PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Jogo de Gamão

Documentação para o Trabalho 3

Grupo:

Luiza Del Negro - 1721251
Iago Farroco - 1712827
Daniel Weil - 1711207

Professor: Flávio Bevilacqua

Resumo

Esta documentação tem como finalidade explicar para programador e usuário o funcionamento do jogo modularizado de gamão desenvolvido pelo grupo.

Índice

1- Especificação de requisitos

1.1 Requisitos funcionais

Do jogo:

I.Gamão é um jogo de tabuleiro para 2(dois) jogadores.

I.iO objetivo do jogo é mover todas as peças para seu tabuleiro interno e, então, fazer a retirada final de cada uma delas do tabuleiro

I.ii O primeiro jogador a retirar todas as suas peças vence a partida.

II.Consiste em um tabuleiro com 24(vinte e quatro)casas

II.i O tabuleiro é dividido em quatro quadrantes, que se refere a seis casas consecutivas.

II.ii As casas de 1 a 6 constituem o quadrante interior ou o *home board*.

II.iii As casas de 7 a 12 constituem o quadrante exterior ou o *outer board*.

II.iv No meio se localiza a barra, para onde as peças que são impedidas temporariamente estão.

III. Possui um total de 30(trinta) pecas, 15(quinze) para cada jogador .

III.i Cada grupo de peças é da cor vermelha ou preta,

III.ii Essas peças estão dispostas inicialmente de forma organizada no tabuleiro.

a)Para um jogador:

2 peças na casa 1

5 peças na casa 6

3 peças na casa 8

5 peças na casa 12

b) Para outro jogador:

A mesma quantidade de peças nas casas opostas do tabuleiro.

IV. O jogo possui dois tipos de dado:

IV.i Dado pontos: Um dado que marca o valor da partida.

a)Ao iniciar a partida, a mesma se inicia com 1(um) ponto.

b)O valor da partida pode ser dobrado de forma alternada por cada jogador.

c) Não é permitido que o mesmo jogador dobre duas vezes.

d) O Dado de pontos pode valer 2,4,8,16,32,64.

e) Não é permitido dobrar além de 64.

IV.ii Dado: Dois dados simples de um a seis lados.

a)Quando lançado, gera dois números aleatórios de um a seis.

b)Se os números forem iguais, o jogador dobra o movimento.

c) O jogador é obrigado a jogar enquanto houver possibilidade de movimento.

V.Na partida:

Vi.Tabuleiro no início do jogo:

a)Com as peças dispostas, o jogador inicia a partida do lado oposto ao seu no tabuleiro.

b)O objetivo é trazer as peças para o quadrante mais próximo e por fim removê-las.

V.ii Dados no início do jogo:

a)O dado simples é lançado, gerando dois números.

b)Cada um se refere a um jogador.

c) O jogador com maior número nos dados inicia o jogo utilizando o ambos dos dados iniciais.

V.iii Dados a cada partida durante o jogo:

a)São gerados dois valores aleatórios de um a seis que são utilizados para movimentar as peças.

V.iv Movimentação no tabuleiro:

a) Condições para mover as peças

a.1)O local que se deseja movimentar está vazio

a.2)O local que se deseja movimentar tem apenas peças do próprio jogador

a.3)O local que se deseja movimentar tem apenas uma peça do jogador adversário(e a mesma é movida para a barra)

b) O jogador pode mover as peças das seguintes formas (se cumprir com as condições para mover as peças):

b.1)Uma peça com o valor total dos dados

b.2)Duas peças, cada uma com o valor de cada dado

b.3) Caso tenha tido dobra, o jogador pode repetir as ações anteriores .

c) Caso de peças na barra

c.1) O jogador da vez só pode mover as peças quando retirar todas suas peças da barra.

c.2) O jogador só pode retirar as peças da barra se o dado lançado na vez permitir realizar algum movimento das Condições para mover peças .

c.3) Se o jogador da vez não puder realizar nenhum movimento com base nos valores de dado recebidos ele perde a vez.

VI. Retirada final:

VI.i O jogo se aproxima do fim quando pelo menos um dos jogadores coloca todas suas peças no seu quadrante final.

VI.ii A partir desse momento o jogador pode começar a retirar suas peças com base no valor dos dados.

VI.iii O jogador tem que conseguir um valor mínimo para retirar as peças do tabuleiro.

a)Um jogador retira uma peça ao tirar um número nos dados que corresponda ao ponto em que a peça se encontra, permitindo que ele a retire definitivamente do tabuleiro. Assim, ao tirar um 6, o jogador retira *uma* peça de seu ponto 6.

VI.iv A saída de uma peça do jogo é definitiva.

VII. Pontuação

VII.i O ganho inicial de uma partida é de 1 ponto. Este ponto é ganho pelo primeiro jogador que consegue retirar todas as suas peças.

