



Pontifícia Universidade Católica do Rio de Janeiro

INF1301 Gamão Especificação de Requisitos

Alunos	1711799 - Fernando Tancini 1710635 - Júlia Rocha 1811208 - Suemy Inagaki
Professor	Flávio Heleno Bevilacqua e Silva

Rio de Janeiro, 8 de Maio de 2019

Conteúdo

1	Hitórico de Revisões	1
2	Introdução	1
2.1	Propósito	1
2.2	Escopo	1
3	Características dos Usuários	1
4	Restrições	1
5	Requisitos Funcionais	1
6	Requisitos Não-Funcionais	4
7	Funções de Interface	4
7.1	Lista - LIS	4
7.2	DadoPontos - DPTS	4
7.3	PecasFinalizadas - PFZ	5
7.4	Tabuleiro - TAB	5
7.5	PecasCapturadas - BAR	5
7.6	Pecas - PEC	5
7.7	Dado - DAD	6
7.8	Jogo - GAM	6

1 Hitórico de Revisões

Data	Versão	Descrição	Autor
22/Abril/2019	1.0	Início da inclusão de requisitos funcionais	Suemy Inagaki
02/Maio/2019	1.0	Documentação completa dos requisitos	Suemy Inagaki

2 Introdução

2.1 Propósito

Este documento especifica os requisitos contemplados pelo jogo de Gamão, fornecendo todas as informações necessárias para o projeto, *software* e testes.

2.2 Escopo

O documento descreve os casos de uso do Gamão, que permite que dois jogadores realizem várias partidas, um contra o outro, em um único computador, de forma que os resultados ao longo de um mesmo jogo serão salvos em um arquivo que poderá ser visualizado antes da finalização do jogo.

3 Características dos Usuários

O Gamão será utilizado por dois perfis de usuário, os *jogadores*, que terão permissão para rolar os dados, mover as peças e visualizar os resultados das partidas jogadas.

4 Restrições

- No início de cada jogo devem ser cadastrados exatamente dois jogadores

5 Requisitos Funcionais

R01 Tabuleiro: O jogo conterà um tabuleiro de 24 casas, divididas em 4 quadrantes de 6 casas cada. Os dois quadrantes da direita são chamados de *internos*. São 12 casas na parte superior e 12 casas na parte inferior.

R02 O programa criará peças vermelhas e pretas, cada uma delas representando um jogador, que serão movimentadas ao longo de cada partida.

R03 Disposição das peças no início de cada partida:

Todas as casas são contadas da direita para a esquerda

- 5 peças vermelhas na 12^a casa da parte superior.
- 5 peças vermelhas na 6^a casa da parte superior.
- 2 peças vermelhas na 1^a casa da parte superior.
- 3 peças vermelhas na 8^a casa da parte inferior.
- 5 peças pretas na 12^a casa da parte inferior.
- 5 peças pretas na 6^a casa da parte inferior.
- 2 peças pretas na 1^a casa da parte inferior.
- 3 peças pretas na 8^a casa da parte superior.

Total: 15 peças vermelhas e 15 peças pretas no tabuleiro.

R04 O jogo possui dois dados para a movimentação das peças, cada um deles numerados de 1 a 6.

R05 Identificar os jogadores: Os jogadores deverão inserir, cada um, um nome no início de um jogo, antes de começarem alguma partida. *Um jogo contém uma ou mais partidas de gamão.*

R06 O programa deve permitir que os jogadores sejam capazes de iniciar uma partida.

R07 Decidir quem começa: no início de cada partida, cada jogador deve lançar os dois dados e, aquele que obtiver o maior valor na soma dos dois dados, iniciará a partida. Em caso de empate os dados devem ser lançados até que se defina o primeiro jogador.

R08 A escolha das cores: O jogador que iniciará a partida jogará com a cor vermelha. O segundo jogador jogará com a cor preta.

R09 O programa informará de qual jogador é a vez de jogar, do jogador 1, ou do jogador 2.

R10 Pontuação: Uma partida deve ser iniciada valendo 1 ponto. Um jogador pode dobrar o valor da partida em qualquer momento. Se o adversário recusar a partida será encerrada e o jogador que propôs a dobra vence. Se o adversário aceitar, ele também poderá propor uma dobra.

- R11** No início de cada jogada, o programa deve solicitar que o jogador da vez lance os dados.
- R12** Jogadas Duplas: Se os valores obtidos nos dois dados lançados forem iguais, a jogada passa a ter 4 movimentações.
- R13** O programa dirá quantas peças existem de cada cor em cada casa no tabuleiro.
- R14** O programa deve ser capaz de identificar os possíveis movimentos do jogador da vez em uma jogada, e deve ser capaz de não permitir os movimentos impossíveis. É uma movimentação válida quando a casa de destino está vazia, quando a casa de destino somente tem peças da cor do jogador, ou quando a casa de destino só tem uma única peça do jogador adversário. Nesse caso a peça do adversário é capturada.
- R15** Movimentação das Peças: Dados uma casa de origem e uma casa de destino válidas, o jogador poderá movimentar a peça.
- R16** Impedir movimenações inválidas: São ditas movimenações inválidas aquelas em que a cada destino não está livre, ou aquela cujo valor obtido no dado não é suficiente para a movimentação requerida pelo jogador. Todas as movimentações inválidas serão notificadas, e o jogador deverá escolher uma movimentação válida.
- R17** Capturar peças: Se uma peça estiver sozinha em uma casa, o adversário poderá capturá-la ao movimentar uma peça dele para lá. O programa deve ser capaz de identificar quando uma peça foi capturada, e ser capaz de alocá-la para o *BAR*.
- R18** Retorno da peça: Cada peça capturada deve ser devolvida a uma casa livre antes de qualquer outra movimentação.
- R19** Retirada das Peças: As peças de um jogador podem ser retiradas caso todas as peças dele estiverem no seu quadrante interno.
- R20** Finalização da Partida: O programa deve ser capaz de identificar a vitória de um jogador, quando ele retirar todas as suas peças do tabuleiro.
- R21** O programa deve permitir que os jogadores encerrem um jogo, desde que todas as partidas estejam encerradas.

