

Projeto Gamão 2019.2

Grupo:

Daniel Faller

Jéssica Pereira

Miguel Sanseverino

Especificação de Requisito de toda a aplicação

Requisitos Funcionais:

A aplicação deve executar um jogo de gamão para dois jogadores.

A aplicação deve armazenar o nome de cada jogador.

A aplicação deve especificar qual jogador deve jogar na rodada.

A aplicação deve executar a rolagem de dois dados de 6 lados para fazer as jogadas.

A aplicação deve exibir o dado de apostas, com numeração de 2, 4, 8, 16, 32 e 64 nas

seis faces.

A aplicação deve contabilizar o valor de cada rodada, que depende do número no dado de

apostas.

A aplicação deve receber as jogadas de cada jogador.

A aplicação deve exibir o tabuleiro do Gamão no início do jogo e depois de cada rodada.

A aplicação deve exibir as peças de cada cor, indicadas com letras correspondentes.

A aplicação deve rejeitar um movimento inválido e exibir um alerta.

A aplicação deve permitir que o jogo seja finalizada a qualquer momento.

A aplicação deve somar os pontos de cada partida, e ao final da execução, exibir o jogador

com mais pontos.

A aplicação deve guardar e exibir as peças que foram capturadas.

A aplicação deve guardar e exibir as peças que foram finalizadas.

Requisitos Não Funcionais:

A aplicação deve ser executável no sistema operacional Windows.

O tempo de resposta após cada jogada não deve ser superior a 10 segundos.

O tabuleiro deve ser exibido em ASCII.

A aplicação deve ser jogada em apenas um computador.

Requisitos Inversos:

A aplicação não terá modo online.

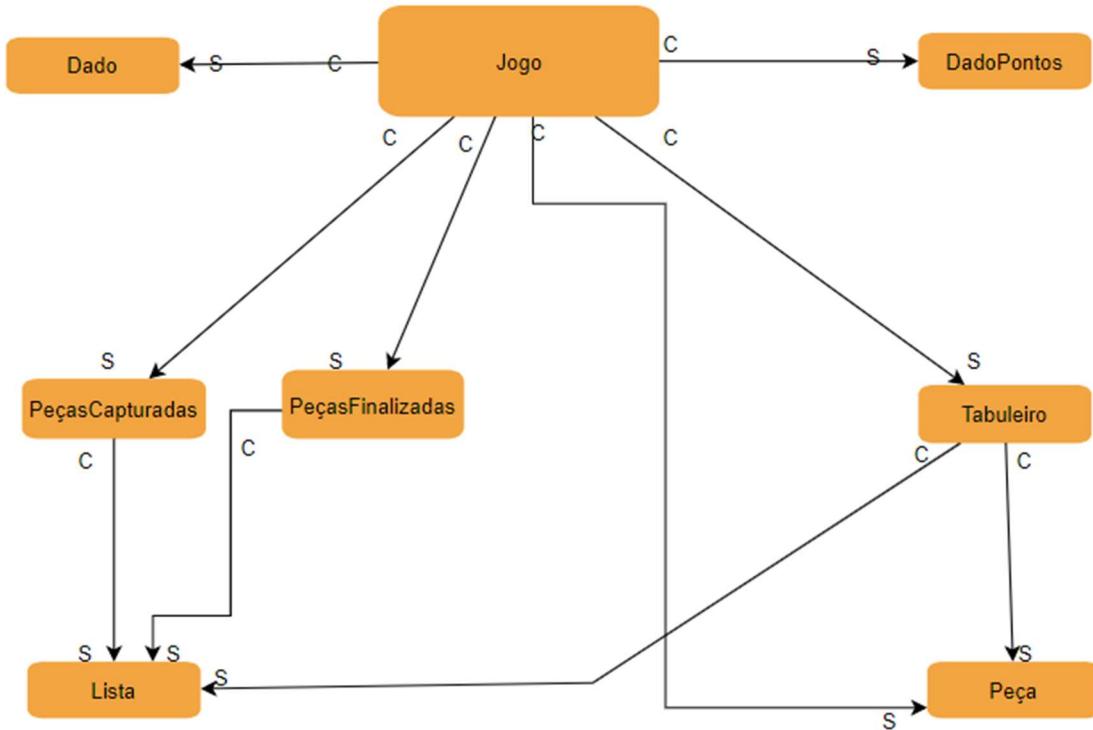
A aplicação não terá interface gráfica dedicada.

