Relatório Técnico do Software: 8-Puzzle Solver com Lógica SAT

1. Resumo

Este documento fala sobre a arquitetura e o funcionamento do software **8-Puzzle Solver (SAT)**, uma aplicação de desktop desenvolvida em Python. O objetivo do software é resolver o clássico quebra-cabeça de 8 peças, demonstrando a aplicação de um **Resolvedor de Satisfatibilidade Booleana (SAT Solver)** para um problema de planejamento e busca.

O sistema possui uma interface gráfica intuitiva construída com a biblioteca Tkinter, que permite ao usuário embaralhar um quebra-cabeça visual e montar uma resolução para ele. A lógica central, sai da interface, modela o problema em uma fórmula booleana complexa e utiliza a biblioteca PySAT para encontrar um modelo satisfatível, que é então traduzido de volta em uma sequência de movimentos para resolver o quebra-cabeça.

2. Arquitetura do Software

O projeto foi estruturado seguindo as boas práticas de engenharia de software, com uma clara separação de responsabilidades que se assemelha ao padrão **Model-View-Controller (MVC)**.

Model (solver_sat/core.py): É o cérebro da aplicação. Contém toda a lógica para traduzir o problema do 8-Puzzle em uma fórmula SAT. É completamente independente da interface gráfica e pode ser reutilizado em outros contextos.

View/Controller (gui/): É a camada de apresentação e interação.

- app.py é responsável por criar a janela, os botões e a grade visual do quebra-cabeça. Ele captura as ações do usuário (cliques), chama o Model para processamento e atualiza a interface com os resultados (animação da solução).
- image_handler.py atua como um serviço para a View, lidando com a tarefa específica de carregar, fatiar e numerar as peças da imagem.

Ponto de Entrada (main.py): Sua única responsabilidade é instanciar e iniciar a aplicação gráfica, garantindo que o programa seja executado a partir do contexto correto.

3. Componentes Principais e Funcionamento

3.1. Interface Gráfica (gui/)

app.py (Classe EightPuzzleSAT_GUI):

Inicialização: Cria a janela principal, carrega as peças da imagem através do image_handler, e inicializa os widgets (botões, labels de status).

Gerenciamento de Estado Visual: Mantém uma representação do tabuleiro (self.current_state_2d) e o atualiza na tela através do método update_grid.

Interação do Usuário:

O botão **"Embaralhar"** chama a função gerar_tabuleiro_inicial do solver_sat.core para obter um novo quebra-cabeça e o exibe na tela.

O botão **"Resolver (SAT)"** inicia o processo de resolução. Para evitar que a interface congele durante o processamento (que pode ser demorado), ele inicia a lógica do solver em uma **threading.Thread** separada.

Animação: Quando a thread do solver termina e retorna uma solução, o método _animate_solution é chamado. Ele executa a sequência de movimentos passo a passo, usando o método self.after() do Tkinter para criar um atraso entre cada movimento, resultando em uma animação suave.

image_handler.py (Função carregar_e_fatiar_imagem):

Recebe o caminho de uma imagem e o tamanho da grade.

Usa a biblioteca Pillow para abrir e redimensionar a imagem.

Divide a imagem em uma grade 3x3.

Para cada uma das 9 peças, utiliza ImageDraw para desenhar seu número correspondente (0 a 8) no centro da peça, com uma sombra para garantir a legibilidade.

Converte cada peça modificada para um objeto ImageTk.PhotoImage, que pode ser exibido em um Label do Tkinter.

3.2. Lógica do Solver SAT (solver_sat/core.py)

Ele transforma o problema de encontrar uma sequência de movimentos em "É possível que esta fórmula booleana seja verdadeira?".

Passo 1: Mapeamento de Variáveis (criar_mapeamento)
O primeiro passo é definir um dicionário que mapeia cada possível evento do jogo a uma variável booleana única (representada por um número inteiro). Existem dois tipos de variáveis:

Variáveis de Posição: tPlcv - É verdadeiro se, no tempo t, a peça v está na linha l e coluna c.

Variáveis de Ação: tAm - É verdadeiro se, no tempo t, a ação m (Cima, Baixo, Esquerda, Direita) é executada.

Passo 2: Geração de Cláusulas (As Regras do Jogo)
O estado do jogo e suas transições são definidos através de um conjunto de regras lógicas, traduzidas para a Forma Normal Conjuntiva (FNC), que é o formato que o solver SAT entende.

min_um e max_um: Garantem que, em qualquer tempo, **exatamente uma** peça ocupe cada uma das 9 casas do tabuleiro.

min_uma_acao e max_uma_acao: Garantem que, em cada passo de tempo, **exatamente uma** ação (movimento) seja executada.

regra_precondicao: Define as condições necessárias para uma ação ser válida. Exemplo: "Se a ação é 'Mover para Cima', então a peça vazia não pode estar na primeira linha". (¬Ação ∨ ¬CondiçãoInválida).

regra_transicao: Descreve os efeitos de uma ação. Exemplo: "Se no tempo t a peça 0 está em (l,c), a peça v está em (l-1,c) E a ação 'Mover para Cima' é executada, ENTÃO no tempo t+1 a peça 0 estará em (l-1,c) E a peça v estará em (l,c)".

regras_inercia (Axioma de Frame): Esta é uma regra crucial. Ela define o que **não muda** quando uma ação é executada. Exemplo: "Se a ação 'Mover para Cima' é executada, todas as peças que não estão nas duas células afetadas pela troca devem permanecer em suas posições originais no próximo passo de tempo". Isso evita que o solver encontre soluções "mágicas" onde peças não relacionadas se movem.

Passo 3: Resolução (resolver_com_sat)

Processo Iterativo: A função não tenta resolver o problema de uma só vez. Ela opera em um loop, testando se uma solução existe em 1 passo, depois em 2, depois em 3, e assim por diante, até o max_passos_limite. Isso garante que a primeira solução encontrada seja a **mais curta possível**.

Construção da Fórmula: Em cada iteração do loop (para um número n de passos), a função:

Define o **estado inicial** como um conjunto de cláusulas unitárias (fatos). Ex: [<var_para_1P11v>] é verdadeiro.

Define o **estado final** desejado ([[0,1,2]...]) no passo n também como um conjunto de cláusulas unitárias.

Combina todas as regras (posição, ação, transição, inércia, etc.) com os estados inicial e final para formar uma única e gigantesca fórmula.

Invocação do Solver: A fórmula completa é passada para o Glucose4 (um solver SAT da biblioteca PySAT).

Decodificação do Modelo: Se solver.solve() retorna True, significa que existe uma atribuição de Verdadeiro/Falso para todas as variáveis que satisfaz todas as regras. solver.get_model() retorna essa atribuição. O código então percorre o modelo e filtra apenas as variáveis de Ação que são verdadeiras (tAm), montando um dicionário que mapeia cada passo de tempo à sua respectiva ação ({1: 'C', 2: 'D', ...}).

Retorno: A função retorna o dicionário de movimentos e o número de passos da solução. Se o loop terminar sem solução, retorna None.

4. Conclusão

O software **8-Puzzle Solver (SAT)** é uma demonstração correta da aplicação de técnicas de lógica computacional a um problema clássico de inteligência artificial. A sua arquitetura modular separa de forma eficaz a lógica de modelagem SAT da interface do usuário, resultando em um código limpo, manutenível e reutilizável. A utilização de um solver SAT, embora exaustivo(computacionalmente intensivo), mostra uma abordagem declarativa para resolver problemas de planejamento, onde o desenvolvedor se concentra em "o que" são as regras do sistema, em vez de "como" encontrar a solução passo a passo.