INF 1301 Programação Modular - 19.1
Prof. Flávio Bevilacqua

Trabalho 2

Felipe de Oliveira - 1720654
Gabriel Garcia Mascheroni Costa - 1811312
Rafael Damazio Monteiro da Silva - 1712990

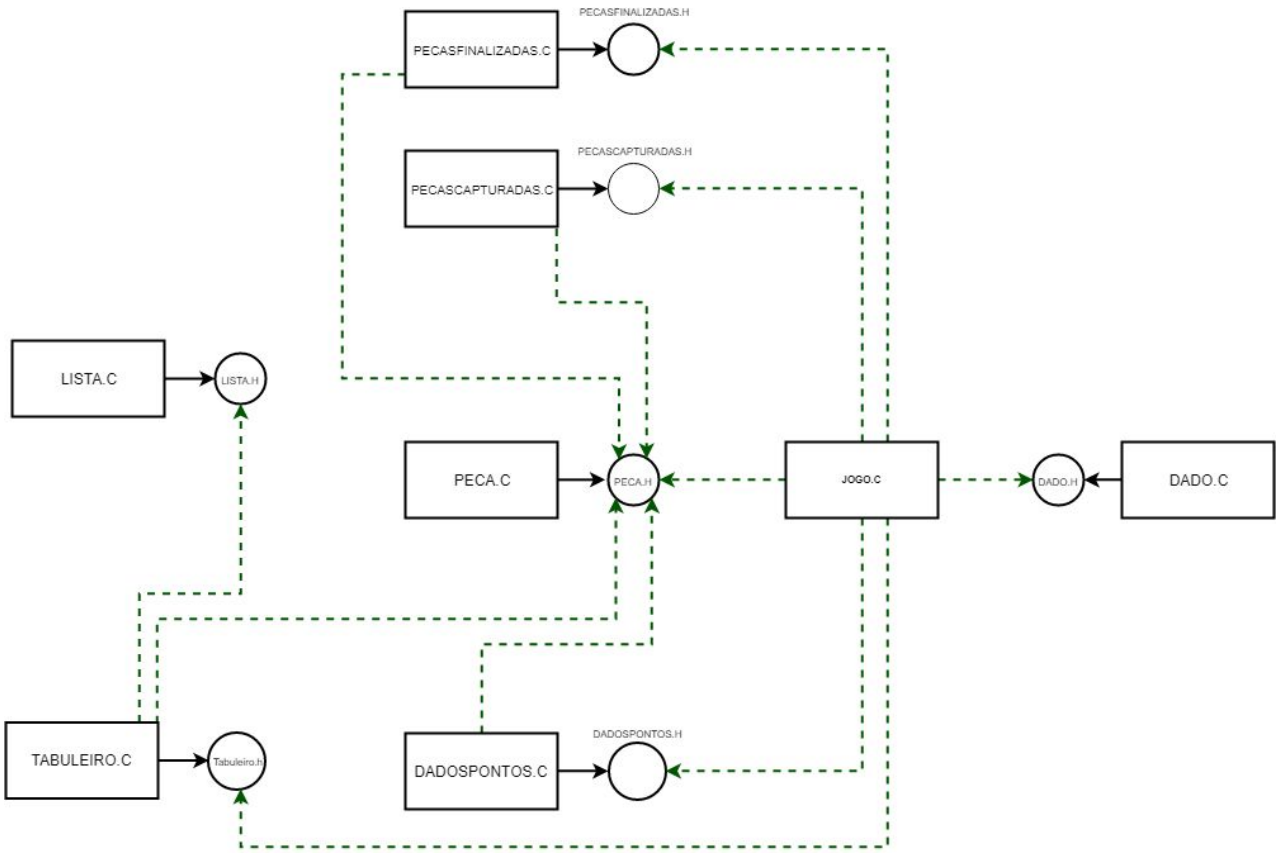
1. Especificação de Requisitos.