A aplicação não terá um modo contra o computador, apenas jogador contra jogador.

A aplicação não será utilizável com o mouse.

A aplicação não armazenará dados do jogo após o fim da execução.

Arquitetura do jogo



Funções de acesso

Módulo Jogo

- JOG_IniciaJogo(void)
- JOG_ResetaJogo(void)
- JOG_RealizaJogada(char playerAtual)

Quando a partida for iniciada deve-se chamar a função JOG_IniciaJogo, esta se encarregará de inicializar suas próprias variáveis, quanto chamar as devidas funções de acesso para atribuir valores necessários aos outros módulos.

- TAB_CriaTabuleiro

- PECC_InicializarPecaCapturada
- PECF_CriarPecaFinalizada
- DAP_CriaDadoPontos.

No caso da inicialização de outra rodada da mesma partida, o comportamento é diferente, chama-se JOG_ResetaJogo, que chamará funções, semelhantes às de inicialização, porém liberando o espaço de memória armazenado para as estruturas prévias.

- TAB_CriaTabuleiro (Esta já verifica a necessidade de destruir estruturas anteriores por padrão)
- PECC_ResetarPecasCapturadas
- PECF_ResetarPecaFinalizada

Ao chamar a função JOG_RealizaJogada, passando o player e o tabuleiro será exibido, e espera-se que o usuário interaja com a interface de forma a executar três possíveis ações, dobrar a quantidade de pontos daquela rodada, desistir, escrever uma posição de origem e destino de uma peça pertencente, movimentando-a conforme os valores que este obteve ao rolar os dados.

- TAB_ExibirTabuleiro
- DAD_RolarDados
- DAP_ObterValorPartida

Primeira sequência de ações:

- DAP_DobrarPontuação

Segunda sequência de ações:

- DAP_DesistirPartida

Terceira sequência de ações

- TAB_MoverPeca

No caso da terceira ser executada, dependendo das posições de origem e destino, poderá ser chamada uma call-back para adicionar tanto ao módulo PeçaFinalizada ou para PeçasCapturadas, afim de adicionar as suas devidas estruturas uma ou mais peças que chegaram ao fim do tabuleiro, ou foram “engolidas”. O módulo jogo verificará então se algum jogador ganhou a partida, dando prosseguimento ao jogo.

Todas as funções de acesso com os devidos atributos.

Módulo Jogo

- JOG_IniciaJogo(void)
- JOG_ResetaJogo(void)
- JOG_RealizaJogada(char playerAtual)
- JOG_PecaCapturada(tpPeca * p)
- JOG_PecaFinalizou(tpPeca * p)
- JOG_Desistir()

Módulo Tabuleiro

- TAB_MoverPeça(char playerCor,int Origem, int Destino)
- TAB_CriaTabuleiro(void)
- TAB_ExibeTabuleiro(void)

Módulo Dado

- DAD_RolarDados(int* Dados)

Módulo DadoPontos

- DAP_CriaDadoPontos(void)
- DAP_DestruirDadoPontos(void)
- DAP_DobrarPontuacao(char valorParm)

-DAP_ObterValorPartida(int * valorParm)

Módulo Peça

- PEC_ObterCorPeca(PEC_tpPeca ** pPeca, char * valorParm)
- PEC_DestruirPeca(PEC_tpPeca ** pPeca)
- PEC_CriarPeca(PEC_tpPeca ** pPeca, char valorParm)
- PEC_ExibePeca(PEC_tpPeca ** pPeca)
- PEC_MudarEstadoPeca(PEC_tpPeca ** pPeca)

