MAC110 Introdução à Computação
EP no. 3

Prof. Dr. Paulo Miranda
Instituto de Matemática e Estatística (IME)
Universidade de São Paulo (USP)

Sokoban (zelador do armazén) é um tipo de jogo de transporte e movimentação de caixas ou engradados em um armazém. O objetivo é pegar e estocar os engradados em determinadas posições. Sokoban foi criado em 1981 por Hiroyuki Imabayashi, e publicado em 1982 por Thinking Rabbit, uma emrpesa de software localizada em Takarazuka.

Regras:

- Os únicos movimentos que o trabalhador pode realizar são andar e empurrar caixas que somente se deslocam nos sentidos horizontal e vertical. Nenhuma caixa pode ser puxada, sendo assim, quando são colocadas nos cantos do cenário não podem mais serem movidas.

- Apenas uma caixa pode ser movimentada por vez. Não é permitido empurrar duas ou mais caixas juntas de uma só vez.

- O jogador não poderá ocupar o mesmo lugar da caixa ou andar nos muros.

- O jogo é finalizado quando as caixas são colocadas nas posições predeterminadas.

Para melhor entender a dinâmica do jogo, nada melhor do que jogar a versão online disponível em:
http://www.game-sokoban.com/

Atividade:

Faça um programa em C que implementa uma versão em modo texto no terminal do jogo Sokoban.
O seu programa deve ler uma fase do jogo (cenário/mapa do jogo) de um arquivo texto fornecido pelo usuário, mostrar no terminal o cenário correspondente, solicitar a entrada de uma lista de comandos (string), processar os comandos, guardar o histórico de movimentos realizados, permitir que movimentos sejam desfeitos (*undo*), e finalizar o jogo com uma mensagem de vitória, quando as caixas forem colocadas nas posições predeterminadas (locais de armazenamento).

Os formatos de arquivo texto contendo os cenários serão os mesmos que são usados pelos sites:
http://www.game-sokoban.com/ e http://www.sokoban-online.de/

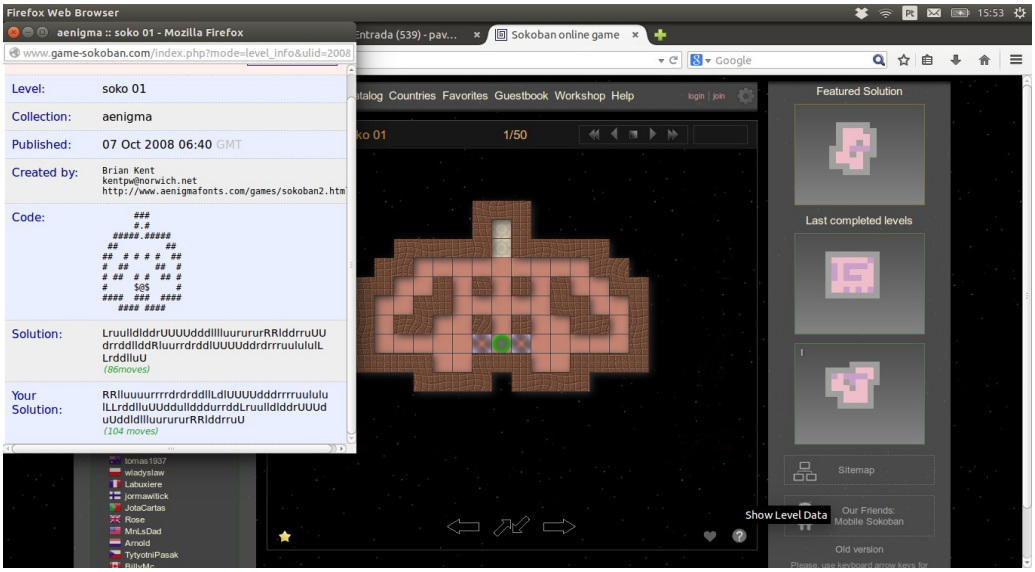


Figura 1: Informações sobre a codificação do cenário podem ser obtidas através do ícone "?" (*Show Level Data*) no site http://www.game-sokoban.com/

Formato padrão:

O formato mais utilizado para representar os níveis Sokoban usa a seguinte tabela de codificação:

Elemento do cenário	Símbolo
Parede	#
Jogador	@
Jogador sobre local de armazenamento	+
Caixa	$
Caixa sobre local de armazenamento	*
Local de armazenamento	.
Piso (posição livre)	(espaço em branco)

O nível mais simples possível corresponde a um arquivo texto com as seguintes linhas:

```
#####
#@$.#
#####
```

É importante que todo o nível seja cercado por muros.

Uma solução do jogo é representada pelo histórico de movimentos do jogador.
O jogador pode mover-se para cima, baixo, esquerda ou direita. Cada movimento é representado por uma das seguintes letras: u, d, l, r (u = *up*, d = *down*, l = *left*, r = *right*).
Letras maiúsculas são usadas no histórico para indicar que uma caixa foi empurrada pelo movimento do jogador. Essa informação é importante para permitir desfazer movimentos (operação de "*undo"*) sem ambiguidade.

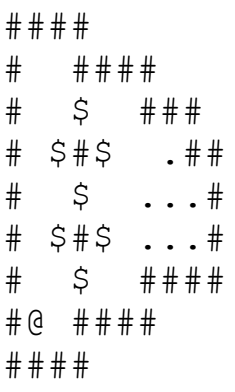
Exemplos de cenários com um possível histórico de solução:

```
#######
#     #
# $+$ #
#.*#*.#
# $.$ #
#     #
#######
```

Solução: LulDrrurrdLrdddlluRdrUdlllluR

```
#####
#   #
#.*.###
#  *  #
# $.**#
#  $ ##
# $ ##
#@ ##
####
```

Solução: uuuuuRldddrUUddlddrUrUdlruLdlUUU



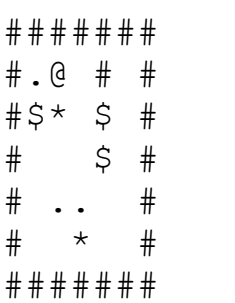
Solução:

uuurRRRRRlululLrrlrddudlllluuRlddrDuluurDlddrUrrrllllluurDldRRRRRurDldRluululLrrddlllluluurDlddddddrUUluudRRRRdrluRdrUlldd lLrruullllddddrUUluRRRRRluullllDldRRRRurDldRluddlUdlllluurrRRuulDullldrdrRurDldR

Formato XSB (Run length encoding):

O formato XSB com RLE (*run-length-encoding*), é mais compacto e eficiente em dispositivos móveis, além de ser menor para enviar por SMS.

Neste formato dígitos mostram quantos elementos do mesmo tipo estão a seguir.
Por exemplo, o padrão de código #### pode ser escrito simplesmente como 4#
Os cenários no formato XSB são fornecidos em uma única linha de texto, sendo a barra vertical "|" usada no lugar dos caracteres de quebra de linha, e os espaços em branco de piso são geralmente codificados usando hífens "-".
Por exemplo, o cenário abaixo:



codificado em XSB com RLE fica: 7#|#.@-#-#|#$*-$-#|#3-$-#|#-..--#|#--*--#|7#

Uma descrição detalhada dos formatos pode ser encontrada em:
http://www.sokoban-online.de/help/sokoban/level-format.html

Durante a leitura do arquivo de cenário, o seu programa deve selecionar o padrão de codificação de acordo com a extensão do arquivo: ".txt" para formato padrão e ".xsb" para codificação XSB com RLE.

O seu programa deve implementar as seguintes funções:

```
void LoadLevel(char map[LIM][LIM], int *m, int *n, char filename[]);
```

Função que lê um cenário, no formato padrão, do arquivo filename na matriz map, e devolve em *m e *n o número de linhas e colunas do mapa, respectivamente.

```
void LoadLevelXSB(char map[LIM][LIM], int *m, int *n, char filename[]);
```

Função que lê um cenário, no formato XSB com RLE, do arquivo filename na matriz map, e devolve em *m e *n o número de linhas e colunas do mapa, respectivamente.

```
void PrintLevel(char map[LIM][LIM], int m, int n);
```

Função que imprime no terminal (saída padrão) o cenário no formato padrão. O cenário é dado pela matriz map com m linhas e n colunas.

```
void GetPlayerPos(char map[LIM][LIM], int m, int n, int *i, int *j);
```

Função que devolve em *i e *j as posições (linha e coluna) do jogador na matriz map.

```
int Move(char *mov, char map[LIM][LIM], int m, int n);
```

Função que realiza o movimento dado por *mov atualizando a matriz de estado do jogo, onde *mov pode assumir um dentre os seguintes valores: u, d, l, r.
O valor de *mov deve ser alterado para letras maiúsculas quando uma caixa é empurrada pelo movimento.
A função deve devolver verdadeiro quando o movimento é válido e falso no caso de movimento impossível (ex: empurrar duas ou mais caixas juntas de uma só vez).

```
void Undo(char mov[MAXMOV], char map[LIM][LIM], int m, int n);
```

Função que desfaz o último movimento presente no histórico de movimentos (string mov), atualizando a matriz de estado do jogo de modo correspondente. A string mov deve ser também atualizada, removendo o seu último caracter.

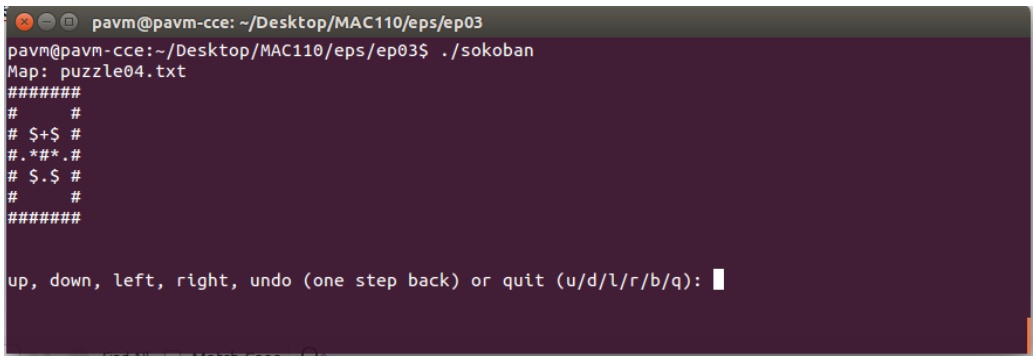
```
int LevelComplete(char map[LIM][LIM], int m, int n);
```

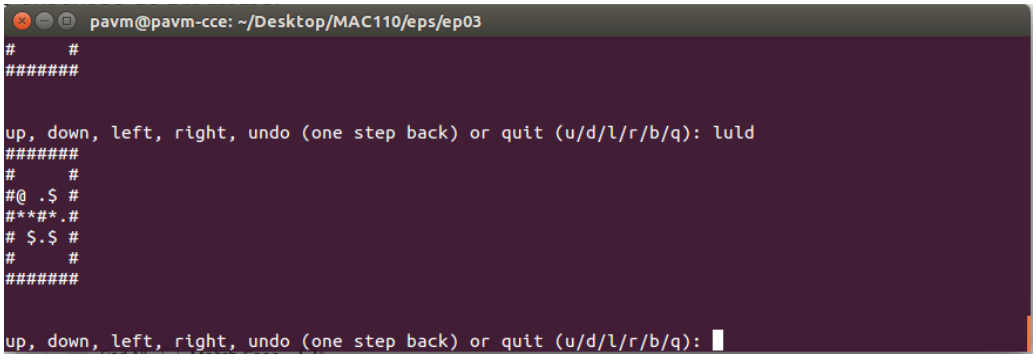
Função que verifica se todas caixas estão colocadas nas posições predeterminadas, indicando o fim do jogo. A função deve devolver verdadeiro (vitória do jogador) ou falso (jogo não encerrado).

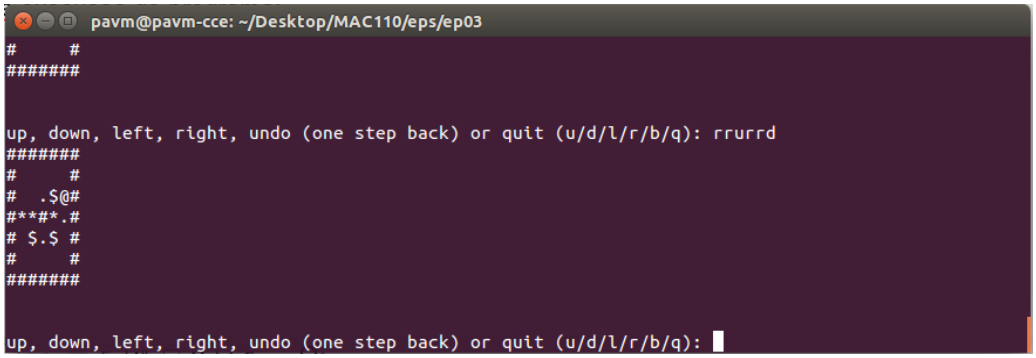
```
int IsValidLevel(char map[LIM][LIM], int m, int n);
```

Função que testa se o cenário é um cenário válido. Todo cenário deve ter um único jogador, quantidade igual de caixas e locais de armazenamento, e pelo menos um lugar de armazenamento. A função deve devolver verdadeiro (cenário válido) ou falso (cenário inválido).
OBS: Por propósitos de simplicidade, a parte de verificar se o cenário está cercado por muros é opcional.

Exemplo de execução do programa:







Após várias iterações: