Universidade Federal do Rio Grande do Norte
Dept. de Informática e Matemática Aplicada — DIMAp

DIM0176 • Programação I

◁ Projeto de Programação Sudoku Interativo ▷

9 de outubro de 2023

O objetivo deste trabalho é implementar uma versão interativa do jogo *puzzle* Sudoku com interface baseada em texto para ser executado em um terminal. Espera-se, com este trabalho, possibilitar a aplicação prática dos conhecimentos adquiridos sobre C++, tais como: uso de *streams*, uso de classes, estruturas de dados, declaração de matriz estática, passagem de matriz por parâmetro, leitura de arquivos de dados, serialização de dados em arquivo, processamento de argumentos de linha de comando, dentre outros.

Além disso, vamos introduzir conceitos relacionados ao desenvolvimento de projetos de programação, como por exemplo: modularização de projeto, arquitetura *game loop* e criação de interface textual.

Por fim, o projeto oferece a oportunidade de se utilizar ferramentas de suporte a programação, como depurador *gdb*, scripts para compilação de projeto com *cmake* ou *Makefile*, bem como a utilização de controle de versionamento de projeto com *git*.

A equipe precisa planejar adequadamente o sistema, para que ele seja eficiente e eficaz. Portanto, é necessário refletir sobre possíveis soluções, para descobrir qual a que resolve o problema da melhor maneira. Além disso, deve existir a preocupação em desenvolver um software de qualidade e robusto.

Sumário

1 Introdução

Um *puzzle* Sudoku consistem de uma matriz 9×9 de células, que podem ou não estar vazias inicialmente. Essa matriz, por sua vez, é organizada em sub-matrizes 3×3 chamadas de **regiões**.

O objetivo do jogador é usar a lógica para preencher as células vazias com dígitos de 1 à 9 de tal maneira que para cada coluna, linha e região **só pode conter um número de cada um dos possíveis dígitos de 1 à 9**. Veja na Figura 1 um exemplo de Sudoku original e sua correspondente solução.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

(a) *Puzzle* Sudoku.

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

(b) Solução *puzzle* Sudoku correspondente.

Figura 1: Exemplo de *puzzle* Sudoku proposto (esquerda) e resolvido (direita). Os números em vermelho foram as dígitos inseridos pelo jogador.
Fonte: https://pt.wikipedia.org/wiki/Sudoku

Normalmente, um bom *puzzle* Sudoku possui apenas uma única solução.

2 O Jogo

Seu objetivo nesse trabalho é o de **projetar** e **implementar** um jogo Sudoku interativo, com interface textual para ser executado em um terminal de comandos.

O seu jogo deve ser capaz de ler uma lista de n *puzzles* a partir de um arquivo de entrada. A partir dessa leitura, o jogo deve oferecer ao usuário um menu principal de opções para

1. escolher qual *puzzle*, dentre os *puzzles* disponíveis, deseja jogar,
2. jogar o puzzle escolhido,
3. ler uma partida em andamento que foi anteriormente salva,
4. pedir ajudar sobre as regras do jogo, ou
5. sair do jogo.

Dentro da escolha **jogar** um *puzzle*, o usuário terá acesso a um submenu de opções para

1. efetuar o posicionamento de um dígito em uma célula vazia,
2. desfazer a jogada anterior,

3. remover um dígito de uma célula que ele preencheu anteriormente,
4. solicitar uma **verificação** no decorrer de uma partida, ou
5. retornar ao menu principal (preservando o conteúdo das células da partida em andamento).

Uma **verificação** é um "recurso" especial que o jogador pode ativar para que o jogo indique quais jogadas realizadas até o momento da solicitação são corretas e quais são incorretas. Esse recurso pode ser usado apenas um número limitado de vezes, que por default são 3 vezes. Obviamente, um bom jogador de Sudoku não precisa utilizar esse recurso para solucionar um *puzzle*.

