ESCOLA SUPERIOR DE TECNOLOGIA - IPS
ANO LETIVO 2015 / 2016
ENGENHARIA DE INFORMÁTICA
PROGRAMAÇÃO AVANÇADA. PROFª PATRICIA MACEDO

DESCRIÇÃO DOS TADS

MIGUEL FURTADO 120221006
TIAGO MONTEIRO 140221001

Índice

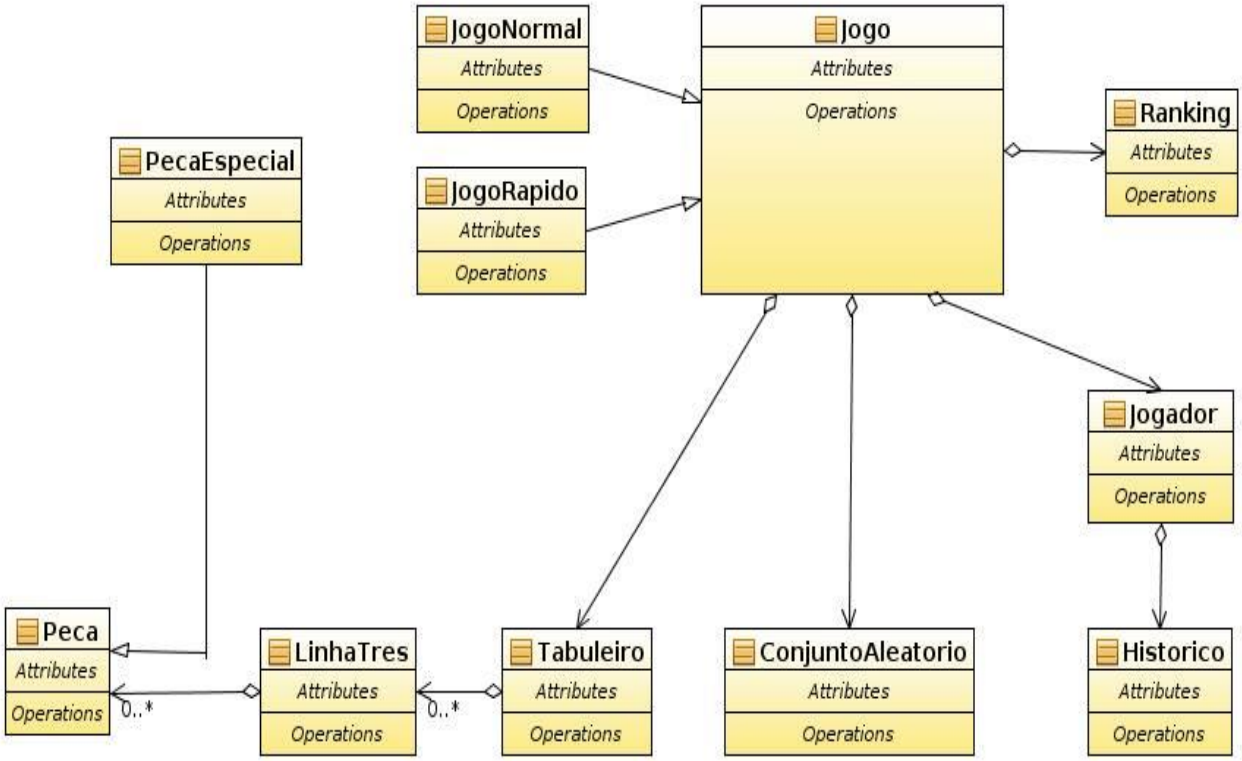
Introdução

O projeto a ser desenvolvido tem como objetivo aplicar todos os conhecimentos que foram e virão a ser adquiridos no âmbito da unidade curricular de Programação Avançada.

O trabalho consiste no desenvolvimento de um jogo do tipo 3 em linha baseado num jogo já desenvolvido e que se encontra em funcionamento na seguinte hiperligação http://www.sheppardsoftware.com/braingames/pumpkins/pumpkins.htm. O jogo tem como objetivo chegar ao máximo de pontos possíveis juntando várias peças e ganha-se pontuação cada vez que se junta três peças iguais, sendo que as opções que temos de peças para jogar são aleatórias.

Numa primeira fase vai ser definido o modelo de análise e implementados os diferentes tipos abstratos de dados para posteriormente estes virem a ser implementados no jogo em questão.

