UFMT – Universidade Federal do Estado de Mato Grosso
IC – Instituto de Computação

Relatório

Projeto Prático

Disciplina – Inteligência Artificial
Professora – Vanessa de Oliveira Campos
Alunos: Bruna Keiko Hatakeyama, Leony Tamio Hatakeyama e Lucimarck Junior da Silva Dias

1. Introdução

A Inteligência Artificial (Artificial Intelligence - AI) figura com um avanço tecnológico que permite aos sistemas simularem uma inteligência similar à humana, mais especificamente a capacidade de aprender, perceber e decidir quais caminhos seguir, de forma racional, diante de determinadas situações, indo além da programação com ordens programadas para tomar decisões.

2. Justificativa

Aplicação dos conhecimentos adquiridos na disciplina de Inteligência Artificial, visando a resolução de um problema específico utilizando técnicas de buscas, funções heurísticas, lógica, árvores de decisão, algoritmos, outros.

3. Apresentação do problema

O Projeto consiste em implementar um agente capaz de resolver autonomamente um desafio de Sudoku. O jogo tradicional começa com um tabuleiro de dimensões 9x9 parcialmente preenchido.

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

A cada passo do jogo, pode-se preencher as posições vazias com um número entre 1 e 9. A solução para o jogo é um tabuleiro completamente preenchido no qual cada número de 1 a 9 apareça exatamente uma vez em cada linha, coluna e bloco.

3.1 O sistema deve:

- ✓ Ler um tabuleiro inicial do jogo sudoku de um arquivo *.txt (estado inicial). O arquivo deve conter 81 dígitos, cada um correspondendo ao valor de uma das posições do tabuleiro;
- ✓ Resolver o problema através de algum método de IA;
- ✓ Exibir a solução encontrada. (Preferencialmente, destacar os valores definidos no estado inicial e valores encontrados pelo sistema).

3.2 O arquivo de estado inicial:

- ✓ O estado inicial do tabuleiro do jogo sudoku deve ser lido de um arquivo *.txt;
- ✓ Como o tabuleiro contém 81 posições, o arquivo deve conter 81 dígitos, cada um correspondendo ao valor de uma das posições do tabuleiro;
- ✓ Os valores “0” do arquivo, correspondem às posições vazias no tabuleiro inicial e, portanto, correspondem às posições cujos valores devem ser encontrados pelo agente;
- ✓ Os demais valores, entre 1 e 9, serão os valores predefinidos no jogo;
- ✓ Os 9 primeiros dígitos do arquivo correspondem as 9 posições da primeira linha do tabuleiro;
- ✓ Os 9 dígitos seguintes, do décimo a décimo oitavo dígito, correspondem as 9 posições da segunda linha do tabuleiro. E assim, por diante.

Exemplo: Considere um arquivo contendo o seguinte:

900000001051200030000980000680740000730000908010058670008100000002007090190004060

O estado inicial do Jogo Sudoku correspondente a esse arquivo seria →

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

3.2 Informações adicionais:

- ✓ O programa pode ser implementado em qualquer linguagem.
- ✓ O programa deverá ser executado em Windows (se bibliotecas auxiliares forem usadas, todos os arquivos necessários deverão ser incluídos no projeto para que ele possa ser executado).

4. Método

Com base no problema apresentado e no que foi exposto em aula, optamos pelo método de **busca por profundidade** para o desenvolvimento da aplicação pois, segundo o conceito de **backtracking,** há possibilidade de possíveis soluções serem removidas antes mesmo de serem avaliadas, possibilitando desta forma buscas alternativas.

A busca de força bruta implementada iria avaliar três condições ao gerar uma coleção de valores para hipotética solução: os valores não contidos na linha, coluna ou área corresponde do valor avaliado.

Com os valores encontrados, um nodo é criado a partir do último (busca de profundidade) em busca de novas soluções. Se porventura alguma solução retornar valores hipotéticos nulos, inicia o gatilho de backtracking, retornando para o nodo resultante do conflito, e uma nova avaliação é feita utilizando o valor anterior ao resultante do conflito.

O processo termina quando todos os campos do Sudoku são resolvidos.

