The N-Queens Problem

Computação Visual - Projeto #1

Pedro Valério 88734, Rodrigo Rosmaninho 88802

Resumo - O presente artigo tem por objetivo documentar o processo de realização do primeiro projeto da cadeira de Computação Visual do Mestrado Integrado em Engenharia de Computadores e Telemática, que visa explorar as capacidades da API WebGL num ambiente web. Neste caso, o tema escolhido foi uma visualização da resolução do conhecido problema das N Damas. Serão, também, apresentados alguns detalhes de implementação e planeamento.

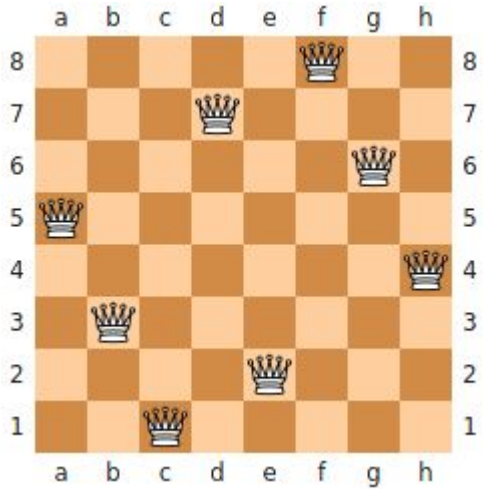
Este relatório começa por apresentar uma breve introdução do tema e estabelecer a sua ligação com o propósito do projecto. De seguida, são explicados os vários elementos implementados para realizar o objectivo principal, detalhando como estes interagem entre si para tal, começando por explicar o algoritmo utilizado para a resolução do problema e de que forma é este utilizado no programa principal. Entrando na componente visual do projecto, são apresentados os vários modelos usados na representação gráfica da solução e outros elementos que os complementam, nomeadamente, o modelo de iluminação e as animações utilizadas. No contexto de animações, é apresentado o fluxo de operações definido, de modo a garantir uma progressão consistente das várias fases de animação.

Estão, neste relatório, também, incluídas todas as interacções possíveis entre o utilizador e o programa.

Por fim, são relatadas algumas dificuldades sentidas ao longo da implementação e a organização geral do projeto.

I. INTRODUÇÃO

O problema das N damas, proposto por Max Bezzel e Franz Nauck em 1850, consiste na colocação de N damas num tabuleiro de xadrez com dimensões NxN de forma a que o estado resultante seja seguro, ou seja, que nenhum par de damas se possa atacar mutuamente. Existem soluções para este problema para todos os números naturais *n* exceto *n*=2 e *n*=3.

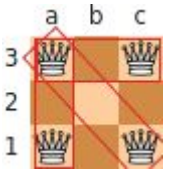


(1)

Fig 1: Solução possível para N=8

Dadas as regras de movimento das damas, define-se uma posição perigosa como qualquer posição em que seja possível estabelecer uma linha reta vertical, horizontal, ou diagonal com uma dama existente.

Assim, como critério base nenhuma coluna ou linha pode conter mais do que uma dama. Sendo ainda necessário verificar todas as diagonais.



(2)

Fig 2: Exemplificação das regras de movimento das damas no tabuleiro

Este puzzle inspirou algumas variantes e problemas relacionados, entre os quais o '*N-Queens Completion Problem*', que consiste na procura de uma solução num tabuleiro que já contém uma ou mais damas pré-colocadas e cuja posição é fixa. Esta variante é também explorada neste projeto.

II. ALGORITMO

Com o decorrer do tempo foram propostos vários algoritmos para a resolução deste problema num sistema computacional.

Para este projeto foi escolhido um algoritmo 'ingénuo' que se apresenta de seguida, de modo a demonstrar claramente o processo e eventuais *backtracks* necessários. A escolha deste tipo de algoritmo em contraste a outros existentes mais eficientes tem como propósito poder realçar a representação gráfica dos vários eventos possíveis na resolução do problema.

A solução consiste em, de uma forma resumida, começando pela primeira coluna, fixar uma dama e verificar recursivamente se essa escolha leva a uma solução possível nas restantes colunas. Caso tal não aconteça é feito o *backtracking* para testar outra posição possível.

---

function solveRec(board, column)
input: matriz bi-dimensional com o estado atual do tabuleiro (board), índice da coluna atualmente considerada (column)
output: valor booleano representativo da descoberta (ou não) de uma solução possível

```
if column >= board.length then
|  return true
for all rows row in board  do
|  board[row][column] ←1
|  if isSafe(board, row, column) then
|  |  if solveRec(board, column +1) then
|  |  |  return true
|  board[row][column] ←0
return false
```

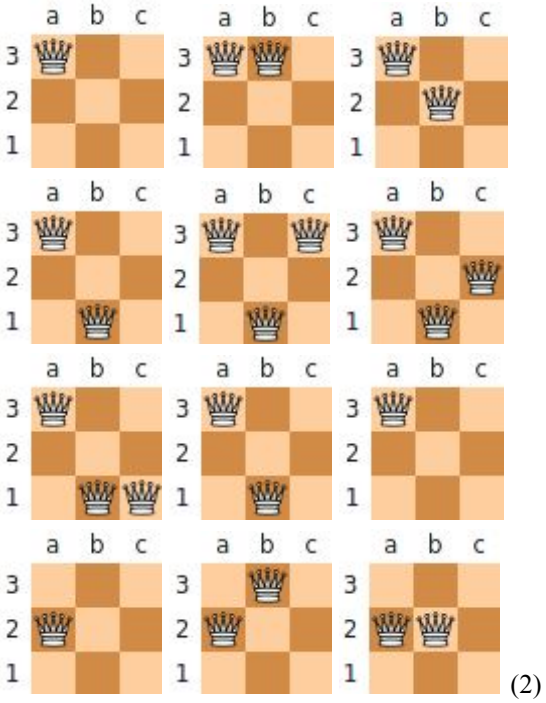
Sendo *isSafe(board, row, column)* uma função que verifica possíveis conflitos na linha, coluna, ou nas diagonais da raínha fornecida como argumento e retorna um valor booleano.

O algoritmo foi implementado em JavaScript e posteriormente adaptado para permitir o registo numa estrutura auxiliar de todos os movimentos feitos ao longo do processo, o respetivo estado do tabuleiro, e uma lista de damas em perigo.
Esta informação é calculada e fornecida à lógica principal da animação assim que o utilizador escolhe o valor de N.

Adicionalmente, foram feitas ligeiras alterações referentes ao funcionamento do sistema com uma raínha fixa pré-posicionada.

N	Nº de passos necessários
2	8 (impossível)
3	23 (impossível)
4	30
5	15
6	196
7	44
8	981
9	365
10	1067
11	558
12	3315
13	1463
14	28380
15	21624

Tab 1: Relação entre o parâmetro de entrada N e o número de passos necessários à resolução do problema



(2)

Fig 3: **Excerto** do processo elaborado pelo algoritmo para N=3 (com duas situações de backtracking)

III. MODELOS

Para a visualização da resolução do problema, foi criado um ambiente 3D interativo, que faz uso de dois modelos principais: Dama e Tabuleiro. Estes são gerados programaticamente e obedecem a uma estrutura específica.

Todos os modelos são instâncias de uma classe caracterizada por um conjunto de grupos de Primitivas que contêm informação sobre:

- tipo: o tipo de primitiva a desenhar no contexto WebGL
- vértices: o conjunto de vértices que formam a primitiva
- normais: os vectores normais associados aos vértices (calculados consoante o tipo de primitiva)
- parâmetros do modelo de iluminação de Phong, nomeadamente, *ambience*, *diffusion*, *specular*, *shininess* e ainda uma cor associada à constante de *diffusion* usada para atribuir uma cor RGB à superfície da primitiva.

Aquando do momento do desenho dos modelos na cena, a função responsável por desenhar um modelo irá iterar sobre e desenhar todas as primitivas contidas neste.

De modo a poder utilizar os modelos no ambiente 3D e atendendo ao facto que há possibilidade de haver modelos que são reutilizados (nomeadamente o da Dama), é utilizada uma classe Objeto que contém um dado modelo. Deste modo, vários objetos distintos podem utilizar um único modelo, visto que é no objecto que se encontram as variáveis de transformação: Translação, Rotação e Escala e ainda uma variável booleana indicativa se o modelo deve ser desenhado ou não (utilizada na animação, de modo a poder esconder temporariamente um objecto sem ter de o remover da cena).

Ambos os modelos referidos (Dama e Tabuleiro) são recalculados quando o valor do parâmetro N do problema muda.

A. Dama

O modelo da Dama é constituído por uma base (Triangle Fan), vários níveis de Triangle Strips, de modo a formar a sua estrutura em coluna, uma face no topo (Triangle Fan) e um conjunto de triângulos para formar o topo. Está, ainda, feita de modo às suas dimensões variarem com o valor de N, sendo possível, desta forma, preservar alguma proporcionalidade entre o tabuleiro e a Dama para os vários valores de N.

Fig 4: Modelo da Dama para os valores de N mais reduzidos

Fig 5: Modelo da Dama para os valores de N mais elevados

B. Tabuleiro

O modelo do Tabuleiro é formado por três primitivas, o conjunto das casas brancas, o conjunto das casas pretas e as bordas castanhas do tabuleiro. Todas estas primitivas são conjuntos de triângulos calculados, consoante o valor de N.

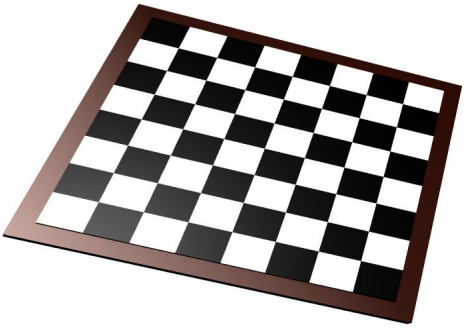


Fig 6: Tabuleiro de N = 8

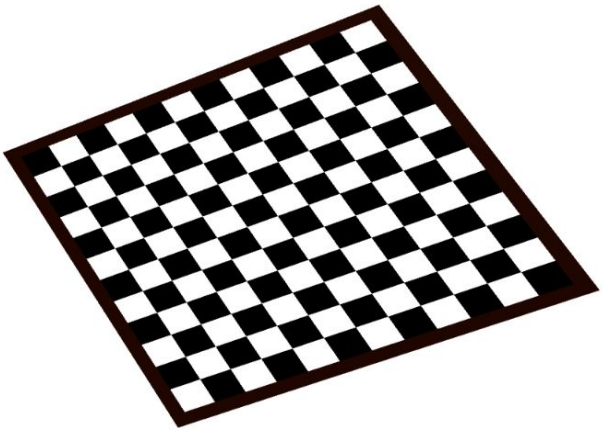


Fig 7: Tabuleiro de N = 12

C. Casa do Tabuleiro

Para efeitos de sinalização de casas em conflito ou para indicação de algoritmo bem sucedido no decorrer da animação, é usado um modelo simples de um quadrado ao qual é atribuída uma cor.

IV. ILUMINAÇÃO

Neste projecto é utilizado o modelo de iluminação de Phong, tendo cada modelo (como foi referido anteriormente) informação relativa ao mesmo. É utilizada uma única fonte de luz fixa branca durante todo o tempo da animação. Para além do cálculo da reflexão da luz nos modelos, é-lhes também atribuída uma cor associada à constante de difusão aquando deste cálculo.

Para o efeito, foram usados os *vertex shader* e *fragment shader* disponibilizados ao longo das aulas, especificamente, os contidos no guião 8. Para permitir a atribuição de uma cor, foi feita uma edição do código do vertex shader para receber um argumento de cor RGB utilizado no cálculo da componente de difusão.

V. ANIMAÇÃO

O objectivo principal deste projecto é demonstrar a resolução do problema das N Damas numa animação. Para tal, um dos modos de funcionamento do programa é a visualização sequencial e automática dos passos do algoritmo de resolução. Nesta animação, foram identificados diferentes eventos com interesse em representar:

- Nova Dama: Uma nova dama é introduzida no tabuleiro.
- Mover Dama: Uma dama existente desloca-se para uma casa diferente.
- Remover Dama: Uma dama existente é removida por completo do tabuleiro, numa situação de *backtracking* do algoritmo.
- Sucesso: O algoritmo termina bem sucedido
- Fracasso: O algoritmo termina não tendo conseguido encontrar uma solução.

