 

Documentação - FreeCell

Aplicação de Pilha. Estrutura de Dados 2017.

Professor Roberto Ferrari.

Ciência da Computação.

Alunos:

Alisson Nunes Vieira Amâncio

Gabriel De Souza Alves

Matheus Bortoleto Da Silva

Rafael Sales Pavarina

Universidade Federal de São Carlos

São Carlos, 2017

# 1 - Autores:

## 1.1 - Nomes:

* Alisson Nunes Vieira Amancio
  + RA: 725862
  + Email: [alynva@gmail.com](mailto:alynva@gmail.com)
* Gabriel de Souza Alves
  + RA: 726515
  + Email: [g4briel.4lves@gmail.com](mailto:g4briel.4lves@gmail.com)
* Matheus Bortoleto da Silva
  + RA: 726570
  + Email: [Matheus.silva.ufscar@gmail.com](mailto:Matheus.silva.ufscar@gmail.com)
* Rafael Sales Pavarina
  + RA: 726583
  + Email: [rspavarina@gmail.com](mailto:rspavarina@gmail.com)

## 1.2 - Área de Atuação:

* Alisson Nunes Vieira Amâncio: Interface e parte gráfica.
* Gabriel De Souza Alves: Implementação da Estrutura dos Dados.
* Matheus Bortoleto Da Silva: Interface e parte gráfica.
* Rafael Sales Pavarina: Documentação.

# 2 – O jogo:

O jogo desenvolvido para o primeiro projeto de Estruturas de Dados foi o jogo de cartas FreeCell. Neste jogo um baralho de 52 cartas é distribuído em 8 pilhas, 4 delas com 7 cartas e as outras 4 com 6 cartas, o jogo também conta com 4 espaços disponíveis para guardar uma carta temporariamente e 4 espaços inicialmente vazios, que deverão receber as cartas do baralho divididas em seus naipes e organizados de maneira crescente.

Durante o jogo, o jogador deve manipular as cartas movendo-as de uma pilha para a outra, respeitando as seguintes condições: A carta a ser movida deve ser menor que a carta do topo da pilha destino, A carta a ser movida deve ser de cor diferente da carta do topo da pilha destino. Satisfeitas as condições, o jogador consegue empilhar novas cartas nas pilhas desejadas e proporcionar uma maior organização das cartas que originalmente foram distribuídas aleatoriamente.

Conforme as pilhas são organizadas, o jogador consegue ter acesso à cartas necessárias para iniciar as pilhas de naipes organizados, inicialmente as pilhas estão vazias e são iniciadas no momento em que o jogador consegue enviar um Ás para uma das 4 pilhas, após isto, a carta seguinte a ser empilhada na pilha de naipes deve possuir o mesmo naipe do Ás (ou da carta anterior àquela) e deve possuir o valor seguinte à atual carta do topo (se na pilha de naipes desejada, a carta do topo for um 5 de paus, a pilha irá aceitar uma carta se e somente se, ela for o 6 de paus).

Ao conseguir enviar as 52 cartas, divididas em 4 naipes com 13 cartas cada, para suas respectivas pilhas definitivas, o jogador ganha a partida e tem a opção de sair ou começar um jogo novo.

# 3 – Estruturação dos Dados:

O jogo foi desenvolvido seguindo as configurações do tipo abstrato de dado (TAD) pilha duplamente encadeada com nó header. No jogo existem as seguintes estruturas:

8 pilhas intermediárias inteligentes: pilhas que são manipuladas para melhor organização das cartas antes de serem enviadas para seu destino final. Uma carta pode ser inserida se e somente se: seu valor é igual ao topo-1 e de cor diferente (copas e ouros = vermelho; espadas e paus = preto);

4 pilhas definitivas inteligentes: pilhas destinadas aos naipes já organizados indicando o progresso na partida. Uma carta pode ser inserida se e somente se: seu naipe é igual ao naipe da pilha destino e se seu valor é igual ao topo+1;

4 pihas auxiliares de 1 espaço cada: pilhas localizadas no canto superior esquerdo, cada espaço admite apenas um elemento temporário, para ajudar no decorrer do jogo;

Além de pilhas auxiliares para ajudar na movimentação de mais de uma carta por vez ( caso hajam subsequências de cartas que possam ser movidas).Tem-se então que cada elemento é armazenado em um nó com três atributos, sendo eles:

* Value: armazena o elemento em si;
* Dir: ponteiro que define o nó do próximo elemento;
* Esq: ponteiro que define o nó do elemento anterior;

Essa pilha tem as funções mais básicas, para a utilização desse TAD temos as funções mais básicas, que são:

* IsEmpty(): retorna um booleano, com valor verdadeiro se a pilha está vazia (isto é o atributo Dir do Header aponta para o Header), e valor falso se a pilha está cheia.
* Push(element, &check): insere element no topo da pilha, e check recebe verdadeiro, se foi possível inserir, ou verdadeiro, se não foi possível.
* Pop(&element): remove o elemento do topo da pilha e passa para element. A função retorna verdadeiro se foi possível remover o elemento no topo (i.e. a pilha não estava vazia), ou falso caso não foi possível.
* clear(): limpa a pilha, removendo todos os elementos, deixando apenas o nó de header.
* getSize(): retorna o tamanho atual da pilha.
* peek(): retorna um ponteiro apontando para o nó no topo da pilha.

Com este tipo básico, derivam-se as funções específicas, responsáveis pelas pilhas do Freecell, a pilha inteligente. Esta implementação herda do tipo abstrato pilha definido anteriormente. Sua função é garantir métodos mais complexos para as pilhas do jogo. Tem-se portanto, que os atributos da pilha também serão herdados, e além deles, existem os seguintes:

* coord: posição dessa pilha na janela criada.
* backTexture: textura da base dessa pilha.
* stateHover: define se o mouse se encontra sobre essa pilha.