Faculdade de Ciências e Tecnologias – Universidade do Algarve

Licenciatura em Engenharia Informática

Unidade Curricular: Inteligência Artificial

Problema 3: N-queens

Ano letivo: 2022/23

Semestre: 1º

Grupo No: 3

Turma: PL1

Trabalho realizado por:

Nome	Número
Daniel Ferros Fernandes	71320
Diogo Afonso Nobre Zacarias	71323
Kartic Hitendra Premgi	71379

Descrição do problema:

Este trabalho consiste em escrever um programa capaz de resolver o problema n-queens, utilizando qualquer algoritmo de IA (inteligência artificial), o problema é considerado resolvido se o programa achar uma solução dentro do melhor tempo possível.

O problema n-queens consiste em colocar n rainhas em um tabuleiro de xadrez n x n, sem que nenhuma das rainhas se ataquem. Uma vez que segundo as regras de xadrez, a rainha anda por todo o tabuleiro na horizontal, vertical e nas suas respetivas diagonais.

Algoritmo “Hill Climb”:

O algoritmo que usamos é o Hill Climb.

De forma geral:

Este algoritmo consiste em começar num ponto aleatório X e criar um estado inicial. A seguir, verificamos se esse estado é a solução, caso seja, acabamos ai, e caso contrário, se estado iremos andar até encontrar essa solução, criando os seus sucessores.

Do nosso problema:

- **Representação de cada variável:**
  - **Checked –** Lista que terá no máximo um tamanho 100. Esta lista terá os layouts que já foram testados;
  - **Contador –** é tipo um buffer, que servirá nas vezes que encontramos um estado igual o atual, serve para dar algum tempo para que possamos progredir de uma situação estagnada;
  - **Atual –** será o estado que iremos analisar;

- **Algoritmo:**

1º Criamos o atual, que irá ter um board com rainhas postas de forma aleatória, uma por linha;

2º Entramos num Loop:

2.1º Verificamos se a lista “checked” está cheia (têm tamanho 100).

2.1.1º Se a lista está cheia, removemos o 1ºelemento;

2.2º Adicionamos à lista “checked”;

2.3.º Verificamos se o atual tem o número de conflitos igual a 0;

2.3.1º Se for igual a 0, então chegamos a solução e retornamos o board;

2.3.2º Caso contrário, calculamos os filhos do estado atual;

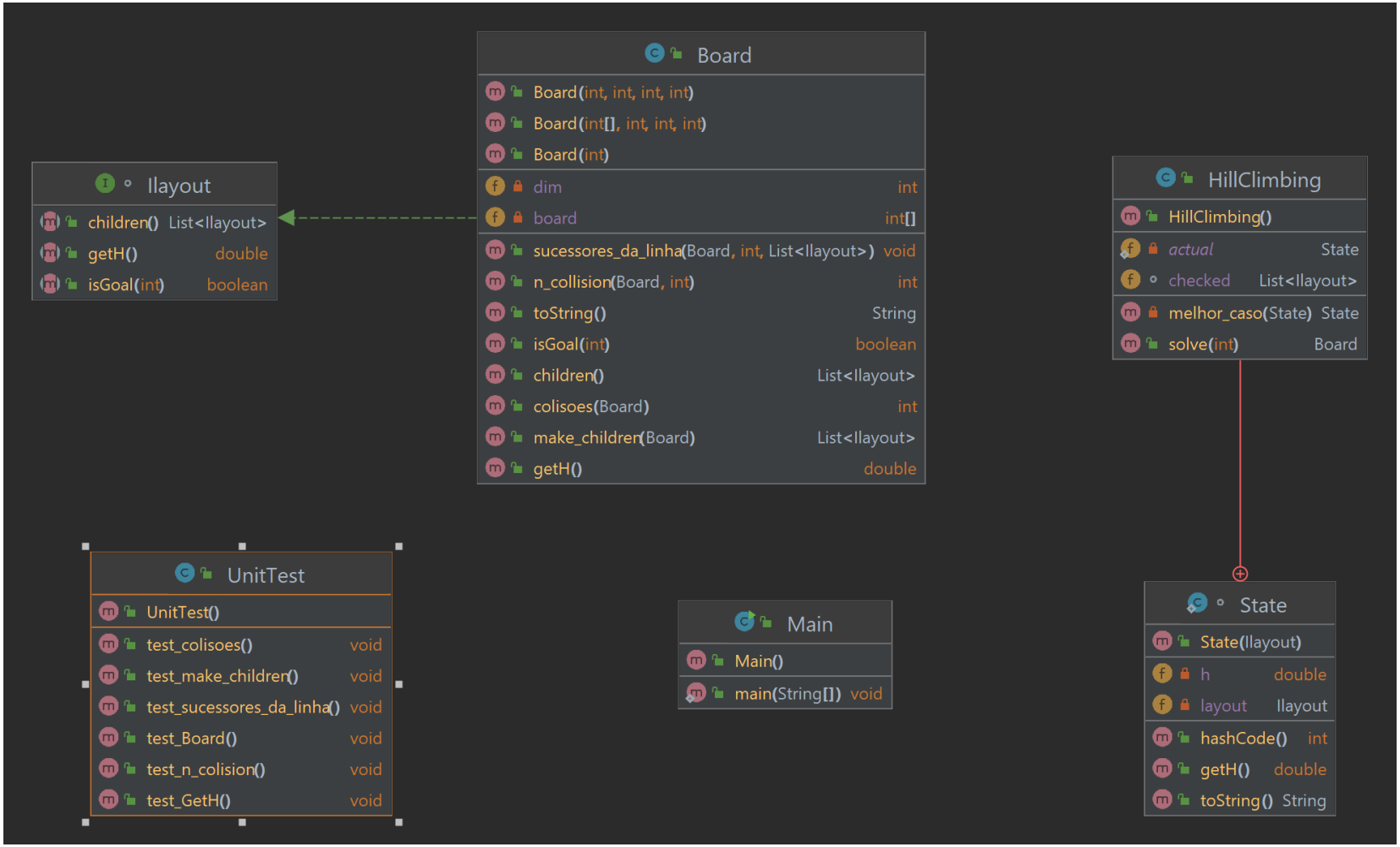
2.3.2.1º Destes filhos criados escolhemos o melhor com base no menor número de conflitos entre rainhas.