A partir de um jogo em andamento, deve ser possível salvar o estado atual do jogo em um arquivo, para posterior retomada da partida. Para salvar um jogo em andamento basta retornar ao menu principal. O programa deve perceber que existe um jogo em andamento e incluir mais uma opção no menu principal, que seria a opção de salvar o jogo atual. Ao escolher esta opção, o programa deve solicitar ao usuário que forneça o nome do arquivo para gravação. A extensão do arquivo sugerida é `.sdk`.

O jogo também deve ser capaz de:

1. identificar jogadas inválidas e comunicar isso ao jogador através de um sistemas de mensagens textuais.
2. informar ao usuário uma lista de dígitos que ainda podem ser jogados, ou seja, dígitos que ainda não foram totalmente utilizados no *puzzle*.
3. identificar quando o jogador finalizar uma partida e informar claramente se o jogador resolveu o *puzzle* corretamente ou cometeu algum erro.

3 Interface do Jogo e Gameplay

A interface inicial do jogo deve exibir uma tela de boas vindas, como a da Figura 2.

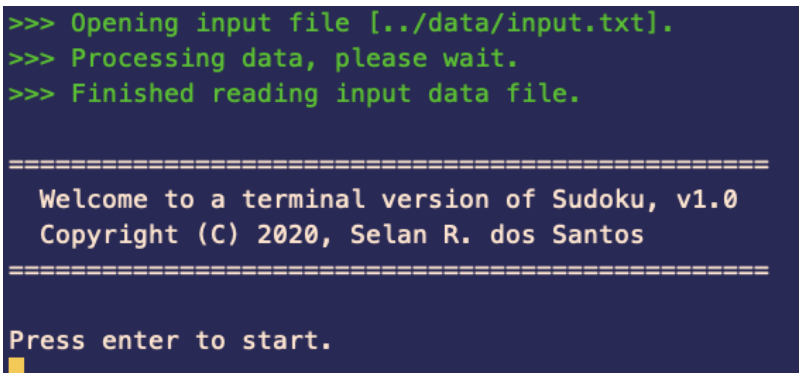


Figura 2: Tela de boas vindas do jogo.

Logo após, deve ser apresentar a tela principal, com exibição do primeiro *puzzle* disponível,

seguido da lista de opções do menu principal, como ilustrado na Figura 3. Note que para cada linha e coluna é exibida uma identificação alfanumérica de coordenadas ao longo da borda superior e borda esquerda do tabuleiro. Essa indicação alfanumérica nas bordas do tabuleiro é essencial para permitir que o jogador identifique de forma única uma célula para realizar uma ação (preenchimento ou remoção) a partir de um par de coordenadas na forma $(linha, coluna)$.

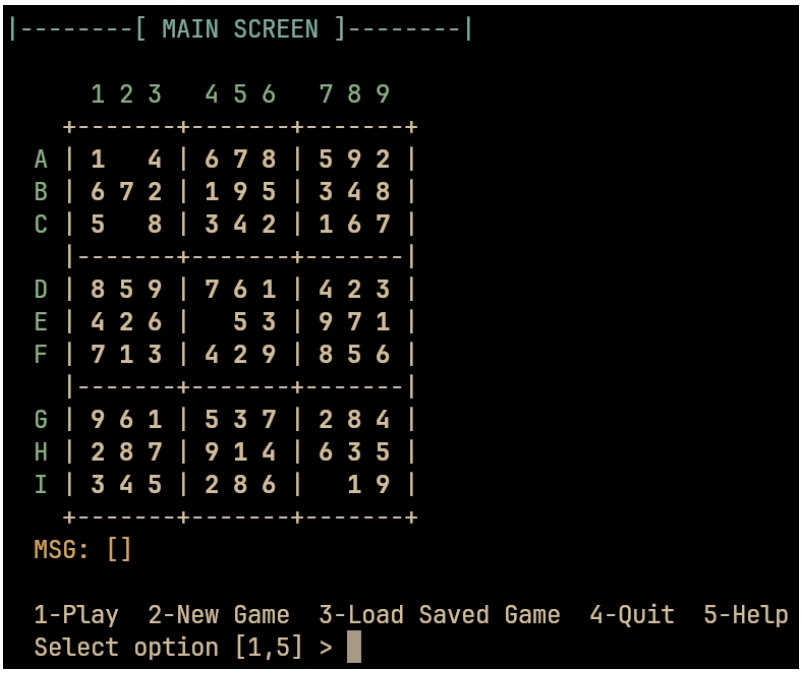


Figura 3: Tela principal do jogo, com apresentação de um *puzzle* a ser resolvido, e o menu de opções logo abaixo do tabuleiro.

A escolha da opção de **Novo Jogo** (*2-New Game*) no menu principal faz com que novos *puzzles* carregados do arquivo de entrada sejam exibidos de forma circular, ou seja, quando se atingir o último *puzzle* deve ser exibido o primeiro *puzzle* da lista novamente.

A escolha da opção de **Carregar Jogo Salvo** (*3-Load Saved Game*) no menu principal permite ao usuário retomar uma partida previamente salva, a partir de um arquivo externo. Se a partida salva for carregada com sucesso, o programa passa imediatamente para o modo de jogo, descrito mais a baixo.

A escolha da opção de **Sair** (*4-Quit*) no menu principal faz com que o jogo seja encerrado. Se por acaso houver alguma partida em andamento no momento que essa opção for escolhida, o jogo deve emitir uma mensagem de alerta e confirmar se o usuário deseja mesmo sair do jogo. Veja uma forma de indicar isso na Figura 4.

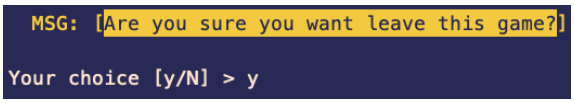


Figura 4: Confirmação de saída do jogo com uma partida em andamento.

A escolha da opção de **Ajuda** (*5-Help*) deve exibir um texto com instruções sobre as regras do jogo, caso o usuário não as conheça, conforme exibido na Figura 5.

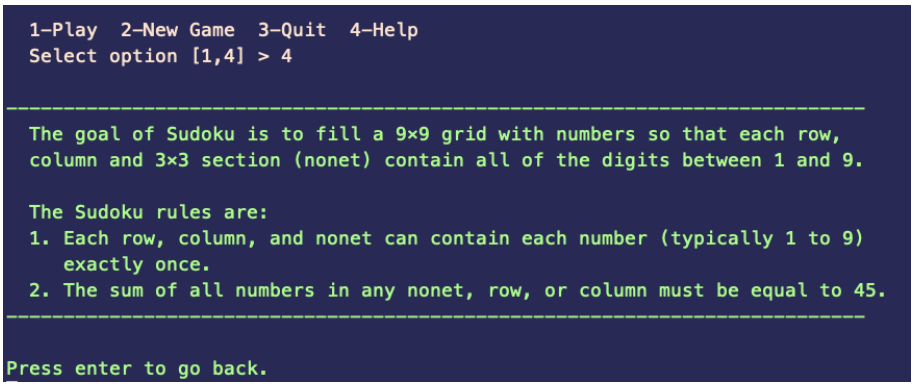


Figura 5: Tela de ajuda geral do jogo.

A escolha da opção de **Jogar** (*1-Play*) ativa a tela de ação, como exibida na Figura 6. As três linhas de informação logo abaixo do tabuleiro correspondem, respectivamente, ao número de verificações restantes que o jogador pode ativar, a lista de dígitos disponíveis a serem utilizados, e a última mensagem emitida pelo jogo direcionada ao jogador.

A lista de dígitos disponíveis é uma mecanismo para auxiliar o jogador à medida que ele progride na partida. Cada vez que ele consegue posicionar (correta ou incorretamente) *nove* dígitos de um tipo, digamos todos os nove dígitos 5, esse dígito desaparece da lista.

