UNIVERSIDADE DE ÉVORA

TRABALHO PRÁTICO

ESTRUTURA DE DADOS E ALGORITMOS I

JOGADOR INFORMADO DO JOGO DO GALO

26-01-2015

Discente:

André Figueira – 31626

Introdução

Com este trabalho pretendo desenvolver os meus conhecimentos na linguagem Java e também os conhecimentos de estrutura de dados e algoritmos através da sua utilização.

Este trabalho consiste num conjunto de classes fornecidas pelo docente cuja função depois de implementadas na sua totalidade possam recriar o jogo do galo. Este jogo do galo consiste em dois jogadores, um inteligente e um “ao acaso”, o jogador inteligente aprende com os jogos e faz uma escolha baseada na informação guardada numa hashtable enquanto o jogador “ao acaso” escolhe aleatoriamente uma posição e joga.

Para facilitar a implementação destas classes notam-se alguns métodos auxiliares que permitem facilitar a resolução dos problemas.

Implementação das Classes

Em relação a implementação das classes irá ser feita uma descrição dos seus métodos. As classes descritas vão ser as que foram fornecidas pelo docente, que são JogoDoGalo, JogInteli, JogAcaso e ProjectGalo. As classes restantes não são descritas porque são as estruturas de dados, HashTable e SimplyLinkedList, que foram implementadas durante a aula prática com a ajuda do docente e da informação das aulas teóricas.

1. Classe JogoDoGalo

1.1 O constructor.

Este constructor é um constructor que não recebe argumentos e tem o propósito de inicializar as variáveis, o tabuleiro em que se vai jogar (um array de duas dimensões) e uma lista ligada simples (cujo propósito vai ser explicado no método **gamePath(JogoDoGalo currentBoard)**).

1.2 Vencedor()

Este método retorna o vencedor do jogo do galo, verificando linhas, colunas e diagonais à procura de um vencedor, se não houver nenhuma linha, coluna, diagonal igual então retorna 0 que significa um empate. Este método permite então saber o vencedor, e com a ajuda deste método permite controlar o número de victórias/derrotas/empates de cada configuração do tabuleiro usando o método **incrementarVDE()**

1.3 Hashcode()

Este método retorna o hashcode do objecto, este código permite que cada tabuleiro tenha um código único pois cada posição do tabuleiro é multiplicado por número primo diferente de todos os outros e multiplicado pelo valor presente na posição (ou seja multiplica por 1 ou por 2 ou por 0) e somado.

1.4 isBoardFull()

Este método retorna um booleano. True se o tabuleiro estiver cheio, false se não estiver (ou seja significa que existe pelo menos uma posição que contem um 0).

1.5 acabou()

Este método verifica o estado do tabuleiro, se após cada jogada do jogador inteligente e jogador "ao acaso" o tabuleiro cumpre os requisitos, ou seja, se já existe um vencedor ou então se o tabuleiro está cheio. Este método permite saber se o jogo terminou ou não.

1.6 joga(int row,int col)

Este método utiliza um contador para saber qual jogador deve jogar. Este contador é incrementado quando a posição que se quer jogar está vazia, se o contador conter um número par é a vez do jogador "O" jogar caso contrario é a vez do jogador "X". Depois da jogada retorna true. Se a posição que se quer jogar não estiver vazia então retorna false e nenhum dos jogadores joga.

1.7 gamePath(JogoDoGalo currentBoard)

De dois em dois turnos, é incrementado para uma lista ligada o estado actual do tabuleiro. Isto permite saber a sua evolução. **O propósito deste método será explicado na classe JogInteli**

1.8 getWinRatio() e getTotalJogos()

Estes dois métodos como o nome indica, um retorna a percentagem de victorias e o outro o número total de jogos

1.9 toString()

Retorna a representação em String do tabuleiro

2. Os Jogadores

2.1 Classe JogAcaso (Jogador “X”)

Esta classe como o nome indica faz uma jogada aleatória. Consiste num constructor e num único método, **joga(JogoDoGalo t)**, este método gera dois números aleatórios e faz uma chamada ao método **joga(int row, int col)** da classe JogoDoGalo e verifica se pode jogar, se puder, realiza uma jogada. Caso contrário volta a gerar números aleatórios ate ser possível.

2.2 Classe JogInteli (Jogador “O”)

2.2.1 O constructor.

Este constructor inicializa uma nova hashtable que usa acesso quadrático que vai ser o aspecto principal desta classe, esta tabela permite guardar todas as jogadas deste jogador e fazer decisões sobre as suas jogadas no jogo a decorrer consoante a sua informação presente na tabela.

Escolha de Acesso Quadrático: Foi escolhido acesso quadrático pois após testes de velocidade de processamento entre acesso linear e duplo hash, notou-se que este foi o mais rápido.