2.3.3º Se esse filho tiver menor número de conflitos que o estado atual, o atual transforma-se no filho e o contador é retornado a 0;

2.3.4º Caso contrário, verificamos se o contador é menor que 50 e se o número de conflitos do atual e do filho são iguais, se for tornamos o atual no melhor filho e aumentamos o contador numa unidade;

2.3.3º Se falhar as duas condições de cima, fazemos um restart, e o atual transformara-se no novo board criado aleatoriamente. Este board criado, será verificado na lista “checked”. Caso este exista, iremos fazer reset até criar um board diferente dos testados;

Diagrama de Classes UML:



Todas as opções de projeto tomadas, designadamente padrões de projeto

- **Iniciação do board:**

Para iniciar o board verificamos que a maneira mais eficiente teria de ser iniciando um board com uma rainha aleatória por linha.

- **Buffer:**

Uma das maneiras que utilizamos para melhorar a eficiência foi criar um "buffer", isto é, permitimos o programa realizar um certo número de iterações seguidas em que não melhore, de maneira que seja possível a procura pela solução estagnar e ainda consiga chegar a solução sem que seja necessário um recomeço total.

- **Melhor caso:**

Sendo o algoritmo escolhido o Hillclimbing precisamos de escolher de todos os filhos possíveis através da movimentação de cada rainha na linha o caso com menor número de conflitos.

- **Escolha do algoritmo:**

Escolhemos o algoritmo Hill Climbing porque nos pareceu o mais eficiente para a resolução deste problema sendo que trabalhamos com conflitos e melhorias rápidas.

- **Uso de array para representar board:**

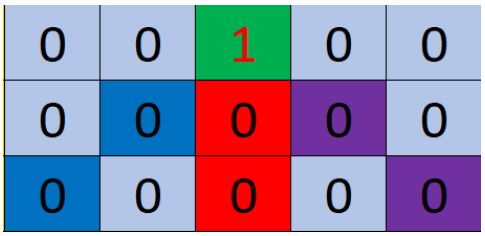
Inicialmente utilizamos uma matriz para representar o board depois notamos que realmente apenas necessitamos de saber onde estava de facto a rainha em cada linha então passamos para um array simplificando assim o código e melhorando a eficiência.

- **Lista checked:**

Criamos uma lista checked que contem os últimos 100 casos testados, utilizamos 100 casos pois pela pesquisa que fizemos vai ser um bom equilíbrio entre melhoramento de tempo sem ser um grande aumento de memoria, esta lista tem objetivo de evitar a testagem de casos recentes já previamente testados, fazemos essa verificação quando escolhemos o melhor filho ou quando criamos um board random.

- **Cálculo Colisões:**

Para calcular colisões dividimos em três casos: se estão na mesma coluna, estão na diagonal direita ou estão na diagonal esquerda.



Para a coluna a verificação é simples,.

Já para as **diagonais da esquerda**, teremos de fazer a comparação entre a soma da linha e da coluna, da linha original , com a linha em análise. Considerando o exemplo da figura 1, estamos a analisar a linha 0. Podemos verificar que a rainha da linha 0 colide com a linha 2. Usando a nossa fórmula, chegamos a essa conclusão:

Para rainha da linha 0: 2 (índice da coluna) + 0 (índice da linha) = 2

Para rainha da linha 2: 0 (índice da coluna) + 2 (índice da linha) = 2

Já para as **diagonais da direita**, teremos de fazer a comparação entre a subtração da coluna e da linha, da linha original, com a linha em análise. Considerando o exemplo da figura 1, estamos a analisar a linha 0. Podemos verificar que a rainha da linha 0 colide com a linha 1. Usando a nossa fórmula, chegamos a essa conclusão:

Para rainha da linha 0: 2 (índice da coluna) - 0 (índice da linha) = 2

Para rainha da linha 1: 3 (índice da coluna) - 1 (índice da linha) = 2

	0	1	2	3
0	0	0	1	0
1	0	0	0	1
2	1	0	0	0
3	0	1	0	0

Figura 1: Board com n=4

Conclusão/Observações:

No âmbito deste trabalho, foi nos dada a liberdade de escolher qualquer algoritmo de IA (inteligência artificial), ou qualquer algoritmo de procura desde que siga o design pattern Strategy, o que estimulou a procura e a analise de vários algoritmos possíveis para a resolução deste problema, no qual escolhemos como melhor abordagem o algoritmo denominado de Hill Climbing.

Com este algoritmo e alguns ajustes, para n = 45, a solução é encontrada em menos de um segundo e para n = 160, a solução é encontrada em mais ou menos 1 minuto.

Em conclusão achamos que a escolha do algoritmo e da abordagem ao problema foi adequada, faltando alguns ajustes em relação à eficiência do programa para que o mesmo ache uma solução em menos tempo, possibilitando assim uma melhor pontuação no concurso, que faz com que a programação tenha uma componente mais competitiva.

Referências bibliográficas:

Última vez visto: 19-11-2022(19:29)

https://www.youtube.com/watch?v=7fjmGWkv-sY&t=547s