Logo abaixo das linhas de informações, vem um conjunto de instruções indicando quais são os **comandos textuais** disponíveis para o jogador indicar seu desejo de **ação** para o jogo. Seu jogo deve suportar as seguintes ações:

- ⋆ Pressionar a tecla <*enter*> sem nada ter sido digitado retorna ao menu principal, preservando o jogo em andamento.
- ⋆ Posicionar um dígito no tabuleiro, com o comando '$p\ linha\ coluna\ digito$' ('p' vem do Inglês *place*).
- ⋆ Remover um dígito inserido pelo jogador anteriormente, com o comando '$r\ linha\ coluna$' ('r' vem do Inglês *remove*).
- ⋆ Ativar uma verificação de erros/acertos, com o comando 'c' (do Inglês *check*).
- ⋆ Desfazer a ação anterior, seja ela um posicionamento de dígito ou remoção de dígito, com o comando 'u' (do Inglês *undo*).

Note que a ativação sucessiva do comando **desfazer**, deve desfazer sequencialmente todos o comandos realizados desde o início da partida. Para tanto, seu jogo deve manter essa lista de <comandos + parâmetros> em alguma estrutura de dados para que seu jogo possa realizar a ação oposta e, dessa forma, 'desfazer' a ação realizada.

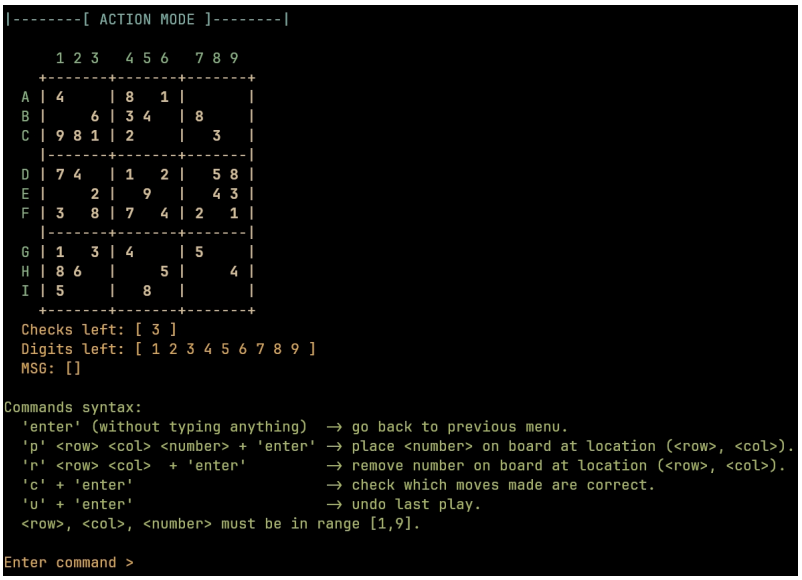


Figura 6: Tela de ação do jogo.

Perceba que as ações são ativadas por comandos digitados, em contraste com escolhas via menu de opções. É exatamente nesse momento que você precisa usar suas habilidades de processamento de cadeias de caracteres (*string*) para identificar o comando digitado pelo jogador. O processamento de um comando textual deve ser robusto o suficiente para **validar** o comando emitido. Isso quer dizer que é necessário identificar comandos inválidos, argumentos inválidos (como por exemplo um dígito fora da faixa permitida ou um caractere inválido informado), falta de argumentos (por exemplo, comando 'p' sem as coordenadas e/ou dígito a ser posicionado), dentre outras possíveis falhas.

Outro aspecto da interface que vale ressaltar aqui é a utilização de cores para auxiliar o jogador a compreender o resultado de suas ações de posicionamento de dígitos no tabuleiro. Existe uma representação por cores para os dígitos, de acordo com a seguinte convenção:

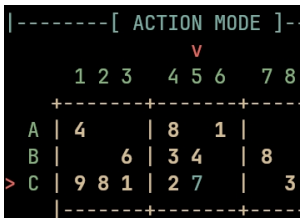
- ⋆ **Dígito em cor branca e negrito**. Dígito fixo, parte do *puzzle* original. Células com esse tipo de dígito não podem ser removidas ou sobrescritas.
- ⋆ **Dígito em cor azul**. No decorrer de uma partida, essa cor representa um dígito em situação válida, podendo ser correto ou não. Ao final da partida, essa cor é utilizada para representar um dígito posicionado corretamente.
- ⋆ **Dígito em cor vermelha**. No decorrer de uma partida, essa cor representa um dígito em situação inválida de acordo com as regras do jogo. Ao final de uma partida, essa

cor é utilizada para representar um dígito posicionado de forma incorreta, com relação à solução do *puzzle*.

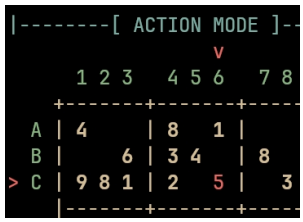
⋆ **Dígito em cor verde**. Indica um dígito válido e correto, aparecendo apenas no modo *verificação* (*check*) do jogo.

Se o jogador solicitar uma ação de posicionamento de dígito em uma célula de maneira a preservar as regras do Sudoku, o dígito deve ser exibido em azul e uma indicação das coordenadas escolhidas deve ser feita no tabuleiro por meio de duas 'setas' (caracteres 'v' para borda superior e '>' para borda esquerda). Confira na Figura 7a o resultado de uma ação de colocação correta de dígito no tabuleiro.

No caso em que o jogador utilize o comando de posicionamento para ocupar uma célula com um dígito que quebra as regras do Sudoku, o jogo deve indicar uma mensagem de erro correspondente, conforme exemplificado na Figura 7b. Contudo, se o jogador desejar prosseguir a partida com o dígito inválido em uma célula, seu jogo deve permitir.



(a) Posicionamento de dígito 7 na célula $(C, 5)$ que preserva a regra do Sudoku.



(b) Posicionamento de dígito 5 na célula $(C, 6)$ que quebra a regra do Sudoku.

Figura 7: Exemplo de possíveis sinalizações da interface, de acordo com o validade da ação de posicionamento de dígito escolhida pelo jogador.

Na utilização de uma verificação, o jogo deve exibir a codificação por cores indicada anteriormente. Veja na Figura 8 um exemplo da utilização do comando de verificação.

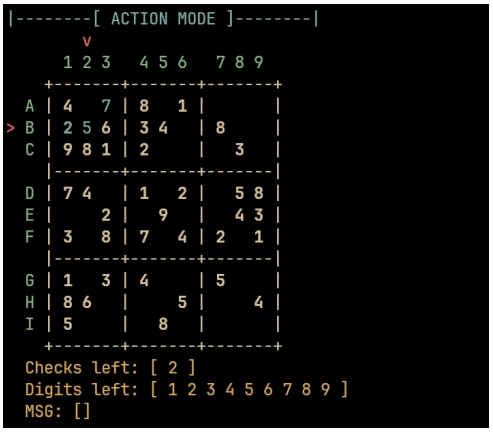
Por fim, quando o jogador finalizar uma partida, seu jogo deve exibir uma mensagem indicando sucesso, ou falha, conforme exemplificado na Figura 9.

4 Entrada de Dados

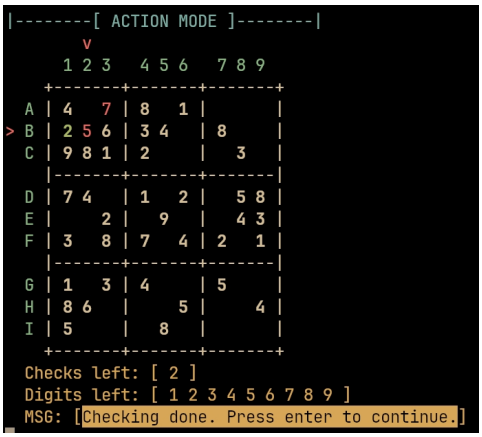
Seu programa deve receber e processar argumentos de linha de comando, de acordo com a seguinte sintaxe de execução:

```
Usage: sudoku [-c <num>] [-h] <input_puzzle_file>
  Game options:
    -c  <num> Number of checks per game. Default = 3.
    -h        Print this help text.
```

Note que de acordo com a sintaxe acima, é **opcional** informar *options* de execução ou um arquivo de entrada contendo Sudoku *puzzles*. As possíveis opções de execução são:



(a) Tabuleiro com algumas jogadas realizadas.



