Implementação modularizada do Jogo Freecell utilizando estruturas de dados lista em C

Especificação de Requisitos
Modelo da Arquitetura
Interface dos módulos
Modelo Físico & Exemplo

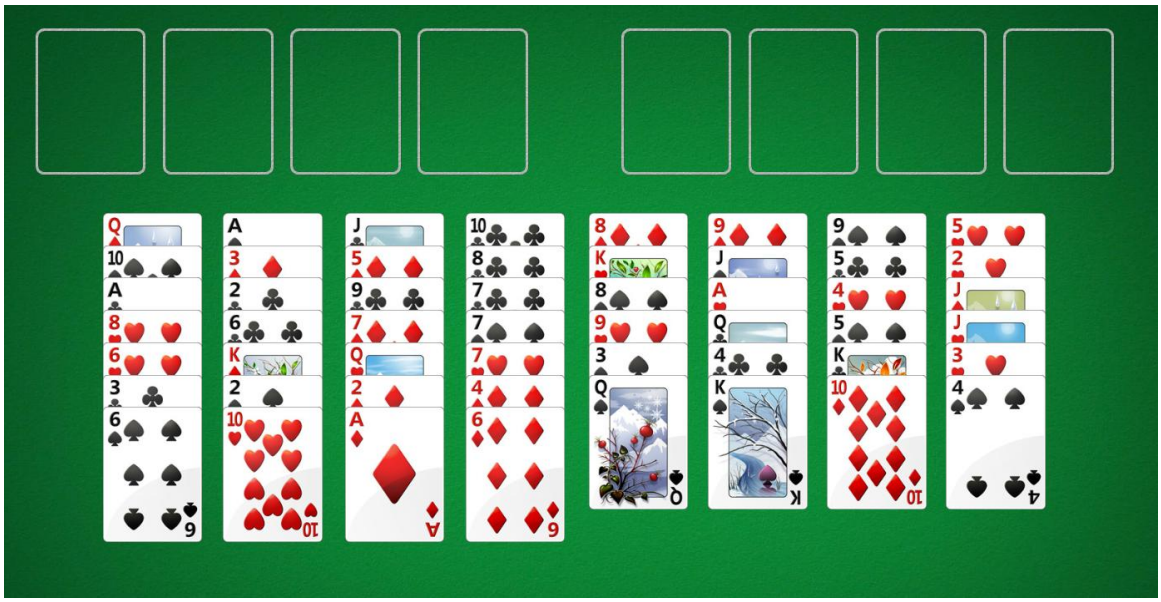
[INF1301 – 3WA] PROGRAMAÇÃO MODULAR 2013.2
Pontifícia Universidade Católica do Rio de Janeiro

Professores	Alessandro Garcia	
	Eiji Adachi	
	Willian Oizumi	
Integrantes	Gabriel Barros	1111061
	Leonardo Giroto	1210817
	Noemie Nakamura	1110743

Especificação de Requisitos

Disposição e Objetivo

FreeCell é um jogo de baralho para apenas um jogador. Inicialmente as 52 cartas de um baralho comum são embaralhadas e distribuídas à mostra em 8 colunas paralelas. São as cartas A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, cada uma disponível em um naipe, que são Paus (♣), Copas (♥), Espadas (♠) e Ouros (♦). A figura abaixo mostra tal disposição de um jogo qualquer:



O objetivo é criar quatro pilhas de 13 cartas, cada pilha de um naipe, da carta mais baixa (A) à mais alta (K), organizando-as nos quatro espaços reservados no canto superior direito da tela. O jogador dispõe de quatro espaços no canto superior esquerdo que servem de estoque temporário de cartas, a fim de ajudar em tal organização. Estes são os espaços livres ou *free cells*, que dão origem ao nome do jogo, e podem, cada um, segurar uma carta qualquer por vez.

Movimentações

O jogador só poderá movimentar uma carta de cada vez. Esta movimentação, porém, deve seguir as regras que serão a seguir apresentadas.

