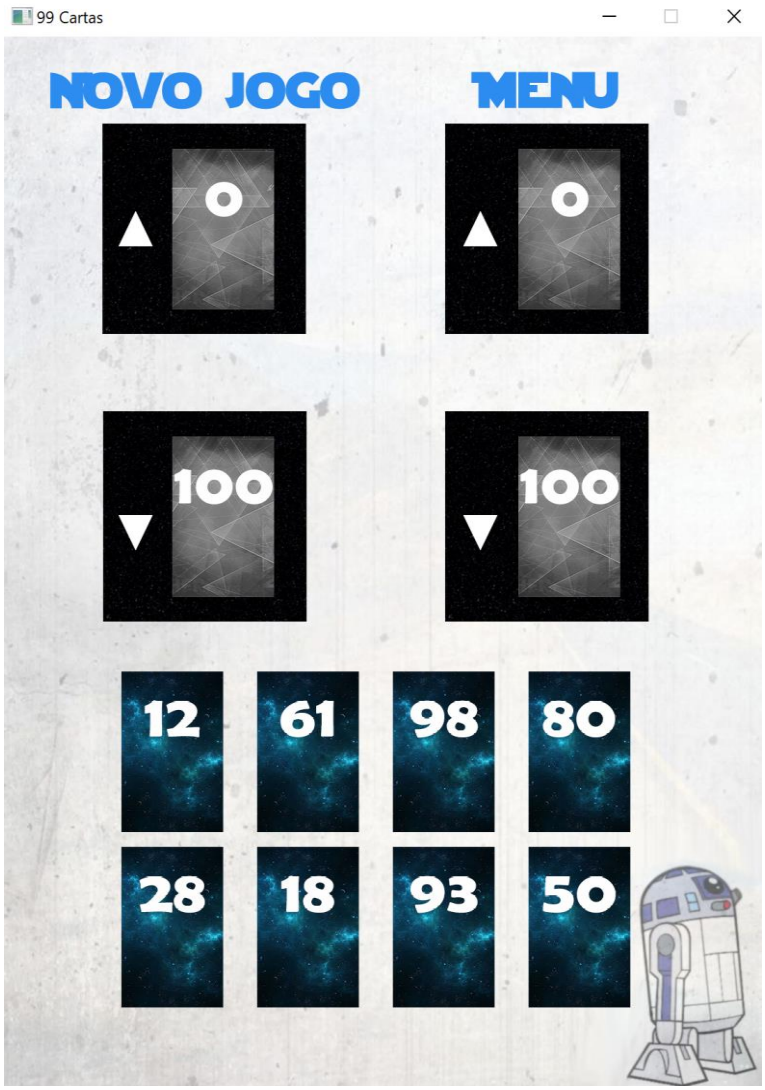
Universidade Federal de São Carlos
Departamento de Computação
Estrutura de Dados

Documentação Trabalho 1

Renata Sarmet Smiderle Mendes, 726586, renatassmendes@hotmail.com.
Rodrigo Pesse de Abreu, 726588, abreu.rodrigo97@gmail.com.
Rafael Bastos Saito, 726580, rafaelbsaito@hotmail.com.

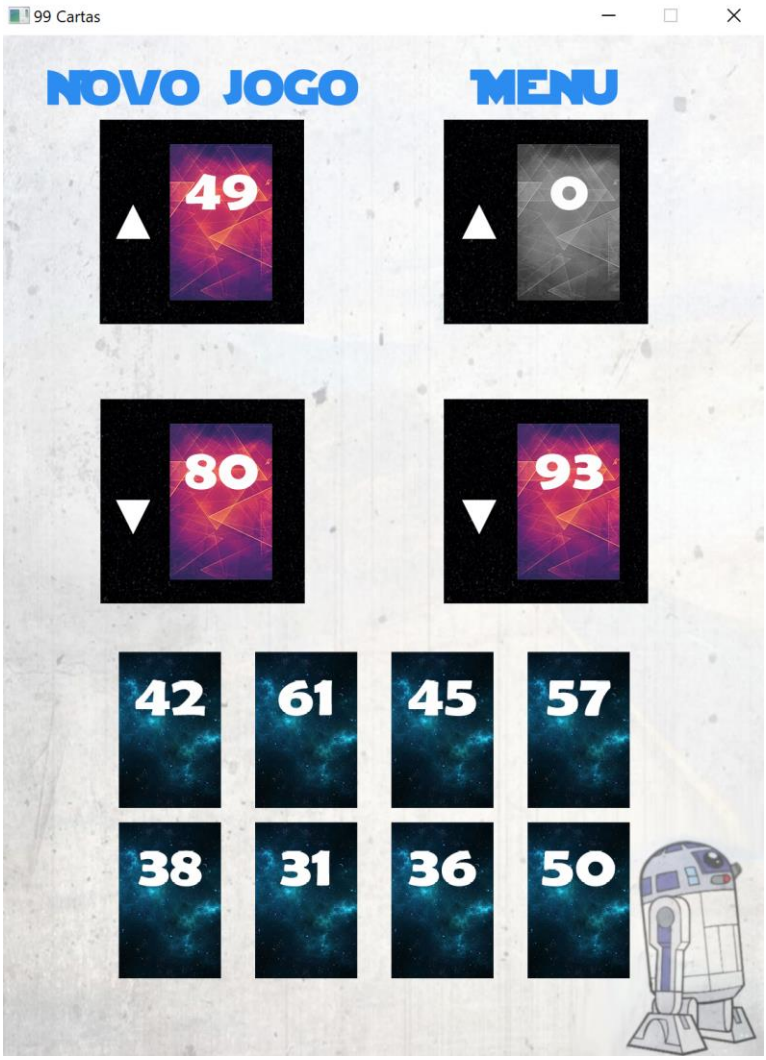
1. Funcionamento do jogo

O jogo 99 Cards, consiste em quatro pilhas diferentes, uma fila (monte de cartas) e uma lista (mesa de cartas). Nas quais, as pilhas, são distribuídas de forma: duas em ordem crescente e duas em ordem decrescente, ambas com capacidade para 99 cartas, sendo o usuário a implementá-las de acordo com suas possibilidades disponibilizadas pela mesa.