VII.ii Um jogador pode a qualquer altura propor dobrar a aposta. Segue com a validade de *IV.i Dado Pontos*

a)Em caso de dobra:

- Abandonar: Jogador não aceita a dobra e abandona imediatamente a partida e perde consequentemente 1 ponto.
- Aceitar: neste caso, a partida continua e os ganhos são duplicados.

VII.iii O vencedor da partida ganha esse número de pontos

VIII. Interface

VIII.i Durante cada mão os seguintes itens são exibidos na tela:

- Nome do jogador da vez;
- Placar da partida;
- Quantos pontos está valendo a partida.
- A opção válida ou inválida de dobrar a quantidade de pontos da partida.
- Nome do próximo a jogar.

VIII.ii No menu inicial, será exibido na tela o que poderá ser escolhido:

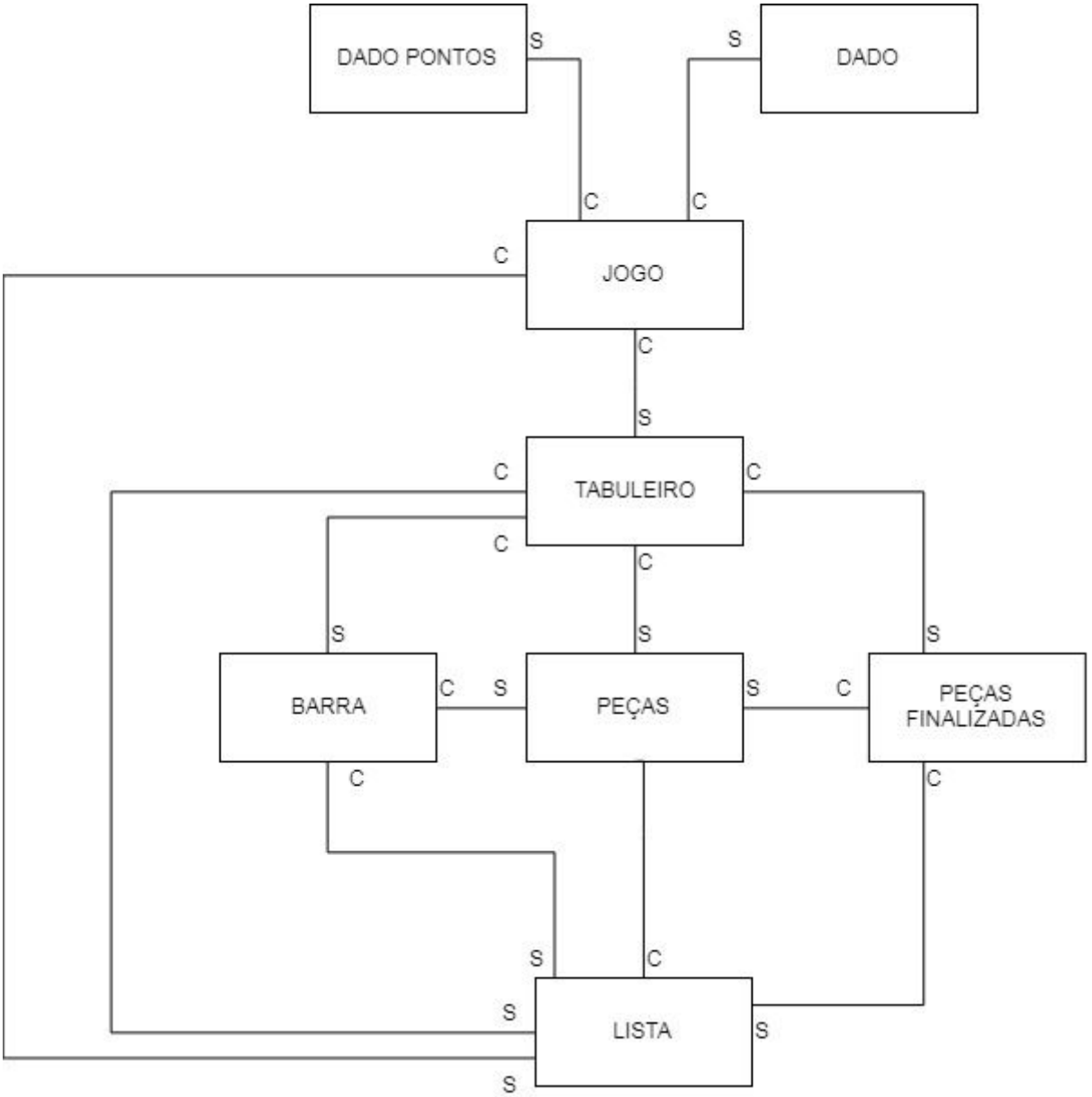
- A opção de continuar um jogo existente.
- A opção de começar um jogo novo.

1.2 Requisitos não-funcionais

I. O sistema deve ser implantado na linguagem C.
II. A aplicação poderá ser utilizada quando desejar.
III.Com auxílio do Arcabouço de teste automatizado todos os módulos(com exceção do módulo jogo, que não precisa do arcabouço) são testados individualmente garantindo a corretude dos módulos.
IV.Os módulos e as funções devem ser desenvolvidos utilizando os métodos e técnicas apresentados no curso, garantindo o fácil entendimento e manutenção.
V. Módulos foram implementados de modo com que sejam reutilizáveis em outros programas.
VI. Houve a utilização de módulos pertencentes ao Arcabouço com o objetivo de otimizar a criação do programa.
VII.O jogo poderá ser interrompido, e ser restaurado posteriormente com as mesmas configurações e pontuações.
IX. Não precisa ser instalado.

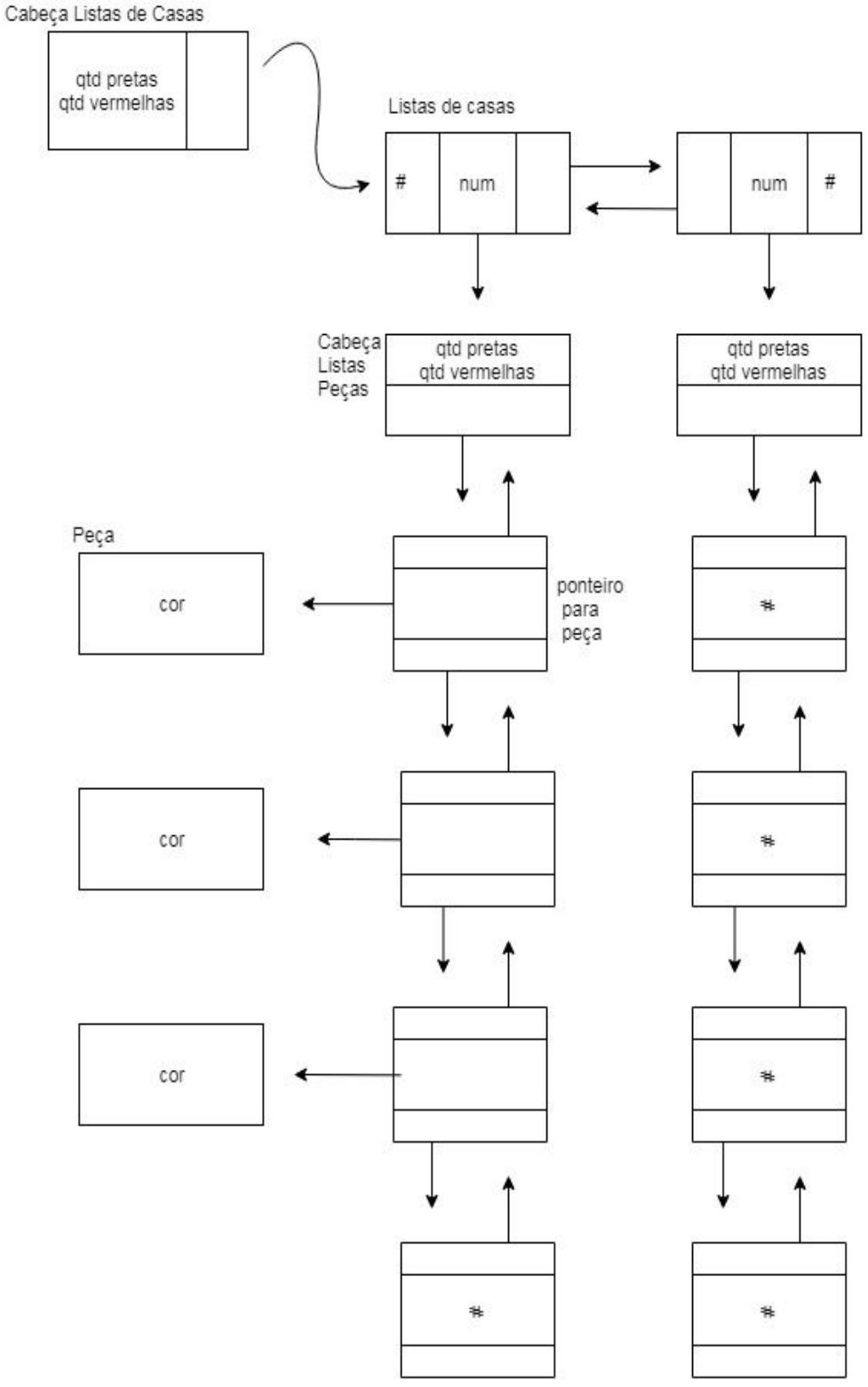
2- Modelo conceitual

Legenda:
C - Cliente
S - Servidor

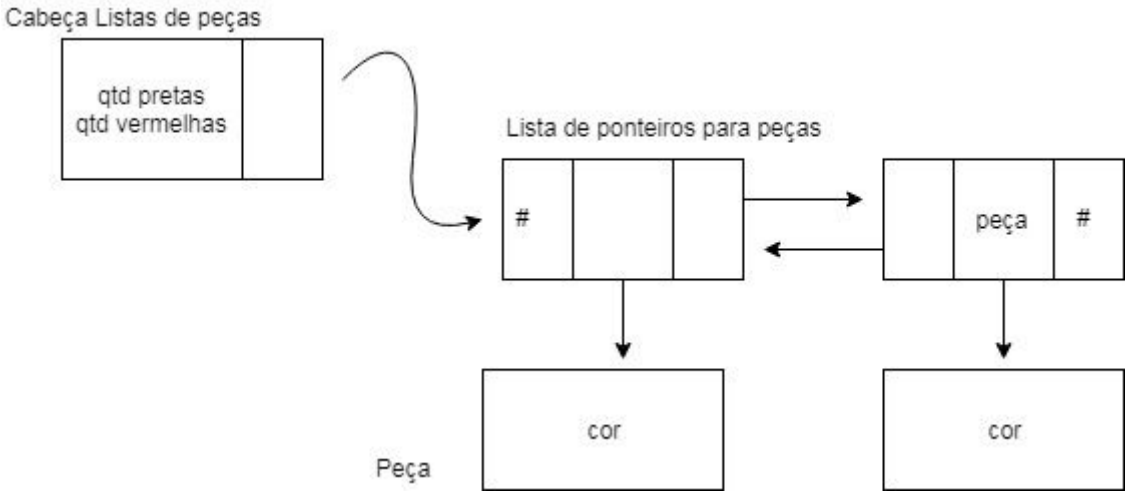


3- Modelo físico

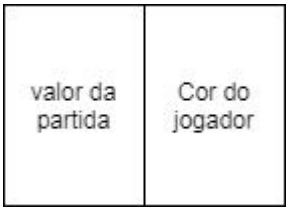
Tabuleiro



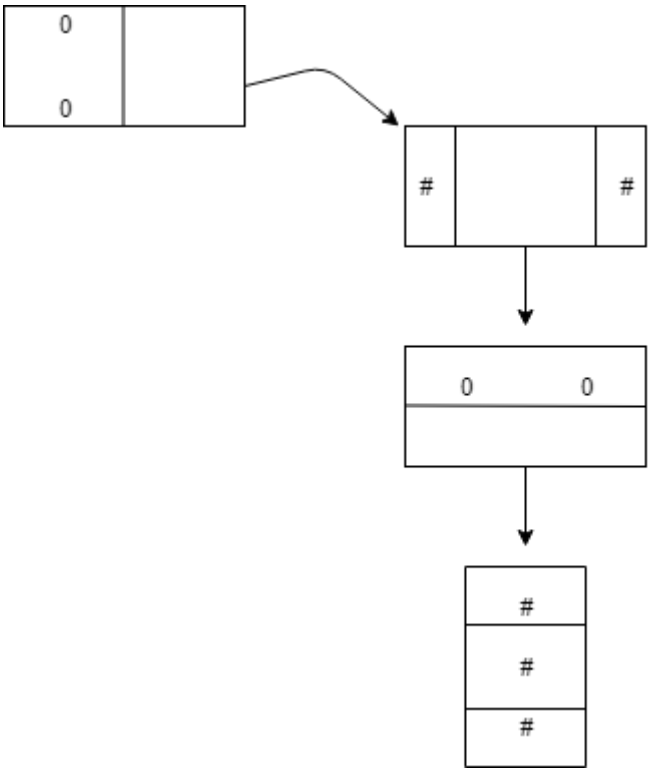
Peças Finalizadas/Barra



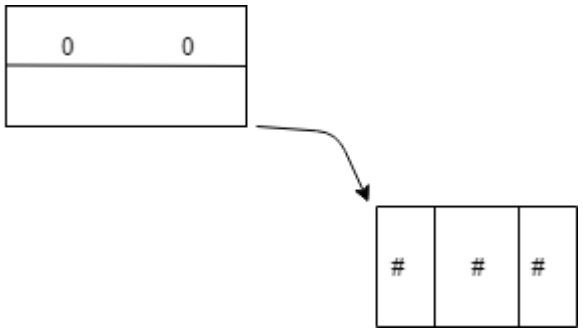
Dado Pontos



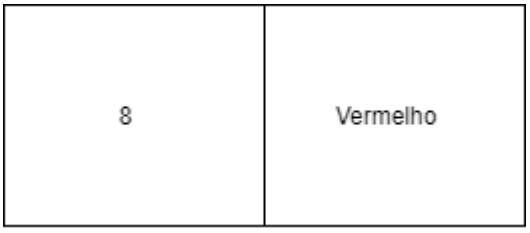
Exemplo Tabuleiro (tabuleiro vazio):



Exemplo Barra/Peças Finalizadas (peças finalizadas/barra vazio):



Exemplo Dado Pontos:



4- Assertivas estruturais

Peças Finalizadas:
CabecaFinalizada é a estrutura para a cabeça das peças finalizadas.
Se cabeçaFinalizada != NULL, aponta para no_finalizadas, uma lista duplamente encadeada que pode conter de 0 a 15 elementos.
As peças que foram para a estrutura de Peças Finalizadas não podem mais voltar pro jogo.