- Requisitos Funcionais:
  1. Ao iniciar uma partida são rolados dois dados e o jogador que tirar o maior valor joga primeiro
  2. O programa deverá guardar uma pontuação de acordo com as partidas vencidas por cada jogador, e essa pontuação deverá ser mostrada na interface
  3. O programa deverá fornecer uma representação visual do tabuleiro que mostre as peças e as casas onde elas se encontram.
  4. A cada rodada o jogador deverá rodar dois dados e selecionar a casa de onde deseja mover uma peça, caso a casa selecionada não possua peças do player a pergunta será repetida
  5. Ao selecionar a casa de onde ele deseja mover peças o player deve escolher para qual casa deseja mover a mesma de acordo com os dados que ele possui (o sistema deverá dar somente as opções válidas de movimento)
  6. O sistema deverá permitir que peças sejam capturadas e essas devem ser alocadas na área de retorno
  7. O sistema deverá contabilizar as peças que foram finalizadas (chegaram até o final do tabuleiro) e utilizar isso como condição de vitória
  8. A cada rodada o jogador deverá ter a opção de dobrar pontos, e o outro jogador deve ter a opção de aceitar ou recusar a dobra, caso seja recusado o jogo deverá ser terminado e o jogador que propôs a dobra recebe 1 ponto, caso seja aceita o jogo continua, mas valendo dois pontos e somente o jogador que aceitou a dobra pode propor futuras dobras na partida
  9. A pontuação deverá ser armazenada em um arquivo externo durante a partida, e quando a mesma for finalizada esse arquivo deverá ser apagado.
  10. Cada peça capturada de um jogador deve ser devolvida para uma casa livre antes de qualquer outra movimentação deste mesmo jogador. O que determina o número da casa é o valor retirado em cada dado, caso seja possível a movimentação.

- Requisitos Não Funcionais
  1. Uma partida em andamento poderá ser salva ao encerrar o programa, e ao iniciar o programa ele deve ter a opção de continuar partidas que foram salvas.
  2. A aplicação será um executável que utilizará o cmd como interface.
  3. Aplicação windows sem suporte para outros sistemas operacionais.

- Requisitos Inversos
  1. Não será criado uma inteligência artificial para jogar contra o jogador humano. Esta aplicação somente suporta jogos contra dois jogadores humanos.

2. Modelo de arquitetura.



Funções e tipos disponibilizados pelos módulos:

I - PECA.C - PEC:

i - Tipos:

Enum PEC_Color: Enum que identifica duas cores de peças, preto e branco.
Enum PEC_CondRet: Enum que enumera os comportamentos das funções.
PecaHead: Tipo que identifica uma peça. Encapsula o espaço de dados.

ii - Funções:

PEC_CondRet PEC_CriarPeca(PEC_color color ,PecaHead* peca);

Assertiva de entrada:

Deve existir um ponteiro para qual a peça criada será retornada por referência.

A cor passada deve ser um enum exportado pelo módulo.

Assertiva de saída:

A peça criada com a cor passada será salva no ponteiro passado por referência.

PEC_CondRet PEC_ObterCor(PEC_color *ret, PecaHead peca);

Assertiva de entrada:

Deve existir uma área de memória do tipo PEC_Color para qual será retornado o tipo da cor.

Assertiva de saída:

Caso não exista a peça a função não retornará a cor na área de memória do retorno da cor

PEC_CondRet PEC_DestruirPeca(PecaHead *peca);

Assertiva de entrada:

O espaço de dados passado deve possuir uma peça alocada

Assertiva de Saída

Após a destruição da peça, o ponteiro passado terá o valor de NULL.

II - DADO.C - DICE:

i- Tipos:

Enum DICA_CondRet: Enum que enumera os comportamentos das funções.

iI - Funções:

DICE_CondRet DICE_RolarDado(int *dado, int numLados);

Assertivas de entrada:

Deve ser fornecido um espaço de dados que irá receber por referência o resultado da rolagem do dado**.**

Deve ser fornecido quantos lados possui o dado.

Assertivas de saída:

Será retornado por referência um número aleatório entre 0 e o número de lados fornecidos.

III - DADOPONTOS.C - DPT:

i-Tipos

Enum DPT_CondRet: Enum que enumera os comportamentos das funções

ii- Funções:

DPT_CondRet DPT_CriarDadoPontos(DPT_DadoPontos **pDadoPontos);

Assertivas de Entrada:

- Deve existir um ponteiro por onde será passado, por referência, o dado de pontos

criado.

