Programação Avançada 2015/2016

Linha3 - 1º Fase

Turma: INF-ES-02
Ricardo José Horta Morais
Nº 140221066

Índice

1 Introdução

O projeto tem como objetivo aplicar os conhecimentos adquiridos no âmbito da unidade curricular de Programação Avançada.

O trabalho consiste em criar um jogo parecido ao 3 em linha em javaFX.

Numa primeira fase vai ser definido o modelo de análise. Também ainda nesta primeira fase vão ser especificados e implementados os tipos abstratos de dados para posteriormente serem utilizados na implementação do Jogo.

2 Diagrama de classes do modelo de análise

2.1 Diagrama de classes de análise

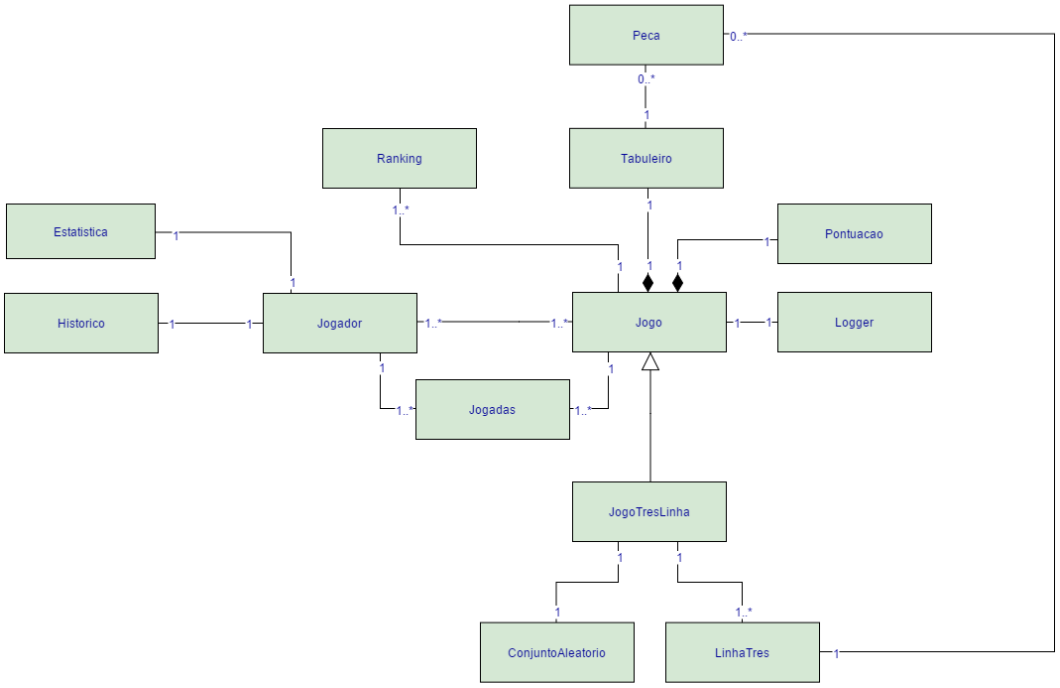

Ilustração 1 - Diagrama de classes de análise

2.2 Descrição do diagrama de classes de análise

O diagrama de classes de análise é a representação gráfica do problema apresentado neste projeto. A sua concepção é simplista e não representa nenhuma solução. Neste diagrama apresento um mapeamento do problema que é portanto um olhar geral no que diz respeito às entidades e relações entre as mesmas.

O diagrama baseia-se principalmente nas entidades Jogo, Jogador e Peça, depois existe um desdobramento para a realização de algumas especificações do problema em sí. Relacionado com o Jogador temos a sua estatística geral e o seu histórico. As cardinalidades apresentadas mostram que cada Jogador terá a sua estatística e o seu histórico. Para a entidade Jogo temos relações com o ranking, a jogada e o logger. Cada jogo pode estar associado a vários Rankings, a várias Jogadas e a um logger. Entretanto um Jogo pode conter um Pontuação e um Tabuleiro. Um Jogo terá um tabuleiro que por sua vez terá peças. O jogoTresLinha herdará de Jogo e irá estar associado ao ConjuntoAleatório e à LinhaTres.
Outros pormenores do problema foram omitidos de forma a simplificar a visualização geral.

3 Descrição dos TAD´s

3.1 LinhaTres - Especificação

Uma LinhaTres é uma coleção de itens com um comportamento semelhante ao TAD Deque permitindo inserir e retirar elementos no início e no fim da fila. No entanto as operações de remover elementos, removem sempre 3 elementos de uma vez e a operações de getFirst e getLast, devolvem os 3 primeiros elementos e os 3 últimos respetivamente. O TAD LinhaTres tem a noção de capacidade máxima, gerando um erro quando essa é excedida. Disponibiliza ainda um iterador para percorrer a coleção.

O TAD LinhaTres suporta os seguintes métodos fundamentais:

- **addFirst(E elemento):**
  - Função: Insere um elemento no início da fila
  - Erros: LinhaCheiaException
  - Entrada: elemento a inserir
  - Saída: nenhuma

- **addLast(E elemento):**
  - Função: Insere um elemento no fim da fila
  - Erros: LinhaCheiaException
  - Entrada: elemento a inserir
  - Saída: nenhuma

- **removeFirst():**
  - Função: Remove os 3 primeiro elementos da fila
  - Erros: LinhaSemTresException
  - Entrada: nenhuma
  - Saída: nenhuma

- **removeLast():**
  - Função: Remove os 3 últimos elementos da fila
  - Erros: LinhaSemTresException
  - Entrada: nenhuma
  - Saída: nenhuma

- **getFirst ():**
  - Função: Devolve os 3 primeiros elementos da fila
  - Erros: LinhaSemTresException
  - Entrada: nenhuma
  - Saída: ConjuntoTriplo

- **getLast ():**
  - Função: Devolve os 3 últimos elementos da fila
  - Erros: LinhaSemTresException
  - Entrada: nenhuma

- Saída: ConjuntoTriplo

- **isEmpty():**
  - Função: Devolve true se a linha estiver vazia false senão
  - Erros: nenhum
  - Entrada: nenhuma
  - Saída: boolean

- **sizeEsquerdo():**
  - Função: Devolve o número de elementos presentes na parte esquerda da linha
  - Erros: nenhum
  - Entrada: nenhuma
  - Saída: int

- **sizeDireito():**
  - Função: Devolve o número de elementos presentes na parte direita
  - Erros: nenhum
  - Entrada: nenhuma
  - Saída: int

3.2 LinhaTres – Implementação

Foi utilizada uma implementação dinâmica, pois queremos ter a possibilidade de ter capacidades de linhas diferentes. A implementação foi feita de raiz e baseou-se num deque circular dinâmico.

3.3 ConjuntoAleatorio – Especificação

Um conjunto aleatório é uma coleção de itens não repetidos, em que não é possível adicionar ou retirar elementos, e que disponibiliza uma única operação: a operação de visualização aleatória de um item. Ou seja, se tivermos um conjunto aleatório com as cores amarelo, verde e azul, a operação de peek (visualização de um item), devolve aleatoriamente uma das 3 cores.

O TAD ConjuntoAleatorio suporta os seguintes métodos fundamentais:

- **peek():**
  - Função: Devolve um dos items da coleção
  - Erros: nenhum
  - Entrada: nenhuma
  - Saída: Elemento

3.4 ConjuntoAleatorio – Implementação

Foi utilizada uma implementação estática, pois serão inseridos todos os elementos na inicialização do ConjuntoAleatorio.

3.5 Historico – Especificação

O Histórico caracteriza-se por ser uma coleção de elementos, com tamanho limitado, onde é apenas possível colocar elementos não sendo possível retira-los. Os elementos são guardados por ordem de entrada para o histórico e ficam associados a uma data (a data de entrada para o histórico). É possível percorrer sequencialmente todos os elementos, assim como saber quais os elementos de um determinado dia, ou dos últimos dias ou do mês corrente etc. Quando o número de elementos é atingido, os elementos mais antigos são descartados automaticamente.

O TAD Historico suporta os seguintes métodos fundamentais:

- **size():**
  - Função: Devolve o número de elementos presentes no historico
  - Erros: nenhum
  - Entrada: nenhuma
  - Saída: int

- **isEmpty():**
  - Função: Devolve true se o historico estiver vazio false senão
  - Erros: nenhum
  - Entrada: nenhuma
  - Saída: boolean

- **getCapacity():**
  - Função: Devolve a capacidade do historico
  - Erros: nenhum
  - Entrada: nenhuma
  - Saída: int

- **add():**
  - Função: Adiciona um elemento ao historico
  - Erros: nenhum
  - Entrada: elemento a adicionar
  - Saída: int

- **getIteradorPorCriterioDeData():**
  - Função: Cria um iterador que itera os elementos que correspondem a um critério
  - Erros: nenhum
  - Entrada: Critério de correspondência
  - Saída: Iterator

3.6 Historico – Implementação

Foi utilizada uma implementação dinâmica pois queremos ter a possibilidade de ter históricos de tamanhos diferentes. Foi usado o padrão de software Strategy para implementar os critérios dos iteradores do histórico.

3.7 Ranking – Especificação

Um ranking é uma lista ordenada de elementos, em que os elementos são ordenados segundo um critério específico.

O TAD Ranking suporta os seguintes métodos fundamentais:

- **size():**
  - Função: Devolve o número de elementos presentes no historico
  - Erros: nenhum
  - Entrada: nenhuma
  - Saída: int
- **isEmpty():**
  - Função: Devolve true se o historico estiver vazio false senão
  - Erros: nenhum
  - Entrada: nenhuma
  - Saída: boolean
- **get():**
  - Função: Devolve o elemento que está no índice
  - Erros: IndiceNaoExistenteException
  - Entrada: Índice ou rank do elemento
  - Saída: Elemento
- **set():**
  - Função: Altera um elemento existente presente no índice
  - Erros: IndiceNaoExistenteException
  - Entrada: Índice ou rank do elemento e o elemento
  - Saída: nenhuma
- **adicionar():**
  - Função: Adiciona um elemento
  - Erros: ElementoExistenteException
  - Entrada: Elemento a adicionar
  - Saída: nenhuma
- **remover():**
  - Função: Remove um elemento presente no índice
  - Erros: IndiceNaoExistenteException
  - Entrada: Índice ou rank do elemento
  - Saída: Elemento removido
- **hasElement():**
  - Função: Verifica se o elemento já existe no ranking
  - Erros: nenhum
  - Entrada: Elemento a verificar

- Saída: boolean

3.8 Ranking – Implementação

Foi utilizada uma implementação dinâmica pois não queremos restrição na capacidade do ranking. Foi usado o padrão de software Strategy para implementar os critérios de ordenação do ranking.