Tela inicial do jogo

Já a fila, que atuará de forma a abastecer a mesa, é inicializada pelo próprio programa de forma aleatória, com valores de 1 a 99, e é distribuída em 8 valores na mesa, dando possibilidades para o usuário empilhar em uma das 4 pilhas. Conforme adicionado duas cartas da mesa para alguma das quatro pilhas, automaticamente a mesa será recomposta novamente para 8 cartas.



Jogo em andamento

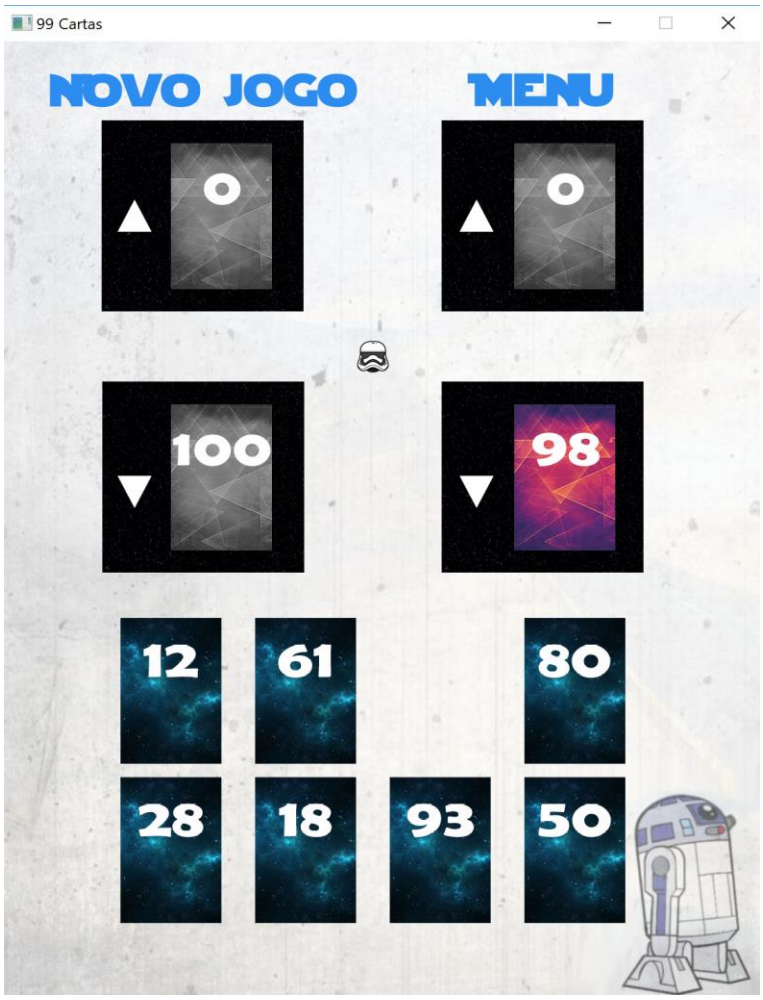
Essa quantidade restrita de cartas na mesa é o clímax da dificuldade do jogo, pois, ao percorrer do mesmo, as possibilidades de transição entre a mesa e as pilhas vão restringindo-se.

Uma válvula de escape, para o aumento do escopo de possibilidades de empilhamento, consiste no recurso de poder empilhar um valor dez números à frente, no caso da pilha decrescente, e dez números a menos, no caso da pilha crescente.

Exemplo:

Caso a pilha seja crescente, esteja com o valor 30 em seu topo e haja o número 20 na mesa, abre-se a possibilidade desse ser empilhado. Tal jogada aumentará a gama de possibilidades de empilhamento ao longo do jogo.

Outra ferramenta disponibilizada para auxiliar o usuário é a opção de desfazer a última jogada. Tal opção, só poderá ser acessada quando existirem sete cartas na mesa ou quando estiver nas últimas sete cartas do jogo. A implementação dessa ferramenta tem o intuito de ajudar o usuário caso tenha empilhado errado ou pensado em outra jogada melhor posteriormente.

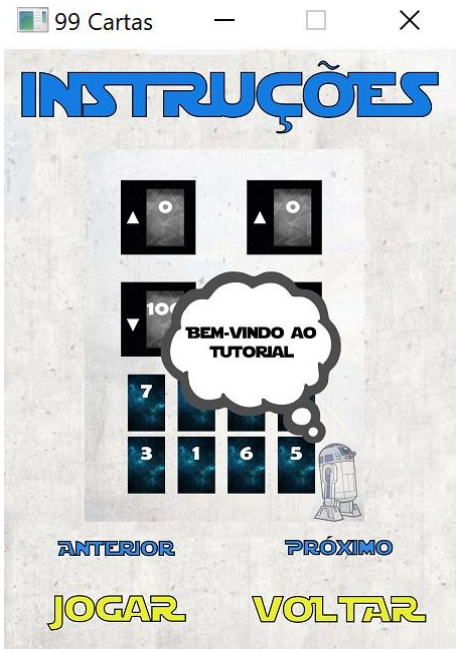


O objetivo do jogo consiste em empilhar todas as cartas da fila nas pilhas. Mediante as alocações ao longo do jogo, a probabilidade de o escopo de jogadas esgotar é grande, resultando na derrota do jogo. Em contrapartida, caso as jogadas forem selecionadas de forma correta, a fila e a mesa esgotar-se-ão e a vitória será contemplada.