Ao movimentar uma, esta pode ser colocada sobre outra carta que esteja imediatamente acima dela na sequência, de cor oposta e sem cartas filhas. Por exemplo, um 9♣ poderia ser colocado sobre um 10♥ ou 10♦ que não tenha nenhuma carta sobre ele. Contudo, para que a carta possa ser movida é necessário que seja a última carta na coluna ou que suas filhas (cartas abaixo) já estejam ordenadas em ordem decrescente e de cor oposta, isto é, se todas as cartas filhas também podem ser removidas. Por exemplo, num grupo 9♣-8♥-7♠ o jogador poderia remover o 9♣.

O jogador pode também movimentar uma carta para um espaço do estoque a qualquer momento do jogo.

Finalmente, o jogador pode movimentar uma carta para o espaço reservado para a sequência ordenada de naipes. Estes espaços originalmente não são exclusivos de um naipe até que o A seja inserido. A partir de então, o naipe deste A define qual será o naipe daquele espaço e da sequência que irá armazenar. O jogo termina quando as 52 cartas estão ordenadas por naipe nestes quatro espaços, se não há possibilidade de movimentos ou se o jogador desistir.

O que implementaremos?

Requisitos funcionais

- O jogo deve apresentar uma interface tipo menu, que apresenta as opções de jogada ao jogador. Estas opções podem ser: Movimentar, Exibir Mesa, Desistir.
- As cartas serão representadas por suas respectivas letras ou números e seus naipes pelos símbolos especiais da tabela ASCII que representam os naipes, de modo que serão variáveis string com até 3 caracteres.
- Ao selecionar a opção de movimentar, o jogador deve informar a origem, a carta e o destino para onde quer que seja movimentada.
- Colunas do tipo naipe não são de um determinado naipe até que seja inserida a primeira carta, e após inserção não será possível remoção.

Requisitos não-funcionais

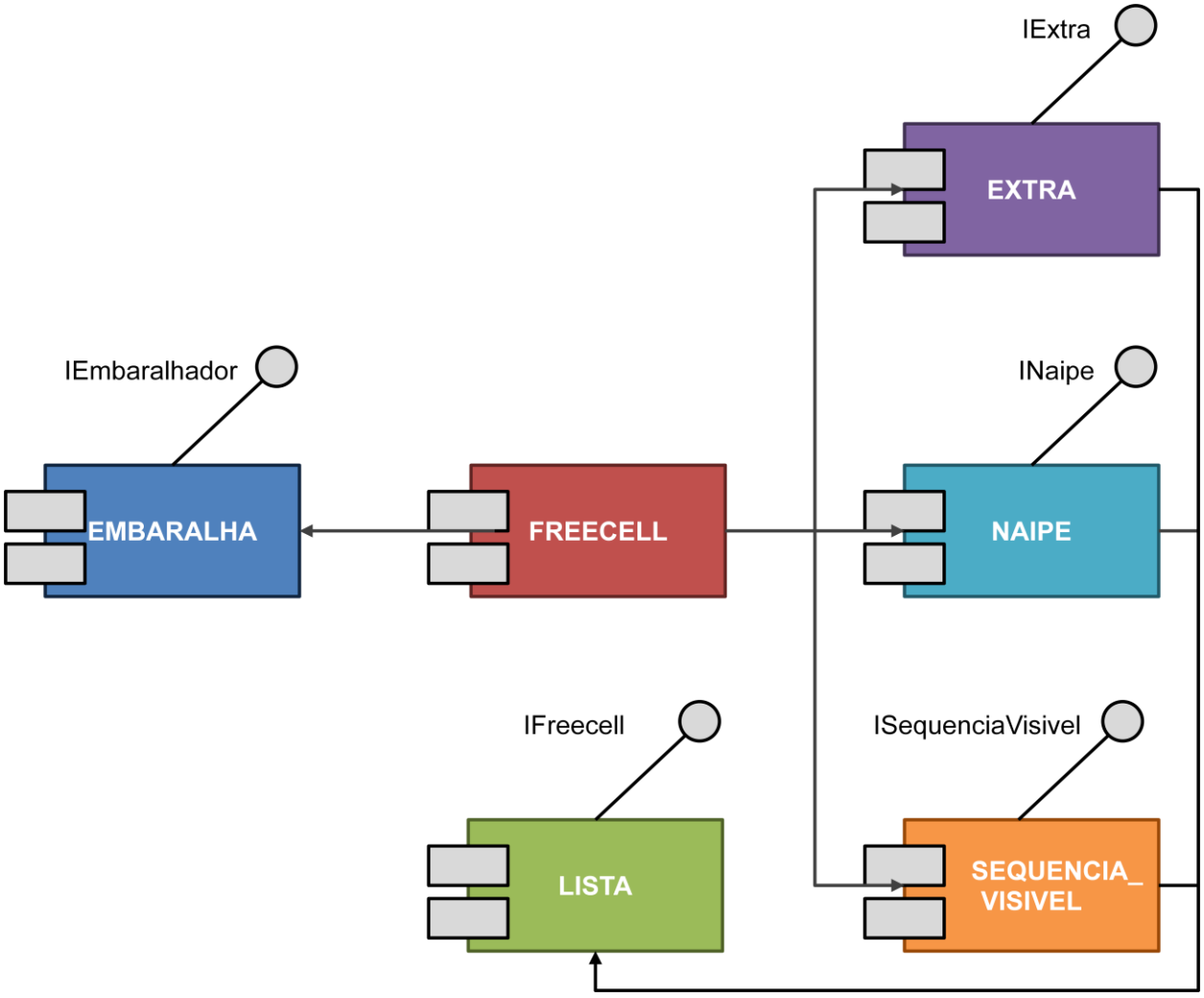
Procuraremos seguir os atributos de qualidade de um programa, tais como:

- Robustez. O jogador sempre será informado quanto à impossibilidade do movimento.
- Manutenibilidade e reusabilidade. Dividiremos o programa, a princípio, em seis módulos, que serão:
    - LISTA, sob o qual montaremos a estrutura do jogo;
    - FREECELL, responsável por exibir a mesa ao jogador, receber o movimento desejado, exibir eventuais mensagens como erros, impossibilidade de movimento, término do jogo;
    - EMBARALHA, responsável por distribuir as cartas randomicamente no inicio do jogo;
    - SEQUENCIA_VISIVEL, que irá receber as 52 cartas inicialmente embaralhadas;
    - EXTRA, que irá receber cartas quaisquer ao longo do jogo;
    - NAIPE, que irá receber as cartas ordenadas por naipe.

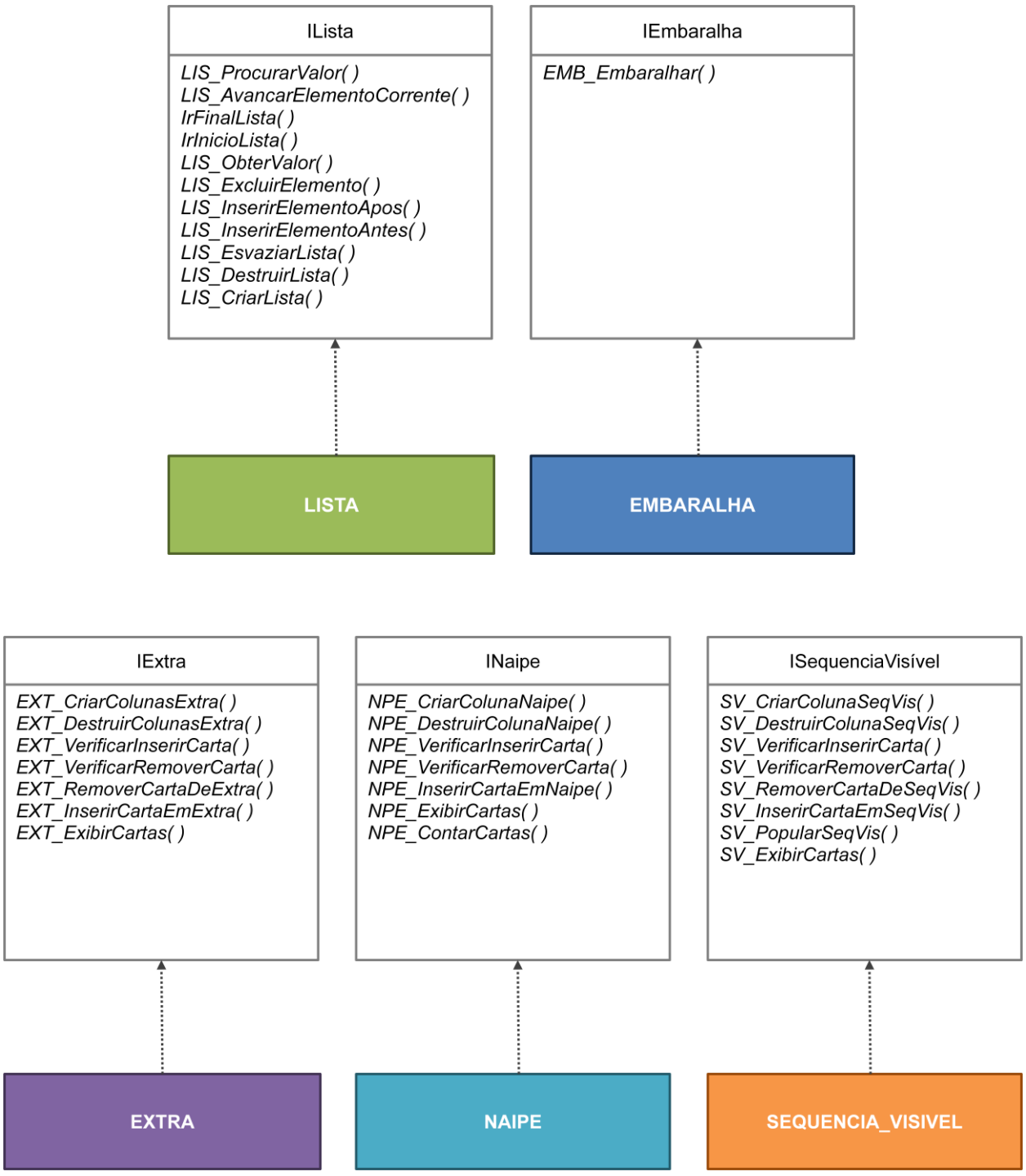
O que não implementaremos?

- Interface gráfica
- Informações e estatísticas do jogador
- Desfazer movimento
- Resolver o jogo
- Salvar o jogo
- Reiniciar o mesmo jogo
- Dicas de movimentação
- Aviso de impossibilidade de movimento
- Movimento de bloco de cartas

Modelo da Arquitetura



Interface dos Módulos



Modelo Físico

Assertivas estruturais

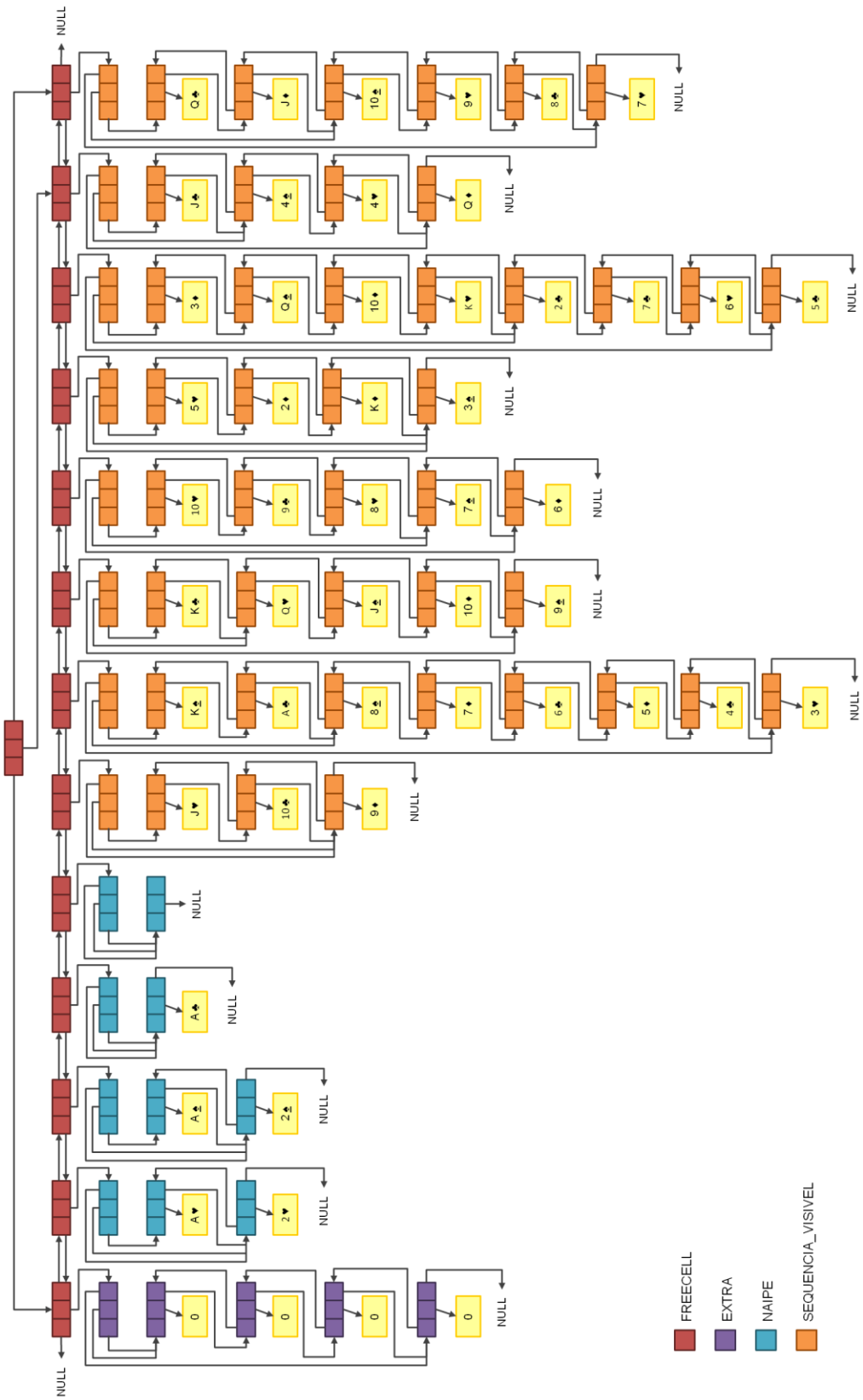
- Uma cabeça do tipo Freecell deve ter 13 elementos que apontam para cabeças de lista, que correspondem às colunas do jogo, neste caso:
  - Uma cabeça de Extra, que no momento da inicialização já aponta para quatro elementos, que por sua vez apontam para cartas vazias.
  - Quatro cabeças de Naipe, que no momento da inicialização estão vazias e podem receber até 13 cartas (de A a K) de um mesmo naipe, definido no momento da inserção da primeira carta.
  - Oito cabeças de Sequência Visível, que no momento da inicialização estão com sete ou oito cartas (52 cartas distribuídas pelas oito colunas) e podem receber até 20 cartas.

Freecell, Extra, Naipe e Sequência Visível utilizam a estrutura lista e cabeça de lista. Desta forma deve-se assegurar que para todo elemento da lista que possui um próximo elemento, o próximo do anterior é o próprio elemento, isto é:

∀ pElem ∈ Lista { pLista } : pElem->pProx != NULL => pElem->pProx->pAnt == pElem

Exemplo de Modelo Físico

(no meio de uma partida)