Para cada um destes eventos, foi desenvolvida uma Sub-Animação implementada numa classe controlada com um temporizador, uma velocidade de incremento do mesmo e uma função que transforma os objectos pretendidos regida por este temporizador.

Para criar os diversos efeitos das diferentes animações, a manipulação dos parâmetros de transformação é feita seguindo regras e fórmulas físicas, especialmente, as equações de movimento de um corpo e funções trigonométricas de sinais, todas estas em função do tempo.

Isto permite que as sub-animações sejam independentes umas das outras e do programa global, podendo ser facilmente criadas e integradas. Em qualquer instante, um máximo de uma sub-animação pode estar a ser reproduzida, por isso, no programa principal, uma variável controla qual o objeto sub-animação atual e chama a sua função de atualização de *frame* no momento adequado.

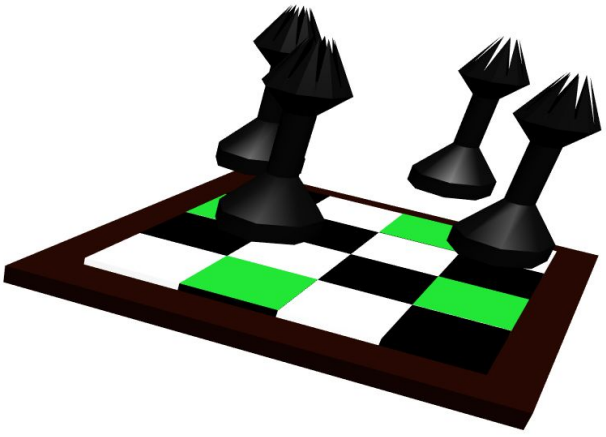


Fig 8: Animação de sucesso num tabuleiro N=4

VI. FLUXO DE EXECUÇÃO

Para assegurar a progressão correta da animação foi utilizada na função *tick()* uma abordagem análoga a uma máquina de estados.

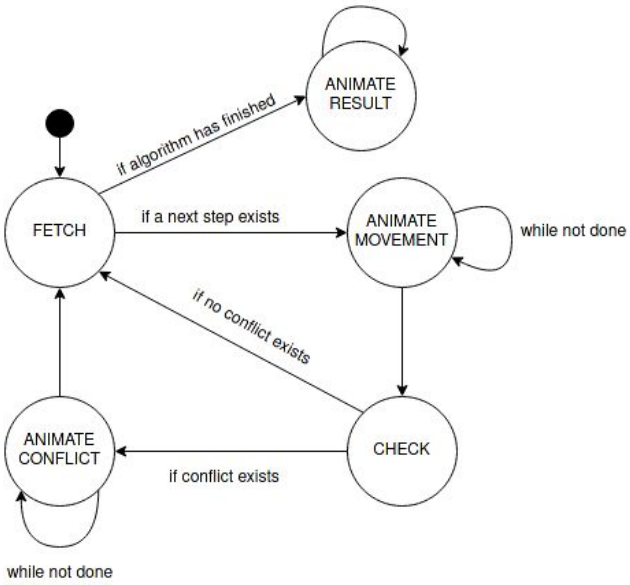


Fig 9: Diagrama de Estados

Em cada chamada da função é executada a lógica referente ao estado atual e o estado seguinte é determinado.

De notar que os estados que consistem na apresentação de sub-animações referentes a eventos permanecem ativos até que essa apresentação chegue ao seu fim.

- FETCH: As variáveis relevantes são atualizadas com o estado do tabuleiro no passo atual, proveniente do algoritmo descrito na secção II.
- ANIMATE MOVEMENT: O passo atual do algoritmo é animado no tabuleiro 3D de uma de três maneiras possíveis:
  - A inserção de uma nova dama do tabuleiro.
  - O movimento de uma dama entre duas posições do tabuleiro.
  - A remoção de uma dama, como consequência de um processo de backtrack.
- CHECK: O estado atual do tabuleiro é verificado com base na informação proveniente do algoritmo.
- ANIMATE CONFLICT: As posições que contêm damas que constituem um perigo para a dama atual são animadas.
- ANIMATE RESULT: É representada a sub-animação de sucesso ou fracasso até que o utilizador escolha um novo valor de N ou reinicie a animação atual.

