetros: nenhum

Saídas: nenhuma.

***Função <*** setSpeed()***>***

Descrição: responsavel por settar a velocidade que o timer ira correr

Entradas: nenhuma.

Parâmetros: nenhum.

Saídas: speed: velocidade do timer

***Função <*** void update(const vector<GameObject \*> &objectVector) ***>***

Descrição: responsavel pela atualização dos GameObject

Entradas: nenhuma.

Parâmetros: const vector<GameObject \*> &objectVector: vetor de GameObject

Saídas: nenhuma

***Função <*** void destroyTube(const vector<Tube \*> &tubeVector) ***>***

Descrição: responsavel pela destruição dos bitmaps do tubes

Entradas: nenhuma.

Parâmetros: const vector<Tube \*> &tubeVector: vetor de Tube

Saídas: nenhuma.

***Função <*** void destroy(const vector<GameObject \*> &objectVector) ***>***

Descrição: responsavel pela destruição dos bitmaps do GameObject

Entradas: nenhuma.

Parâmetros: const vector<GameObject \*> &objectVector: vetor de GameObject

Saídas: nenhuma.

***Função <*** checkCollision(const vector<Tube \*> &tubeVector, Player \*InitialPlayer) ***>***

Descrição: responsavel por fazer a checagem se o player colidiu com algum tubo ou se passou

Entradas:

Parâmetros: const vector<Tube \*> &tubeVector : vetor de Tube

Player \*InitialPlayer: player

Saídas: caso true o jogar colidiu caso false fez um ponto

***Função <*** void drawTube(const vector<Tube \*> &tubeVector) ***>***

Descrição: responsavel pela exibição dos tubos na tela

Entradas: nenhuma.

Parâmetros: const vector<Tube \*> &tubeVector : vetor de Tube

Saídas: nenhuma.

***Função <*** void mustInit(bool isWorking, string errorMesage) ***>***

Descrição: responsavel por conferir se as instancias estão funcioando corretamente

Entradas: nenhuma.

Parâmetros: bool isWorking: retorno da função

string errorMesage: mensagem de erro a ser exibida

Saídas: nenhuma.

***Função <*** void getTubes(vector<Tube \*> &TubeVector, const char \*imagePath[]) ***>***

Descrição: responsavel por popular o vector<Tube>

Entradas: nenhuma

Parâmetros: vector<Tube \*> &TubeVector: vetor a ser populado

const char \*imagePath[]: endereço das imagens

Saídas: nenhuma.

### Testes do sistema

Casos de testes do sistema: função < bool collision(*Player* \**InitialPlayer*)>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Número* | *Varáveis de Entrada* | *Valores válidos* | *Resultado Esperado* | *Valores inválidos* | *Resultado Esperado* |
| *1* | **Valores do tipo int do**  **eixo x e y do player que sobrepõem os valores x e y do objeto cano** | **Valores do tipo int, maiores ou iguais a 0** | **Colisão ocorreu, o botão de restart irá aparecer** | **Valores de outro tipo(char ,floot ,bool)** | **Colisão não ocorreu, o jogo continua** |
| *2* | **Valores do tipo int do**  **eixo x e y do player que não sobrepõem os valores x e y do objeto cano** | **Valores do tipo int, maiores ou iguais a 0** | **Colisão não ocorreu, o jogo continua** | **Valores de outro tipo(char ,floot ,bool)** | **Colisão não ocorreu, o jogo continua** |
| *3* |  |  |  |  |  |
| *4* |  |  |  |  |  |
| 5 |  |  |  |  |  |