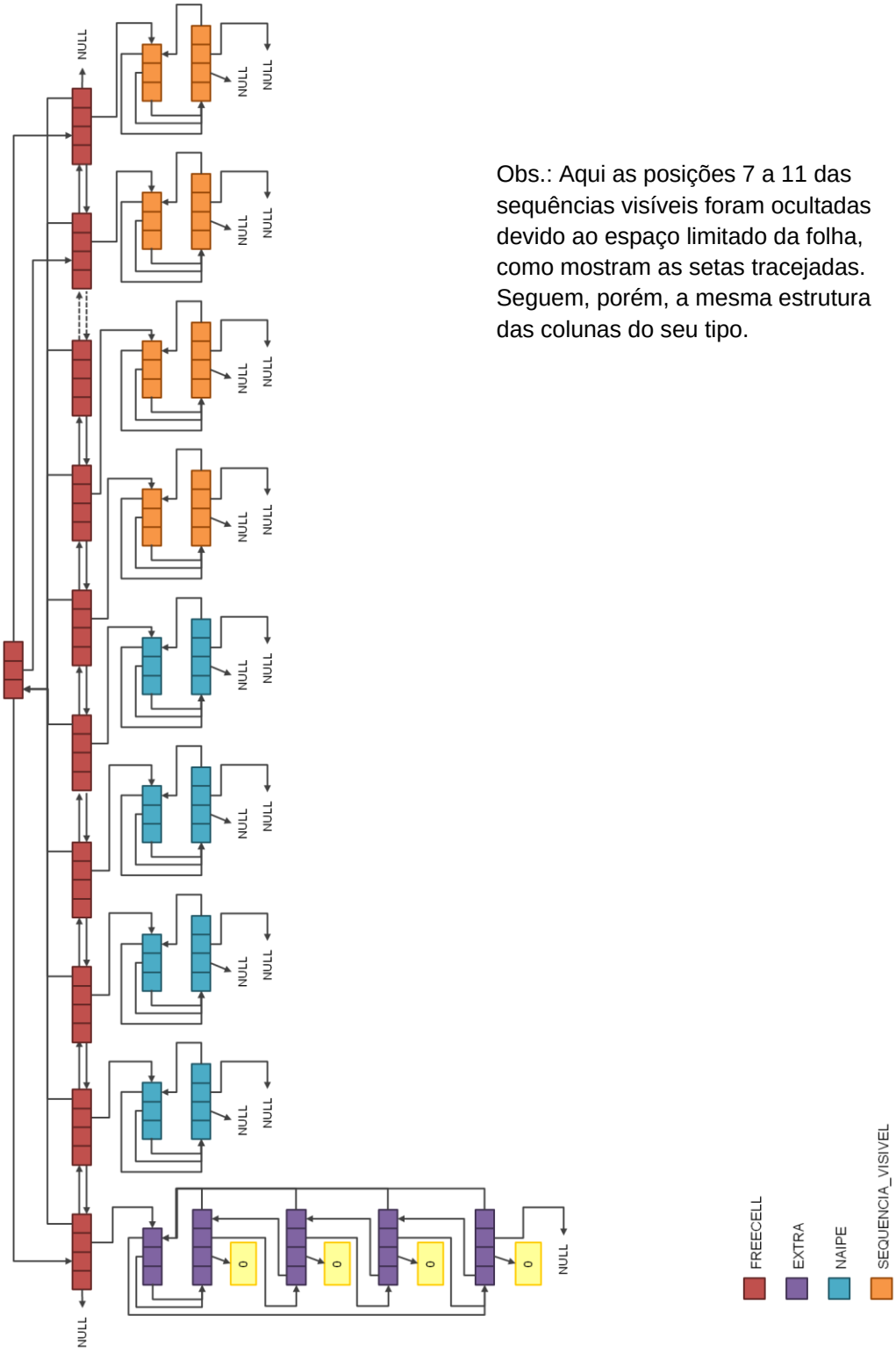
Lista Genérica Auto-Verificável

Assertivas estruturais

Ao elemento do tipo lista acrescenta-se agora mais um campo, que é um ponteiro (de volta) para sua cabeça (em cores mais claras na figura). Esta mudança não altera as assertivas anteriores, porém acrescenta a seguinte: A cabeça que aponta para um elemento deve ser a mesma a qual o ponteiro para cabeça aponta.

Exemplo de Modelo Físico

(criação de colunas do jogo no início)



Obs.: Aqui as posições 7 a 11 das sequências visíveis foram ocultadas devido ao espaço limitado da folha, como mostram as setas tracejadas. Seguem, porém, a mesma estrutura das colunas do seu tipo.