Diagrama de Classes do Modelo de Análise



Para o desenvolvimento deste projeto foi criada uma classe jogo, que é responsável por gerir tudo o que é associado ao jogo. Este contem dois tipos de jogo, rápido e normal. O jogo tem ainda o ranking com o ranking dos jogadores. No tabuleiro é onde se irá jogar e este irá ter um conjunto de linhas três com peças, e dessas peças existe uma que é especial. O conjunto aleatório servirá para gerar as novas peças. A par destas ainda temos o jogador que irá ter um historio.

TAD Conjunto Aleatório

O conjunto aleatório foi desenvolvido com uma interface, TADAleatorio, e com uma class conjuntoAleatório.

A interface usada foi a interface TADAleatorio e apenas tem um método:

- **Peek():** que devolve o objeto no topo.
  Entrada: NULL; Saída:elemento generico

A class ConjuntoAletorio é uma coleção de itens não repetidos, em que não é possível adicionar ou retirar elementos, e que disponibiliza uma única operação, implementada através da interface Stack, que é o peek. Esta permite visualizar um item alectório do conjunto de itens criados nesta classe. Para isto foi criado inicialmente um array de objetos que foi inicializado no construtor ao receber um array.

TAD Histórico

O Histórico foi usado com duas interfaces, TADHistorico e Iterator, e duas classes a NodeHistorico e Historico.

A interface TADHistorico tem quatro métodos e esta vai definir o comportamento da classe Historico. Inclui os seguintes métodos:

- **Add (E element):** que adiciona um elemento.
  Entrada:elemento generico; Saída:nenhuma
- **getSize():** que devolve o tamanho.
  Entrada:nenhuma; Saída: int
- **isEmpty():** que devolve true se estiver vazio.
  Entrada:nenhuma; Saída:boolean
- **getElementsByDate(Date inicio, Date fim):** que devolve um conjunto de elementos a partir da data.
  Entrada:Date início e Date fim; Saída:Array elementos genéricos

A classe NodeHistorico tem sete métodos e esta classe representa cada elemento no Historico. Como tal foram criados três atributos, um NodeHistorico anterior e próximo, um elemento genérico e por último um atributo data. Os métodos são os seguintes:

- **getAnterio():** que devolve o Node anterior a um Node.
  Entrada:nenhuma; Saída: NodeHistorico
- **setAnterior(NodeHistorico anterior):** que define qual será o Node anterior a um certo Node.
  Entrada: NodeHistorico; Saída: nenhuma;
- **getProximo():** que devolve o Node a seguir a um Node.
  Entrada:nenhuma; Saída:NodeHistorico proximo
- **setProximo(NodeHistorico proximo):** que define qual será o Node a seguir a um certo Node.
  Entrada:NodeHistorico próximo; Saída:nenhuma
- **getElement():** que devolve o elemento pertence-te a um Node.
  Entrada:nenhuma; Saída: elemento genérico

- **getData():** que devolve a data de um Node.

A classe Historico tem implementada a interface TADHistorico e é uma coleção de elementos, com tamanho limitado, dai ao início se criar uma variável maxSize do tipo int e que irá ser inicializada no construtor, e onde é apenas possível inserir elementos e não retirá-los. Estes elementos são guardados por ordem de entrada para o histórico e ficam associados a uma data. Para tal criamos inicialmente para além da variável maxSize, a variável header do tipo NodeHistorico e uma variável do tipo int com o nome size.

Criámos para esta classe cinco métodos, sendo quatro deles implementados a partir da interface TADHistorico, e um último método privado:

- **removeLast():** usado para remover um elemento no método add quando o número máximo de elementos é atingido.
  Entrada:nenhuma; Saída:nenhuma

Para esta classe ainda foi implementado uma classe privada HistoricoInterator que implementa a interface Iterator de modo a conseguirmos percorrer mais tarde os elementos da nossa coleção.

TAD LinhaTres

O TAD LinhaTres é um TAD dinâmico com um comportamento semelhante ao TAD Deque e foi desenvolvido com duas interfaces e cinco classes sendo que três destas classes são classes de exceção.