VII. CONTROLO DE UTILIZADOR

A animação e o funcionamento do programa também permitem algum nível de *feedback* e controlo por parte do utilizador.

Durante uma animação corrente, é possível ver exatamente em que passo da solução se encontra a animação, bem como quantos *backtracks* já foram efectuados.

Ao nível de controlo, é possível controlar o fluxo da animação com funcionalidades tradicionais de um visualizador de vídeo:

- Pausa: coloca a animação em pausa, desligando as sub-animações, sendo possível ver a animação passo-a-passo (através de um botão ou da barra de espaços do teclado).
- Reproduzir: restaurar a progressão automática de passos da solução (mesmo método da anterior).
- Step-Back/Forward: avançar um passo à frente ou atrás. No caso de a animação estar colocada em pausa, as sub-animações não são reproduzidas (através de botões ou das teclas de direção esquerda ou direita, respectivamente).
- Reiniciar: voltar ao primeiro passo da solução (através de botão ou da tecla R do teclado).
- Controlo da velocidade de reprodução: utilizando a barra de velocidade, é possível aumentar a velocidade da animação (através da barra de velocidade do GUI ou das teclas +/- do teclado).

Os controlos destas ações por teclado podem ser consultados movendo o rato sobre os botões equivalentes no GUI.

Para poder visualizar vários casos de resolução do algoritmo, o utilizador também pode mudar o valor de N. No contexto deste projecto, o valor de N pode variar entre 2 e 15 inclusive. No caso de mudança de N, os modelos são recalculados, o algoritmo é novamente resolvido para o novo valor de N e a animação é reiniciada com a nova solução.

Foi, ainda, adicionada a possibilidade de escolher fixar uma posição inicial que deve ser incluída na solução final (se for possível). Desta forma representando também o problema de '*N-Queens Completion*'.

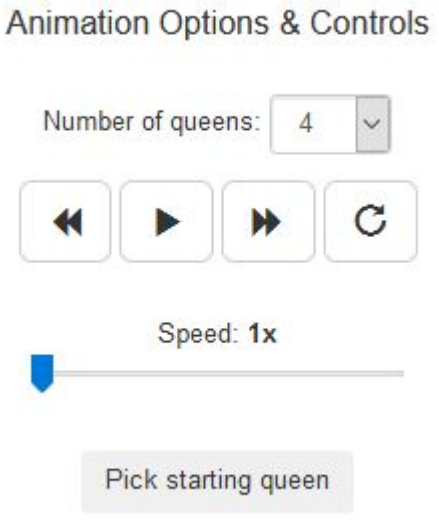


Fig 10: Controlo de fluxo de animação e parâmetros do algoritmo

No que toca à posição e características da câmara, é possível rodá-la verticalmente e horizontalmente usando o botão esquerdo e movimentos do rato, assim como aproximar e afastar com a roda do rato. Estes controlos foram implementados, recorrendo a variáveis de rotação e escala globais aplicadas a todos os objectos da cena, sobre as suas transformações locais.

Também é possível colocar a câmara nas suas definições de rotação e zoom iniciais. De notar que, apesar de não haver efeito na rotação, o nível de zoom aplicado no estado inicial da câmara é calculado de acordo com o N actual, de modo a ser possível visualizar todo o tabuleiro.

Em alternativa, pode-se escolher uma vista *top-down* tradicional de jogos de xadrez virtuais, sendo esta vista também usada na escolha de uma posição inicial para uma dama, que pode ser selecionada com recurso ao rato.

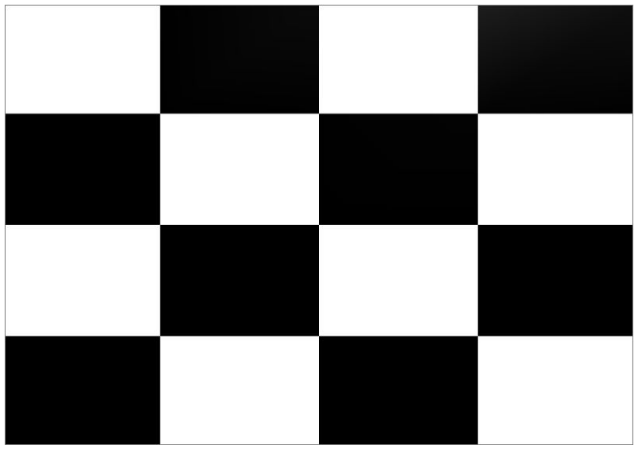


Fig 11: *Vista top-down*

Quanto às características, o utilizador pode escolher entre uma vista em perspectiva e uma vista ortogonal e, ainda, ligar ou desligar o *back-face-culling*.