Assertivas de Saída:

- Dado de pontos foi criado e pDadosPontos foi atualizado.

DPT_CondRet DPT_AtualizarJogadorDobra(DPT_DadoPontos *pDadoPontos, Pec_color CorPeca);

Assertivas de Entrada:

- Deve existir um ponteiro para o dado de pontos que indica a dobra;
- Deve existir uma cor associada ao jogador que está dobrando a pontuação.

Assertivas de Saída:

- O jogador que pode dobrar a partida é atualizado através da cor da peça;

DPT_CondRet DPT_DobrarPontuacaoPartida(DPT_DadoPontos *pDadoPontos, Pec_color CorPeca);

Assertivas de Entrada:

- Deve existir ponteiro para o dado de pontos que será utilizado;
- Deve existir uma cor de peça associada ao jogador que está dobrando a pontuação.

Assertivas de Saída:

- Se dado pontos não existir ou cor da peça não for igual a cor autorizada a dobrar a pontuação o ponteiro pDadosPontos não é atualizado;
- Caso contrário, a pontuação é dobrada e o ponteiro é atualizado.

DPT_CondRet DPT_ObterJogadorDobraPartida(DPT_DadoPontos *pDadoPontos, Pec_color *pCorPeca);

Assertivas de Entrada:

-Deve existir um ponteiro para o dado de pontos que será analisado.

-Deve existir um ponteiro para ser atualizado com a cor do Jogador possuidor do Dado Pontos

Assertivas de Saída:

-Se dado pontos não existir ou não tiver possuídor no momento, o ponteiro não será atualizado.

-Caso contrário, o ponteiro é atualizado com a cor correta.

DPT_CondRet DPT_ObterPontuacaoPartida(DPT_DadoPontos *pDadoPontos, int *pPontuacao);

Assertivas de Entrada:

- Deve existir um ponteiro para o dado de pontos que será utilizado;
- Deve existir um ponteiro para ser atualizado com a pontuação da partida.

Assertivas de Saída:

- Se dado de pontos existir, a pontuação será obtida e o conteúdo do ponteiro pPontuacao será atualizado.

DPT_CondRet DPT_DestruirDadoPontos(DPT_DadoPontos **pDadoPontos);

Assertivas de Entrada:

- Deve existir um ponteiro para o dado de pontos que será destruído, passado por referência.

Assertivas de Saída:

- Dado de pontos foi destruído e ponteiro agora aponta para NULL.

IV - TABULEIRO.C - TBL:

i-Tipos

Enum TBL_CondRet: Enum que enumera os comportamentos das funções.
TabuleiroHead: Tipo que identifica um Tabuleiro. Encapsula a lista de listas que representa as casas do tabuleiro.

ii- Funções:

TBL_CondRet TBL_CriarTabuleiro(TabuleiroHead* tabuleiro);

Assertiva de entrada:

Deve existir um ponteiro para qual o tabuleiro criado será retornado por referência.

Assertiva de saída:

O tabuleiro criado será salvo no ponteiro passado por referência, ele terá uma lista de 24 casas com as peças iniciais posicionadas.

TBL_CondRet TBL_DestruirTabuleiro(TabuleiroHead tabuleiro);;

Assertiva de entrada:

O espaço de dados passado deve possuir um tabuleiro alocado

Assertiva de Saída

Após a destruição do tabuleiro, o ponteiro passado terá o valor de NULL.

TBL_CondRet TBL_ObterCasas(LIS_tppLista* casas, TabuleiroHead tabuleiro);

Assertiva de entrada:

Deve existir uma área de memória do tipo LIS_tppLista para qual será retornada a lista de casas.

Assertiva de saída:

Caso não exista o tabuleiro a função não retornará a lista de casas.