(b) Tabuleiro com indicações de dígitos corretos (em verde) e incorretos (em vermelho).

Figura 8: Exemplo de utilização do comando de verificação (*check*) para identificar jogadas incorretas. Na tela da esquerda temos o tabuleiro original e na tela da direita a exibição da verificação correspondente, através da qual é possível identificar que os dígitos 5 e 7 são incorretos (mas a jogada é legal) e o dígito 2 está correto.

- ⋆ `-h`: solicita ao programa instruções sobre como informar os argumentos de linha de comando, ou seja, exibe a sintaxe de uso apresentada acima.
- ⋆ `-c <number>`: indica a quantidade de verificações que o jogo deve oferecer ao jogador. Note que `<number>` deve ser um inteiro positivo.

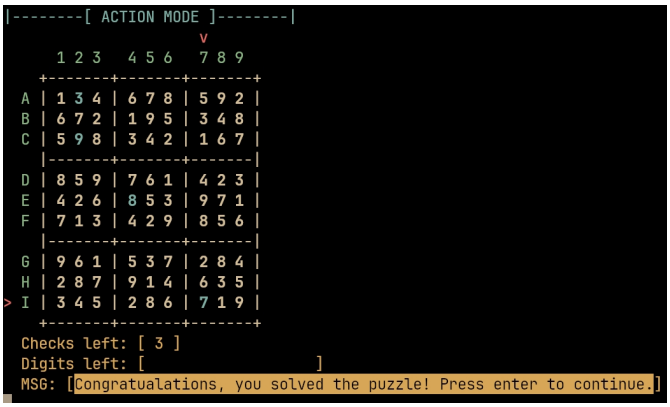
O último argumento da linha de comando, `<input_puzzle_file>`, deve ser um arquivo texto contendo uma sequência de *puzzles* de Sudoku, de acordo com o formato indicado na Figura 10a. Se nenhum arquivo de entrada for informado, o jogo deve procurar ler o arquivo `input.txt` na pasta em que o programa for executado. Se nem o arquivo de entrada indicado pelo usuário nem o arquivo padrão forem encontrados, o jogo deve emitir uma mensagem de erro e finalizar.

O arquivo de entrada pode conter uma série de tabuleiros de no formato da Figura 10a. Note que o *puzzle* está solucionado, porém alguns dígitos são negativos. Os dígitos negativos indicam as células que deverão ser exibidas vazias para o jogador. Veja o resultado correspondente na Figura 10b. Dessa forma em uma única entrada, temos o gabarito completo do *puzzle* e a indicação de quais células deverão ser exibidas em branco.

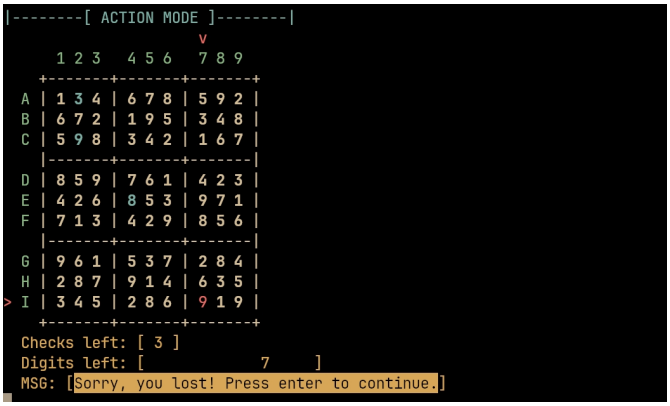
Um arquivo de entrada pode contar n *puzzles*, separados por uma ou mais linhas em branco.

5 Saída de Dados

Em termos de saída de dados, o programa deve ser capaz de armazenar em um arquivo binário externo o estado atual de uma partida em andamento. Isso inclui, o tabuleiro original com



(a) Indicação de partida vitoriosa.

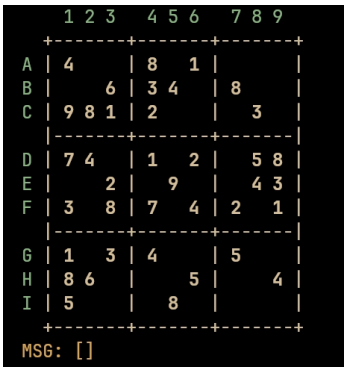


(b) Indicação de derrota na partida.