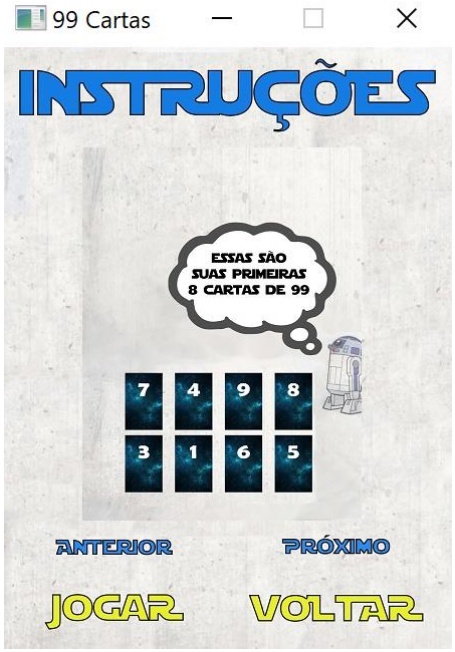
2. Telas do jogo

Tela Inicial

Menu

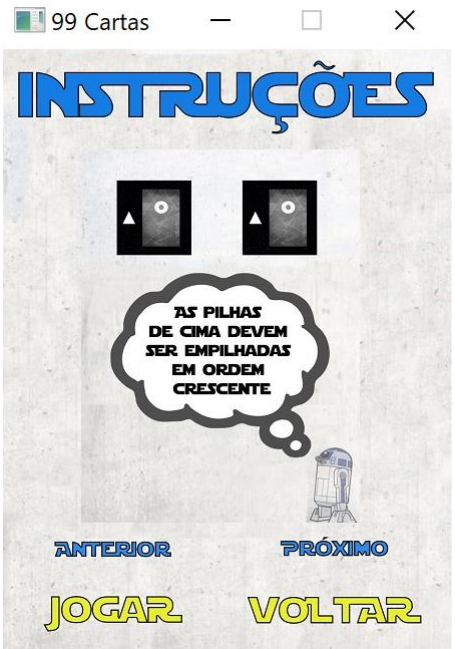


Instruções parte1

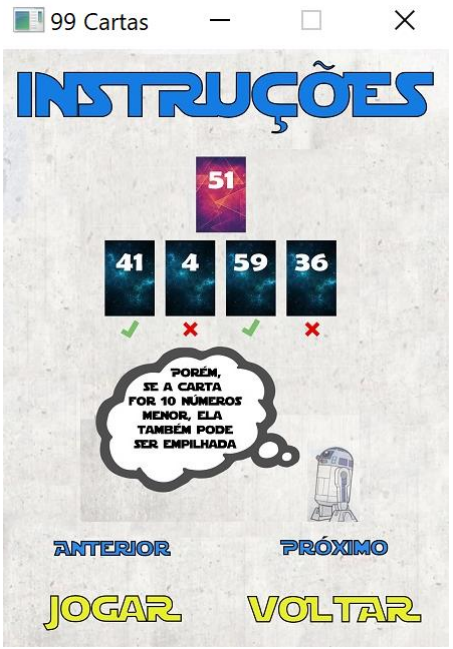


Instruções parte 2

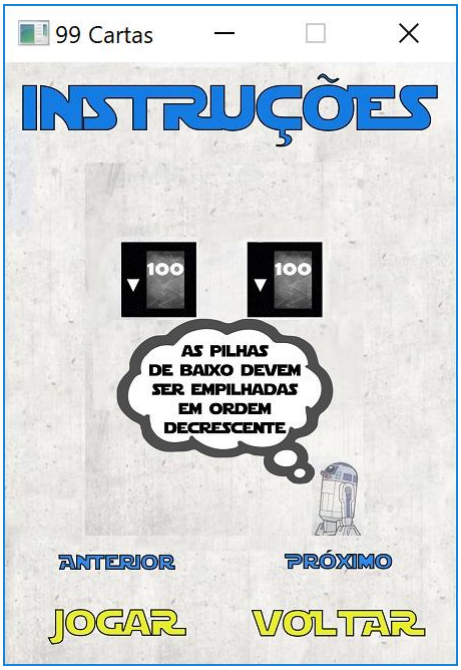
Instruções parte 3



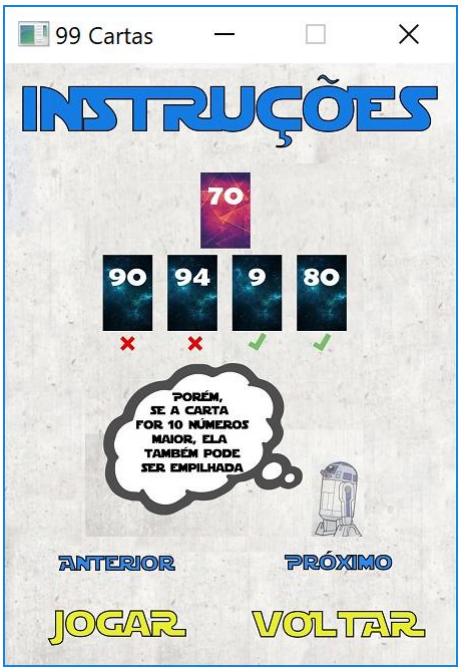
Instruções parte 4



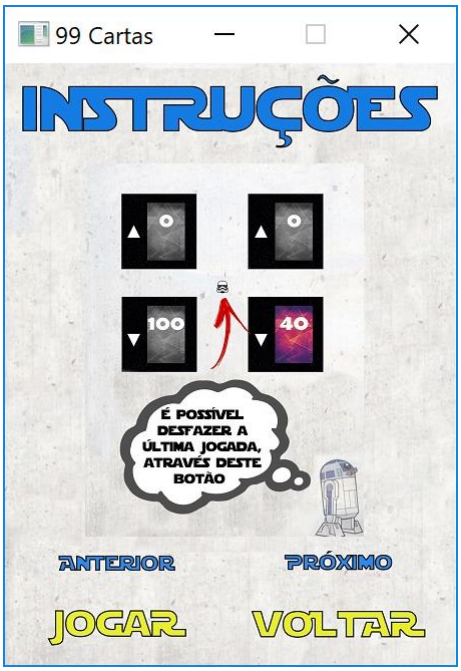
Instruções parte 5



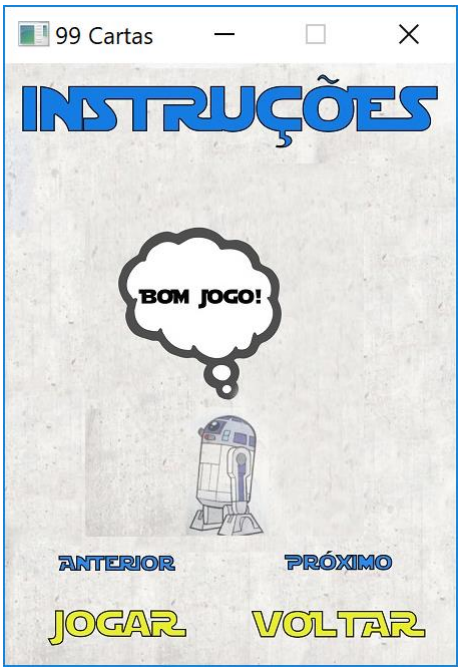
Instruções parte 6



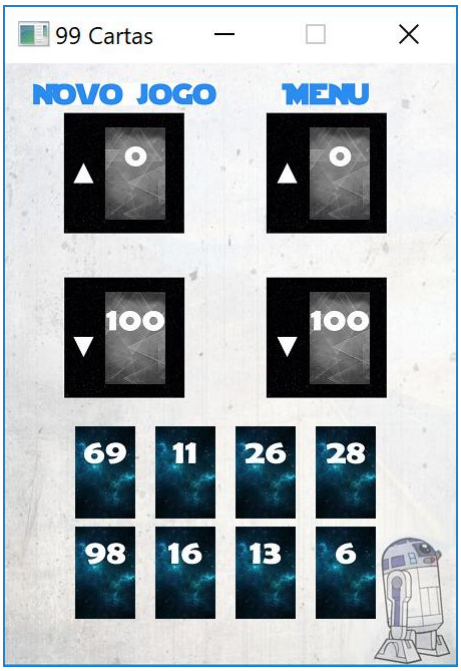
Instruções parte 7



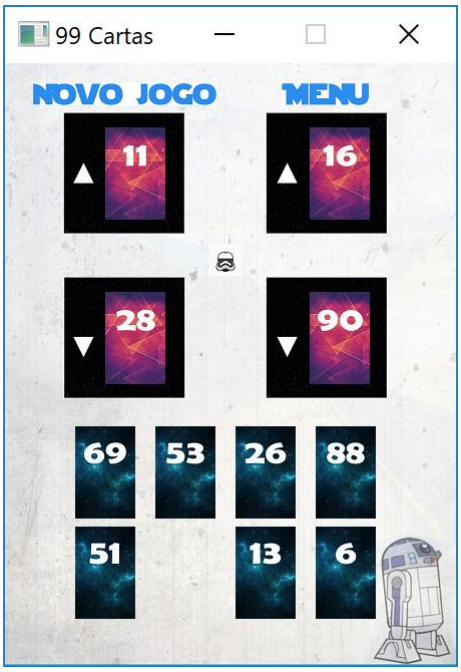
Instruções parte 8



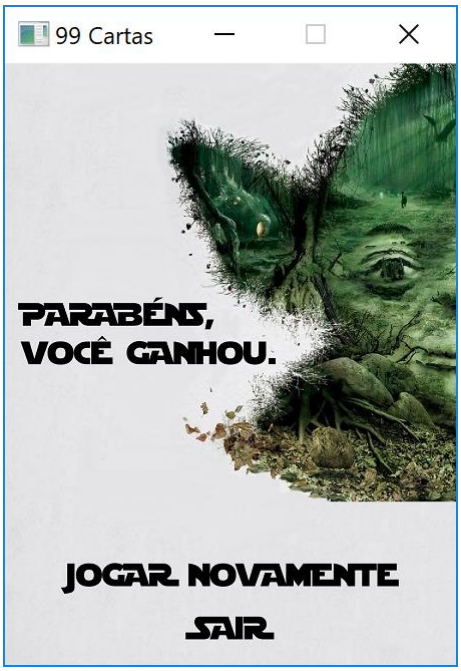
Instruções parte 9



Situação inicial de jogo



Situação de jogo



Ganhou

Perdeu

3. Estruturação do código

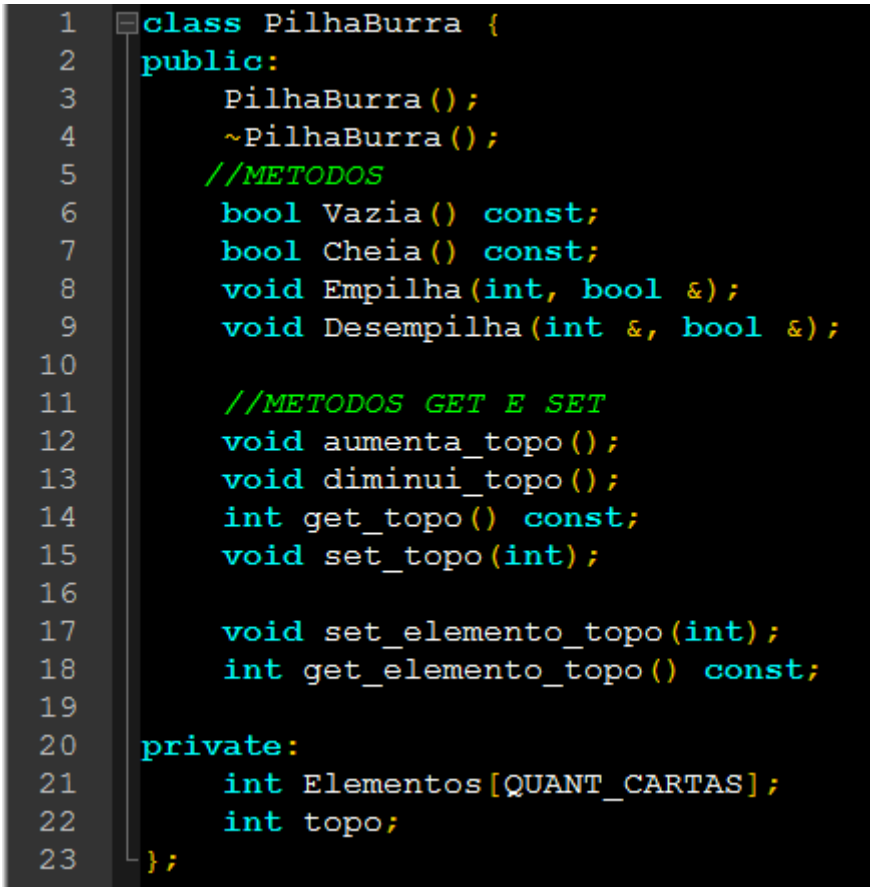
A estruturação do código foi formulada por meio de tópicos: Pilha, Fila, Lista e parte gráfica.