Tabuleiro:
Tabuleiro é o ponteiro para estrutura da cabeça do tabuleiro.
Se tabuleiro != NULL aponta para Casa. Casa consiste em uma lista duplamente encadeada. Devem haver exatamente 24 casas.
Cada cada casa aponta para uma estrutura chamada Triangulo que é a cabeça para a lista de peças.
Se Triangulo!=NULL aponta para CasaTriangulo que consiste em uma lista duplamente encadeada que pode conter 0 ou mais peças.
Quantidade de peças azuis e vermelhas deve ser 15 no inicio da partida, distribuidas pelo tabuleiro..
Caso inicializado, O tabuleiro consiste em uma lista com 24 casas e cada casa apontando para uma outra lista, que são as peças.

Peça
Um jogador só pode ter uma cor : preto(0) ou vermelho(1).

Dado Pontos
Contém informações do último jogador que dobrou, o mesmo não pode dobrar duas vezes seguidas. E também armazena o valor atual da partida.

Peça Barra
Cabeca_barra é o ponteiro para estrutura da cabeça das peças na barraa.
Se Cabeca_barra!= NULL aponta para no_barra, que consiste em uma lista duplamente encadeada. Pode conter de 0 a 15 elementos.

O jogador só pode mover as peças do tabuleiro quando não houverem peças suas na barra.

5- Funções de interface e Assertivas

Legenda:
AE = Assertiva de Entrada **AS = Assertiva de Saída**

BARRA.H

- void criaBarra();

AE: O struct cuja função usa necessita de seu formato definido
AS: A função retorna void.
Se ocorreu corretamente, A função aloca um espaço e cria a cabeça para a barra que aponta para o início de uma lista vazia.
Senao, nao aloca espaço.

- void adicionaPecaBarra(int corPeca);

AE: O struct com a estrutura para barra tem que existir.
Recebe um inteiro que indica a cor de uma peça que será adicionada, tem que ser um número válido.
AS: A funçao retorna void.
Se ocorreu conforme o esperado, é adicionado uma peça a lista de barra.
Senao, nao adiciona.

- void tiraPeca(int corPeca);

AE: O struct com a estrutura para barra tem que existir.
Recebe um inteiro que indica a cor de uma peça que será removida, tem que ser um número válido.
AS:A funçao retorna void.
Se ocorreu conforme o esperado, é removido uma peça a lista de barra.
Senao, nao remove.

- void qtdCadaJogadorBarra(int* qtdPreto, int* qtdVermelho);

AE:O struct com a estrutura para barra tem que existir.
Os ponteiros de inteiro recebidos tem que ser válidos

AS:A função retorna void.
O conteúdo dos ponteiros são alterados com os valores determinados pelo corpo da função.

DADO.H

- DD_tpCondRet DD_DadoSimples(int * a, int * b);

AE: Os ponteiros de inteiro recebidos tem que ser válidos.
AS:O conteúdo dos ponteiros são alterados com os valores determinados pelo corpo da função.
A função retorna uma condição de retorno específica do módulo.
Se DD_CondRetOK tudo ocorreu conforme o esperado e os valores são diferentes.
Se DD_CondRetDadosIguais Tudo ocorreu conforme o esperado e os valores são iguais

DADOPONTOS.H

- PTS_tpCondRet PTS_CriaDadoPontos( );

AE: A funcao recebe void.O struct cuja função usa necessita de seu formato definido.
AS:A função retorna uma condição de retorno específica do módulo.
Se PTS_CondRetFaltouMemoria, nao foi possivel alocar espaço.

Se PTS_CondRetOK Dobra o valor do jogo.

- void PTS_DestroiDadoPontos( );

AE:O struct a ser excluido tem que existir
AS:Funcao ocorreu conforme esperado

- PTS_tpCondRet PTS_DobraDadoPontos(int * quemdobra);

AE: O struct a ser utilizado tem que existir. O char temque existir e ser válido.
AS: Se PTS_CondRetOK : Funcao ocorreu conforme esperado. Valor do dado foi dobrado, valor da partida foi alterado e o jogador que dobrou na vez foi alterado
Se PTS_CondRetRepetiu : Alteracoes nao foram feitas pois o mesmo jogador não pode dobrar o dado vezes seguidas
Se PTS_CondRetDobraMaxima : Alteracoes não foram feitas uma vez que se atingiu o valor máximo da partida
Se PTs_CondRetErro : Não havia struct a ser utilizado

- PTS_tpCondRet PTS_ObtemPontos(int* Pontuacao);

AE: O struct a ser utilizado tem que existir e não ser vazio. O ponteiro para inteiro em que o valor da partida será armazenado tem que existir.
AS:Se PTS_CondRetOK : Função ocorreu conforme esperado.O valor da partida foi resgatado.
Se PTs_CondRetErro : Não havia struct a ser utilizado.