5. Aplicação

Para o desenvolvimento do software, foi utilizado a linguagem C# juntamente com o framework ASP.NET Core utilizando Blazor como modelo de projeto. O código fonte foi dividido em três projetos:

Sudoku.Web.Common – responsável pela implementação da lógica bem como para ser utilizado como biblioteca aos modelos de hospedagem, contendo classes em C# e componentes Blazor escrito na linguagem Razor;

Sudoku.Web.Server – responsável por toda a arquitetura responsável para a geração do executável como um servidor ASP.NET Core visando permitir realizar a execução de um aplicativo Web utilizando o projeto de referência como container do seu conteúdo;

Sudoku.Web.Assembly – responsável por permitir a geração de aplicações Web estáticas escritas em C# por meio da tecnologia Web Assembly. Para utilizar a aplicação, há duas opções:

- Executável – O arquivo Sudoku.Web.Server dentro da pasta _exe inicializa um servidor Web nos endereços http://localhost:80 e https://localhost:443 podendo ser acessado de qualquer navegador da preferência do usuário.
- Endereço Web – Por meio do projeto Sudoku.Web.Assembly, uma página no Github Pages foi provisionada permitindo interagir com a aplicação pela internet.

6. Resultados

Durante a execução da aplicação, notamos uma sequência de ações que, ao final, refletia muito bem o conceito da **busca em profundidade**. A cada iteração, novos valores hipotéticos eram criados, avaliados e permitia a criação de novos nodos.