Módulo PeçasCapturadas

- PECC_InicializarPecaCapturada(void)
- PECC_AdicionarPecaCapturada(PEC_tpPeca * p)
- PECAP_DestruirPecaCapturada(void)
- PECC_RetirarPecaCapturada(LIS_tppLista * lista)
- PECC_ResetarPecasCapturadas(void)

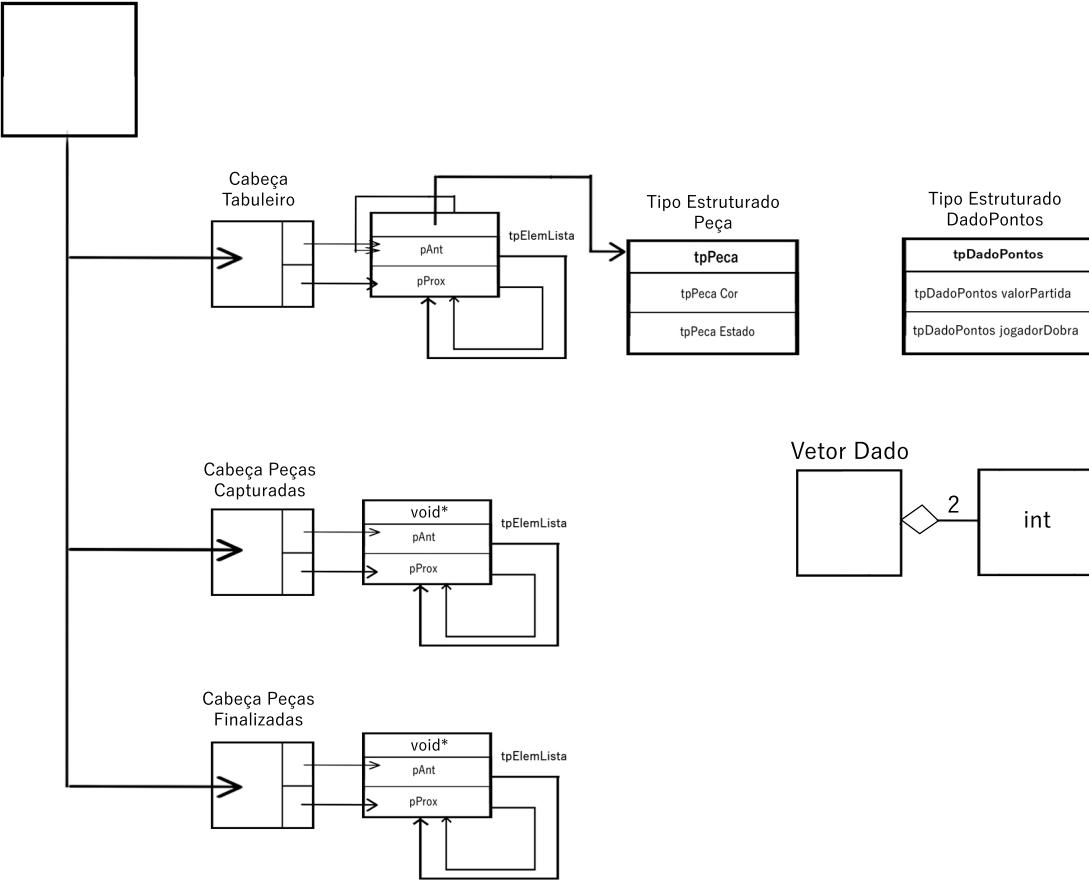
Módulo PeçasFinalizadas

- PECF_CriarPecaFinalizada(void)
- PECF_ResetarPecaFinalizada(LIS_tppLista * lista)
- PECF_AdicionarPeça(PEC_tpPeca * p)
- PECF_ObterCorPecaFinalizada (PEC_tpPeca * p)