2.2.2 novoJogo(int jogador)

Este método não foi utilizado pois não se verificou a sua utilidade pois visto que a docente permitiu aos discentes que podiam assumir que o Jogador inteligente seria sempre X ou O durante os 100 jogos tornou-se então um método desnecessário.

2.2.3 joga(JogoDoGalo t)

Quando é a vez do jogador inteligente jogar é feito uma chamada a este método. Caso seja o primeiro jogo, o jogador não pode ir hashtable verificar qual é a melhor opção pois está vazia logo é forçado a jogar aleatoriamente. Caso contrario já tenha algo na tabela faz uma chamada ao método **findBestPosition(JogoDoGalo t)**

findBestPosition(JogoDoGalo t): este método tem o propósito de, como o nome indica, encontrar a melhor posição. O método atinge esse objectivo através da procura na hashtable da jogada seguinte a "t" com maior percentagem de victorias.

Como a hashtable só contem as jogadas do jogador "O". E "t", a última jogada foi efectuada pelo jogador "X", é percorrer dois ciclos e é necessário clonar "t" e usar o clone (clona-se em cada iteração) para realizar a jogada seguinte (de forma a evitar alterações desnecessárias no "t" original) e verificar a sua existência na hashtable, caso exista na hashtable retorna o objecto (usando o método **procurarJogo(JogoDoGalo x)** da hashtable que retorna o objecto) e utiliza-se o seu **getWinRatio()** para efectuar comparações com **maxWinRatio**. Como o nome da variável indica, contem a percentagem mais alta de victorias. Inicialmente encontra-se a 0. Caso haja uma percentagem igual a **maxWinRatio** então decide-se aleatoriamente entre o conjunto de coordenadas a usar.

Caso o objecto não exista na hashtable então passa para a próxima iteração.

Quando o ciclo termina obtém-se então a posição mais correcta.

Pode acontecer que a tabela não esteja vazia e que não exista nenhuma jogada seguinte na hashtable. Nesse caso a posição é aleatória.

2.2.4 acabou(JogoDoGalo terminal)

Este método é utilizado quando **terminal.acabou()** retorna true.

Utilizando a lista que contém a evolução do jogo (que foi conseguido através do **gamePath(JogoDoGalo currentBoard)**) é feita uma procura elemento a elemento dessa lista na hashtable para verificar a sua existência. Caso não exista incrementa-se o vencedor usando **incrementVDE(int vencedor)** e insere-se na tabela. Caso contrário procura o jogo na hashtable que contém uma representação String igual, retorna o jogo (através do método **procuraJogo(JogoDoGalo x)**) e incrementa o vencedor usando **incrementVDE(int vencedor)**. Este processo repete-se para todos os elementos na lista.

2.2.5 movimentoFavorito()

Retorna o primeiro movimento mais usado, ou seja, o movimento com mais victorias, este método cria um objecto do tipo JogoDoGalo temporário e executa um jogada usando o método **joga(int row, int col)** da classe JogoDoGalo semelhante ao processo utilizado em **joga(int row, int col)** para encontrar objectos na hashtable da classe JogInteli, em seguida é feito uma chamada ao método **sucessores(JogoDoGalo t)** que irá procurar na hashtable todas as configurações que veem a seguir. Depois é feito uma serie de comparações como foram feitas em **joga(int row, int col)** na classe JogInteli para conseguir obter o que tem mais victorias. Após feitas as comparações, retorna o favorito.

2.2.6 sucessores(JogoDoGalo t)

Este método cria um clone de “t”, e realiza uma jogada usando o método **joga(int row, int col) da classe JogoDoGalo** e procura na hashtable, se encontrar adiciona o jogo para um array cujo tamanho é igual ao numero de ocupados. Esta decisão garante que não haja problemas de espaço, visto que como dito anteriormente, tem de se procurar na hashtable,

2.2.7 numeroDeVezesVisto(JogoDoGalo t)

Se não existir na hashtable retorna 0, caso contrario procura na hashtable o jogo e retorna o número total de jogos

3. Classe ProjectGalo

Esta classe irá gerir os jogos entre os dois jogadores: JogAcaso e JogInteli. A main desta classe faz uma chamada a **play(JogAcaso playerX, JogInteli playerO, int numJogos)** que executa 100 jogos através de um ciclo. Sempre que o jogador "O" realiza uma jogada faz uma chamada a **gamePath(JogoDoGalo currentBoard)**, e no final de cada jogo invoca o método **acabou(JogoDoGalo terminal)**, e incrementa num contador (**playerO e playerX)** o vencedor. No final do ciclo faz uma chamada ao método **movimentoFavorito()** e imprime um relatório final com as estatísticas.

4. Conclusao

Pode-se concluir que os objectivos foram cumpridos. Através da resolução deste trabalho contribuiu para a aquisição de conhecimentos e compreensão de hashtables através da sua utilização e implementação e também contribuiu para a compreensão e conhecimentos de inteligência artificial através do seu uso.