



## **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.

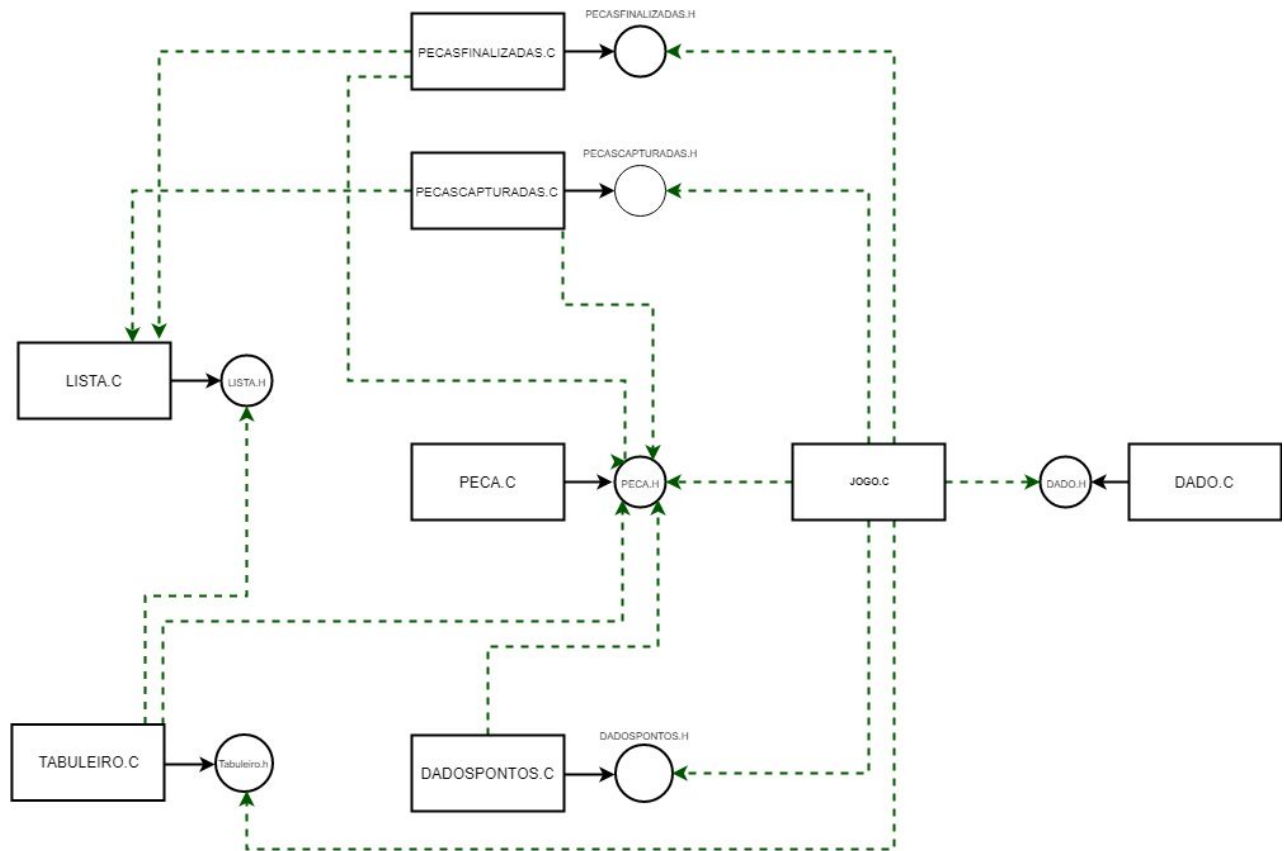
## - Requisitos Não Funcionais

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);

PEC\_CondRet PEC\_ObterCor(PEC\_color \*ret, PecaHead peca);

PEC\_CondRet PEC\_DestruirPeca(PecaHead \*peca);

