UNIVERSIDADE FEDERAL DE SANTA CATARINA CENTRO JOINVILLE

RELATÓRIO DA CONSTRUÇÃO DE UM CÓDIGO PARA O JOGO “Mastermind”

Disciplina EMB5031: Programação III
Professora: Analize Zomkowski Salvi

Nome: Jéssica Medalha Ferri e Pedro Luiz de Faria da Luz
Matriculas: 15250188 e 16250681

04/07/2019

Sumário

1. INTRODUÇÃO

A partir dos conhecimentos apresentados na disciplina de programação três – bem como a sintaxe da linguagem de programação C++ e suas estruturas de linguagem orientada a objeto -, este documento tem como objetivo relatar os processos envolvidos na implementação do jogo conhecido como *"Mastermind"*. A construção do código foi norteada pela descrição do jogo e pelos requisitos e diretrizes apresentadas, pela professora da disciplina, para a elaboração do jogo. O código que será retratado não apresenta interface gráfica.

2. DESENVOLVIMENTO

2.1. Descrição do jogo e requisitos

Como fornecido na descrição do trabalho, o jogo funciona da seguinte maneira:

> *"Mastermind (também conhecido como senha, em português) é um jogo muito famoso nas décadas de 70 e 80 para ser jogado em 2 pessoas, que ganhou várias versões e variações para ser jogado em computador. A dinâmica do jogo é a seguinte:*
>
> *1. O jogador 1(codemaker) escolhe uma sequencia de cores que o jogador 2 (codebreaker) tem que adivinhar.*
> *2. O codebreaker dá um palpite, escolhendo sua própria sequencia de cores. O objetivo do codebreaker é adivinhar tanto as cores quanto a ordem.*
> *3. A cada palpite, o codemaker retorna uma sequencia com o mesmo número de peças da sequencia a ser descoberta, indicando o quão correto está o palpite dado. Nesta sequencia, uma peça preta indica cor e posição corretos,uma peça branca indica cor correta em posição errada e uma peça cinza indica que ambos cor e posição estão errados.*
> *4. O codebreaker deve usar essa informação para tentar novamente um novo palpite, até ele acerte a sequencia decores na ordem correta ou o número de palpites acabem."*

Os requisitos sugeridos para a elaboração do jogo foram:

> *"Você deve programar o jogo em C++ em estrutura de classes. Não use códigos prontos senão sua nota será zero! Esta parte pode ser feita em duplas. Para quem optar por no máximo 20% nota final, o jogo deve possibilitar:*
>
> *•Escolher o número de cores utilizados no jogo (4-10);*
> *•Escolher o tamanho do código (4 – 6);*
> *•Escolher o número máximo de palpites (4-10);*
> *•Escolher se pode haver cores repetidas;*
> *•que duas pessoas possam jogar uma partida;*
> *•que uma pessoa possa jogar contra o computador com diferentes níveis de dificuldade. A implementação deve conter:*
> *•Classe(s);*
> *•Encapsulamento respeitados por todas as classes;*
> *•Uso de associação;*
> *•Alocação dinâmica de memória;*
> *•Template(s).Uma boa implementação deve ter:*
> *•Heranças e suas derivações;*
> *•Polimorfismo;*
> *•Padrões de projeto."*

2.2 . Estruturas básicas do jogo

2.2.1. Classes e Herança

O jogo é estruturado por herança, onde a classe base *"Tabuleiro"* é herdada pela classe derivada *"Partida"*. Essa divisão foi estabelecida pelo fato de que o tabuleiro é o centro do jogo, pois é lá onde o jogo está registrado e necessariamente o que dá existência ao jogo e suas normas. O tabuleiro, no entanto, varia algumas de suas características de acordo com a partida desejada. Para lidar com essa situação, as especificidades de cada partida são construídas dentro da classe *"Partida"*, que herda as características básicas do tabuleiro e monta a particularidade de cada partida.

2.2.2. Polimorfismo

O polimorfismo aparece no código no método que identifica se as cores digitadas pelo usuário existem no vetor de cores disponível para o jogo. O protótipo do método que realiza essa tarefa é o "virtual bool validaCor(string)".

2.2.3. Template

Foi utilizado o template de classe Vector, da C++ Standard Library, nos atributos *"senha"*, *"cores"* e *"matriz"* da classe *"Tabuleiro"*. Como os atributos citados variam seu tamanho de acordo com cada jogada, o uso do vector possibilita algumas facilidades para o código, pois as bibliotecas disponíveis auxiliam, por exemplo, na alocação dinâmica dos vetores e o fazem com maior segurança.

2.3. Class Tabuleiro

A classe *"Tabuleiro"* contém os principais atributos necessários para o jogo - a senha de cores ("senha"), a matriz do tabuleiro ("matriz") e as cores disponíveis o jogo ("cores") – e seus respectivos métodos de acesso e manipulação. Os atributos e os métodos da classe foram definidos da seguinte forma:

```
1  class Tabuleiro{
2  private:
3    vector<string> cores =
4{"amarelo","verde","azul","vermelho","roxo","laranja","marrom","rosa","lilas","violeta
"};
5    vector<vector<string>> matriz;
6    vector<string> senha;
7    bool vitoria = false;
8  public:
9   ~Tabuleiro(){}
10    vector<string> getCores();
11    string getCor(int );
12    void setSenha(vector<string> );
13    vector<string> getSenha() const;
14    void setVitoria(bool);
25    bool getVitoria();
16    virtual void imprime_Tabuleiro();
17    void adicionaJogadaNaMatriz(vector<string>);
18    vector<string> retornaResposta(vector<string>);
19    vector<string> getUltimaJogadaERespota();
20    vector<vector<string>> getMatriz()const;
21    virtual bool validaCor(string cor);
22 };
```

Figura 1: Class Tabuleiro

O uso do template vector possibilita a declaração do vetor *“senha”* e da matriz *“matriz”* sem definir um tamanho para ambos, pois seus respectivos tamanhos variam com as características de cada partida. A variável *“bool vitoria”, **na linha 7,*** auxilia, em cada jogada, a avaliar se o jogador acertou a senha ou não.

Foram criados métodos *“gets”* e *“sets”* para o acesso e a modificação de cada atributo da classe, assim como os métodos: *“virtual void imprime_Tabuleiro()”**(***responsável por imprimir o tabuleiro), *“vector<string> retornaResposta(vector<string>)”* (***responsável por verificar cada jogada e retornar um vetor com as respostas)***, *“void adicionaJogadaNaMatriz(vector<string>)”* **(**responsável por registrar cada jogada do jogador e o respectivo resultado na matriz) e *“virtual bool validaCor(string)” **(responsável*** com verificar se as cores digitadas pelo jogador são validas). .

Algumas funções vector foram usadas nos métodos. Em *“virtual void imprime_Tabuleiro()”* a função *“size”* retorna o tamanho da vetor, em *“vector<string> retornaResposta(vector<string>)”* as funções *“end, begin, insert e push_back”* retornam, respectivamente, o iterador do último elemento, do primeiro elemento, insere elemento no vetor e aloca espaço no final do vetor.

2.4. Class Partida

A classe partida contem as características específicas de cada partida. Ela é responsável pela criação do tabuleiro, pela criação do formato da partida e pelo controle das ações de cada jogada. A classe está estruturada da seguinte forma:

```
1  class Partida: public Tabuleiro{
2  private:
3    bool adversario;
4    int num_cores;
5    int tamanho_codigo;
6    int num_palpites;
7    bool cores_repetidas;
8    int numJogada;
9     bool vitoria = false;
10   void setAdversario(bool const );
11    bool getAdversario() const;
12    void setNum_cores(int const);
13    int getNum_cores() const;
14    void setTamanho_codigo(int const );
15    int getTamanho_codigo() const;
16    void setNum_palpites(int const );
17    int getNum_Palpites();
18    void setCores_repetidas(bool const );
19    bool getCores_repetidas() const;
20    void setNumJogada(int const );
21    int getNumJogada();
22    void escolheDificuldade(int *)
23    void setaAtributosPelaDificuldade(int const );
24    static void escolheAtributosManualmente(int *, int *, int *, bool *) ;
26    static bool escolheAdversario();
27
28    void setSenhaAleatoria() ;
29    vector<string> retornaCoresRepetidas();
30    vector<string> retornaCoresNaoRepetida();
31    void setSenhaManual();
32    void realizaJogada();
33    bool validaCor(string);
34  public:
35    Partida(){
36        int dificuldade = -1;
37        escolheDificuldade(&dificuldade);
38         setaAtributosPelaDificuldade(dificuldade);
39          setAdversario(escolheAdversario());
40          getAdversario() ? setSenhaManual() :     setSenhaAleatoria();}
41
42    void jogar();
43    void imprime_Tabuleiro();
44};
```

Figura 2: Class Partida

Os atributos da classe guardam e definem o tamanho da senha e consequentemente o numero de colunas do tabuleiro (dentro da variável *"tamanho_codigo"*), o numero de palpites determinado e consequentemente o numero de linhas do tabuleiro (dentro da variável *"num_palpites"*), o número de cores disponível para a partida (dentro da variável *"num_cores"*), se a partida é jogada por dois jogadores ou por um (dentro da variável "adversário"), se a partida possibilita cores repetidas ou não (dentro de *"cores_repetidas"*) e se o jogador acertou a senha ou não (dentro de "*vitoria"*).

Foram feitos métodos *“gets”* e *“sets”* para a maioria do atributos da classe - excluindo apenas a o atributo *“vitoria”*, pois a sua manipulação é feita por outras funções dentro da classe. A classe *“Partida”*, por herdar publicamente a classe “Tabuleiro”, tem acesso aos métodos públicos da mesma.

Além dos gets e sets, é possível classificar dois tipos de métodos da classe: os responsáveis por receber (ou gerar) e preencher os atributos da classe e os responsáveis pela criação de cada jogada.

Os primeiros são chamados pelo construtor da classe (veja na linha 35), quando a classe é criada os métodos são invocados e o formato da partida vai sendo construído. O primeiro método chamado é o “*escolheDificuldade(&dificuldade)”*, ele possibilita ao jogador escolher entre montar deu próprio tabuleiro ou escolher entre os níveis de dificuldades fornecidos pelo programa. A resposta do usuário é registrada e enviada ao segundo método do construtor (linha 38). Caso o usuário tenha escolhido montar seu jogo, o programa fornecerá os atributos do jogo para que o usuário digite os valores. Caso contrário, o programa disponibiliza dez níveis de dificuldade em que três foram classificados como *“Fácil”*, três como *“Médio”*, três como *“Difícil”* e um como *“Super difícil”*. Cada nível da classificação tem como padrão o número de elementos da senha, que vai de 4 a 6 e aumenta um elemento a cada nível (com exceção do último nível, que possui o número de elementos igual ao seu antecessor). Os subníveis foram estabelecidos a partir de noções de cálculos de probabilidade discreta para a vitória do jogo, considerando a variação das seguintes características: repetição de cores ou não, numero de cores e número de palpites. As características progridem ou regridem seu valor a cada nível de acordo com a sua interferência na probabilidade de vitória do jogo. Após a coleta das características mencionadas acima, o programa chama o terceiro método (linha 39) que pergunta ao usuário se ele deseja jogar contra o computador ou contra outro usuário. A informação é registrada e o último método do construtor (linha 40) é invocado. Essa etapa é responsável pelo registro da senha de cores da partida, caso o jogador tenha optado por jogar contra outro jogador o método *“setSenhaManual()”* é invocado e pede ao segundo usuário que forneça as cores para a senha. Para isso, o método cria um vetor *“vector<string>”* sem tamanho definido e usa a função *“push_back”* para alocar os espaços necessários para o vetor senha e registrar as cores fornecidas pelo jogador. E, por fim, registra a senha de cores no atributo “senha” da classe “Tabuleiro”. Caso o usuário tenha escolhido jogar contra o computador, o método *“setSenhaAleatoria()”* é chamado e uma senha é gerada pela função *“rand()”* e seu conteúdo é registrado da mesmo forma do último método mencionado.

Os métodos responsáveis pela criação de cada jogada são: o *“void imprime_Tabuleiro()”*, o *“void jogar()”* e o *“void realizaJogada()”*. Eles têm a função de recolher as cores fornecidas pelo jogador a cada jogada, registrar o conteúdo na matriz do tabuleiro, verificar e registrar os acertos e erros da jogada e imprimir os resultados a cada jogada.

2.5. Main

A main é responsável por inicializar a jogada. Ela cria a classe *“Partida”* onde seu construtor será invocado, para a construção das características da partida, e em seguida o método *“jogar()”* da classe é chamado e, como descrito nas seções anteriores, os métodos de ação das jogadas são chamados. A main está estruturada da seguinte forma:

```
1  int main(){
2          bool inicializa;
3          cout<<"Deseja inicializar uma partida? (Sim - 1, Nao - 0)\n";
4          cin>>inicializa;
5          cout<<"\n";
6
7       if(inicializa){
8          Partida novoJogo;
9          novoJogo.jogar();
10    }
11 }
```

Figura 3: Main

2.6 Bibliotecas Utilizadas

As bibliotecas utilizadas no código foras as seguintes:

```
1 #include <iostream>
2 #include <cstdlib>
3 #include<ctime>
4 #include<string>
5 #include <vector>
6 #include<algorithm>
```

Figura 4: Bibliotecas

3. DISCUSSÃO

Discutiu-se muito sobre qual seria a estrutura principal do código, quais objetos seriam necessários criar, quais atributos e métodos cada classe deveria ter e como classificar cada objeto de maneira genérica e objetiva. Toda essa problemática levantada possibilitou a definição da estrutura geral do código e o enquadramento dos atributos e métodos necessários para o funcionamento do jogo.

Como o jogo possui várias características e elas podem variar, foi preciso prever todas as possibilidades das partidas e pensar na construção de um código que garantisse uma maior versatilidade.

4. CONCLUSÃO

As estruturas de C++ possibilitam a construção mais simplificados de códigos que possuem vários objetos e se relacionam. No código apresentado, a escolha por estruturar os objetos em herança possibilitou uma classificação mais precisa e uma construção mais genérica e

eficiente dos objetos. O polimorfismo também contribuiu para esses aspectos.

A utilização do template *"vector"* e suas funções facilita uma construção mais genérica do código. A possibilidade de usar suas funções para alocar dinamicamente o tamanho do vetor senha e da matriz do tabuleiro, para preencher os espaços alocados, para identificar o tamanho do vetor e entre outras funções utilizadas, reduziu significativamente a quantidade de códigos necessárias para se realizar as mesmas tarefas em comparação com o uso de ponteiros para a criação de vetores e uma manipulação mais genérica dos mesmos.

5. BIBLIOGRAFIA

1. H M *Deitel,* P J *Deitel. C++ Como Programar;*