

MANUAL DOS CÓDIGOS DO PROGRAMA CAVALO.EXE

Daiane Araujo

Leonardo Gouvêa

Luana Vieira

Universidade Federal da Bahia

10 de março de 2016

Salvador, Bahia

Análise do programa

Observações importantes:

- Todos os códigos foram feitos no ambiente Windows x64 com o aplicativo notepad++.
- O compilador utilizado foi o MinGW.
- Foi utilizado a API OpenGL com o freeglut para a realização da interface gráfica.
- Foi utilizado como base da interface gráfica o código Knigh's Tour/C¹ do Rosetta Code e adaptado para a finalidade final do programa.
- O usuário pode terminar o programa a qualquer momento.

Funcionamento:

Objetivo do código:

Dado as especificações do tabuleiro retangular, a posição inicial do cavalo e uma busca, o programa procura se existe algum caminho onde o cavalo visite todas a casas do tabuleiro somente uma única vez.

Entrada do código:

```
Informe as dimensoes x e y do tabuleiro separados por espaco: 5 5
Informe a posicao inicial x e y do cavalo separados por espaco: 2 2

Tipos de buscas
1) Busca em profundidade.
2) Busca em Largura.
3) Busca gulosa (heuristica).
Digite o indice da busca desejada: 3
```

Figura 1: Exemplo de entrada. Um tabuleiro 5x5, começando no espaço (2,2) e realizando uma busca gulosa.

O código recebe como entrada a largura e a altura do tabuleiro retangular, posição inicial x e y do cavalo, e o tipo de busca desejado pelo usuário. Altura e Largura podem ser quaisquer números naturais diferente de 0. As posições iniciais x e y do cavalo podem ser quaisquer números naturais, mas não podem ser maiores ou iguais que a largura e altura respectivamente. O programa implemente 3 buscas, Profundidade, Largura e Gulosa, e cada uma pode ser escolhida digitando o índice correspondente.

Saída do código:

Caso exista alguma solução, o programa exibe uma interface gráfica mostrando os movimentos realizados pelo cavalo. Após isso, exibe em forma de string o caminho tomado pelo cavalo e algumas características da busca: Tempo total, Nós visitados e memória máxima alocada.

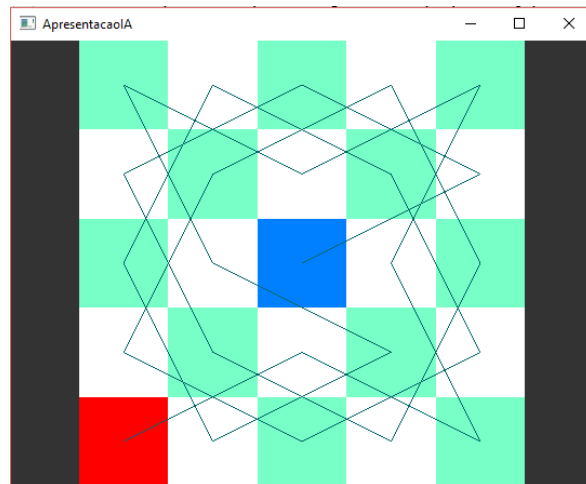


Figura 2: Exemplo de saída gráfica da entrada anterior.

```
Passeio completo
Aperte espaco para continuar
Solucao: (2 2)->(4 3)->(2 4)->(0 3)->(1 1)->(3 0)->(4 2)->(3 4)->(1 3)->(0 1)->(2 0)->(4 1)->(3 3)->(1 4)->(0 2)->(1 0)->(3 1)->(1 2)->(0 4)->(2 3)->(4 4)->(3 2)->(4 0)->(2 1)->(0 0)

Tempo decorrido: 0.00e+000s
Nos visitados: 2.50e+001
Memoria maxima utilizada: 2.46e+003 Bytes

Pressione qualquer tecla para terminar o programa.
```

Figura 3: Exemplo de saída estatística.

Se não existe solução ou o programa tem termino forçado pelo usuário, ele exibe somente as estatísticas.

Comportamento do programa

O programa pega a posição atual e verifica se a mesma é final. Caso não seja, ela pode ser caminho para a solução, então ela é inserida no caminho da solução. Então determina quais as próximas posições legais o cavalo pode assumir e os coloca como filhos do nó atual. Depois decide visitar o sucessor. O sucessor do nó atual depende da busca que estamos realizando.