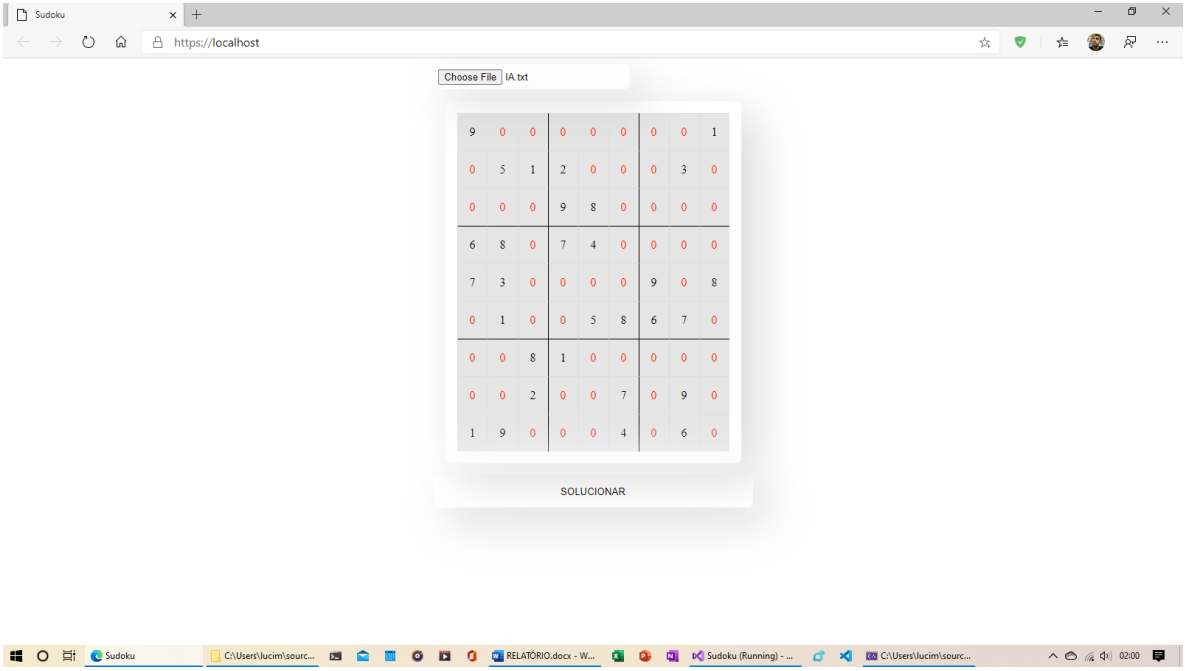

Fig.1 – Estado inicial do tabuleiro após ler um arquivo .txt contendo 81 dígitos

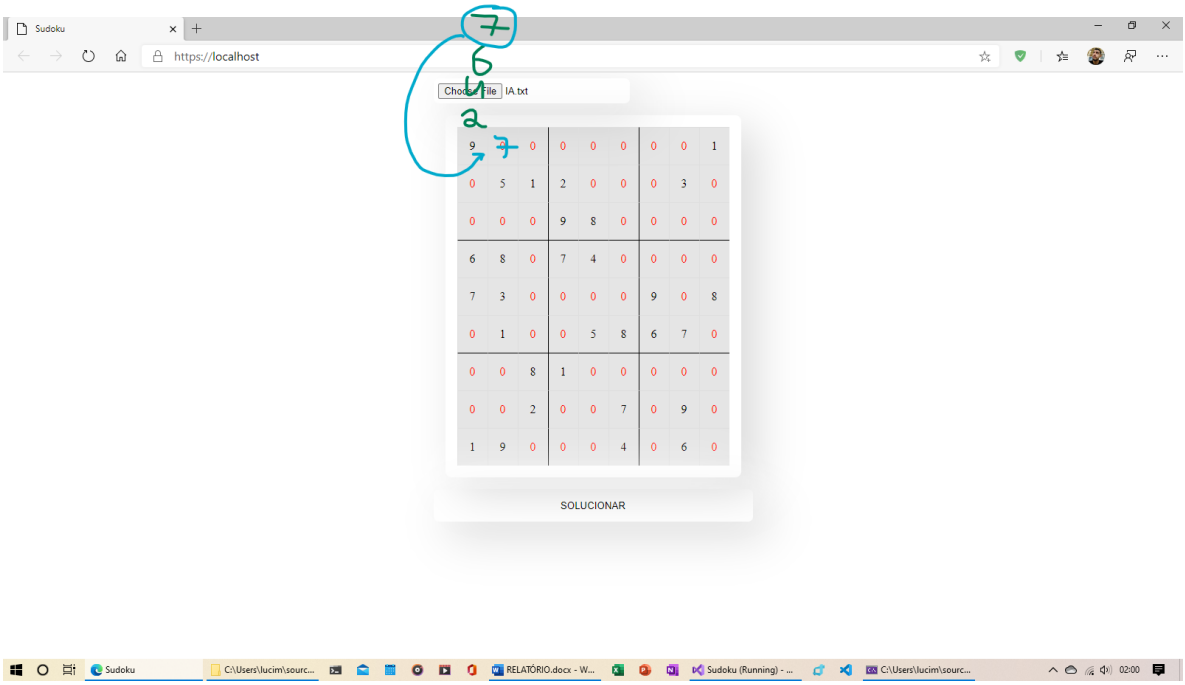

Fig.2 – Valores encontrados durante a depuração da aplicação na primeira iteração.