Para as interfaces usámos a interface Iterator que tem os seguintes métodos:

- **hasNext():** que devolve se existir um elemento.
  Entrada:nenhuma; Saída:boolean
- **next():** que devolve o elemento atual e passa ao próximo.
  Entrada:nenhuma; Saída:elemento genérico

A interface DequeLinhaTres que é baseado na interface Deque, mas com algumas diferenças nos métodos:

- **removeLast()** e **removeFirst():** devolvem um array com três elementos e remove três elementos.
  Entrada: nenhuma; Saída: array de elementos genéricos
- **getLast()** e **getFirst():** devolvem um array de 3 elementos.
  Entrada:nenhuma; Saída: array de elementos genéricos
- **size():** devolve o tamanho.
  Entrada:nenhuma; Saída:int
- **isEmpty():** devolve true se estiver vazio.
  Entrada:nenhuma; Saída:boolean
- **addFirst(E element):** que adiciona um elemento no início.
  Entrada: elementos genérico; Saída:nenhuma
- **addLast(E element):** que devolve um elemento no fim.
- Entrada: elementos genérico; Saída:nenhuma

As exceções são a DequeInsufienteElements que só é usada caso o elemento não exista, a DequeFullException que é usada no caso de a coleção estar cheia e por último a exceção DequeEmptyException que é usada para o caso de se fazer algo e a coleção estar vazia.

Para as classes foi criado a classe DTresNode e a classe LinhaTres.

A classe DTresNode representa o elemento que vai ser usado no LinhaTres e tem três variáveis. A variável anterior e próximo do tipo DTresNode e a variável elemento do tipo genérico. O método construtor inicia estas três variáveis e os seguintes métodos:

- **getAnterior():** que nos devolve o node anterior ao que está em causa.
  Entrada:nenhuma; Saída:DTresNode
- **getProximo():** que devolve o node a seguir ao que está em causa.
  Entrada:nenhuma; Saída:DTresNode
- **getElement():** que devolve o elemento do node em causa.
  Entrada:nenhuma; Saída: elemento genérico
- **setAnterior(DTresNode):** que define o node anterior ao em causa.
  Entrada: DTresNode anterior; Saida: nenhuma
- **setProximo(DTresNode**): que define o node a seguir ao em causa.
  Entrada:DTresNode próximo; Saída:nenhuma

A classe LinhaTres implementa a interface DequeLinhaTres e os seus respetivos métodos descritos acima. Para além disto foi implementada uma classe privada IteratorTresEmLinha que implementa a interface Iterator e os métodos desta que são descritos acima.

TAD Ranking

O TAD Ranking é uma lista ordenada de elementos, em que estes são ordenados segundo um critério, como tal foram criadas duas interfaces, a List e a StrategyComparator e duas classes, a classe Ranking e a classe ComparatorInt.

A interface List implementa os seguintes métodos:

- **size():** que devolve o tamanho da lista de ranks.
  Entrada:nenhuma; Saída:int
- **isEmpty():** devolve true se estiver vazia.
  Entrada:nenhuma; Saída:boolean
- **add(E elem):** adiciona um elemento à lista.
  Entrada:elemento genérico; Saída:nenhuma

A interface StrategyComparator implementa o seguinte método:

- **compareTo(E e1, E e2):** que compara dois elementos do rank.
  Entrada: dois elementos genéricos; Saída: int

A classe ComparatorInt é usada para implementar o método compareTo descrito acima.

A classe Ranking foi implementada com a interface List e contém os métodos descritos anteriormente, para além destes ainda contém dois métodos privados nodeAtRank e removeChecker e um publico getRankingIterator:

- **nodeAtRank(E elem):** que compara dois Nodes para que no método add um novo Node adicionado fique na posição correta.
  Entada:elemento genérico; Saída: RankingNode
- **removeChecker(RankingNode newNode):** que vai percorrer a lista de ranks e remover se existir um Node com um rank mais baixo, mas com o mesmo nome.
  Entrada:RankingNode; Saída:nenhuma
- **getRankingIterator():** devolve o iterator ranking
  Entrada:nenhuma; Saída:Iterator

Para além destes métodos a classe Ranking ainda contém duas classes privadas a RankingNode e a RankingIterator. A RankingNode é a classe que cria o elemento que irá ficar na lista de rankings, a rankingIterator é o Iterador para percorrer a lista de ranking e foi usada a interface Iterator que se encontra descrita no Tad LinhaTres.