- PTS_tpCondRet PTS_QuemDobra(char * quemdobra);

AE: O struct a ser utilizado tem que existir e não ser vazio. O ponteiro para char em que o último jogador será armazenado tem que existir.
AS: Se PTS_CondRetOK : Função ocorreu conforme esperado.O último jogador que dobrou foi resgatado.
Se PTs_CondRetErro : Nao havia struct a ser utilizado

LISTA.H

- void adicionaPeca(Triangulo* triangulo, int corPeca);

AE: Ponteiro para a cabeça da casa existe e o char com a cor é válida.
AS: A função retorna void. Se AE válida, adiciona peça da cor já existente na casa.

- void rmvPeca(Triangulo* triangulo, int corPeca);

AE:Ponteiro para a cabeça da casa existe e o char com a cor é válida.
AS:A função retorna void. SE AE válida, remove peça da cor equivalente que está na casa.

- void criaCasaTriangulo(casaTriangulo** casaTri);

AE: Recebe ponteiro de ponteiro válidopara a cabeça da casaTri.
AS: A função retorna void. Se AE válida, aloca o espaço para o elemento tipo lista.

- void criaTriangulo(Triangulo** Tri);

AE: Recebe ponteiro de ponteiro válido para a cabeça de Tri, que contém a quantidade de cad apeça e aponta para o inicio da lista de peças .
AS:A função retorna void. Se AE válida, aloca o espaço para o elemento tipo cabeça.

- void povoaTriangulo(Triangulo* triangulo);

AE: Recebe um ponteiro válido para a casa.
AS: A função retorna void. Se AE, adiciona elementos.

- void qtdCadaCor(Triangulo* triangulo, int* qtdPreto, int* qtdVermelho);

AE:Recebe um ponteiro válido para a casa e dois ponteiros de inteiro válidos.
AS:A função retorna void. SE AE válida, altera a quantidade de elementos de um dos ponteiros recebidos.

- void destroiTriangulo(Triangulo* Tri);

AE:Recebe um ponteiro válido para a cabeça da casa
AS:A função retorna void. SE AE válida, destrói o elemento.

- void destroiCasaTriangulo(casaTriangulo* casaTri);

AE:Recebe um ponteiro válido.
AS:A função retorna void. SE AE válida, destrói o elemento.

- void criaTriangulo(Triangulo** Tri);

AE:Recebe um ponteiro de ponteito válido para a cabeça da casa
AS:A função retorna void. SE AE válida, cria cabeça para lista de peças.

- void destroiTriangulo(Triangulo* Tri);

AE:Recebe um ponteiro válido para a cabeça da casa.
AS:A função retorna void. SE AE válida, destrói o elemento.

- void criaCasaTriangulo(casaTriangulo** casaTri);

AE:Recebe um ponteiro de ponteiro válido para a cabeça da casa.
AS:A função retorna void. SE AE válida, cria lista duplamente encadeada para as peças.

- void destroiCasaTriangulo(casaTriangulo* casaTri);

AE:Recebe um ponteiro válido para a cabeça da casa.
AS:A função retorna void. SE AE válida, destrói o elemento.

PECA.H

- PECA_tpCondRet PECA_CriarPeca (int cor, PECA_tppPeca* pPeca);

AE:Tem que haver espaço na memória. Recebeu um elemento do enum COR em seus parâmetros de entrada, com valor válido. Recebeu um ponteiro para a PECA_tppPeca em seus parâmetros de entrada.

AS: Funcao ocorreu conforme esperado, não resultando em erros. Função criou a peça e guardou ela na variável pPeca. A Função retornou PECA_tpCondRet.

- void PECA_DestruirPeca(PECA_tppPeca *pPeca);

AE: Recebeu um ponteiro para a PECA_tppPeca em seus parâmetros de entrada e PECA_tpp contém uma peça válida
AS:A função retorna void. Função ocorreu conforme esperado, não resultando em erros. Função liberou o espaço de memória alocado para a peça.

- PECA_tpCondRet PECA_ObterCor(PECA_tppPeca pPeca, int *pCor);

AE: Recebeu um ponteiro para a PECA_tppPeca em seus parâmetros de entrada e PECA_tpp contém uma peça válida. Recebeu um ponteiro para o inteiro pCor, onde guardará a cor da peça.
AS: Função ocorreu conforme esperado, não resultando em erros. Função guardou a cor da peça na variável pCor. A Função retornou PECA_tpCondRet.

PECASFINALIZADAS.H

- void criaFinalizadas();

AE: A função não recebe nada.
AS:A função retorna void. Cria lista de peças finalizadas.

- void adicionaPecaFinalizadas(int corPeca);

AE:Recebe um número válido que indica a cor da peça.
AS:A função retorna void. Adiciona uma peça de determinada cor a quantidade de peças finalizadas da mesma cor .

- void qtdPecasFinalizadasJogador(int* qtdPreto, int* qtdVermelho);

AE: Recebe dois ponteiros de inteiro válidos.

AS:A função retorna void. Contudo, altera o conteúdo dos ponteiros para o numero existente de peças finalizadas por cada jogador.

TABULEIRO.H

- int criaTabuleiro( ) ;

AE: Recebe void.
AS: Retorna inteiro. Se Não foi possível alocar o espaço retorna 1. Se foi possível e a função ocorreu conforme o previsto, retorna o e é criado a cabeça para o tabuleiro.

- Casa* criaCasa(int numero);

AE: Recebe um inteiro que indica o número da casa a ser criada.
AS: Se o espaço não for alocado corretamente, retorna NULL. Se ocorrer conforme o previsto, cria-se a casa para o tabuleiro e retorna esse elemento.

- TAB_tpCondRet addPeca(int numeroCasa, int cor);

AE: Recebe dois inteiros, um com um número de casa em que se deseja adicionar a peça, outro com a cor da peça a ser adicionada.
AS: Retorna uma condição de retorno padrão(TAB_CondRetOK) que indica que a função funcionou conforme o esperado

- void addCasaComTriangulo();

AE:Recebe void. Tem que existir uma estrutura de cabeça paratabuleiro e de casas para o tabuleiro.
AS: A função retorna void. A função cria a cabeça para as peças que irão pertencer ao tabuleiro.

- void qtdPecaCasa(int nCasa, int*qtd, int* cor);

AE: Recebe um número para casa e dois ponteiros, um para quantidade de peças e um para a cor.
AS: Retorna void. Altera o conteúdo dos ponteiros com a quantidade de elementos e a cor das peças na casa desejada.

- TAB_tpCondRet movePeca(int casaInicial, int casaDestino) ;

AE:casaInicial e casaDestino sejam números entre 1 e 24.
AS:Se TAB_CondRetAindaExistePecaNaBarra, não é possível mover peça pois existe peça na barra.

Se TAB_CondRetNaoJogadorNaoPossuiPecaNaBarra, o jogador pode jogar pois não possui peças na barra.
Se TAB_CondRetOK Peça é adicionada a casa. Se nela tiver peça do oponente sozinha, esta peça é enviada para a barra. Ou caso tenha peças suficientes no último quadrante, são enviadas para a barra.
Se TAB_CondRetTrianguloBloqueado Não é possível jogar nessa casa.
SeTAB_CondRetNemTodasAsPecasEstaoNoQuadranteFinal não pode realizr o movimento pois
Se TAB_CondRetPecaNaoEncontrada movimento não é realizado pois não foram encontradas peças no local.

- TAB_tpCondRet montaTabuleiro( ) ;

AE: Recebe void.
AS: Se TAB_CondRetFaltouMemoria não foi possível criar o tabuleiro por falta de memória.
Se TAB_CondRetOK a função ocorreu conforme o esperado, tabuleiro foi criado incluindo peças nos locais certos.

- TAB_tpCondRet destroiTabuleiro( ) ;

AE:Recebe void. Estrutura para o tabuleiro deve existir.
AS: TAB_CondRetOK Destruiu o tabuleiro.

- TAB_tpCondRet fimDoJogo( ) ;

AE:Recebe void. Estrutura para o tabuleiro, peças finalizadas devem existir.
AS: Se TAB_CondRetJogadorPretoVenceu jogo termina, jogador preto ganha.
Se TAB_CondRetJogadorVermelhoVenceu jogo termina, jogador vermelho ganha.
Se TAB_CondRetJogoNaoAcabou, jogo não acabou.

- TAB_tpCondRet qtdPecaTabuleiro(int nCasa, int* qtdPreto, int* qtdVermelho) ;

AE:Inteiro com número de casa válido. Dois ponteiros para inteiros válidos.
AS: Se TAB_CondRetCasaInvalida, ultrapassou os elementos, está em uma casainexistente. Se TAB_CondRetNumeroDaCasaInvalido, encontrou a casa desejada porém ela é vazia
Se TAB_CondRetOK, contou a quantidade de elementos do tabuleiro naquela casa.

- TAB_tpCondRet qtdPecaBarra(int* qtdPreto, int* qtdVermelho);

AE: Dois ponteiros para inteiros válidos.
AS: Se TAB_CondRetOK, contou a quantidade de elementos da barra.Altera o conteúdo dos ponteiros.

- TAB_tpCondRet qtdPecaFinalizadas(int* qtdPreto, int* qtdVermelho);

AE:Dois ponteiros para inteiros válidos.
AS:Se TAB_CondRetOK, contou a quantidade de elementos finalizados. Altera o conteúdo dos ponteiros.

6- Argumentação de Corretude

```
AE
TAB_tpCondRet qtdPecaTabuleiro( int nCasa, int* qtdPreto, int*
qtdVermelho )
{
AI1
      Casa* casaAtual ;
AI2
      casaAtual = Tab->primeiraCasa ;
AI3
      while( casaAtual->numeroCasa != nCasa ){
AI6
AE2
            if (casaAtual == NULL){
                  return TAB_TAB_CondRetCasaInvalida;
            }

AI7
```

```
            casaAtual = casaAtual->proximaCasa ;
      }
AS2
AI4
AE3
      if (casaAtual == NULL){
            return TAB_CondRetNumeroDaCasaInvalido;
      }

AI5   qtdCadaCor( casaAtual->ponteiroTriangulo, qtdPreto,
qtdVermelho ) ;
AS3
      return TAB_CondRetOK ;
}
AS
```

Argumentação de seleção

AE2:Existe um elemento chamado ncasa que indica um número referente a uma lista
AS2: Elemento não era vazio, continua a função
AE&&(C==T)->AS
Pela AE se o ponteiro apontar para nulo indica que a lista esta vazia
AE&&(C==F)->AS
Pela AE se nao, procuramos a próxima casa, andamos.
AE3=AE2 : O número de casa tem que ser válido
AS2: Casa atual encontrada tem o numero da casa desejado
AE&&(C==T)->AS
Pela AE se o ponteiro a casa or vazia é um valor inválido
AE&&(C==F)->AS
Pela AE se nao,Encontramos o desejado na casa desejada,

Argumentação da sequência:

AE: Existe dois ponteiros de inteiro para as quantidades de cor de cada casa e um número para a casa.
AS: Se TAB_CondRetNumeroDaCasaInvalido, a casa que tenta acessar não é válida. Se TAB_CondRetOK, as peças da casa desejada foram contadas, a função ocorre como desejado.
Se return TAB_CondRetCasaInvalida a casa não existe.
AI1: A variavel *casaAtual* foi corretamente criada.
AI2: A variavel *casaAtual* aponta para o primeiro elemento da lista
AI3:O elemento corrente da lista é o primeiro
AI4: O elemento corrente está na posição desejada. O elemento corrente é vazio, portanto, não é o elemento desejado. Retorna indicador do que ocorreu.

AI5: O elemento corrente está na posição desejada e é o elemento desejado. O lugar respectivo na lista não é vazio, pode-se realizar a operação de conta. Retorna indicador do que ocorreu.
AI6:O elemento corrente não é o que desejamos. O elemento é vazio, portanto, indica que chegou ao fim e o número desejado não foi encontrado. Retorna indicador do que ocorreu.
AI7:O elemento corrente não é o desejado, buscamos o próximo. Elemento corrente é atualizado.

Argumentação de repetição:

AE⇒AI3
AS⇒AI5
AINV: Existe uma casa que tem que ser encontrada. O elemento corrente aponta para o elemento a visitar.
Existem dois conjuntos teóricos:
- Já visitados
- Não visitados

1) AE⇒AINV
   Pela AE, *casaAtual* aponta para o primeiro elemento da lista. Nesse caso, todos os elementos estão no grupo dos elementos não visitados e o conjunto de visitados está vazio. Vale assertiva invariante.
2) **AE&&(C==F)⇒AI4 E/OU AI5**
   Pela AE nCasa é a casa que buscamos e *casaAtual* = primeira casa do tabuleiro
   Para a condição ser falsa, o número da casa visitada tem que ser igual ao número da casa desejado. Se a casa existir, a sua quantidade é contada. Senão, condição de erro.
3) AE&&(C==T)⊕B⇒AINV
   Pela AE nCasa é a casa que buscamos e *casaAtual* = primeira casa do tabuleiro e nCasa é o elemento que deseja ser encontrado com a execução do bloco B. Se o elemento não for encontrado, o elemento foi visitado e vai para o grupo dos já visitados que não são o número desejado. Vale AINV.
4) AINV&&(C==T)⊕B⇒AINV
   Para que continue valendo aINV, B deve garantir que o elemento visitado ão foi encontrado e *casaAtual* seja atualizado.
5) AINV&&(C==F)⊕B⇒**AI4 E/OU AI5**
   Para C==F, *casaAtual* é encontrado e não precisamos mais percorrer. Vale AS
6) Término
   Como a cada ciclo um elemento passa para o grupo dos já visitados e a quantidade de elementos é finita, a repetição termina em um número finito de passos ou quando a casa desejada é encontrada.