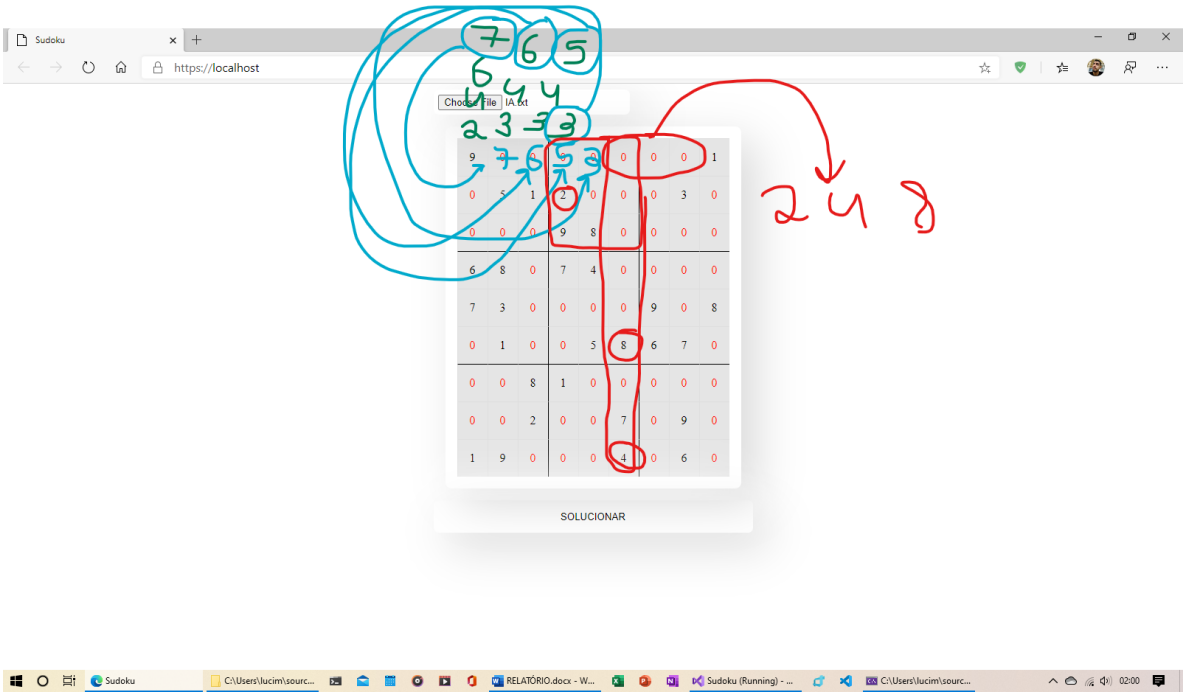

Fig.3 – Valores encontrados depois de uma sequência de iterações durante a depuração do programa mostrando um exemplo de gatilho de backtracking.

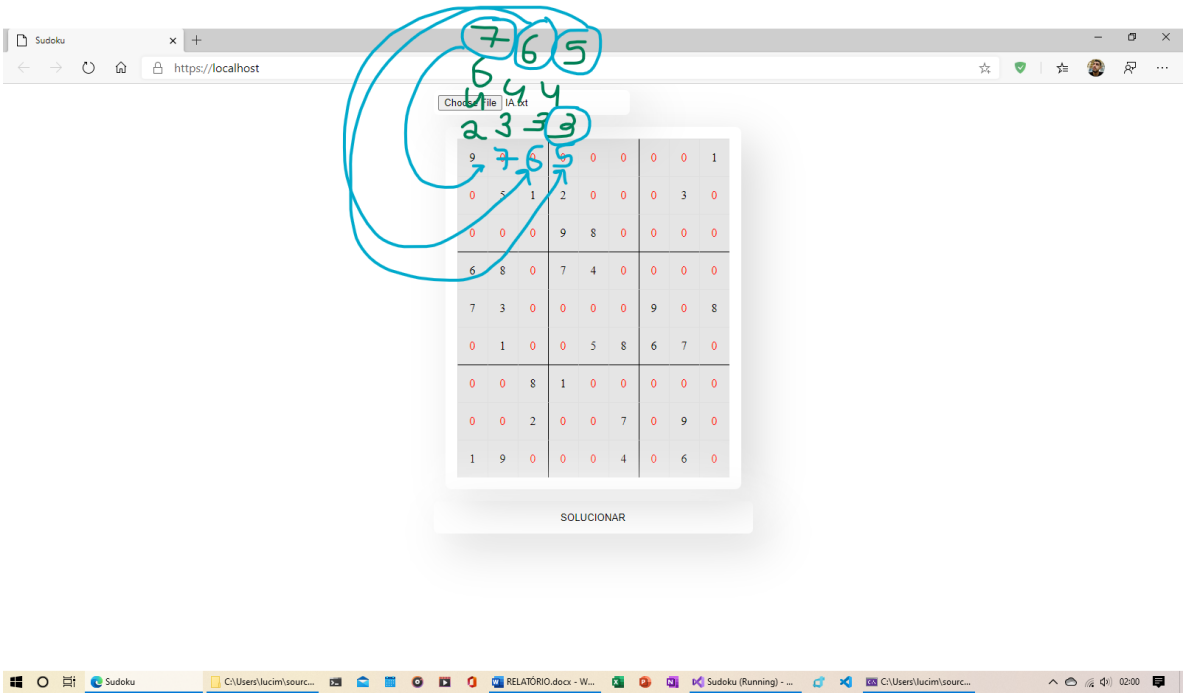

Fig.4 – Fluxo da busca de profundidade realizando um backtracking até o nodo resultante do conflito (nodo terminal).