TBL_CondRet TBL_MoverPeca(TabuleiroHead tabuleiro, int casaInicio, int casaFim);

Assertiva de entrada:

O espaço de dados passado deve possuir um tabuleiro alocado
Os paramêtros casaInicio e casaFim devem ser casas válidas do tabuleiro

Assertiva de saída:

Uma peça da casa inicial deve ser movida para a casa final

V - PECASFINALIZADAS.C - PFIN:

I - Tipos:

Enum PFIN_CondRet: Enum que enumera o comportamento das funções exportadas por este módulo

II - Funções:

PFIN_CondRet ObterPecasFinBrancas(int *ret);
PFIN_CondRet ObterPecasFinPretas(int*ret);
PFIN_CondRet InserirPecaFinBranca();
PFIN_CondRet InserirPecaFinPreta();
PFIN_CondRet Resetar();

VI - PECASCAPTURADAS.C - PCAP:

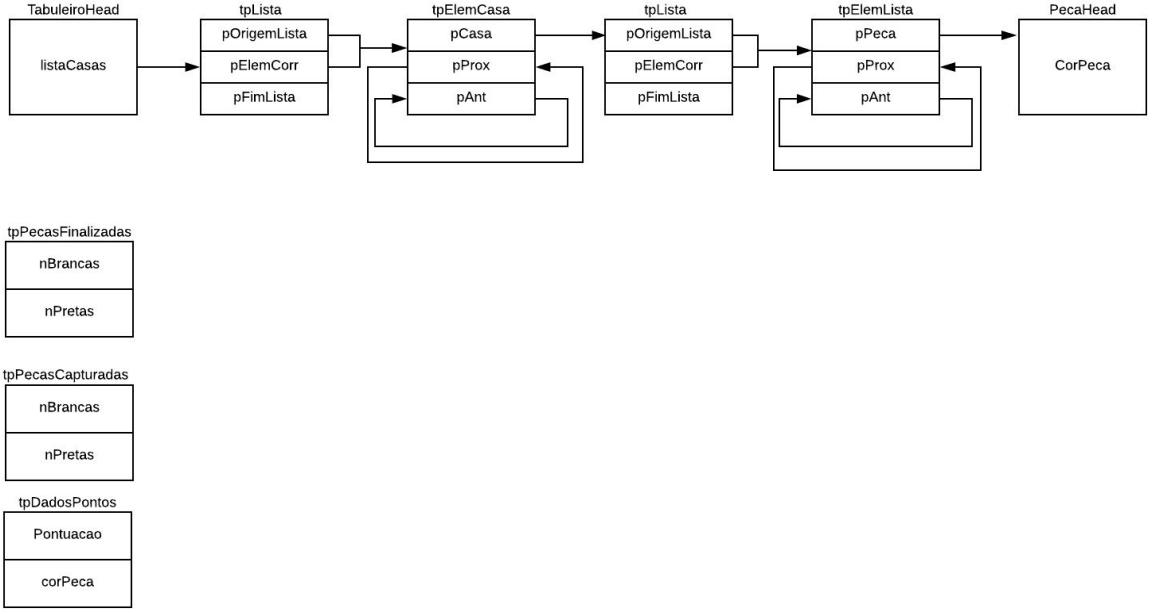
I - Tipos:

Enum PCAP_CondRet: Enum que enumera o comportamento das funções exportadas pelo módulo.

II - Funções:

PCAP_CondRet ObterPecasCapBrancas(int *ret);
PCAP_CondRet ObterPecasCapPretas(int*ret);
PCAP_CondRet InserirPecaCapBranca();
PCAP_CondRet InserirPecaCapPreta();
PCAP_CondRet Resetar();

3. Modelo estrutural.



Assertivas:

I - DadoPontos:

Estrutura composta por dois campos int, uma para guardar o multiplicador de da atual partida e o atual dono do dado. Este último campo somente é preenchido quando já houver uma multiplicação em andamento, conforme as regras do gamão.