As pilhas foram implementadas de forma geral inicialmente, por meio de uma Pilha Burra, com todos os métodos gerais necessários (como Cria, Vazia, Cheia, Empilha, Desempilha) que uma Pilha normal possui.

Em seguida, foram feitas as ramificações entre Pilha Crescente e Pilha Decrescente, as quais herdam de Pilha Burra, além de modificarem as condições que permitem ou não empilhar as cartas.

O código foi implementado em C++, organizado através de classes.

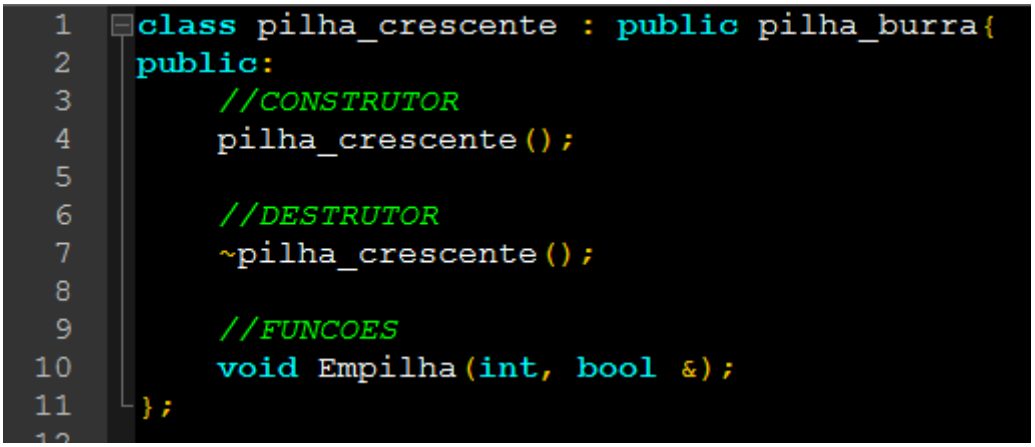
A Classe Pilha Burra é composta por métodos genéricos para a manipulação de uma pilha comum.



```
class PilhaBurra {
public:
    PilhaBurra();
    ~PilhaBurra();
   //METODOS
    bool Vazia() const;
    bool Cheia() const;
    void Empilha(int, bool &);
    void Desempilha(int &, bool &);

    //METODOS GET E SET
    void aumenta_topo();
    void diminui_topo();
    int get_topo() const;
    void set_topo(int);

    void set_elemento_topo(int);
    int get_elemento_topo() const;

private:
    int Elementos[QUANT_CARTAS];
    int topo;
};
```

Código Pilha Burra

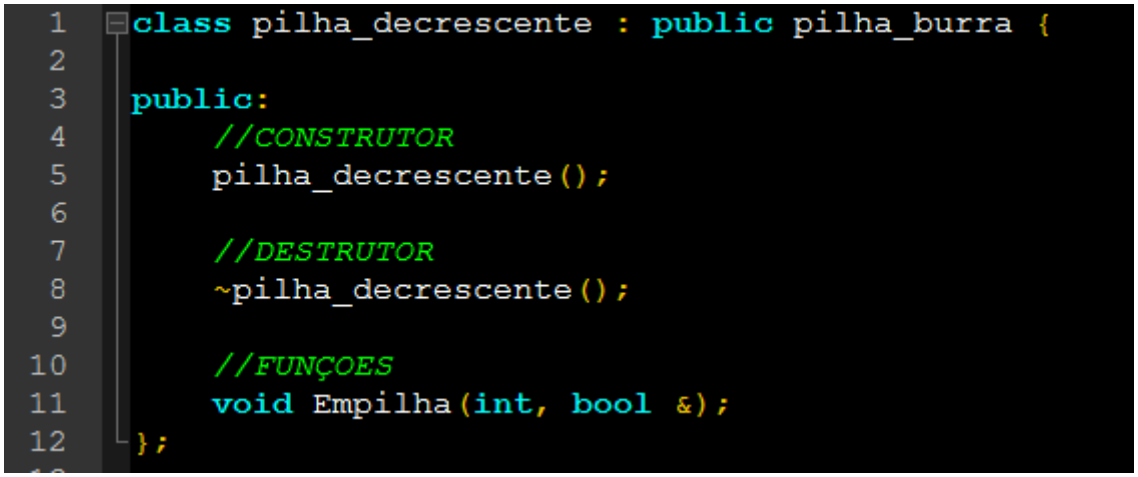
A Classe Pilha Crescente é subclasse de Pilha Burra e altera apenas o método Empilha, a fim de moldá-lo de acordo com as especificações do jogo, isto é, aceitar cartas maiores que o topo ou a carta 10 números a menos.



```
class pilha_crescente : public pilha_burra{
public:
    //CONSTRUTOR
    pilha_crescente();

    //DESTRUTOR
    ~pilha_crescente();

    //FUNCOES
    void Empilha(int, bool &);
};
```

Código Pilha Crescente

Da mesma forma, a Classe Pilha Decrescente é subclasse de Pilha Burra e apenas altera o método Empilha de tal forma que aceita cartas menores que o topo ou a carta 10 números a mais.



```
class pilha_decrescente : public pilha_burra {

public:
    //CONSTRUTOR
    pilha_decrescente();

    //DESTRUTOR
    ~pilha_decrescente();

    //FUNÇOES
    void Empilha(int, bool &);
};
```

Código Pilha Decrescente

A Classe FilaMonte é uma aplicação de Fila que tem como objetivo compor o Monte, que possui cartas de 1 a 99 em ordem aleatória, o qual irá abastecer a Mesa, de acordo com as Distribuições durante o jogo.

Para isso, é importante 2 condições serem verificadas: Todas as cartas (de 1 a 99) estarem na fila e nenhuma aparecer mais de uma vez, isto é, não pode haver repetição de cartas.

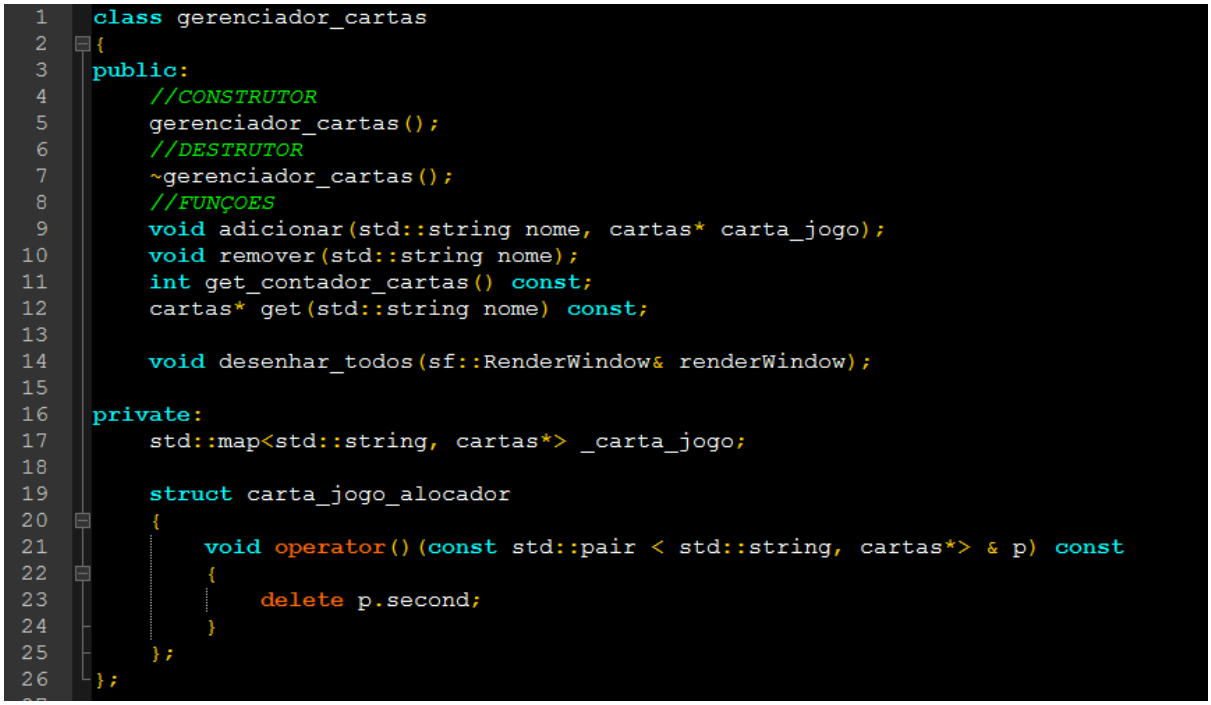
Para que isso ocorra, o código de sua implementação possui várias peculiaridades a serem consideradas.

A princípio, a fila é inicializada com valores de 1 a 99 em ordem crescente.

Em seguida, é chamada uma função Embaralhar, a qual percorre toda a fila e altera ou não as posições de forma aleatória, resultando em uma Fila toda embaralhada.

```
class fila_monte
{
public:

    //CONSTRUTOR
    fila_monte();

    //DESTRUTOR
    ~fila_monte();

    //METODOS
    bool Vazia() const;
    bool Cheia() const;
    void Insere(int X, bool &DeuCerto);
    void Retira(int &X, bool &DeuCerto);
    void InicializaElementos();
    void EmbaralhaElementos();


    //METODOS GET E SET
    void proximo_primeiro();
    int get_primeiro() const;
    void proximo_ultimo();
    int get_ultimo() const;
    void set_primeiro(int);
    int get_NroElementos() const;
    void diminui_NroElementos();
    void aumenta_NroElementos();
    void set_elemento_ultimo(int);
    int get_elemento_ultimo() const;
    int get_elemento_primeiro() const;
```

Código Fila Monte - parte 1



```
class gerenciador_cartas
{
public:
    //CONSTRUTOR
    gerenciador_cartas();
    //DESTRUTOR
    ~gerenciador_cartas();
    //FUNÇOES
    void adicionar(std::string nome, cartas* carta_jogo);
    void remover(std::string nome);
    int get_contador_cartas() const;
    cartas* get(std::string nome) const;

    void desenhar_todos(sf::RenderWindow& renderWindow);

private:
    std::map<std::string, cartas*> _carta_jogo;

    struct carta_jogo_alocador
    {
        void operator()(const std::pair < std::string, cartas*> & p) const
        {
            delete p.second;
        }
    };
};
```

Código Fila Monte - parte 2

A Classe Mesa é uma aplicação de Lista, a qual exibe as cartas disponíveis para o jogador empilhar. Essa Mesa é alimentada pela FilaMonte, através de métodos de Distribuição, e tem a função de abastecer as Pilhas ao longo do jogo.

Ela é implementada como um vetor de Cartas com capacidade máxima igual a 8 e possui todos os métodos básicos para manipulação (como Vazia, Cheia, Insere, Remove, Gets e Sets), além de duas diferentes Distribuições: PrimeiraDistribuicao e NovaDistribuicao.

A PrimeiraDistribuicao insere 8 cartas na mesa e é chamada apenas ao iniciar o jogo. A NovaDistribuicao insere 2 cartas e é chamada sempre que a mesa fica com 6 cartas, até que não haja mais cartas na fila para abastecê-la.

A Mesa também está ligada com o método que desfaz a última jogada, pois este é disponibilizado ou não de acordo com o número de elementos presente nesta.

Quando o número de elementos da Mesa e da fila chegam ambos a zero, o jogador venceu o jogo. Quando ainda há cartas disponíveis na Mesa porém essas já não fornecem nenhuma possibilidade de jogada, o jogador perdeu o jogo.

```cpp
class mesa
{
public:

    //CONSTRUTOR
    mesa();

    //DESTRUTOR
    ~mesa();

    //METODOS
    bool Vazia() const;
    bool Cheia() const;
    bool insereMesa(int X, bool &DeuCerto);
    void removeMesa(sf::RenderWindow& window, int i);
    void LimpaMesa();
    void distribuicao(fila_monte &f, int quantidade);
    void NovaDistribuicao(fila_monte &f, bool &DeuCerto);
    void PrimeiraDistribuicao(fila_monte &f, bool &DeuCerto);

    //METODOS GET E SET
    int get_NroElementos() const;
    void set_NroElementos(int);
    void diminui_NroElementos();
    void aumenta_NroElementos();
    int get_elemento_i(int i, bool &DeuCerto) const;
    void set_elemento_i(int i, int X, bool &DeuCerto);

    virtual void set_posicao(float x, float y);
```

Código Mesa - parte 1

```cpp

    virtual float get_altura() const;
    virtual float get_largura() const;

    void adicionar_mesa(std::string nome, cartas* carta_mesa);

    virtual sf::Rect<float> get_bounding_rect();

    cartas *teste[TAMANHO_MESA];

private:
    int Elementos[TAMANHO_MESA];
    int NroElementos;
    std::map<std::string, cartas*> _carta_mesa;
    sf::Sprite _sprite;
    sf::Texture _imagem;
    std::string nome_arquivo;
    bool carregou;

    struct carta_mesa_alocador
    {
        void operator()(const std::pair < std::string, cartas*> & p) const
        {
            delete p.second;
        }
    };
};
```

Código Mesa - parte 2

Para desenvolver a parte gráfica do jogo, foi utilizada a ferramenta SFML (Simple and Fast Multimedia Library), que é uma biblioteca orientada a objetos, multiplataforma e livre.

A integração entre a parte lógica e a parte gráfica aconteceu graças à criação de algumas classes:

A Classe Tela Inicial simplesmente cria uma nova tela e define o background da mesma.

```
class Tela_Inicial
{
    public:
        void Mostrar(sf::RenderWindow& window);
};
};
```

Código tela inicial

A Classe Menu é responsável pelo menu do jogo, desde a criação de uma nova tela até obter a resposta do mouse para saber qual opção o jogador escolheu.

```
class Menu
{
public:
    enum menu_inicial {Nada, Sair, Jogar, Instrucoes};

    struct Item_Menu
    {
    public:
        sf::Rect<int> rect;
        menu_inicial action;
    };

    menu_inicial Mostrar(sf::RenderWindow& window);

private:
    menu_inicial obter_resposta_menu(sf::RenderWindow& window);
    menu_inicial clique(int x, int y);
    std::list<Item_Menu> itens_menu;
};
```

Código Menu

A Classe Cartas cria e controla (definindo sua posição, background e tamanho) todas as cartas do nosso jogo, sejam elas pertencentes à mesa ou às Pilhas Crescente e Decrescente.

```
class cartas
{
public:
    //CONSTRUTOR
    cartas();

    //DESTRUTOR
    virtual ~cartas();

    //FUNCOES
    virtual void carregar(std::string nome_arquivo);
    virtual void desenhar(sf::RenderWindow& window);

    virtual void set_posicao(float x, float y);
    virtual float get_x() const;
    virtual float get_y() const;

    virtual void set_origem(float x, float y);
    virtual float get_origem_x() const;
    virtual float get_origem_y() const;

    virtual float get_altura() const;
    virtual float get_largura() const;

    virtual sf::Rect<float> get_bounding_rect();

    bool selecionado(sf::RenderWindow& window);
    bool pressionado(sf::RenderWindow& window);

    bool mouse_intersects(sf::RenderWindow& window);
```

Código Cartas - parte 1

```
    bool mouse_intersects(sf::RenderWindow& window);

    sf::Sprite _sprite;

private:

    sf::Texture _imagem;
    std::string nome_arquivo;
    int valor;
    bool carregou;
    bool estado;


};
```

Código Cartas - parte 2

A Classe Gerenciador de Cartas, como o próprio nome já diz, gerencia as cartas, ou seja, é uma classe que facilita a operação de desenhar as cartas na tela. Isso porque nesta classe existe a função adicionar, que adiciona determinada carta a um mapa, e graças à função desenhar_todos é possível desenhar na tela todas as cartas que foram adicionadas a esse mapa.

