Mastermind

Ligia Benassi Yamashita, *Engenharia Mecatrônica, UFSC - Campus Joinville*

I. Introdução

ESSE trabalho tinha como objetivo realizar uma recriação do jogo *Mastermind* como um programa em C++, visando a implementação do conteúdo aprendido na matéria de Programação III; incluindo uso de classes, encapsulamento respeitado por todas as classes, alocação dinâmica de memória e associação de classes.

II. Desenvolvimento

A. Regras

O jogo se inicia com a escolha das definições da partida *(Figura 1):* o número de cores utilizados no jogo, o tamanho do código, o número máximo de palpites e a possibilidade de utilização de cores repetidas no código. O jogador deve então escolher seu adversário: outro jogador ou o computador.

Figura 1. Inserção das definições

1) 2 Jogadores: O primeiro jogador, denominado *codemaker*, determina o seu código escolhendo uma sequência de pinos de variadas cores através uma sequência numérica no terminal *(Figura 2).*

Figura 2. Rodada do *codemaker*

O objetivo do outro jogador é adivinhar tanto as cor quanto a posição de cada pino do código. Para fazer isso, o segundo jogador, denominado *codebreaker*, tenta adivinhar o código escolhido entrando com sua própria sequência de pinos *(Figura 3).*

Em sua rodada, o *codemaker* classifica a tentativa do *codebreaker*, colocando um marcador branco para cada pino com a cor correta e a posição incorreta; um marcador preto para cada pino com a cor e posição correta; e nenhum marcador para pinos com a cor incorreta *(Figura 4).*

Figura 3. Rodada do *codebreaker*

Figura 4. Resposta do *codemaker*

O jogo continua em rodadas alternadas até que o *codebreaker* acerte o código, ou exceda a quantidade de tentativas permitidas. É exibida a tela de vitória ou derrota e o código *(Figuras 5 e 6).*

Figura 5. Tela de vitória

2) 1 Jogador: Nesse modo, o computador irá jogar na posição de *codemaker*, criando uma sequência aleatória (mas que ainda segue as definições escolhidas inicialmente). O *codebreaker* terá os mesmos objetivos do que em um jogo de 2 jogadores, mas receberá a sequência de marcadores automaticamente gerada pelo computador.

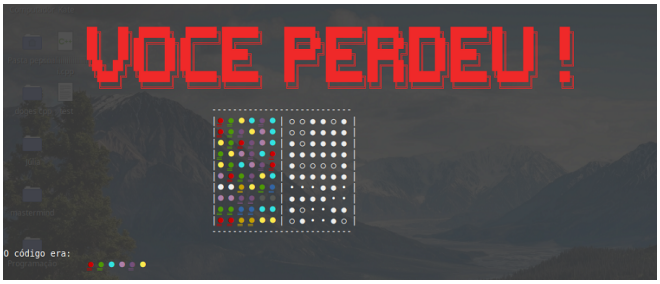

Figura 6. Tela de derrota

III. Discussão

A. Implementação

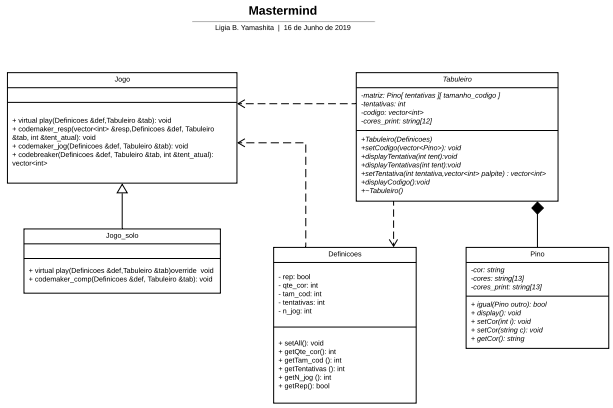

Figura 7. Diagrama UML do projeto

A Figura 7 apresenta as relações entre as classes implementadas no código. Como pode ser visto, há uma relação de composição entre a classe Pino e Tabuleiro, e uma relação de herança entre a classe Jogosolo e a classe Jogo, sendo utilizado polimorfismo para a sobreposição da função *play*. Todas as outras relações entre classes são de dependência.

A escolha do polimorfismo com relação à classe Jogo foi motivada pela reutilização da função codebreaker, e a escolha da implementação de composição na classe Tabuleiro foi útil na criação de uma matriz de objetos Pino.

O código do jogo se inicia com a função main chamando a função Title que imprime o título *Mastermind*, depois é chamada a função *setAll* da classe Definicoes que aceita entradas de usuário para definir as definições de jogo. Um objeto da classe Tabuleiro é inicializado com as definições inseridas. Um ponteiro da classe Jogo recebe um objeto Jogo ou Jogosolo dependendo da quantidade de jogadores escolhida e é chamada a função *play* do objeto partida. Dentro desta função são chamadas funções alternadas para a jogada do *codemaker* e do *codebreaker*, até que o código seja descoberto ou que o *codebreaker* ultrapasse a quantidade de tentativas máxima definida.

Ambas as funções consistem, de maneira simplificada, em preenchimento do objeto matriz da classe Tabuleiro, comparação da inserção com a resposta correta e impressão do objeto Tabuleiro criado em tela.

As entradas de usuário possuem checagem de tamanho e validade (em relação à repetição e cores permitidas) conforme necessário.

A implementação do jogo é explicada em maiores detalhes em comentários no arquivo *"Implement.cpp"*.

B. Jogabilidade

A instruções de jogo seguem o indicado na subseção II-A desse trabalho, e o formato de entradas e mais detalhes podem ser encontrados no arquivo *"README.txt."*

IV. Conclusão

Nesse decorrer desse trabalho foi possível perceber como uma modelagem apropriada e associações de classes podem beneficiar um código, e as vantagens e desvantagens relacionadas a programação orientada a objetos.