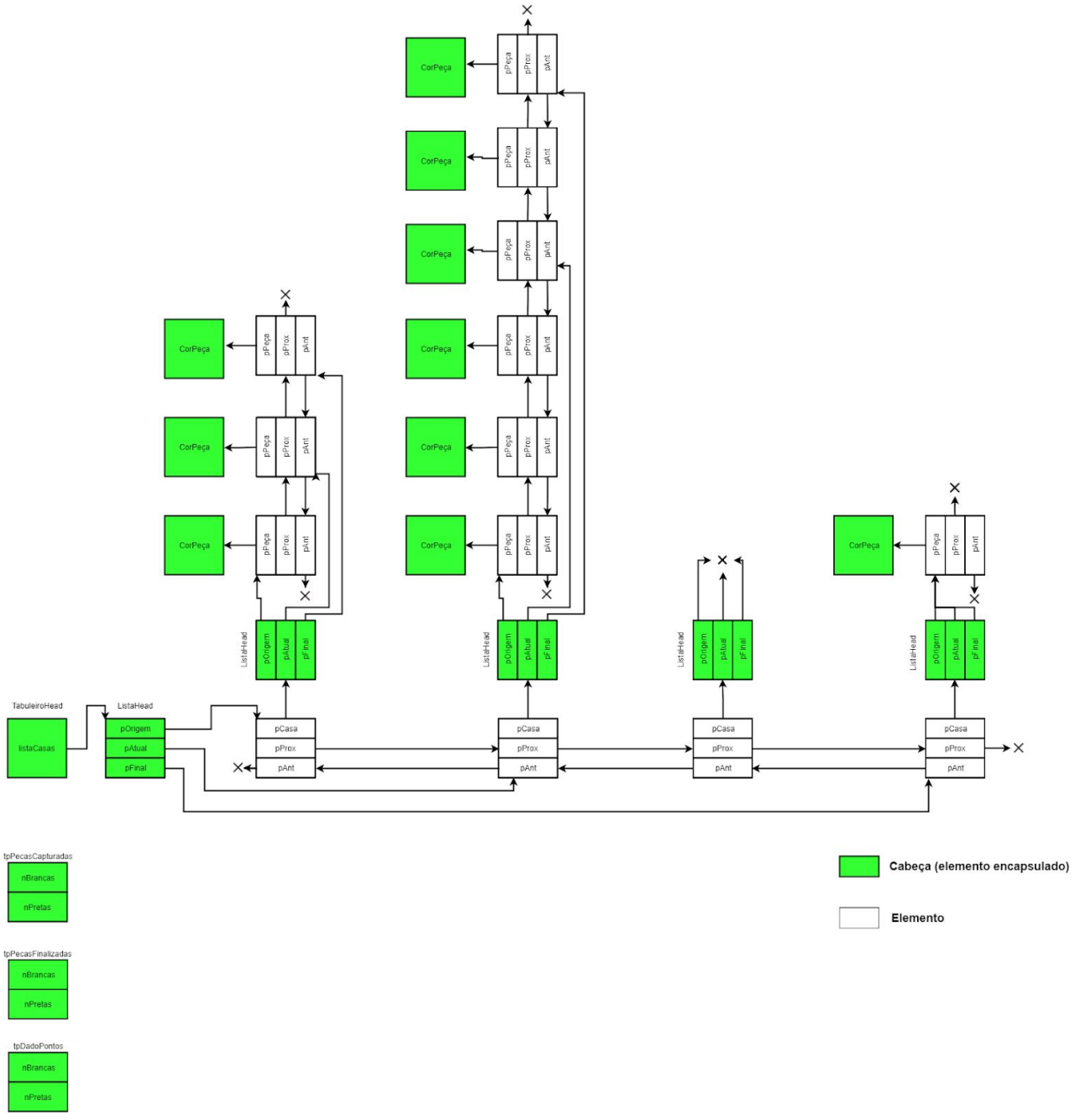
II - PecasCapturadas:

Estrutura composta por dois campos int, encapsulada, uma para guardar o número de peças brancas capturadas e uma para guardar o número de peças pretas capturadas.

III - PecasFinalizadas:

Estrutura composta por dois campos int, encapsulada, uma para guardar o número de peças brancas finalizadas e uma para guardar o número de peças pretas finalizadas.

Exemplo:


CorPeça
pPeça
pProx
pAnt
ListaHead
pOrigem
pAtual
pFinal
TabuleiroHead
listaCasas
pCasa
tpPecasCapturadas
nBrancas
nPretas
tpPecasFinalizadas
tpDadoPontos
Cabeça (elemento encapsulado)
Elemento