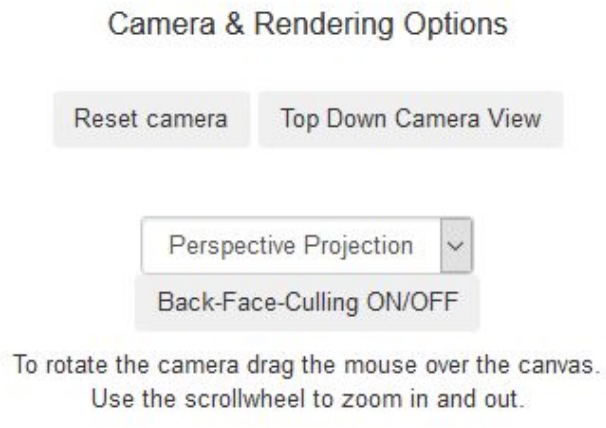


Fig 12: Opções de câmara

VIII. ORGANIZAÇÃO DE CÓDIGO FONTE

O código fonte desenvolvido ao longo do projeto encontra-se organizado da seguinte forma:

- **main.js:** programa top-level do projecto, responsável pelo controlo e gestão dos dados e entidades utilizados.
- **scene.js:** definição das entidades a serem utilizadas no ambiente gráfico.
- **maths.js:** lógica associada a funções matemáticas relevantes em múltiplos ficheiros
- **initShaders.js:** rotinas de inicialização dos shaders usados na visualização gráfica.
- **lightSources.js:** definição e inicialização dos dados de fontes de luz.
- **subanims.js:** implementação das várias classes de sub-animações e o seu fluxo.
- **webgl-utils.js:** funções e lógica de simplificação do código WebGL relevante.
- **nqueens.js**: implementação do algoritmo de resolução do problema das damas apresentado na secção II.
- Ficheiros relativos à interface web e à sua correta apresentação no browser (UI).
    - **index.html, bootstrap.min.css, glyphicons-regular.woff2**

IX. DIFICULDADES A CONSIDERAR

Ao longo da realização do projecto, foram encontrados alguns problemas provenientes de dificuldade de implementação de certos aspectos, bem como de erros causados por funcionalidades já existentes, pelo que foi necessário agilizar e adaptar certos detalhes.

O mais relevante destes aspectos foi a escolha de não integrar texturas no projecto, pois foi decidido que seria bastante complexo conseguir integrá-las no código existente em conjunto com outros aspectos já implementados. A principal razão para evitar esta abordagem foi a necessidade associada ao tema do projecto de ter um tabuleiro de tamanho variável, pelo que a textura utilizada para a face também teria de ser substituída (neste caso, um simples redimensionamento da textura não seria eficaz, pois não iria adicionar mais casas ao tabuleiro, só redimensionar as existentes), de modo que a edição das faces do modelo permite, neste caso, uma maior flexibilidade.

Uma outra dificuldade sentida foi, também, conciliar um modelo de iluminação com a utilização de diferentes cores para diferentes modelos, sendo que esta foi resolvida, editando o código dos shaders para também incluir uma cor RGB passada como argumento no cálculo da componente de difusão.

X. REFERÊNCIAS

A implementação desenvolvida utilizou como base o código fornecido nas aulas práticas de Computação Visual.

Foram, ainda, utilizados alguns recursos online para esclarecimento de dúvidas pontuais sobre JavaScript e WebGL (ie: StackOverflow). Não foram, no entanto, utilizados excertos significativos de código de terceiros.

Por último, foi usada a framework de CSS *Twitter Bootstrap*, por fim de auxiliar na estruturação da interface web (UI).

(1) - Imagem proveniente da Wikipédia, https://en.wikipedia.org/wiki/Eight_queens_puzzle

(2) - Adaptações manuais de imagens base provenientes da Wikipédia, https://en.wikipedia.org/wiki/Eight_queens_puzzle