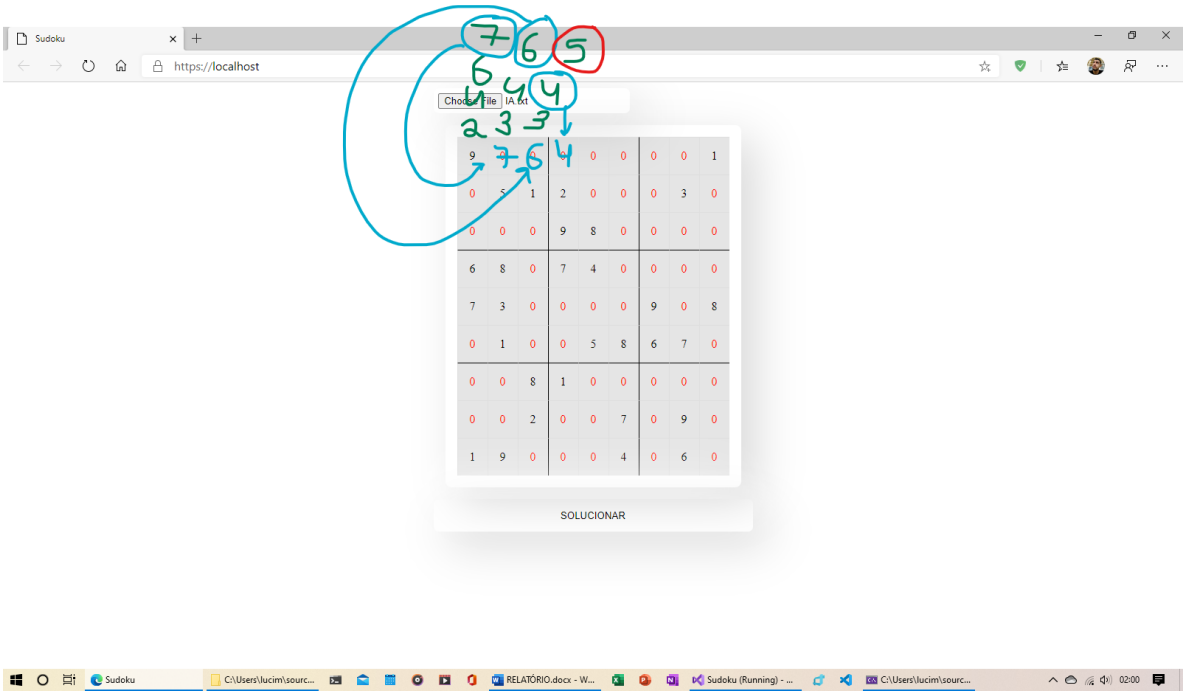

Fig.5 – Destacando que a solução avaliada não foi aprovada logo uma solução alternativa que, no caso, o valor anterior à solução previamente avaliada.