Figura 9: Exibindo os possíveis resultados de uma partida.

```
 4 -3 -5  8 -7  1 -9 -6 -2
-2 -7  6  3  4 -9  8 -1 -5
 9  8  1  2 -5 -6 -4  3 -7
 7  4 -9  1 -3  2 -6  5  8
-6 -1  2 -5  9 -8 -7  4  3
 3 -5  8  7 -6  4  2 -9  1
 1 -9  3  4 -2 -7  5 -8 -6
 8  6 -7 -9 -1  5 -3 -2  4
 5 -2 -4 -6  8 -3 -1 -7 -9
```

(a) Exemplo de arquivo de entrada (*puzzle*).



(b) Exibição do *puzzle* correspondente.

Figura 10: Exemplo de arquivo de entrada com a descrição do *puzzle* (imagem à esquerda) e a correspondente exibição do *puzzle* no jogo (imagem à direita).

a solução "escondida", da mesma maneira que foi apresentado na Seção 4, os números que foram jogados no tabuleiro, a quantidade de *verificações* que ainda restam e a lista de ações realizadas para permitir ativar o *undo*, se necessário.

A escolha pelo formato binário, ao invés do tradicional ascii (modo texto), é para evitar que o jogador consiga abrir este arquivo em um editor de textos e visualizar a solução do puzzle.

A processo de de salvar e recuperar essas informações na forma de uma sequência bytes enviados para um arquivo binário externo é chamada de **serialização**.

6 Avaliação

Seu programa deve ser escrito em um *bom estilo de programação*. Isto inclui (mas não fica limitado a) o uso de nomes significativos para identificadores e funções, um cabeçalho de comentário no início de cada arquivo, cabeçalho no formato Doxygen para cada função/método criado, uso apropriado de linhas em branco e indentação para ajudar na visualização do código, comentários significativos nas linhas de código, etc.

O programa completo deverá ser entregue sem erros de compilação, testado e totalmente documentado. O projeto é composto de 115 pontos e será avaliado sob os seguintes critérios:-

- ⋆ Trata corretamente os argumentos de linha de comando (5 pts);
- ⋆ Lê e armazena os vários *puzzles* contidos em um arquivo de entrada, no formato indicado na Seção 4 (5 pts);
- ⋆ Exibe corretamente a tela principal com o menu de ações (5 pts);
- ⋆ Permite a escolha dos vários *puzzles* lidos do arquivo (5 pts);
- ⋆ Executa corretamente a ação de posicionamento de dígitos no tabuleiro (comando posicionar) com a identificação jogadas inválidas (11 pts);
- ⋆ Executa corretamente a remoção de dígitos posicionados pelo jogador (comando remover) (10 pts);
- ⋆ Executa corretamente o comando de desfazer ação (12 pts);
- ⋆ Executa corretamente o comando de verificação do tabuleiro (12 pts);
- ⋆ Salva corretamente, em arquivo externo, uma partida em andamento (7 pts);
- ⋆ Carrega corretamente, de um arquivo externo, uma partida em andamento previamente salva e consegue retormar a partida (8 pts);
- ⋆ Exibe as regras do jogo quando solicitado (5 pts);
- ⋆ Indica corretamente se uma partida finalizada foi vencida ou perdida (5 pts);
- ⋆ Apresenta uma interface com usuário igual ou similar a que foi especificada nesse documento (5 pts);
- ⋆ Apresenta uma validação robusta dos comandos textuais de ação fornecidos pelo usuário (5 pts);
- ⋆ Exibe corretamente a lista de dígitos disponíveis, conforme o progresso do jogo (5 pts);
- ⋆ Apresenta as indicações com ‘setas‘ da coordenada da célula que foi alvo da última ação (5 pts);
- ⋆ Apresenta um fluxo correto entre telas, com opção de retomar uma partida suspensa e solicitação de confirmação de finalização de partida, caso exista uma partida em anda-

mento (5 pts);

A pontuação acima não é definitiva e imutável. Ela serve apenas como um guia de como o trabalho será avaliado em linhas gerais. É possível a realização de ajustes nas pontuações indicadas visando adequar a pontuação ao nível de dificuldade dos itens solicitados.

Os itens abaixo correspondem à descontos, ou seja, pontos que podem ser retirados da pontuação total obtida com os itens anteriores:-

- Presença de erros de compilação e/ou execução (até -20%)
- Falta de documentação do programa com Doxygen (até -10%)
- Vazamento de memória (até -10%)
- Falta de um arquivo texto `README` contendo, entre outras coisas, identificação da dupla de desenvolvedores; instruções de como compilar e executar o programa; lista dos erros que o programa trata; e limitações e/ou problemas que o programa possui/apresenta, se for o caso (até -20%).

Boas práticas de programação

Recomenda-se fortemente o uso das seguintes ferramentas:-

- ⋆ Doxygen: para a documentação de código e das classes;
- ⋆ Git: para o controle de versões e desenvolvimento colaborativo;
- ⋆ Valgrind: para verificação de vazamento de memória;
- ⋆ gdb: para depuração do código; e
- ⋆ CMake/Makefile: para gerenciar o processo de compilação do projeto.

Procure organizar seu código em várias pastas, conforme vários exemplos apresentados em sala de aula, com pastas como `src` (arquivos `.cpp`), `include` (arquivos `.h`), `bin` (arquivos `.o` e executável) e `data` (arquivos de entrada e saída de dados).

7 Autoria e Política de Colaboração

O trabalho pode ser realizado **individualmente** ou em **dupla**, sendo que no último caso é importante, dentro do possível, dividir as tarefas igualmente entre os componentes. A divisão de tarefas deve ficar evidente através do histórico de *commit* do git.

Após a entrega, qualquer equipe pode ser convocada para uma entrevista. O objetivo da entrevista é duplo: confirmar a autoria do trabalho e determinar a contribuição real de cada componente em relação ao trabalho. Durante a entrevista os membros da equipe devem ser capazes de explicar, com desenvoltura, qualquer trecho do trabalho, mesmo que o código tenha sido desenvolvido pelo outro membro da equipe. Portanto, é possível que, após a entrevista, ocorra redução da nota geral do trabalho ou ajustes nas notas individuais, de maneira a refletir a verdadeira contribuição de cada membro, conforme determinado na entrevista.

O trabalho em cooperação entre alunos da turma é estimulado. É aceitável a discussão de ideias e estratégias. Note, contudo, que esta interação **não** deve ser entendida como permissão para utilização de código ou parte de código de outras equipes, o que pode caracterizar a situação de plágio. Em resumo, tenha o cuidado de escrever seus próprios programas.

Trabalhos plagiados receberão nota **zero** automaticamente, independente de quem seja o verdadeiro autor dos trabalhos infratores. Fazer uso de qualquer assistência sem reconhecer os créditos apropriados é considerado **plagiarismo**. Quando submeter seu trabalho, forneça a citação e reconhecimentos necessários. Isso pode ser feito pontualmente nos comentários no início do código, ou, de maneira mais abrangente, no arquivo texto `README`. Além disso, no caso de receber assistência, certifique-se de que ela lhe é dada de maneira genérica, ou seja, de forma que não envolva alguém tendo que escrever código por você.

8 Entrega

Você pode enviar seu trabalho de duas formas possíveis: via GitHub Classroom (GHC), ou via tarefa de submissão SIGAA. Caso você decida enviar seu trabalho via GHC você **deve** enviar também um arquivo texto via tarefa de submissão do Sigaa com o link do github para seu repositório. Caso opte por enviar seu trabalho apenas via SIGAA, envie um arquivo zip contendo todo o código necessário para compilar e executar o projeto.

Em qualquer uma dessas duas maneiras, lembre-se de remover todos os arquivos executáveis (ou seja, a pasta `build`) do seu projeto antes de entregar o seu trabalho.

◄ FIM ►