## 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);

## III - DADOPONTOS.C - DPT:

### i-Tipos

Enum DPT\_CondRet: Enum que enumera os comportamentos das funções

### ii- Funções:

DPT\_CondRet DPT\_CriarDadoPontos(DPT\_DadoHead\* pDadoPontos);  
DPT\_CondRet DPT\_AtualizarJogadorDobra(DPT\_DadoHead pDadoPontos,  
PEC\_color CorPeca);

DPT\_CondRet DPT\_DobrarPontuacaoPartida(DPT\_DadoHead pDadoPontos,  
PEC\_color CorPeca);

DPT\_CondRet DPT\_ObterJogadorDobraPartida(DPT\_DadoHead pDadoPontos,  
PEC\_color \*pCorPeca);

DPT\_CondRet DPT\_ObterPontuacaoPartida(DPT\_DadoHead pDadoPontos, int  
\*pPontuacao);

DPT\_CondRet DPT\_DestruirDadoPontos(DPT\_DadoHead\* pDadoPontos);

## 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);  
TBL_CondRet TBL_DestruirTabuleiro(TabuleiroHead tabuleiro);  
TBL_CondRet TBL_ObterCasas(LIS_tppLista* casas, TabuleiroHead tabuleiro);  
TBL_CondRet TBL_MoverPeca(TabuleiroHead tabuleiro, int casaInicio, int casaFim);
```

## 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(PecaHead peca);  
PFIN_CondRet InserirPecaFinPreta(PecaHead* peca);  
PFIN_CondRet Resetar();
```

## VI - PECASCAPTURADAS.C - PCAP:

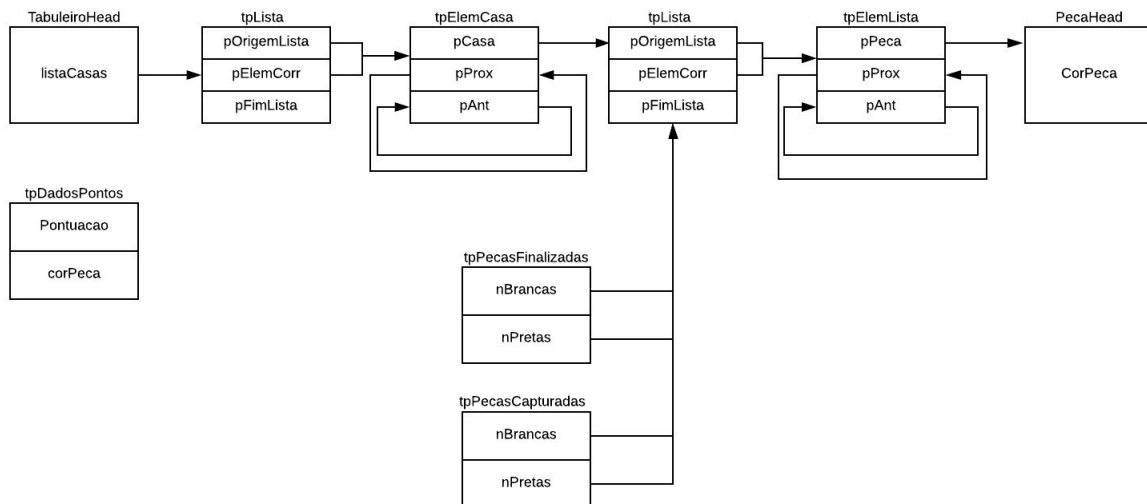
### I - Tipos:

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(PecaHead peca);  
PCAP_CondRet InserirPecaCapPreta(PecaHead* peca);  
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 dois campos de lista duplamente encadeada, encapsulada, para guardar o as peças capturadas, um campo para peças brancas e outra para as pretas.

##### III - PecasFinalizadas:

Estrutura composta dois campos de lista duplamente encadeada, encapsulada, para guardar o as peças finalizadas, um campo para peças brancas e outra para as pretas.

## Exemplo:

