Universidade Federal do Espírito Santo
Centro Tecnológico
Departamento de Informática
Estrutura de Dados I

Primeiro Trabalho Prático

Jogo de Bisca

Aluno: Elias Pereira Gomes Junior

Professor: Vinícius Fernandes Soares Mota

11/05

Jogo de Bisca

1. **Introdução**

O intuito do trabalho é criar um jogo de bisca de fácil execução e de jogabilidade intuitiva. No programa há a possibilidade de jogar contra um ou contra três jogadores do tipo máquina. Contrastando com o jogo de bisca tradicional, cada jogador busca sua vitória individual, ou seja, não há a formação de duplas. No fim do jogo é mostrado o placar e as cartas ganhas por cada jogador.

Além disso é possível optar entre duas dificuldades: Fácil ou Difícil. Quando escolhida a primeira opção os jogadores do tipo máquina jogaram suas cartas de modo aleatório, e quando escolhida a última, haverá consideração das regras do jogo no descarte.

No Jogo existem duas modalidades: A padrão ou a expositiva. No modo padrão, o jogo é executado de forma mais próxima a realidade, onde nenhum jogador tem a visibilidade do baralho dos outros jogadores. Porém na opção de execução expositiva é possível a visualização do baralho do jogo, e das cartas na mão de cada jogador. Essa última opção foi criada para análise estrutural da jogabilidade do software.

2. **Descrição das estratégias implementadas**

A seguir serão citadas as estratégias utilizadas na implementação das principais funções do jogo, as funções serão apresentadas em uma ordem lógica para uma melhor compreensão da estrutura do programa.

- **criaBaralho**: Essa função recebe como parâmetro um ponteiro de ponteiros do tipo *tBaralho*. Alocando espaço na memória para a criação de um baralho com quarenta cartas. Utiliza-se dois laços de *for,* um dentro do outro, com o primeiro percorrendo os números de naipes possíveis (quatro naipes) e o segundo percorrendo todas as naturezas possíveis para uma carta (K, J, A etc...). Assim com o auxílio da função *criaCarta* e *insereCartaNoBaralho* as cartas que vão sendo criadas são inseridas no baralho. Essa função é utilizada unicamente na criação do baralho do jogo. Seu custo computacional é O(1), pois a são sempre criadas a mesma quantidade de cartas no baralho.

- **embaralhar**: Recebe como parâmetro um ponteiro do tipo *tBaralho.* A função embaralha todas as cartas de modo aleatório. Inicialmente ocorre a inserção de um número variável como semente para na função *srand*. Após isso, a função *rand* gera números diferentes a cada utilização da função possibilitando o embaralhamento das cartas. Logo após isso ocorre a abertura de um laço *for* que percorre todas as cartas do baralho. Além disso é importante destacar que as cartas do baralho são implementadas como uma lista encadeada, essa lista é do tipo *tCartaDoBaralho* e que cada célula da lista possui seu índice identificador. Assim, através das funções buscaCarta e insereCartaNoIndice é possível realizar a troca das cartas. Seu custo computacional é $O(n^2)$.

- **buscaCarta:** Essa função recebe como parâmetro um ponteiro do tipo *tBaralho* e um inteiro representando um índice de uma carta. O objetivo principal é retornar a carta de índice igual ao índice passado como parâmetro. Não conseguindo achar a carta no baralho, a função imprime uma mensagem de erro e finaliza o programa. Seu custo computacional é O(n).

- **insereNoIndice**: A função recebe como parâmetros um ponteiro do tipo *tBaralho*, um ponteiro do tipo *tCarta* e um valor inteiro do índice de uma carta. A função percorre todo o baralho e só insere se o índice passado for válido. Seu custo computacional é O(n).

- **insereNoBaralho**: A função recebe como parâmetro um ponteiro de ponteiro do tipo *tBaralho* e um ponteiro do tipo *Carta*. Sua ação é inserir a carta passada como a última do baralho. Seu custo computacional é O(1).

- **retiraCartaBaralho**: Recebe como parâmetro um ponteiro de ponteiro do tipo *tBaralho*, um valor inteiro e um ponteiro do tipo *tCarta*. O valor inteiro passado é igual um índice de uma carta pertencente ao baralho. Assim, o objetivo da função é retirar uma carta de um índice específico do baralho. Para isso coloca-se um laço *while* para achar a carta desejada. Possuindo o endereço desejado, realizam-se as verificações necessárias e, por fim, ocorre a remoção. Seu custo computacional é O(n).