- Busca em Profundidade: O sucessor é sempre o filho mais à esquerda. Caso este não exista, o sucessor será o filho mais à esquerda não visitado do nó pai. Tem natureza recursiva.
- Busca em Largura: O nó atual é inserido numa fila e seus filhos também, da esquerda para a direita. Este nó é retirado da fila e então o próximo da fila é visitado. Tem natureza iterativa.
- Busca Gulosa: Semelhante ao de profundidade, mas ao invés de escolher o filho mais à esquerda, escolhe que apresenta o menor número de movimentos possíveis. Tem natureza recursiva.

Caso o estado atual não faça parte da solução, ele é retirado da solução.

O programa encontra solução se ela existir. Caso ela não existe, a busca visitará TODO espaço de estados. Como tal procedimento pode demorar muito, o usuário pode para o programa a qualquer hora apertando qualquer tecla e será exibida as estatísticas da busca até aquele momento.

Durante a animação gráfica da solução, o usuário pode acelera-la, diminuir sua velocidade ou pula-la completamente. A para acelerar, basta apertar ‘>’ ou ‘.’. Para diminuir a velocidade, basta pressionar ‘<’ ou ‘.’. Para pular a animação, basta pressionar barra de espaço.

Utilização de memória

A memória é alocada dinamicamente para cada nó novo guardado na agenda. As buscas em profundidade e gulosa tem um limite bem comportado de memória, de ordem linear. Como a agenda da busca em profundidade é maior e mais complexa, ela torna-se exponencial.

Resultados experimentais

A busca em profundidade funciona bem para tabuleiros de até 6x6, tornando-se inviável para tabuleiros maiores. O de largura já apresenta demora num tabuleiro 5x5, demorando mais de 10 minutos para encontrar uma solução. A busca gulosa consegue encontrar a solução de maneira eficiente, caso exista. Se não existe solução, todas as buscas tornam-se inviáveis em tabuleiros muito grandes.

Análise do código

Observações importantes:

- Não iremos analisar o código `stuff.c`. O código é o de interface gráfica adaptado de um código já existente, como dito anteriormente. Explicar tal código mostra-se desnecessário e foge da razão desse manual.

Organização

O programa é composto por dois códigos principais, `Buscasjogos.c` e `stuff.c`, e um header personalizado, `stuff.h`. A lógica principal por trás do programa encontra-se em `Buscasjogos.c`. O código `stuff.c` encontra-se a lógica pela interface gráfica, utilizando-se do OpenGL e do GLUT. O header `stuff.h` contém as estruturas utilizadas no código principal e a assinatura da função de `stuff.c` que chamamos em `Buscasjogos.c` para realizar a interface.

Estruturas e constantes globais

No código existem três estruturas principais: *Nodo*, *Agenda* e *Lista*. *Nodo* é usado em todas as buscas e é uma representação do estado e do espaço de estados. Um nó contém o ponto daquele estado, as próximas possíveis posições ainda não visitado naquele ramo e a função de avaliação. A função de avaliação é somente usada em busca gulosa e representa o número de movimentos possíveis que aquele estado tem.

Agenda é uma fila de nós e é utilizada somente em busca por largura. A agenda contém um nodo (estado), seu sucessor e a lista de posições visitadas no ramo na árvore de estados. Note que não é necessário guardar as posições visitadas para as outras buscas, pois a natureza recursiva delas já armazena.

Lista é uma lista de posições. Ela serve tanto para guarda as posições visitadas para cada nó na busca em largura, como devolver a solução, caso ela exista, para as buscas.

Utilizamos de uma série de constantes globais. Uma apresentação rápida nas mais simples, *nos* guarda o número de nós visitados; *altura* é a altura em que se encontra o estado atual na árvore de estados. O vetor *board* são as dimensões do tabuleiro. *begin*, *end* e *time_spent* são contadores de tempo.

As constantes que merecem certa observação são *estadofinal*, *memoria[3]* e *memoriamax*. Como apresentado nos slides, dado a natureza do problema, sabemos que qualquer nó que se encontre na altura igual à largura vezes altura do tabuleiro menos 1 será um estado final. Logo sempre comparamos a altura do estado atual com a altura onde se encontra a solução, se ela existir, para saber se aquele estado é solução. *memoria[3]* guarda a quantidade de alocação de memória utilizado para criar as estruturas descritas anteriormente. Ela serve para nos indicar a quantidade máxima de memória que a busca utilizou e guarda em *memoriamax*.

Funções

int main()

- Função principal do programa, realiza a obtenção de entradas, faz a chamada das buscas e da impressão de saída, tanto graficamente quanto estatisticamente.
- Verifica se as entradas do usuário são corretas. Pode apresentar comportamento inesperado se inserido qualquer elemento que não seja um número.
- Todas as buscas retornam 2 caso o usuário force a parada da busca.

void possiveisPosicoes(struct nodo **p, struct lista *visitados);

- Tem como entrada um nodo e a lista das posições já visitadas no ramo daquele nodo.
- A função não retorna nada, mas atualiza o nodo atual com seus filhos, caso existam.
- A função realiza todos os movimentos do cavalo da posição atual e verifica se o movimento é legal. Caso seja, verifica se a posição futura já não foi visitada. Caso não tenha sido, essa posição torna-se nó filho do nó atual.
- Atualiza fa, indicando quantos filhos aquele nó tem.

void possiveisMovimentos (struct nodo **p, struct lista *visitados)

- Semelhante ao anterior, mas não gera mais filhos. Somente indica a quantidade de movimentos legais dos filhos do nó atual, caso existam.

int buscaProf(struct nodo *p, struct lista **solu)

- Tem como entrada o nó a ser visitado e a lista dos nós visitados naquele ramo.
- Caso a lista não exista, cria-se uma nova e insere a posição atual nela. Caso exista, somente insere a posição atual no final da lista.
- Verifica se o estado atual é final. Caso não seja, encontra os filhos daquele nó e visita o sucessor, recursivamente.
- Caso o estado atual seja final, retorna 1 e devolve a lista atualizada com a solução.
- Caso o estado atual seja uma folha ou não pertença à solução, remove-o da lista e retorne 0.

int buscaLarg(struct agenda **memory, struct lista **solu)

- Tem como entrada uma agenda e a lista da possível solução.
- Verifica se o primeiro elemento da fila é estado final. Caso não seja, encontra os filhos do nó atual e os insere na fila, da esquerda para a direita.
- Retira o elemento atual da fila.
- Caso o elemento atual da fila seja estado final, retorna 1 e devolve a lista de nós visitados do estado atual.
- Caso a agenda fique vazia, retorne 0.

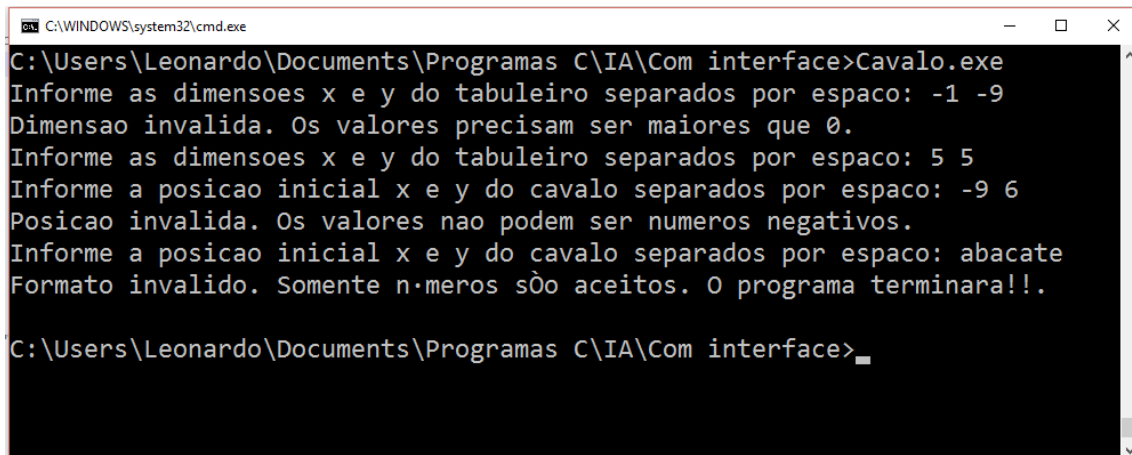
int buscaGulosa (struct nodo *p, struct lista **solu)

- Funcionamento recursivo idêntico a busca em profundidade. Com a única diferença que utilizada a função possiveisMovimentos().

Testes

Caso 1: Entradas incorretas (valores ilegais ou formatos inválidos)

Verificação: Em caso de valores ilegais, o programa consegue reagir ao erro e repara-lo em tempo de execução, pedindo ao usuário que informe os valores corretos. Em casos de formatos inválidos, dado a natureza do scanf(), foi preferível adotar a estratégia de abortar o programa caso o usuário insira algo que não seja um número inteiro. Outra estratégia seria limpar o buffer de entrada, mas não foi adotada nesse programa por não ver razão para fazê-lo.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>Cavalo.exe
Informe as dimensoes x e y do tabuleiro separados por espaco: -1 -9
Dimensao invalida. Os valores precisam ser maiores que 0.
Informe as dimensoes x e y do tabuleiro separados por espaco: 5 5
Informe a posicao inicial x e y do cavalo separados por espaco: -9 6
Posicao invalida. Os valores nao podem ser numeros negativos.
Informe a posicao inicial x e y do cavalo separados por espaco: abacate
Formato invalido. Somente n-meros s-ao aceitos. O programa terminara!!.
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>
```

Figura 4: Exemplo de entradas com valores ilegais e ou formato inválido e como o programa se comporta.

Caso 2: Tabuleiro pequeno (1x1, 2x1, 1x2 e 2x2)

Verificação: Foi observado o programa em tabuleiros pequenos demais para que o cavalo tivesse qualquer movimento. Com exceção do 1x1, não a solução. O retorno, para todas as buscas, apresentara o esperado.


```
C:\WINDOWS\system32\cmd.exe - Cavalo.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>Cavalo.exe
Informe as dimensoes x e y do tabuleiro separados por espaco: 1 1
Informe a posicao inicial x e y do cavalo separados por espaco: 0 0

Tipos de buscas
1) Busca em profundidade.
2) Busca em Largura.
3) Busca gulosa (heuristica).
Digite o indice da busca desejada: 1

Passeio completo
Aperte espaco para continuar
Solucao: (0 0)

Tempo decorrido: 0.00e+000s
Nos visitados: 1.00e+000
Memoria maxima utilizada: 5.60e+001 Bytes

Pressione qualquer tecla para terminar o programa._
```

Figura 5: Busca em 1x1 encontra resultado

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>Cavalo.exe
Informe as dimensoes x e y do tabuleiro separados por espaco: 1 2
Informe a posicao inicial x e y do cavalo separados por espaco: 0 0

Tipos de buscas
1) Busca em profundidade.
2) Busca em Largura.
3) Busca gulosa (heuristica).
Digite o indice da busca desejada: 1

Nao ha solucao

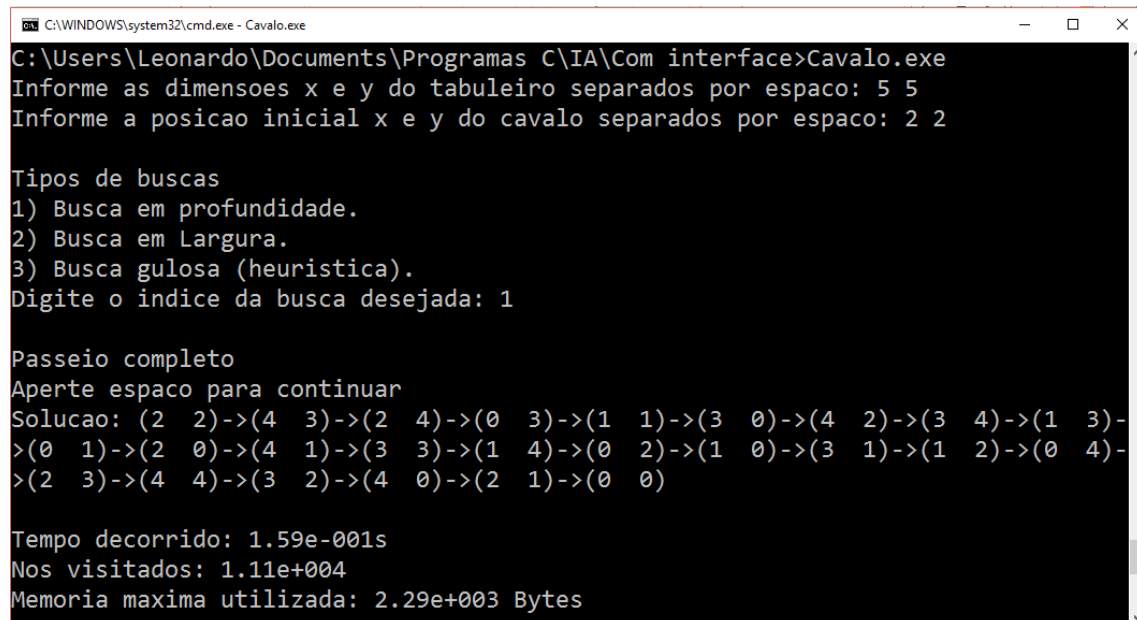
Tempo decorrido: 0.00e+000s
Nos visitados: 1.00e+000
Memoria maxima utilizada: 5.60e+001 Bytes

Pressione qualquer tecla para terminar o programa.
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>_
```

Figura 6: Busca em 1x2 não encontra solução

Caso 3: Verificar se todas as buscas encontram solução quando existe.

Verificação: Para este caso, foi testado o limiar da busca em profundidade. Para todas as buscas, foram definidas um tabuleiro 5x5 começando do centro (2,2). Sabe-se que a solução para esse caso existe. O resultado foi o esperado.



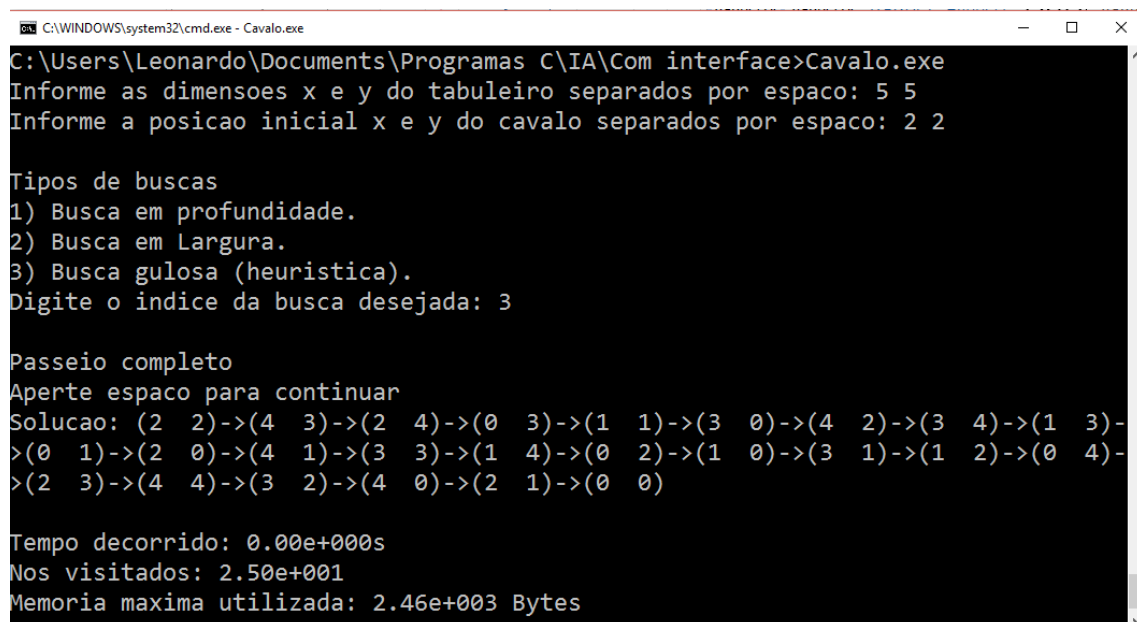
```
C:\WINDOWS\system32\cmd.exe - Cavalo.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>Cavalo.exe
Informe as dimensoes x e y do tabuleiro separados por espaco: 5 5
Informe a posicao inicial x e y do cavalo separados por espaco: 2 2

Tipos de buscas
1) Busca em profundidade.
2) Busca em Largura.
3) Busca gulosa (heuristica).
Digite o indice da busca desejada: 1

Passeio completo
Aperte espaco para continuar
Solucao: (2 2)->(4 3)->(2 4)->(0 3)->(1 1)->(3 0)->(4 2)->(3 4)->(1 3)->(0 1)->(2 0)->(4 1)->(3 3)->(1 4)->(0 2)->(1 0)->(3 1)->(1 2)->(0 4)->(2 3)->(4 4)->(3 2)->(4 0)->(2 1)->(0 0)

Tempo decorrido: 1.59e-001s
Nos visitados: 1.11e+004
Memoria maxima utilizada: 2.29e+003 Bytes
```

Figura 7: Busca em Profundidade



```
C:\WINDOWS\system32\cmd.exe - Cavalo.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>Cavalo.exe
Informe as dimensoes x e y do tabuleiro separados por espaco: 5 5
Informe a posicao inicial x e y do cavalo separados por espaco: 2 2

Tipos de buscas
1) Busca em profundidade.
2) Busca em Largura.
3) Busca gulosa (heuristica).
Digite o indice da busca desejada: 3

Passeio completo
Aperte espaco para continuar
Solucao: (2 2)->(4 3)->(2 4)->(0 3)->(1 1)->(3 0)->(4 2)->(3 4)->(1 3)->(0 1)->(2 0)->(4 1)->(3 3)->(1 4)->(0 2)->(1 0)->(3 1)->(1 2)->(0 4)->(2 3)->(4 4)->(3 2)->(4 0)->(2 1)->(0 0)

Tempo decorrido: 0.00e+000s
Nos visitados: 2.50e+001
Memoria maxima utilizada: 2.46e+003 Bytes
```

Figura 8: Busca Gulosa

```
C:\WINDOWS\system32\cmd.exe - cavalo.exe
Deseja fazer nova busca? (s/n)s
Informe as dimensoes x e y do tabuleiro separados por espaco: 5 5
Informe a posicao inicial x e y do cavalo separados por espaco: 2 2

Tipos de buscas
1) Busca em profundidade.
2) Busca em Largura.
3) Busca gulosa (heuristica).
Digite o indice da busca desejada: 2

Solucao: (2 2)->(4 3)->(2 4)->(0 3)->(1 1)->(3 0)->(4 2)->(3 4)->(1 3)->(0 1)->(2 0)->(4 1)->(3 3)->(1 4)->(0 2)->(1 0)->(3 1)->(1 2)->(0 4)->(2 3)->(4 4)->(3 2)->(4 0)->(2 1)->(0 0)

Tempo decorrido: 1.42e+003s
Nos visitados: 6.42e+005
Memoria maxima utilizada: 2.30e+007 Bytes
```

Figura 9: Busca em Largura

Caso 4: Verificar se todas as buscas não retornam solução, quando ela não existe.

Verificação: Este teste só é viável ser feito em ambientes pequenos. Se a solução não existe, todas as buscas visitam TODOS os estados do espaço de estados. O espaço de estado cresce de maneira exponencial com o aumento da altura da árvore, que é a multiplicação da altura do tabuleiro com a largura do tabuleiro. Por isso, o teste foi realizado num espaço 4x4 e posição inicial (2,3), que não há solução. Observe que todas visitam o mesmo número de nós, ou seja, todo espaço de estados.

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>Cavalo.exe
Informe as dimensoes x e y do tabuleiro separados por espaco: 4 4
Informe a posicao inicial x e y do cavalo separados por espaco: 2 3

Tipos de buscas
1) Busca em profundidade.
2) Busca em Largura.
3) Busca gulosa (heuristica).
Digite o indice da busca desejada: 3

Nao ha solucao

Tempo decorrido: 1.50e-002s
Nos visitados: 1.89e+003
Memoria maxima utilizada: 1.24e+003 Bytes

Pressione qualquer tecla para terminar o programa.
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>
```

Figura 10: Busca gulosa

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>Cavalo.exe
Informe as dimensoes x e y do tabuleiro separados por espaco: 4 4
Informe a posicao inicial x e y do cavalo separados por espaco: 2 3

Tipos de buscas
1) Busca em profundidade.
2) Busca em Largura.
3) Busca gulosa (heuristica).
Digite o indice da busca desejada: 2

Nao ha solucao

Tempo decorrido: 3.30e-002s
Nos visitados: 1.89e+003
Memoria maxima utilizada: 7.82e+004 Bytes

Pressione qualquer tecla para terminar o programa.
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>
```