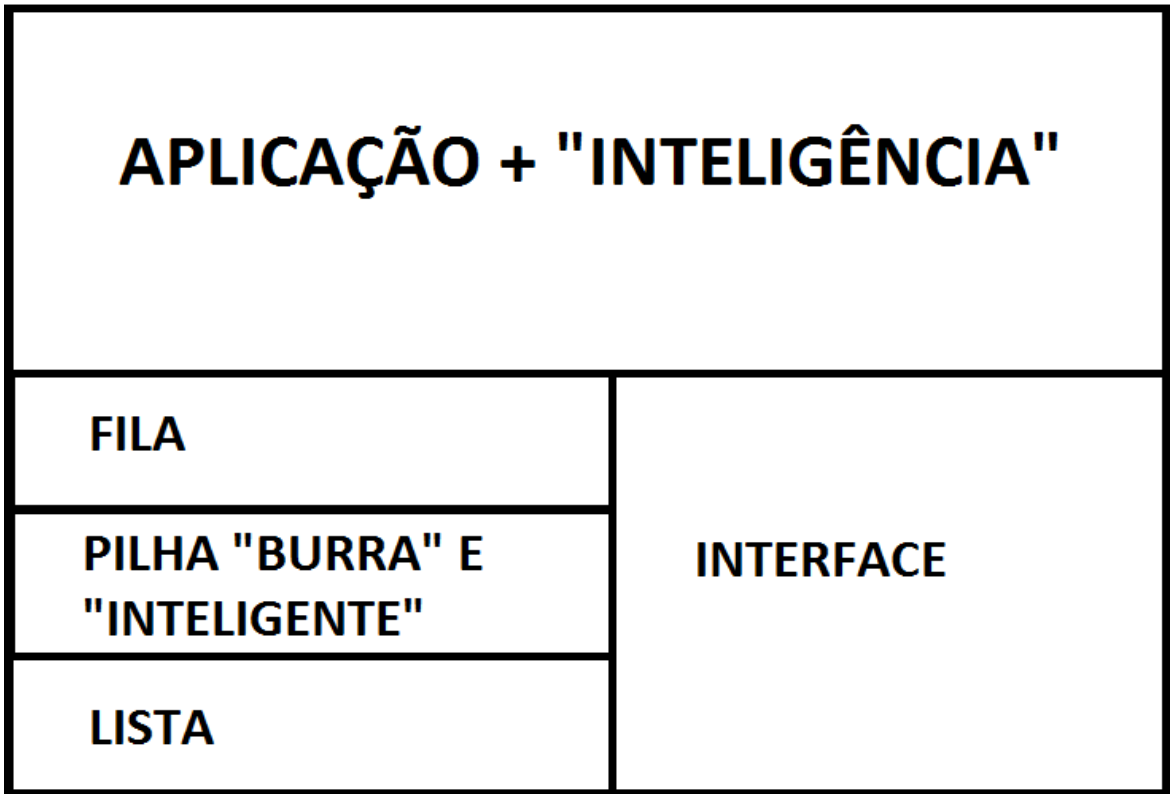
```
class gerenciador_cartas
{
public:
    //CONSTRUTOR
    gerenciador_cartas();
    //DESTRUTOR
    ~gerenciador_cartas();
    //FUNÇOES
    void adicionar(std::string nome, cartas* carta_jogo);
    void remover(std::string nome);
    int get_contador_cartas() const;
    cartas* get(std::string nome) const;

    void desenhar_todos(sf::RenderWindow& renderWindow);

private:
    std::map<std::string, cartas*> _carta_jogo;

    struct carta_jogo_alocador
    {
        void operator()(const std::pair < std::string, cartas*> & p) const
        {
            delete p.second;
        }
    };
};
```

Código gerenciador cartas

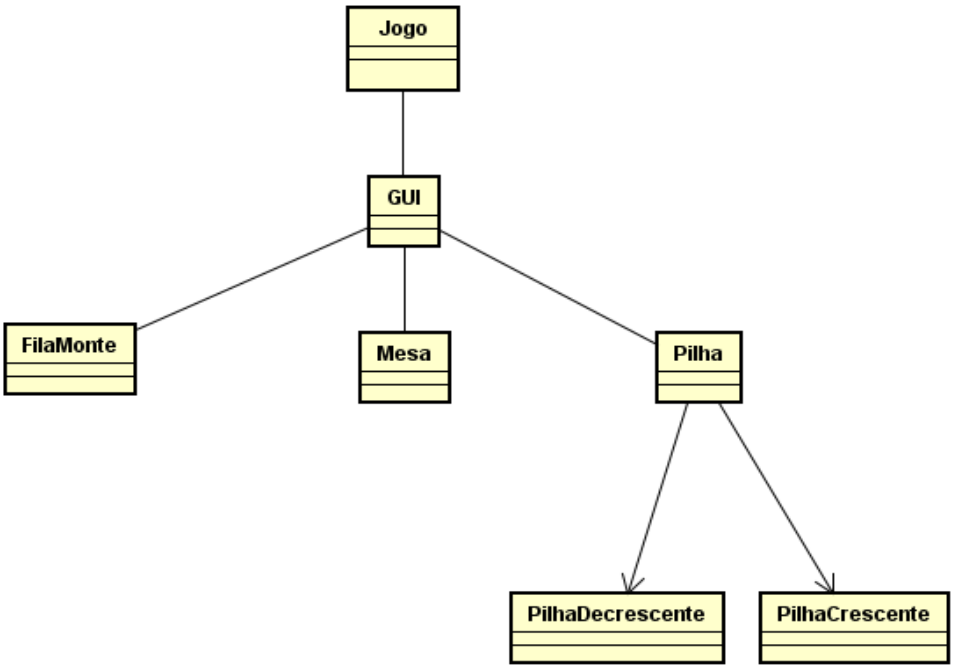
Observação: Para que o executável funcione, ele necessita estar na mesma pasta que os arquivos “.dll” da biblioteca SFML.

4. Diagramas

a. Diagrama da arquitetura do software



b. Diagrama das classes do software



5. Conclusão

O desenvolvimento do jogo teve seu estopim com a divulgação da realização de um trabalho prático na matéria Estrutura de Dados, realizada no Departamento de Computação da Universidade Federal de São Carlos (UFSCar), ministrada pelo professor Roberto Ferrari.

Com a divulgação do trabalho e de suas especificações (utilização do conceito de Fila, Pilha e/ou Lista, lecionados em sala), o grupo, gradativamente, foi juntando ideias, dúvidas e conclusões, até resultar na decisão de implementar o jogo 99 cartas.

O grupo concluiu que o projeto, como um todo, trouxe um grande desenvolvimento pessoal para cada integrante. O desafio da implementação foi muito empolgante para os integrantes, pois, de uma forma prática, utilizou-se os conceitos aprendidos em sala. Já o Design do game, foi a parte mais difícil do projeto, pelo fato do pouco contato com esta área ao longo do curso.

As divisões do projeto deram-se da seguinte forma: O código fonte (sem GUI) foi implementado pela Renata e pelo Rodrigo. A parte gráfica do projeto, na qual linkava o código fonte com a interface foi feita pelo Rafael e pela Renata. A parte de design foi feita pelo Rodrigo e pelo Rafael. A documentação do projeto foi realizada por todos os membros do grupo.