- **criaJogador**: Essa função recebe um ponteiro de ponteiro do tipo *tJogador*, um número inteiro representando o tipo do jogador e um inteiro representando o id do jogador. Sua finalidade é criar um jogador. Antes de tudo há a alocação do ponteiro para o jogador na memória. A seguir ocorre a definição do nome do jogador. Existem três possibilidades para o id do jogador, quando id é igual a zero, o jogador é humano, caso contrário o jogador é do tipo máquina. Se for humano, ocorre a solicitação da inserção do nome, caso contrário o nome é definido pelo sistema de forma automática. Após isso há a alocação dos baralhos pertencentes ao jogador. Seu custo computacional é O(1).

- **compraCarta**: Os parâmetros de entrada são um ponteiro do tipo *tBaralho* e um ponteiro do tipo *tJogador*. O propósito da função é colocar uma carta no baralho da mão do jogador, se o baralho passado como parâmetro estiver vazio, a função printa uma mensagem de erro e finaliza o programa. Caso contrário, utiliza-se a função *retiraCartaBaralho* para tirar a última carta do baralho do jogo e a função *insereNoBaralho* para inserir a carta no baralho da mão do jogador. Seu custo computacional é O(n).

- **jogaCartaModoFacil**: A entrada desta função é um ponteiro do tipo *tJogador*, dois ponteiros do tipo *tCarta* e um valor inteiro que indica a modalidade do jogo. Existem dois casos a considerar, se o jogador for humano ou se é do tipo máquina. Se for humano, o baralho da mão do jogador aparecerá na tela e aparecerá na tela as possíveis opções de escolha para o lançamento da carta. Caso o jogador for no tipo máquina, haverá o descarte de uma carta aleatório. Se a modalidade escolhida for a expositiva,o baralho da mão do jogador do tipo máquina ficará exposto, caso contrário ficará oculto. Seu custo computacional é O(n).

- **jogaCartaModoDificil**: Os parâmetros de entrada são um ponteiro do tipo *tJogador*, três ponteiros do tipo *tCarta* e dois inteiros indicando a indicando a quantidade de cartas presentes no monte da partida e a modalidade do jogo. Caso o jogador for humano, não haverá mudanças em relação a função

jogaCartaModoFacil. Porém se o jogador for do tipo máquina, ocorrerá um descarte da seguinte maneira: *Verifica-se a quantidade de cartas no monte*. Caso seja zero, o jogador é por conseguinte, o primeiro a jogar, com isso a carta lançada é uma carta sem valor ou com o menor valor do baralho da mão do jogador. Do contrário, *ocorre uma tentativa de buscar uma carta de maior prioridade que a carta vencedora atual.* Caso não exista tal carta, o jogador lançará a carta de menor valor da sua mão. Seu custo computacional é O(n).

- **buscaCartaComMenorValor**: A função recebe como parâmetros um ponteiro do tipo *tBaralho* e um ponteiro do tipo *tCarta*. Um laço *for* percorre todas as cartas do baralho, comparando o valor atual com o menor valor definido inicialmente (A menor carta é iniciada com a primeira carta). No fim do for haverá o retorno do índice da menor carta mais um. Seu custo computacional é O(n).

- **buscaCartaDeMaiorPrioridade**: A entrada desta função é um ponteiro do tipo *tBaralho*, e dois ponteiros do tipo *tCarta*, representando a carta vencedora atual e a carta trunfo. Inicialmente verifica-se se a carta vencedora atual é um corte, caso seja, a função tenta buscar o menor corte na mão do jogador que seja maior que o corte vencedor. Se não houver tal corte, o sistema busca jogar a menor carta da mão. Entretanto, se a carta vencedora atual não for um corte, haverá a tentativa de um encarte, caso não tenha um encarte, haverá a tentativa de cortar. No fim, se também não houver corte na mão do jogador o sistema jogará a menor carta da mão. Seu custo computacional é O(n).

- **temCorteNoBaralho**: A entrada desta função é um ponteiro do tipo *tCarta* representando a carta trunfo e um ponteiro do tipo *tBaralho* representando o baralho. A função percorre todo o baralho o baralho passado através de um laço *for* e verifica se o naipe de alguma carta é igual ao naipe da carta passada como trunfo. Se for igual, haverá o retorno do inteiro um, senão haverá o retorno do inteiro zero. Seu custo computacional é O(n).

- **TemEncarteNoBaralho**: Recebe como parâmetro um ponteiro do tipo *tBaralho* representando o baralho e um ponteiro do tipo *tCarta* representando a carta vencedora do monte. Através de um laço *for* a função percorre todo o baralho verificando se existe alguma carta do mesmo naipe que a carta vencedora e que possua uma prioridade maior. Caso haja, a função retorna um, do contrário retorna zero. Seu custo computacional é O(n).

- **encarteVencedor**: Os parâmetros de entrada são um ponteiro do tipo *tBaralho* e um ponteiro do tipo *tCarta* que representa a carta vencedora anterior. Ocorre uma comparação carta a carta para determinar a carta que encarte a carta vencedora. A função retorna o índice do encarte vencedor. É importante destacar que essa função só é acessada se primeiro haver um retorno positivo da função *temEncarteNoBaralho.* Seu custo computacional é O(n).