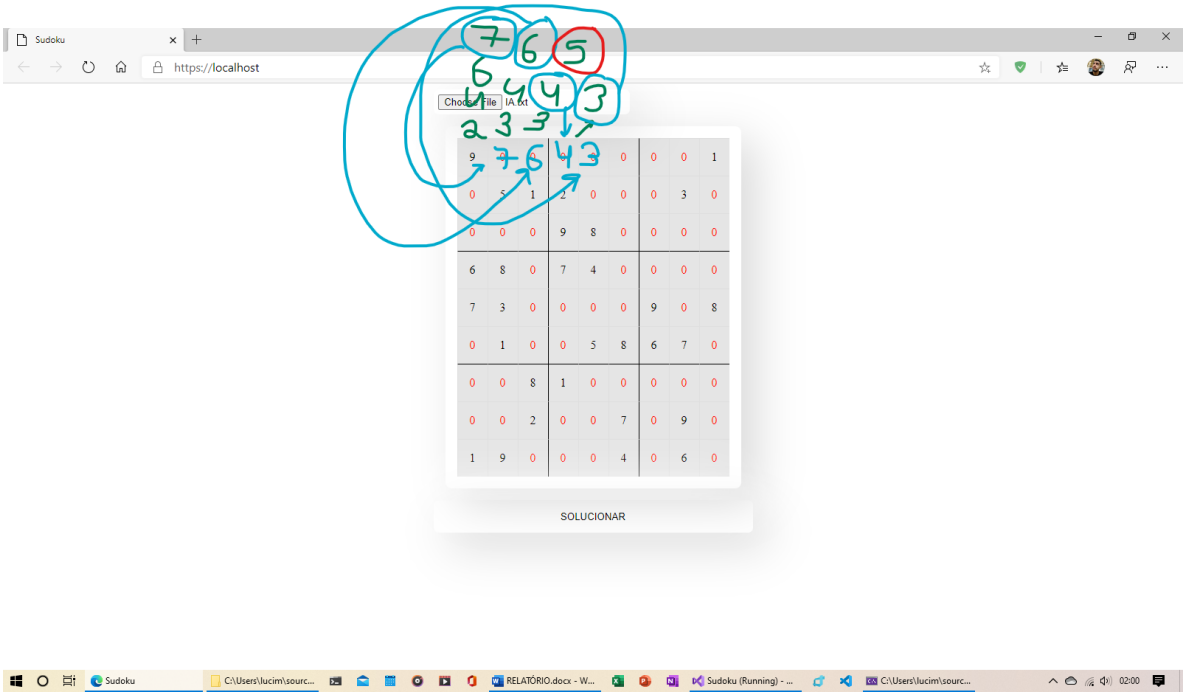

Fig.6 – Continuação do fluxo da aplicação mostrando os valores encontrados e utilizados.

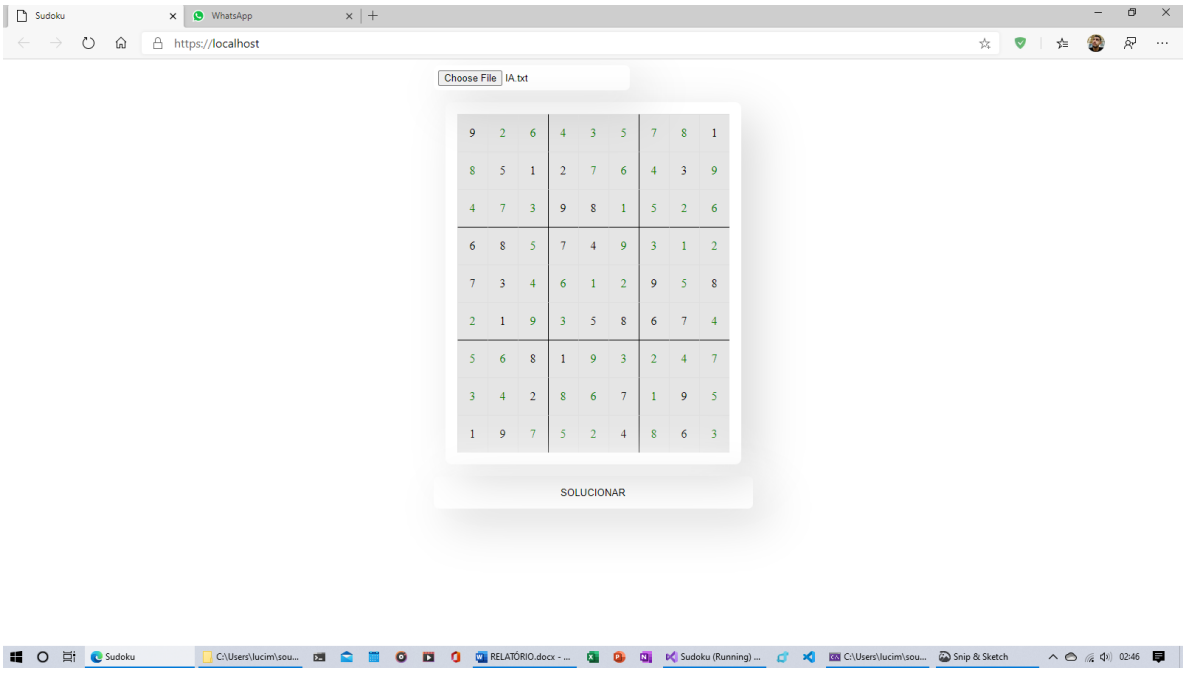

Fig.7 – Aplicação no estado final contendo o tabuleiro Sudoku complemente solucionado.