Figura 11: Busca em Largura

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>Cavalo.exe
Informe as dimensoes x e y do tabuleiro separados por espaco: 4 4
Informe a posicao inicial x e y do cavalo separados por espaco: 2 3

Tipos de buscas
1) Busca em profundidade.
2) Busca em Largura.
3) Busca gulosa (heuristica).
Digite o indice da busca desejada: 1

Nao ha solucao

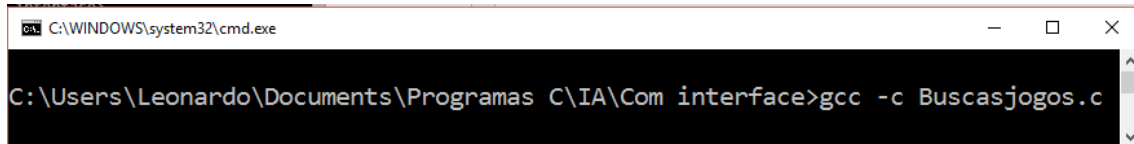
Tempo decorrido: 2.00e-002s
Nos visitados: 1.89e+003
Memoria maxima utilizada: 1.22e+003 Bytes

Pressione qualquer tecla para terminar o programa.
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>
```

Busca 12: Busca em Profundidade.


Compilação

Para compilar o código, é necessário ter a API OpenGL e a biblioteca freeglut. O código de interface foi escrito em estilo C99 e por isso é necessário indicar no compilador. Será mostrado abaixo como foi realizada a compilação do código em ambiente Windows, sendo facilmente adaptável a qualquer ambiente.

A screenshot of a Windows command prompt window. The title bar shows 'C:\WINDOWS\system32\cmd.exe'. The command prompt shows the directory 'C:\Users\Leonardo\Documents\Programas C\IA\Com interface' and the command 'gcc -c Buscasjogos.c' being entered.

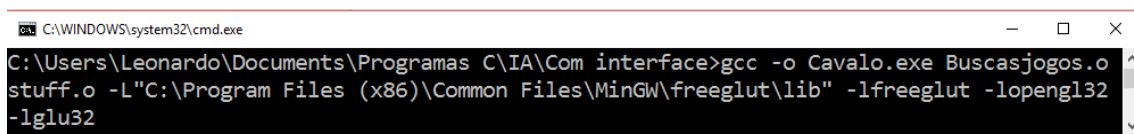
```
C:\WINDOWS\system32\cmd.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>gcc -c Buscasjogos.c
```

Figura 13: Compilando o programa principal

A screenshot of a Windows command prompt window. The title bar shows 'C:\WINDOWS\system32\cmd.exe'. The command prompt shows the directory 'C:\Users\Leonardo\Documents\Programas C\IA\Com interface' and the command 'gcc -c -o stuff.o stuff.c -I"C:\Program Files (x86)\Common Files\MinGW\freelut\include" -std=c99' being entered.

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>gcc -c -o stuff.o stuff.c
-I"C:\Program Files (x86)\Common Files\MinGW\freelut\include" -std=c99
```

Figura 14: Compilando a interface gráfica

A screenshot of a Windows command prompt window. The title bar shows 'C:\WINDOWS\system32\cmd.exe'. The command prompt shows the directory 'C:\Users\Leonardo\Documents\Programas C\IA\Com interface' and the command 'gcc -o Cavalo.exe Buscasjogos.o stuff.o -L"C:\Program Files (x86)\Common Files\MinGW\freelut\lib" -lfreetut -lopengl32 -lglu32' being entered.

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Leonardo\Documents\Programas C\IA\Com interface>gcc -o Cavalo.exe Buscasjogos.o
stuff.o -L"C:\Program Files (x86)\Common Files\MinGW\freelut\lib" -lfreetut -lopengl32
-lglu32
```

Figura 15: Linkando.

Referência

- ROSETTA CODE. Knight's Tour/C. Disponível em: http://rosettacode.org/wiki/Knight%27s_tour/C.¹ Acessado em: 08/03/2016
- TRANSMISSION ZERO. Using freeglut or GLUT with MinGW. Disponível em: <http://www.transmissionzero.co.uk/computing/using-glut-with-mingw/>. Acessado em: 09/03/2016