Módulo Lista

- LIS_tppLista LIS_CriarLista(void (* ExcluirValor)(void * pDado))
- void LIS_DestruirLista(LIS_tppLista pLista)
- void LIS_EsvaziarLista(LIS_tppLista pLista)
- void LIS_EsvaziarLista(LIS_tppLista pLista)
- LIS_tpCondRet LIS_InserirElementoAntes(LIS_tppLista pLista ,void * pValor)
- LIS_tpCondRet LIS_InserirElementoApos(LIS_tppLista pLista ,void * pValor)
- LIS_tpCondRet LIS_ExcluirElemento(LIS_tppLista pLista)
- void * LIS_ObterValor(LIS_tppLista pLista)
- void IrInicioLista(LIS_tppLista pLista)
- void IrFinalLista(LIS_tppLista pLista)
- LIS_tpCondRet LIS_ProcurarValor(LIS_tppLista pLista , void * pValor)

Cabeça da Partida



Assertivas Estruturais

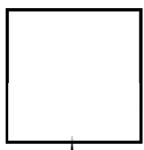
- Cabeça Tabuleiro aponta para uma lista de listas que contém tipos estruturados Peça.
- Cabeça Peças Capturadas aponta para uma lista que contém um ponteiro **void***. Nessa estrutura serão armazenados tipos estruturados Peça que foram capturadas, e que serão repostos no tabuleiro.
- Cabeça Pelas Finalizadas aponta para uma lista que contém um ponteiro **void***. Nessa estrutura serão armazenados tipos estruturados Peça que foram finalizados, e não serão repostos no tabuleiro.

-**tpElemLista** é um elemento de uma lista duplamente encadeada genérica ,em que **pAnt** aponta para **pProx** e **pProx** aponta para ele mesmo. O conteúdo de cada elemento é um ponteiro **void***.

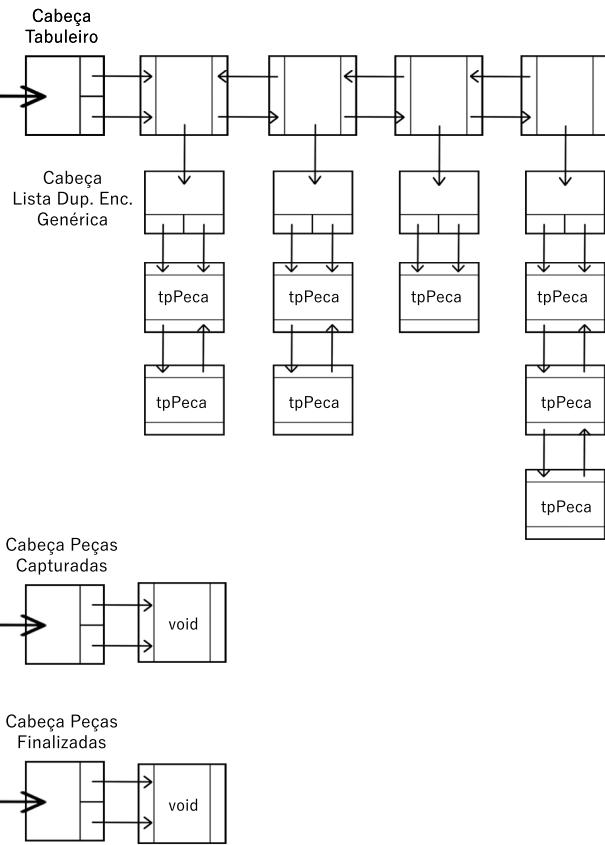
No caso da lista de listas, o conteúdo é um ponteiro para lista cujo conteúdo é um tipo estruturado.

- Vetor Dado é um vetor de 2 elementos **int**.
- Tipo Estruturado DadoPontos possui um **int ValorPartida** e um **char JogadorDobra**.
- Tipo Estruturado Peca possuei um **char CorPeca** e um **int Estado**.

Cabeça da Partida



Exemplo de Estrutura Jogo



Tipo estruturado Peca



Tipo estruturado DadoPontos



Vetor Dado