- **menorCorte**: A função recebe como parâmetros de entrada um ponteiro do tipo *tBaralho* e um ponteiro do tipo *tCarta* que representa a carta trunfo do jogo. Através de um laço *for* a função percorre todas as cartas do baralho. Ocorre uma comparação se a carta atual passada é um corte, e caso sim faz as comparações necessárias para retornar o menor corte da mão do jogador. No fim, a função retorna o índice do menor corte. Seu custo computacional é O(n).

- **menorCorteVencedor**: A entrada desta função é um ponteiro do tipo *tBaralho* e um ponteiro do tipo *tCarta*. A principal diferença desta para a função *menorCorte* reside na necessidade de retornar o índice do menor corte que seja maior que a carta vencedora(que também é um corte). Esta função só é utilizada do software caso haja um retorno positivo da função *temCorteNoBaralho.* Se não for achado uma carta vencedora na mão do jogador a função retorna um índice negativo, logo inválido. Seu custo computacional é O(n).

- **criaJogo**: Um único parâmetro de entrada é necessário para essa função: um ponteiro de ponteiro do tipo *tJogo*. O principal objetivo dessa função é criar um jogo completamente configurado. Inicialmente ocorre a alocação do ponteiro do jogo, a seguir ocorre a criação e iniciação das variáveis pertencentes ao jogo, como os jogadores e o baralho. Entre as funções auxiliares dessa função ocorre existem aquelas que verificam se a entrada está conforme o esperado pelo sistema, caso não esteja o programa retorna a inserção dos valores e só continua se a entrada for válida, conforme o explicitado no momento da execução do software. Seu custo computacional é $O(n^2)$.

- **comecaJogo**: O parâmetro de entrada desta função é um ponteiro de ponteiro do tipo *tJogo*. A função é abrangente e são necessárias várias funções auxiliares para que ela possa ser realizada com sucesso. Inicialmente ocorre a verificação se o jogo é do tipo expositivo, se for, o baralho do jogo é mostrado e é oferecido a opção de embaralhar o baralho, senão nenhuma dessas funcionalidades são ofertadas. Após isso, ocorre a abertura de laço *do-while* e a execução da função *jogaRodada.* Após o baralho estar vazio, ocorre a abertura de um laço

para terminar de jogar as cartas restantes das mãos dos jogadores. No fim, é imprimido os dados do jogo.

- **jogaRodada**: A função recebe como parâmetros um ponteiro de ponteiro do tipo *tJogo*, um inteiro representando o número da rodada e um inteiro informando o id do vencedor da rodada anterior. A função imprime o cabeçalho da partida e distribui as cartas aos jogadores. Retorna o id do vencedor da rodada. Seu custo computacional é $O(n^2)$.

- **distribuiCartas**: A função recebe como parâmetros um ponteiro de ponteiro do tipo *tJogo*, o valor inteiro do número da rodada e o valor inteiro do id do jogador vencedor anterior. Caso a rodada seja a primeira, três cartas são distribuídas a cada jogador. Caso contrário haverá a distribuição de uma carta por jogador começando no jogador que venceu a rodada anterior. A função só distribui enquanto o baralho do jogo não estiver vazio. Seu custo computacional é O(n).

- **verficaCartaGanhadora**: Os parâmetros de entrada desta função são três ponteiros do tipo *tCarta*, representando a carta ganhadora, a carta atual e a carta trunfo. Sua finalidade é definir qual de duas cartas passadas possui a maior prioridade. No fim, a função retorna o id do jogador que jogou a carta vencedora. Seu custo computacional é O(1).

- **cartaVencedoraDoMonte**: Os parâmetros de entrada são um ponteiro do tipo *tMonte*, um inteiro indicando a quantidade de cartas no monte e um ponteiro do tipo *tCarta* indicando a carta trunfo do jogo. A função determina a carta vencedora entre todas as cartas jogadas em uma rodada. Com isso, retorna a carta vencedora no final da função. Seu custo computacional é O(n).

- **recebeCartasRodada**: Recebe como parâmetros de entrada um ponteiro de ponteiros do tipo *tJogador*, um ponteiro do tipo *tMonte* e o valor inteiro da quantidade de cartas no monte. Sua principal finalidade é inserir as cartas do monte no baralho das cartas ganhas do jogador passado como parâmetro. Seu custo computacional é O(n).

3. **Conclusão**

O trabalho foi concluído com sucesso e cumpre todos os requisitos propostos. Vale ressaltar mais uma vez que jogando o jogo no modo expositivo é possível averiguar que as funções funcionam como deveriam. Ademais, apesar de não possuir interface gráfica, foi possível a construção de um jogo intuitivo e fácil de se jogar.