R22 O programa deve permitir que os jogadores acessem ao arquivo contendo os resultados das partidas jogadas no jogo, desde que este ainda não esteja encerrado. Após o encerramento do jogo, o arquivo é apagado.

6 Requisitos Não-Funcionais

RN01 O programa foi desenvolvido para ser executado em sistemas operacionais Windows 7 ou superior, e não é necessário nenhuma instalação.

7 Funções de Interface

7.1 Lista - LIS

```
LIS_tpplista LIS_CriarLista(void (* ExcluirValor)(void *pDado));  
void LIS_DestruirLista( 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_AvancarElementoCorrente( LIS_tpplista pLista ,int numElem);  
LIS_tpCondRet LIS_ProcurarValor( LIS_tpplista pLista ,void * pValor);
```

7.2 DadoPontos - DPTS

```
DPTS_CondRet DPTS_CriarDadoPontos(DPTS_DadoPCabeca *pDadoPontos);  
DPTS_CondRet DPTS_JogadorDobraAtualiza(DPTS_DadoPCabeca pDadoPontos, char CorPeca);  
DPTS_CondRet DPTS_DobrarPontuacaoAtual(DPTS_DadoPCabeca pDadoPontos, char CorPeca);  
DPTS_CondRet DPTS_ObterJogadorDobraPonto(DPTS_DadoPCabeca pDadoPontos, char *pCorPeca);  
DPTS_CondRet DPTS_ObterPontuacaoPartida(DPTS_DadoPCabeca pDadoPontos, int *pPonto);  
DPTS_CondRet DPTS_DestruirDadoPontos(DPTS_DadoPCabeca *pDadoPontos);
```

7.3 PecasFinalizadas - PFZ

```
PFZ_tpCondRet PFZ_CriarListaPecasFinalizadas(PFZ_tpPecasFinalizadas
                                              **pPecasFinalizadas);
PFZ_tpCondRet PCF_InserirPeca(PFZ_tpPecasFinalizadas *pPecasFinalizadas,
                              PCA_tpPeca *pPeca);
PFZ_tpCondRet PFZ_ContaPecas(PFZ_tpPecasFinalizadas *pPecasFinalizadas,
                              char CorPeca, int *pContagem);
PFZ_tpCondRet PFZ_DestruirListaPecasFinalizadas(PFZ_tpPecasFinalizadas
                                              **pPecasFinalizadas);
```

7.4 Tabuleiro - TAB

```
TAB_tpCondRet TAB_CriarTabuleiro( void );
TAB_tpCondRet TAB_DestruirTabuleiro( void );
TAB_tpCondRet TAB_InserirPecasCasa(int n, char cor, int nCasa);
TAB_tpCondRet TAB_RemovePecasCasa(int n, int nCasa);
TAB_tpCondRet TAB_NumPecasCasa(int nCasa, int* numPecas);
TAB_tpCondRet TAB_CorPecasCasa(int nCasa, char* cor);
```

7.5 PecasCapturadas - BAR

```
BAR_tpCondRet BAR_CriarListaPecasCapturadas(BAR_tpPecasCapturadas
                                              **pPecasCapturadas);
BAR_tpCondRet BAR_InserirPeca(BAR_tpPecasCapturadas *pPecasCapturadas,
                              PCA_tpPeca *pPeca);
BAR_tpCondRet BAR_RemoverPeca(BAR_tpPecasCapturadas *pPecasCapturadas,
                              char CorPeca, PCA_tpPeca **pPeca);
BAR_tpCondRet BAR_ContaPecas(BAR_tpPecasCapturadas *pPecasCapturadas,
                              char CorPeca, int *pContagem);
BAR_tpCondRet BAR_DestruirListaPecasCapturadas(BAR_tpPecasCapturadas
                                              **pPecasCapturadas);
```

7.6 Pecas - PEC

```
PEC_tpCondRet PEC_CriarPeca(tppPeca * PECCriado, char CorDaNovaPeca);  
void PEC_DestruirPeca(void * Peca);  
PEC_tpCondRet PEC_ObterCor(tppPeca Peca, char *cor);
```

7.7 Dado - DAD

```
DAD_tpCondRet DAD_Jogar(int *Numero);
```

7.8 Jogo - GAM

Este módulo não possui funções externas