Laporan Artificial Intelligence

SUDOKU MENGGUNAKAN ALGORITMA BACKTRACKING DAN BACKTRACKING+MRV

disusun untuk memenuhi tugas

mata kuliah Artificial Intelligence

oleh :

Anis Raysa	250820701100003
Zahra Zafira	2208107010040
Cut Sula Fhatia Rahma	2208107010048

PROGRAM STUDI MAGISTER KECERDASAN BUATAN

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS SYIAH KUALA

DARUSSALAM, BANDA ACEH

2025

BAB I
PENDAHULUAN

1.1 Latar Belakang

Permainan Sudoku merupakan salah satu teka-teki logika berbasis angka yang populer di seluruh dunia. Sudoku memiliki aturan sederhana, namun kompleksitas penyelesaiannya membuat permainan ini menjadi topik menarik dalam kajian kecerdasan buatan (Artificial Intelligence/AI) dan ilmu komputer. Setiap teka-teki Sudoku berbentuk matriks berukuran 9×9, yang dibagi ke dalam subgrid 3×3, dengan aturan bahwa setiap baris, kolom, dan subgrid harus diisi angka 1 sampai 9 tanpa pengulangan (Russell & Norvig, 2021).

Secara komputasional, penyelesaian Sudoku dikategorikan sebagai permasalahan Constraint Satisfaction Problem (CSP), yang umumnya diselesaikan dengan teknik pencarian (search algorithms) seperti backtracking, forward checking, maupun heuristik seperti Minimum Remaining Value (MRV) untuk mempercepat proses pencarian solusi (Simonis, 2005).

Dalam beberapa penelitian, algoritma backtracking terbukti efektif dalam menemukan solusi Sudoku karena sifatnya yang sistematis, meskipun memiliki kelemahan pada efisiensi ketika ruang pencarian sangat besar (Stojanovic & Milinkovic, 2018). Untuk mengatasi hal tersebut, pendekatan heuristik seperti MRV digunakan agar proses pencarian solusi lebih cepat dengan cara memilih variabel yang paling ketat batasannya terlebih dahulu (Kumar et al., 2020).

Berdasarkan hal tersebut, penelitian dan implementasi ini bertujuan untuk mengembangkan aplikasi Sudoku Solver berbasis Python yang mampu menyelesaikan teka-teki Sudoku menggunakan kombinasi algoritma backtracking dan MRV. Aplikasi ini juga dilengkapi antarmuka interaktif berbasis pygame untuk memvisualisasikan proses pencarian solusi, sehingga dapat memberikan gambaran lebih jelas tentang bagaimana algoritma bekerja.

1.2 Rumusan Masalah

1. Bagaimana mengimplementasikan algoritma backtracking dan backtracking dengan heuristik MRV untuk menyelesaikan teka-teki Sudoku secara otomatis?
2. Bagaimana perbandingan kinerja kedua algoritma tersebut berdasarkan waktu eksekusi?

3. Apakah tingkat kesulitan puzzle Sudoku berpengaruh terhadap efisiensi algoritma backtracking dan backtracking dengan MRV?

1.3 Tujuan Penelitian

1. Mengimplementasikan algoritma backtracking dan backtracking dengan heuristik Minimum Remaining Value (MRV) untuk menyelesaikan teka-teki Sudoku secara otomatis.
2. Membandingkan kinerja algoritma backtracking dan backtracking dengan heuristik MRV berdasarkan waktu eksekusi yang dibutuhkan.
3. Menganalisis pengaruh tingkat kesulitan puzzle Sudoku terhadap efisiensi algoritma backtracking dan backtracking dengan heuristik MRV.

BAB II
KAJIAN PUSTAKA

2.1 Permainan Sudoku

Sudoku merupakan teka-teki logika berbasis angka yang populer secara global. Sudoku berbentuk matriks 9×9 yang dibagi menjadi 9 subgrid berukuran 3×3, di mana setiap baris, kolom, dan subgrid harus berisi angka 1 sampai 9 tanpa pengulangan (Russell & Norvig, 2021). Secara komputasional, Sudoku dikategorikan sebagai Constraint Satisfaction Problem (CSP), karena penyelesaiannya melibatkan pencarian nilai variabel yang memenuhi sekumpulan batasan (Simonis, 2005).

2.2 Sudoku sebagai Permasalahan CSP (Constraint Satisfaction Problem)

Sudoku sering dipandang sebagai sebuah Constraint Satisfaction Problem (CSP), di mana setiap kotak kosong dianggap sebagai variabel, dan domain variabel tersebut adalah angka 1 sampai 9. Aturan bahwa tiap baris, kolom, dan sub-kotak 9×9 tidak boleh ada angka yang sama dijadikan sebagai constraint atau kendala. Dengan model ini, teknik-teknik dari bidang kecerdasan buatan (AI) atau teori CSP dapat diterapkan untuk mempercepat penyelesaian. Pendekatan CSP ini memungkinkan penggunaan berbagai heuristik dan propagasi constraint (misalnya arc-consistency, forward checking) untuk mengeliminasi nilai-nilai yang tak mungkin sejak dini, sehingga mengurangi ruang pencarian.

2.3 Algoritma Backtracking dalam Menyelesaikan Sudoku

Backtracking adalah metode klasik dan intuitif dalam menyelesaikan Sudoku : program akan mengisi kotak kosong dengan salah satu angka dari 1 sampai 9, memeriksa apakah pengisian itu tidak melanggar aturan baris, kolom, sub-kotak, lalu melanjutkan ke kotak berikutnya secara rekursif. Jika pada suatu titik tidak ada angka yang valid, maka mundur (backtrack) ke langkah sebelumnya dan coba angka lain. Metode ini dijamin menemukan solusi jika ada, tetapi bisa sangat lambat jika ruang pencarian besar.

Dalam banyak kasus, backtracking murni (tanpa heuristik) akan mengeksplorasi banyak cabang yang tidak perlu, terutama dalam puzzle yang sulit atau yang memiliki banyak kotak kosong. Beberapa penelitian mencoba meningkatkan backtracking dengan teknik tambahan.

Misalnya, pada makalah “A pencil-and-paper algorithm for solving Sudoku puzzles”, peneliti menambahkan teknik naked pairs (dua kotak yang memiliki kandidat sama) untuk mengeliminasi kandidat di kotak lain dalam baris/kolom/sub-kotak sehingga mengurangi langkah backtracking.

2.4 Heuristik MRV (Minimum Remaining Value) dan Teknik Tambahan

Heuristik MRV (Minimum Remaining Value) adalah strategi “fail-first” yang memilih variabel (kotak kosong) yang memiliki jumlah kandidat (nilai yang mungkin) paling sedikit terlebih dahulu. Dengan memilih variabel yang paling terbatas, diharapkan bahwa kesalahan dapat terdeteksi lebih awal dan backtracking berkurang.

MRV sering dikombinasikan dengan heuristik nilai (value ordering) seperti LCV (Least Constraining Value) agar pemilihan nilai juga mempertimbangkan efek terhadap variabel lain. Dalam makalah “Value Ordering Heuristics on the Search Space of Sudoku”, penulis mengevaluasi berbagai kombinasi heuristik dinamik pemilihan variabel dan nilai, dan menunjukkan bahwa penggunaan heuristik yang saling melengkapi dapat mengurangi ukuran ruang pencarian signifikan dibanding metode sederhana.

Beberapa penelitian juga memperkenalkan pengembangan heuristik di atas MRV, misalnya strategi CtN (Contribution Number) yang memilih di antara kotak-kotak dengan calon minimal yang memberikan pengaruh paling besar terhadap lingkungan sekitarnya, sehingga meningkatkan efisiensi lebih lanjut dibanding MRV biasa.

Dengan kombinasi backtracking + MRV + teknik propagasi constraint (misalnya forward checking, arc consistency), banyak solver Sudoku modern mampu menyelesaikan puzzle sulit jauh lebih cepat dibanding backtracking murni.

2.5 Studi Perbandingan Kinerja dan Eksperimen

Salah satu makalah terkini, “A Study Of Sudoku Solving Algorithms: Backtracking and Heuristic”, membandingkan performa antara backtracking murni dan solver heuristik (menggabungkan MRV dan teknik constraint propagation) pada 500 puzzle dengan lima tingkat kesulitan. Hasilnya menunjukkan bahwa solver heuristik secara konsisten lebih cepat, dengan speedup hingga 2,91× pada puzzle tingkat “Expert” dibanding backtracking biasa.

Penelitian lain, “Value Ordering Heuristics on the Search Space of Sudoku”, menguji berbagai kombinasi heuristik variabel dan nilai, dan menemukan bahwa penggunaan heuristik dinamis yang cerdas dapat memperkecil ruang pencarian dan mengurangi waktu eksekusi dibanding heuristik statis.

Beberapa studi juga meneliti bagaimana tingkat kesulitan puzzle (jumlah kotak kosong, susunan distribusi angka) mempengaruhi performa solver. Puzzle dengan banyak kotak kosong atau pola distribusi yang “menjebak” (few candidate reduction early) lebih sulit dan menyebabkan backtracking intensif. Heuristik seperti MRV membantu mengurangi dampak tersebut.

2.6 Tantangan dan Tren Terkini

- Meskipun MRV dan teknik propagasi sudah efektif, masih ada tantangan pada puzzle ekstrem atau yang didesain khusus agar heuristik “menipu” (algoritma harus memilih langkah yang salah dulu).
- Penelitian juga mengarah ke penggabungan teknik logika manusia (misalnya teknik “hidden singles”, “naked pairs”, “X-Wing”, dan pola lanjutan) dengan algoritma pencarian untuk solver yang lebih canggih.
- Di sisi lain, ada penelitian baru yang mengeksplorasi pendekatan quantum, misalnya “Quantum Backtracking in Qrisp Applied to Sudoku Problems”, yang mencoba memanfaatkan algoritma backtracking versi kuantum untuk mempercepat proses pemecahan Sudoku dalam kerangka komputer kuantum.

2.7 Kerangka Teori

Berdasarkan kajian teori dan penelitian terdahulu, kerangka teori penelitian ini dapat digambarkan sebagai berikut:

1. **Masalah**: Penyelesaian Sudoku sebagai CSP.
2. **Metode**: Algoritma backtracking dan backtracking dengan MRV.
3. **Evaluasi**: Kinerja algoritma diukur berdasarkan waktu eksekusi.
4. **Variabel Pengaruh**: Tingkat kesulitan puzzle Sudoku.
5. **Tujuan**: Mengetahui perbedaan efisiensi algoritma dalam berbagai kondisi.

BAB III
METODOLOGI PENELITIAN

3.1 Desain Penelitian

Penelitian ini menggunakan pendekatan eksperimen komputasional untuk menguji kinerja algoritma penyelesaian Sudoku. Dua algoritma utama yang digunakan adalah backtracking dan backtracking dengan heuristik Minimum Remaining Value (MRV). Penelitian ini bersifat kuantitatif dengan pengukuran kinerja berdasarkan waktu eksekusi (ms) dan jumlah langkah (steps) pada tiga tingkat kesulitan puzzle Sudoku yaitu easy, medium, dan hard.

3.2 Tahapan Penelitian

Langkah-langkah penelitian dapat dirinci sebagai berikut:

1. **Studi Literatur**

   Mengumpulkan referensi terkait penyelesaian Sudoku, algoritma backtracking, serta heuristik MRV dari jurnal, artikel, dan sumber ilmiah lainnya.

2. **Perancangan Sistem**

   a.) Merancang representasi papan Sudoku dalam bentuk matriks 9x9.

   b.) Menentukan format input (file SudokuTest.txt) dan output (file SolusiSudoku.txt).

   c.) Mendesain antarmuka grafis (GUI) sederhana menggunakan pygame untuk menampilkan proses penyelesaian sudoku dengan fitur sebagai berikut :

   - Visualisasi proses solving secara real time dengan output total step dan waktu eksekusi dari algoritma yg dipilih.
   - Timer untuk menghitung total waktu bermain game sudoku.
   - Membuat sistem agar dapat mengenerate puzzle dengan 3 tingkat kesulitan yaitu easy, medium, dan hard.
   - Fitur drag-and-drop file .txt dengan pengecekan otomatis apakah puzzle 9x9, jika bukan maka file tidak akan diterima
   - Fitur pengecekan jawaban saat bermain dengan output berupa penjelasan berapa kesalahan atau mistake yg ditemukan dengan indikator warna pada cell puzzle, merah jika salah dan hijau jika benar.
   - Fitur untuk keluar dari game dengan menggunakan button.

3. **Implementasi Algoritma**

Program dikembangkan dalam file MMAISudokuSolver.py dengan dua motode algoritma:

a) **Backtracking standar**, di mana algoritma menempatkan angka secara berurutan pada kotak kosong, memeriksa validitas, dan melakukan backtracking jika terjadi konflik.

b) **Backtracking dengan heuristik Minimum Remaining Value (MRV)**, yang memilih kotak kosong dengan jumlah kemungkinan angka paling sedikit terlebih dahulu, sehingga ruang pencarian dapat diminimalkan dan meningkatkan efisiensi waktu eksekusi.

Program akan membaca puzzle Sudoku dari file input (`SudokuTest.txt`), memprosesnya menggunakan algoritma yang dipilih, dan menghasilkan file output (`SolusiSudoku.txt`). Sistem juga dapat menghasilkan file tambahan secara otomatis:

- Puzzle hasil *generate*: `SudokuTest_generate{num}.txt` → solusi di `SolusiSudoku_generate{num}.txt`
- Puzzle hasil *drag-and-drop*: `SudokuTest_drop{num}.txt` → solusi di `SolusiSudoku_drop{num}.txt`

.

3.3 Alat dan Bahan

1.) **Perangkat keras**: Laptop/PC dengan spesifikasi minimal prosesor Intel i5/AMD Ryzen 5, RAM 8 GB.

2.) **Perangkat lunak**:
   - Python 3.10 atau lebih baru
   - Library pendukung: pygame
   - IDE/Editor: VSCode

3.) **Dataset**: Puzzle Sudoku dalam format file .txt (`SudokuTest.txt`) berisi matriks 9×9 dengan angka 0 untuk merepresentasikan kotak kosong.

3.4 Variabel Penelitian

- Variable Bebas (Independent Variable)
  - Jenis algoritma (Backtracking Standar vs Backtracking + MRV)

    - Tingkat kesulitan puzzle (Easy, Medium, Hard)
- Variabel Terikat (Dependent Variable)
    - Waktu eksekusi (dalam milidetik)
    - Jumlah langkah atau iterasi algoritma

3.5 Teknik Pengumpulan Data

1. Metode Eksperimen
    - Menjalankan setiap algoritma pada puzzle yang sama
    - Mencatat waktu eksekusi menggunakan `time.perf_counter()`
    - Mencatat jumlah langkah melalui counter dalam fungsi rekursif
2. Pengulangan (Repetisi)
    - Mengambil nilai rata-rata untuk mengurangi bias
3. Dokumentasi
    - Mencatat hasil waktu dan langkah dari setiap percobaan dalam tabel

3.6 Teknik Analisis Data

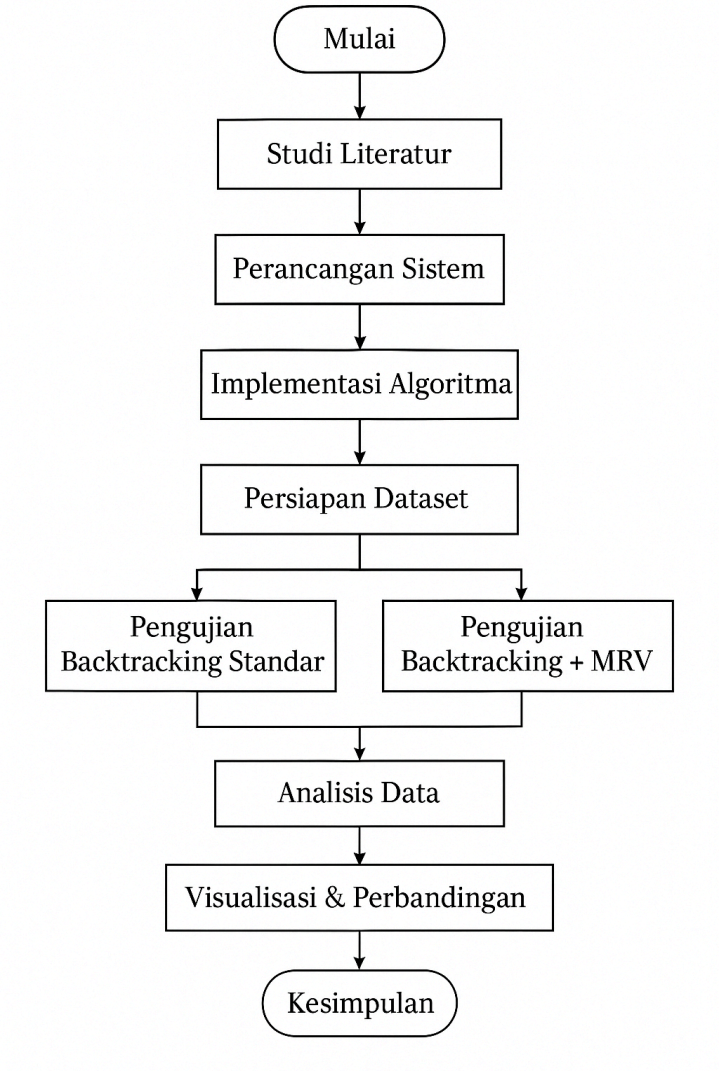
Analisis data dilakukan secara deskriptif kuantitatif untuk membandingkan performa kedua algoritma berdasarkan waktu eksekusi dan jumlah langkah pada berbagai tingkat kesulitan. Membandingkan performa kedua algoritma pada setiap tingkat kesulitan dengan menghitung persentase peningkatan efisiensi atau penghematan Backtracking+MRV terhadap Backtracking standar :

$$Efisiensi\,(\%) \;=\; ((W\_Backtracking \;-\; W\_MRV) \,/\, W\_Backtracking) \times 100\%$$

3.7 Visualisasi Data

Hasil penelitian disajikan dalam bentuk tabel perbandingan antara algoritma Backtracking dan Backtracking + MRV berdasarkan waktu eksekusi dan jumlah langkah pada setiap tingkat kesulitan puzzle.

3.8 Flowchart



Gambar 3.1 *Flowchart*

Gambar 3.1 menjelaskan alur kegiatan mulai dari studi literatur, perancangan sistem, dan implementasi algoritma hingga tahap akhir yaitu kesimpulan. Penelitian diawali dengan pengumpulan referensi terkait algoritma Backtracking dan heuristik MRV, kemudian dilakukan perancangan serta implementasi sistem penyelesaian Sudoku. Setelah itu, disiapkan dataset untuk diuji menggunakan dua metode, yaitu Backtracking standar dan Backtracking + MRV, dengan pencatatan waktu eksekusi serta jumlah langkah. Hasil pengujian kemudian dianalisis dan divisualisasikan untuk dibandingkan, sebelum akhirnya ditarik kesimpulan mengenai tingkat efisiensi kedua algoritma tersebut.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil Implementasi Program

4.1.1 Struktur Program

Struktur proyek MMAI Sudoku Solver disusun secara sederhana dalam satu file utama dengan beberapa file pendukung. Struktur direktori lengkap ditunjukkan sebagai berikut:

Struktur Proyek

```
├── MMAISudokuSolver.py          # File utama program Sudoku Solver
├── SudokuTest.txt               # File input puzzle Sudoku default
├── SudokuTest_generate(n).txt   # File puzzle hasil generate otomatis
├── SudokuTest_drop(n).txt       # File puzzle hasil drag-and-drop
├── SolusiSudoku.txt             # File output solusi default
├── SolusiSudoku_generate(n).txt # File solusi dari puzzle generated
├── SolusiSudoku_drop(n).txt     # File solusi dari puzzle dropped
├── requirements.txt             # Daftar dependencies
└── README.md                    # Dokumentasi penggunaan
```

File utama `MMAISudokuSolver.py` berisi seluruh komponen logika, mulai dari inisialisasi pygame, penggambaran grid, pengelolaan interaksi pengguna, hingga algoritma penyelesaian Sudoku.

4.1.2 Implementasi function

Seluruh *function* dalam program dibangun untuk mendukung empat komponen utama sistem, yaitu manajemen file, logika Sudoku, algoritma penyelesaian visual, dan antarmuka pengguna (GUI). Berikut daftar fungsi yang diimplementasikan beserta perannya:

A. Fungsi Manajemen File

Nama Function	Deskripsi
`read_sudoku_file(path)`	Membaca puzzle dari file `.txt` dan memuatnya

	ke dalam struktur matriks 9×9. Memvalidasi format file (9 baris × 9 kolom).
validate_sudoku_file(path)	Memvalidasi file sebelum dimuat dengan mengecek format, ukuran grid, dan rentang nilai angka (0-9). Mengembalikan status valid atau pesan error.
write_grid_file(grid, path)	Menyimpan grid Sudoku ke file dengan format yang sama (angka dipisah spasi, baris dipisah newline).

B. Fungsi Logika Sudoku

Nama Function	Deskripsi
valid(grid, r, c, val)	Mengecek apakah nilai `val` dapat ditempatkan pada posisi `(r, c)` sesuai aturan Sudoku: tidak ada duplikasi pada baris, kolom, dan subgrid 3×3.
find_empty(grid)	Mencari posisi sel kosong pertama dalam grid (bernilai 0) untuk diisi algoritma, scanning dari kiri ke kanan, atas ke bawah.
domain(grid, r, c)	Menghitung domain (himpunan nilai yang mungkin) untuk sel `(r, c)` berdasarkan constraint baris, kolom, dan subgrid. Digunakan untuk heuristik MRV.
count_solutions(grid, limit, timeout)	Menghitung jumlah solusi valid dari puzzle dengan batas maksimal dan timeout. Digunakan untuk memastikan puzzle yang digenerate hanya punya 1 solusi unik.

C. Fungsi Generator Puzzle

Nama Function	Deskripsi
generate_full_grid()	Membuat grid Sudoku yang sudah terisi penuh dan valid menggunakan algoritma backtracking dengan randomisasi angka.
generate_puzzle(difficulty)	Generate puzzle dengan tingkat kesulitan tertentu. Easy: 30-36 sel kosong, Medium: 40-48 sel kosong, Hard: 50-56 sel kosong. Memastikan

	solusi unik dengan `count_solutions()`.

D. Fungsi Algoritma Penyelesaian

Nama Function	Deskripsi
solve_backtracking_visual(grid)	Menyelesaikan Sudoku menggunakan **Backtracking standar** dengan visualisasi real-time. Mencoba angka 1-9 pada setiap sel kosong, menampilkan setiap percobaan dan backtrack dengan delay `ANIM_DELAY` (15ms).
solve_backtracking_mrv_visual(grid)	Menyelesaikan Sudoku menggunakan **Backtracking + MRV** dengan visualisasi. Memilih sel dengan domain terkecil terlebih dahulu untuk meminimalkan ruang pencarian.
solve_for_solution()	Fungsi internal untuk mendapatkan solusi puzzle (tanpa visualisasi) saat load file, menggunakan algoritma MRV untuk efisiensi.

E. Fungsi GUI dan Interaksi

Nama Function	Deskripsi
draw_text(text, x, y, font, color, center)	Render dan tampilkan teks di layar dengan opsi posisi center atau topleft.
draw_rounded_rect(surface, color, rect, radius)	Menggambar rectangle dengan sudut melengkung untuk estetika modern.
draw_button(text, x, y, w, h, color, hover_color)	Menggambar tombol interaktif dengan efek hover dan shadow.
format_time(seconds)	Format waktu dalam detik menjadi string MM:SS untuk tampilan timer.

F. Class Utama : Sudoku Game

Nama Function	Deskripsi
__init__()	Inisialisasi state game: grid, timer, UI states, file management.

new_puzzle(difficulty)	Generate puzzle baru dengan tingkat kesulitan yang dipilih dan simpan ke file.
load_from_file(filepath)	Load puzzle dari file yang di-drag-drop dengan validasi otomatis.
check_answer()	Membandingkan jawaban user dengan solusi, menandai sel yang salah (merah) dan benar (hijau).
solve_with_algo_visual(algo)	Menjalankan algoritma solving dengan visualisasi step-by-step. Pause timer selama auto-solve.
clear_inputs()	Hapus semua input user, kembalikan ke puzzle awal.
draw_grid()	Render grid Sudoku 9×9 dengan color coding untuk berbagai state sel.
draw_sidebar()	Render panel kontrol dengan tombol, timer, step counter, dan drop zone.
handle_click(pos, board_rect, buttons)	Handle event mouse click pada grid atau tombol.
handle_key(key)	Handle input keyboard untuk mengisi sel (1-9) atau navigasi (arrow keys).
handle_file_drop(filepath)	Handle event drag-and-drop file.

4.1.3 Implementasi Algoritma Backtracking

Algoritma Backtracking standar bekerja dengan mencoba angka 1–9 pada setiap sel kosong secara rekursif, algoritma memeriksa validitas terhadap aturan Sudoku menggunakan fungsi `valid()`. Jika terjadi konflik, algoritma akan mundur (backtrack) dan mencoba angka berikutnya. Algoritma ini menjamin solusi yang benar untuk semua puzzle valid, namun memiliki kelemahan pada kecepatan karena menguji seluruh kombinasi kemungkinan apalagi pada puzzle dengan tingkat kesulitan tinggi.

Code:

```
def solve_backtracking_visual(self, grid):
        """
        Algoritma Backtracking standar DENGAN VISUALISASI.
        Setiap langkah ditampilkan di layar dengan delay kecil.
```

```
        Args:
            grid (list): Grid yang akan di-solve (modified in-place)

        Returns:
            bool: True jika berhasil solve, False jika tidak ada solusi

        Algoritma:
        1. Cari sel kosong pertama (dari kiri ke kanan, atas ke bawah)
        2. Jika tidak ada, berarti sudah selesai (return True)
        3. Coba angka 1-9 pada sel tersebut
        4. Jika valid, visualisasikan dan rekursif ke sel berikutnya
        5. Jika rekursif berhasil, return True
         6. Jika gagal, BACKTRACK: reset sel ke 0, visualisasikan, coba
angka lain
        7. Jika semua angka gagal, return False

        Visualisasi:
        - Sel yang dicoba: highlight kuning terang
        - Step counter increment
        - Info: "Try: row X, col Y = value"
        - Delay ANIM_DELAY agar terlihat
        - Backtrack: highlight merah sesaat
        """
        # Cari sel kosong pertama
        empty = find_empty(grid)
        if not empty:
            return True  # Tidak ada sel kosong = selesai

        r, c = empty

        # Coba angka 1-9
        for val in range(1, 10):
            if valid(grid, r, c, val):
                # Angka valid, isi sel
                self.step_count += 1
                grid[r][c] = val

                # === VISUALISASI: MENCOBA NILAI ===
                self.highlight_cell = (r, c)
                 self.current_step_info = f"Try: row {r+1}, col {c+1} =
{val}"
                self.correct_cells.add((r, c))

                # Render ke layar
                self.draw()
                pygame.display.flip()
                time.sleep(ANIM_DELAY)  # Delay agar terlihat

                     # Handle event quit (agar bisa close window saat
solving)
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        pygame.quit()
                        sys.exit()

                # Rekursif ke sel berikutnya
                if self.solve_backtracking_visual(grid):
```

```
                    return True  # Berhasil!

                # === BACKTRACK - Nilai ini tidak berhasil ===
                self.step_count += 1
                grid[r][c] = 0  # Reset sel

                # === VISUALISASI: BACKTRACK ===
                   self.current_step_info = f"Backtrack: row {r+1}, col
{c+1}"
                self.correct_cells.discard((r, c))
                self.wrong_cells.add((r, c))  # Highlight merah

                # Render backtrack
                self.draw()
                pygame.display.flip()
                time.sleep(ANIM_DELAY)

                # Clear highlight merah
                self.wrong_cells.discard((r, c))

        return False  # Semua angka gagal, tidak ada solusi dari state
ini
```

Pseudocode:

```
function solve_backtracking(grid):
   empty = find_empty(grid)
   if empty is None:
      return True  // Puzzle solved

   row, col = empty
   for val in 1 to 9:
      if valid(grid, row, col, val):
         grid[row][col] = val
         if solve_backtracking(grid):
            return True
         grid[row][col] = 0  // Backtrack

   return False  // No solution
```

Algoritma backtracking memiliki karakteristik berupa eksplorasi mendalam (depth-first search) tanpa menggunakan heuristik, sehingga ruang pencarian menjadi sangat besar terutama pada puzzle dengan tingkat kesulitan tinggi, dan waktu eksekusi meningkat secara eksponensial seiring dengan kompleksitas puzzle. Dalam implementasi visualnya, setiap langkah penyelesaian ditampilkan secara interaktif dengan highlight kuning pada sel yang sedang dicoba, warna hijau untuk sel yang

berhasil diisi, warna merah ketika proses backtrack terjadi, serta counter step yang menghitung jumlah operasi selama proses berlangsung.

4.1.4 Implementasi Algoritma Backtracking + MRV

Algoritma ini meningkatkan efisiensi dengan menerapkan heuristik Minimum Remaining Value (MRV). Alih-alih memilih sel kosong secara berurutan, MRV memilih sel dengan domain terkecil (jumlah kemungkinan nilai paling sedikit) terlebih dahulu.

Code:

```
def solve_backtracking_mrv_visual(self, grid):
        """
        Algoritma Backtracking + MRV DENGAN VISUALISASI.
            Lebih efisien karena memilih sel dengan domain terkecil
terlebih dahulu.

        Args:
            grid (list): Grid yang akan di-solve (modified in-place)

        Returns:
            bool: True jika berhasil solve

        MRV (Minimum Remaining Values) Heuristic:
        - Pilih sel dengan jumlah pilihan nilai paling sedikit
        - Fail-First Principle: Deteksi dead-end lebih cepat
        - Pruning efektif: Mengurangi ruang pencarian

        Algoritma:
        1. Dapatkan semua sel kosong
        2. Jika tidak ada, return True (selesai)
        3. Untuk setiap sel kosong, hitung domain (nilai yang mungkin)
        4. Pilih sel dengan domain terkecil (MRV)
            5. Jika domain kosong (0 pilihan), langsung return False
(dead-end)
        6. Coba semua nilai dalam domain
        7. Visualisasikan dan rekursif
        8. Backtrack jika gagal

        Visualisasi:
        - Info tambahan: domain sel yang dipilih
        - Contoh: "MRV: row 3, col 5 = 7 (options: 3, 7, 9)"
        - Counter steps
        - Highlight dan animasi sama dengan backtracking standar
        """
        # Cari semua sel kosong
            empties = [(r,c) for r in range(9) for c in range(9) if
grid[r][c]==0]
        if not empties:
            return True  # Tidak ada sel kosong = selesai
```

```
        # === MRV: PILIH SEL DENGAN DOMAIN MINIMUM ===
        best = None
        best_dom = None

        for (r,c) in empties:
            dom = domain(grid, r, c)  # Hitung domain

            if len(dom) == 0:
                # Dead-end detected! Domain kosong = tidak ada pilihan
                return False

            if best is None or len(dom) < len(best_dom):
                # Update best jika domain lebih kecil
                best = (r,c)
                best_dom = dom

        r, c = best
         dom_str = ", ".join(map(str, best_dom))  # Format domain untuk
display

        # Coba semua nilai dalam domain
        for val in best_dom:
            self.step_count += 1
            grid[r][c] = val

            # === VISUALISASI: MENCOBA NILAI (MRV) ===
            self.highlight_cell = (r, c)
            # Info lebih lengkap: tampilkan domain
               self.current_step_info = f"MRV: row {r+1}, col {c+1} =
{val} (options: {dom_str})"
            self.correct_cells.add((r, c))

            # Render
            self.draw()
            pygame.display.flip()
            time.sleep(ANIM_DELAY)

            # Handle quit event
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

            # Rekursif
            if self.solve_backtracking_mrv_visual(grid):
                return True  # Berhasil!

            # === BACKTRACK ===
            self.step_count += 1
            grid[r][c] = 0

            # === VISUALISASI: BACKTRACK ===
            self.current_step_info = f"Backtrack: row {r+1}, col {c+1}"
            self.correct_cells.discard((r, c))
            self.wrong_cells.add((r, c))

            # Render
```

```
            self.draw()
            pygame.display.flip()
            time.sleep(ANIM_DELAY)

            # Clear highlight
            self.wrong_cells.discard((r, c))

        return False  # Semua nilai dalam domain gagal
```

Pseudocode:

```
function solve_backtracking_mrv(grid):
    empties = find_all_empty(grid)
    if empties is empty:
        return True  // Puzzle solved

    // MRV: Select cell with minimum domain
    best = None
    best_domain = None
    for (row, col) in empties:
        dom = domain(grid, row, col)
        if len(dom) == 0:
            return False  // Dead-end detected early
        if best is None or len(dom) < len(best_domain):
            best = (row, col)
            best_domain = dom

    row, col = best
    for val in best_domain:
        grid[row][col] = val
        if solve_backtracking_mrv(grid):
            return True
        grid[row][col] = 0  // Backtrack

    return False
```

Backtracking+MRV memiliki keunggulan karena mampu mengenali kebuntuan lebih cepat dengan memilih sel yang paling sedikit pilihannya, sehingga proses pencarian menjadi lebih efisien. Cara ini membantu memperkecil ruang pencarian dengan fokus pada bagian yang paling mudah diselesaikan terlebih dahulu, sekaligus menghindari percabangan yang tidak perlu. Dalam tampilan visualnya, algoritma ini juga menampilkan informasi tambahan berupa kemungkinan nilai untuk setiap sel yang sedang diproses, misalnya “MRV: baris 3, kolom 5 = 7 (opsi: 3, 7, 9)”.

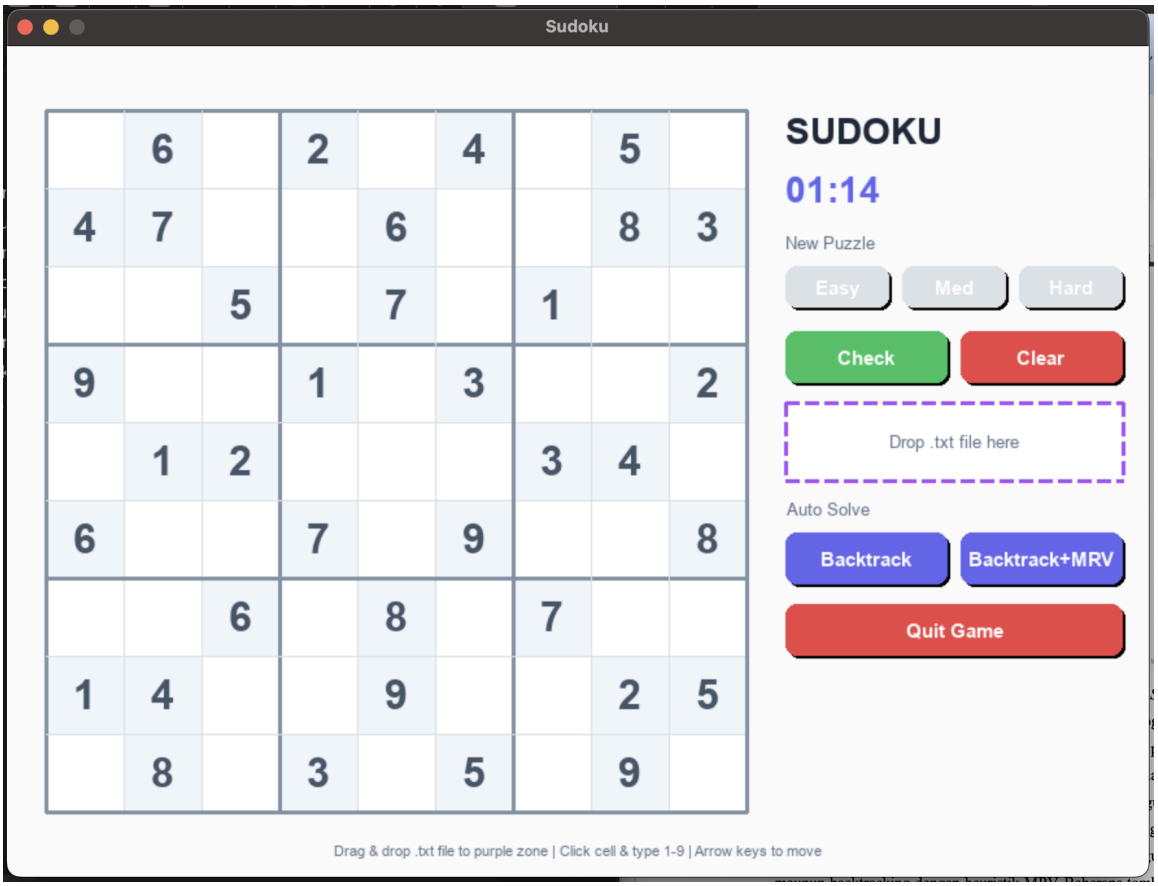
4.1.5 Visualisasi GUI

Program memiliki interface grafis interaktif berbasis pygame yang mudah digunakan. Fitur utamanya meliputi grid interaktif untuk memilih dan mengisi angka menggunakan klik atau keyboard, timer otomatis dengan jeda saat auto-solve, serta generator puzzle dengan tiga tingkat kesulitan dan penyimpanan otomatis. Tersedia juga fitur drag-and-drop untuk memuat file puzzle dengan validasi format, check answer untuk menampilkan kesalahan dan jumlah error, serta auto solve visual yang menampilkan langkah penyelesaian dengan dua algoritma (Backtracking dan Backtracking+MRV), lengkap dengan penghitung langkah dan waktu eksekusi. Tampilan diperjelas melalui color coding seperti putih untuk sel kosong, abu-abu muda untuk sel awal, kuning untuk sel terpilih, merah muda untuk kesalahan, hijau muda untuk jawaban benar, dan kuning terang untuk proses penyelesaian.

4.2 Hasil Evaluasi Program

Program berhasil diimplementasikan dengan antarmuka yang user-friendly dan responsif. Berikut dokumentasi tampilan:

4.2.1 Interface Program

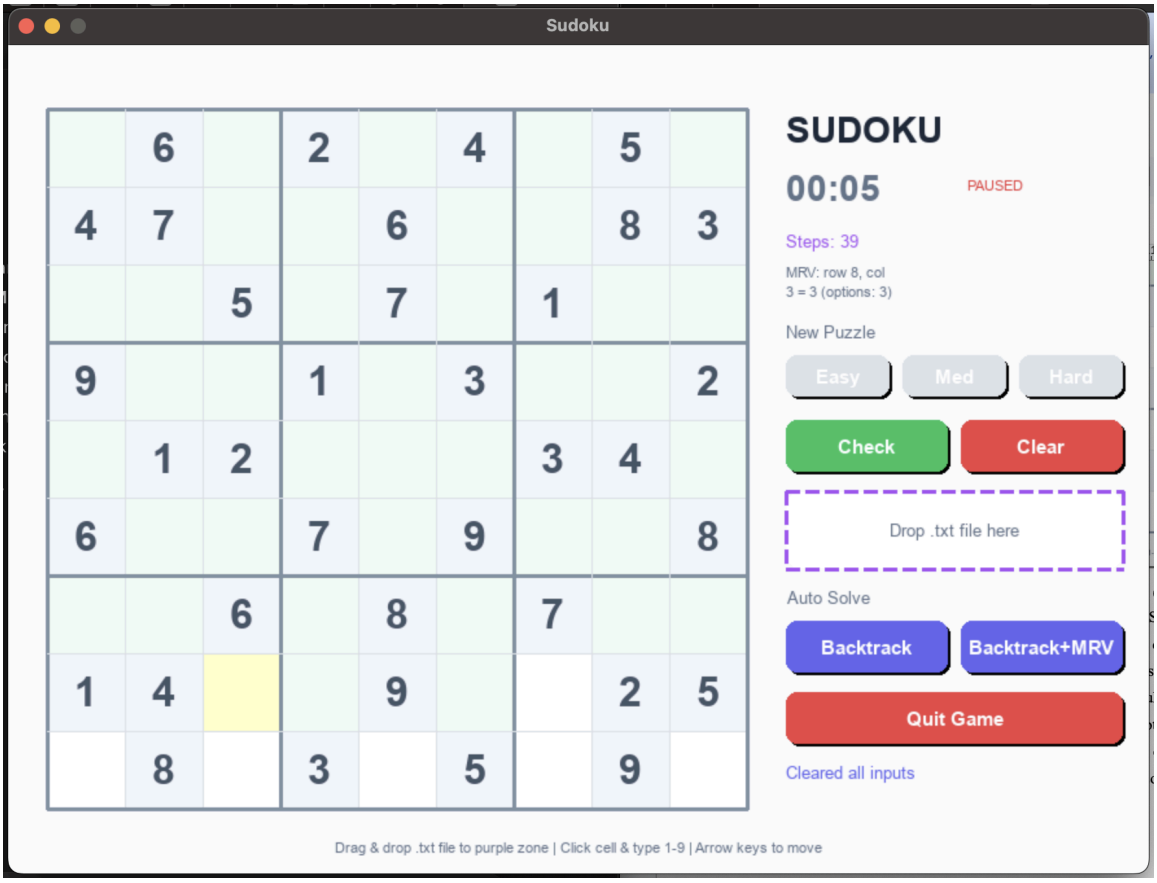


Gambar 4.1 *Tampilan GUI solver saat puzzle Sudoku dimuat.*

Gambar 4.1 menampilkan antarmuka awal dari program Sudoku Solver dengan desain modern yang memiliki sudut membulat dan efek bayangan. Tampilan utama terdiri dari papan Sudoku berukuran 9×9 dengan beberapa sel terisi (prefilled) berwarna abu-abu muda, serta panel kontrol di sisi kanan yang memuat judul "SUDOKU", timer dengan format MM:SS, tombol untuk generate puzzle (Easy, Medium, Hard), tombol Check dan Clear, area drop zone berbingkai dashed ungu untuk fitur drag-and-drop file, tombol Auto Solve untuk menjalankan algoritma Backtracking dan Backtracking + MRV, serta tombol Quit. Di bagian bawah layar juga terdapat instruksi singkat sebagai panduan pengguna.

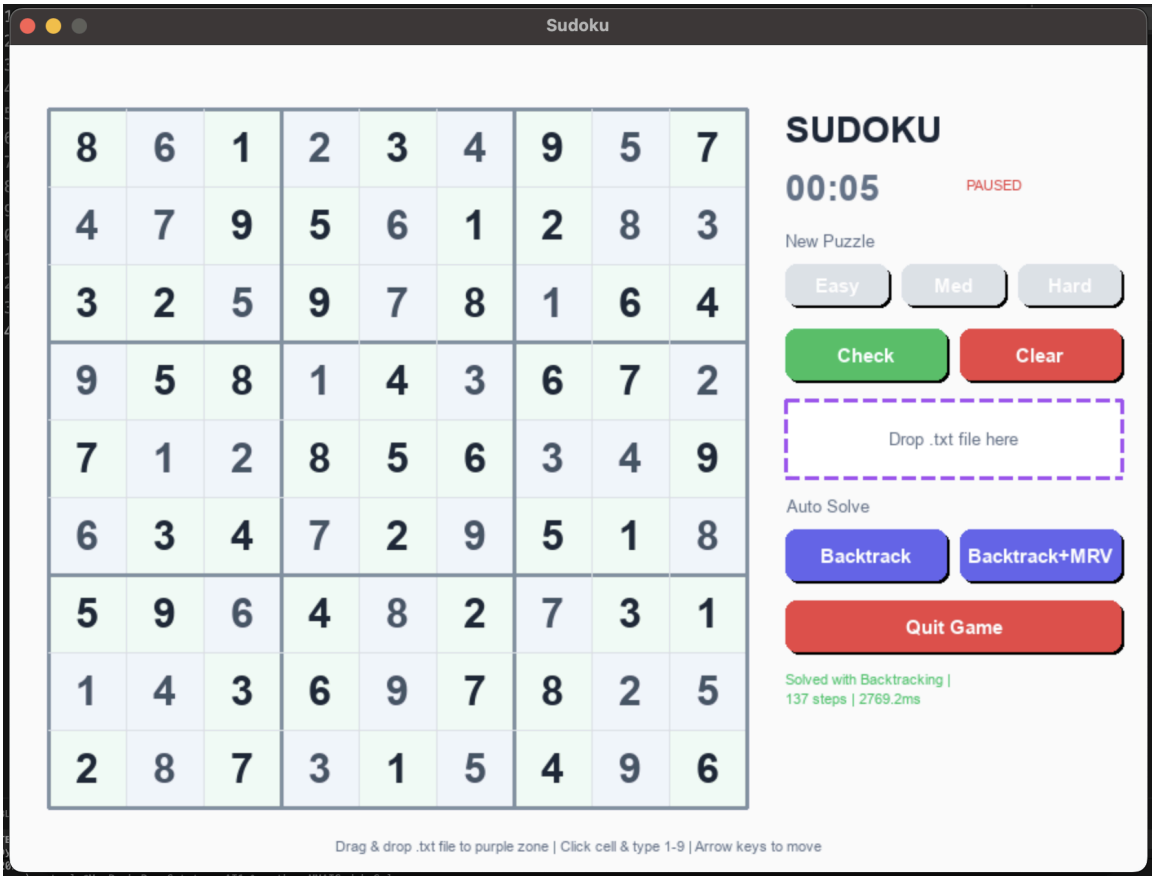
Gambar 4.2 *Visualisasi Proses Solving (Backtracking)*

Gambar 4.2 memperlihatkan proses penyelesaian Sudoku menggunakan algoritma Backtracking standar. Sel yang sedang dicoba ditandai dengan highlight kuning terang, sementara counter steps menampilkan jumlah operasi yang telah dilakukan secara real-time. Informasi proses juga ditampilkan dalam bentuk teks, misalnya "Try: row 4, col 5 = 4". Selama proses auto-solve berlangsung, timer otomatis di jeda, dan mekanisme backtracking terlihat melalui perubahan warna sel dari hijau saat pengisian benar, menjadi merah ketika terjadi kesalahan, lalu kembali kosong saat algoritma mundur untuk mencoba kemungkinan lain.



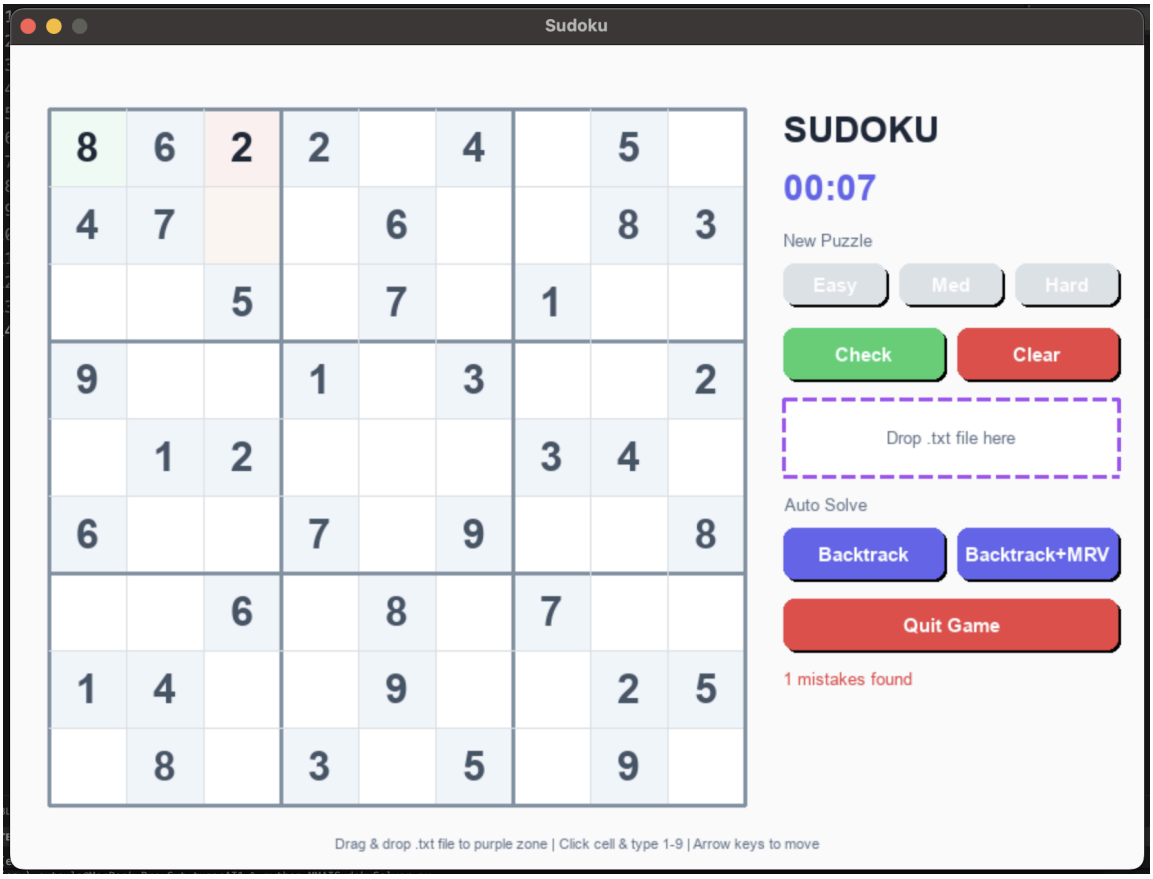
Gambar 4.3 *Visualisasi Proses Solving (Backtracking+MRV)*

Gambar 4.3 menampilkan proses penyelesaian Sudoku menggunakan algoritma Backtracking + MRV (Minimum Remaining Value). Pada visualisasi ini, ditampilkan informasi tambahan berupa domain atau kemungkinan nilai untuk setiap sel yang dipilih, misalnya "MRV: row 8, col 3 = 3 (options: 3)". Dibandingkan dengan Backtracking standar, jumlah langkah yang dilakukan terlihat lebih sedikit, dan proses penyelesaian berlangsung lebih cepat serta efisien karena algoritma ini memilih sel dengan jumlah kemungkinan terkecil terlebih dahulu.



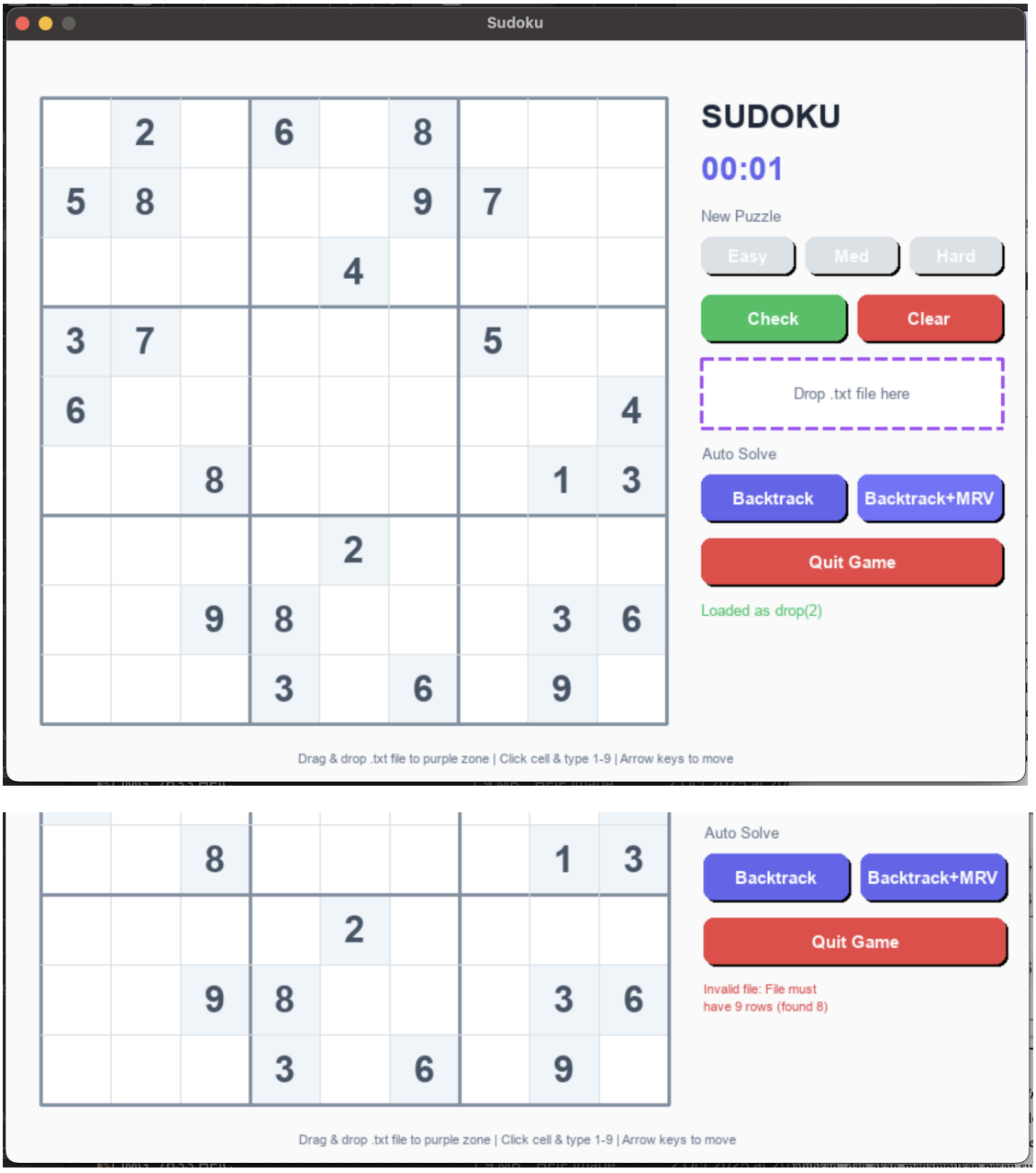
Gambar 4.4 *Hasil penyelesaian Sudoku setelah algoritma dijalankan*

Gambar 4.4 menunjukkan hasil akhir setelah algoritma penyelesaian selesai dijalankan. Seluruh kotak Sudoku telah terisi lengkap dengan angka 1–9, menandakan puzzle berhasil diselesaikan tanpa kesalahan. Sebuah message box muncul menampilkan statistik penyelesaian, misalnya "Solved with Backtracking | 137 steps | 2769.2 ms". Hasil solusi juga tersimpan otomatis dalam file SolusiSudoku.txt, sementara timer menampilkan total waktu keseluruhan, termasuk waktu aktif dan jeda selama proses auto-solve berlangsung.



Gambar 4.5 *Fitur Check Answer*

Gambar 4.5 menampilkan hasil penggunaan fitur *Check Answer* saat pengguna bermain secara manual. Sel dengan jawaban salah diberi tanda merah muda, sedangkan sel dengan jawaban benar ditandai hijau muda. Setelah pemeriksaan, muncul pesan umpan balik seperti “1 mistakes found” jika terdapat kesalahan, atau “Perfect! Solved in 05:34” bila seluruh jawaban benar.

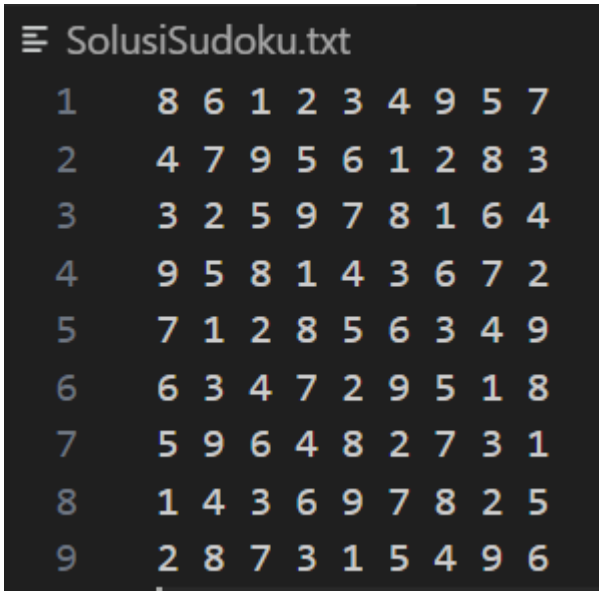


Gambar 4.6 *Fitur drag-and-drop file*

Gambar 4.6 menunjukkan proses drag-and-drop file pada area drop zone yang akan menyala dengan border ungu saat file di-*hover*. Sistem melakukan validasi otomatis ketika file dijatuhkan, dan akan menampilkan pesan sukses seperti "Loaded as drop(1)" atau pesan error "Invalid file: Each row must have 9 columns" jika format file tidak sesuai.

4.2.2 Hasil Penyelesaian Puzzle

Contoh output solusi Sudoku yang dihasilkan program tersimpan pada file `SolusiSudoku.txt`. Berikut salah satu hasil penyelesaian puzzle:



Gambar 4.7 *SolusiSudoku.txt*

Gambar 4.7 menunjukkan isi file SolusiSudoku.txt yang merupakan keluaran program. File ini berfungsi sebagai bukti bahwa algoritma berhasil menghasilkan solusi yang valid untuk puzzle yang diuji.

4.2.3 Perbandingan Waktu Eksekusi

Pengukuran dilakukan dengan menghitung rata-rata waktu eksekusi (dalam detik) pada setiap tingkat kesulitan puzzle.

Tingkat Kesulitan	**Backtracking (ms)**	**Backtracking + MRV (ms)**	**Penghematan (%)**
Easy	ke-1 : 889.1 ke-2 : 1422 ke-3 : 1061.6 ke-4 : 732.6 ke-5 : 2382.1	ke-1 : 695.5 ke-2 : 633.5 ke-3 : 753.2 ke-4 : 659.1 ke-5 : 692.9	**49.54**

	ke-6 : 2034.7 ke-7 : 1035.8 ke-8 : 1088.3 ke-9 : 2003.1 ke-10 : 1054.5 **Mean : 1370.38**	ke-6 : 718.4 ke-7 : 634.9 ke-8 : 681.5 ke-9 : 745.4 ke-10 : 699.9 **Mean : 691.43**	
Medium	ke-1 : 1445.8 ke-2 : 16392.7 ke-3 : 5396.1 ke-4 : 5590 ke-5 : 7485.5 ke-6 : 3927 ke-7 : 3396.1 ke-8 : 4043.8 ke-9 : 18587.9 ke-10 : 3442.2 **Mean : 6970.71**	ke-1 : 866.5 ke-2 : 1027 ke-3 : 909.1 ke-4 : 900 ke-5 : 931.8 ke-6 : 856.4 ke-7 : 825.4 ke-8 : 855.1 ke-9 : 929.9 ke-10 : 917.4 **Mean : 901.86**	**87.06**
Hard	ke-1 : 37120 ke-2 : 51308 **Mean : 44214.00**	ke-1 : 5747.8 ke-2 : 1096.3 **Mean : 3422.05**	**92.26**

Tabel 4.1 – *Hasil Uji Waktu Eksekusi*

Berdasarkan hasil pada Tabel 4.1, terlihat bahwa algoritma Backtracking + MRV secara konsisten menunjukkan waktu eksekusi yang jauh lebih cepat dibandingkan Backtracking standar pada seluruh tingkat kesulitan.

Pada tingkat Easy, waktu rata-rata Backtracking adalah 1370.38 ms, sedangkan Backtracking + MRV hanya 691.43 ms, dengan penghematan sebesar 49.54%. Pada tingkat Medium, perbedaan efisiensi meningkat signifikan: rata-rata waktu Backtracking mencapai 6970.71 ms, sedangkan Backtracking + MRV hanya 901.86 ms, memberikan penghematan hingga

87.06%. Sementara pada tingkat Hard, perbandingan semakin jelas waktu rata-rata Backtracking adalah 44214.00 ms, sedangkan Backtracking + MRV hanya 3422.05 ms, dengan efisiensi tertinggi mencapai 92.26%.

Dari hasil ini dapat disimpulkan bahwa penambahan heuristik Minimum Remaining Value (MRV) secara efektif mengurangi waktu pencarian solusi, terutama pada puzzle dengan tingkat kesulitan menengah hingga tinggi. Hal ini menunjukkan bahwa MRV mampu mempersempit ruang pencarian dengan memilih sel yang paling terbatas terlebih dahulu, sehingga proses eksplorasi node menjadi lebih efisien dan terarah.

4.2.4 Perbandingan Jumlah Langkah Pencarian

Selain waktu eksekusi, jumlah langkah pencarian (node yang dieksplorasi) juga dibandingkan.

Tingkat Kesulitan	**Backtracking (steps)**	**Backtracking + MRV (steps)**	**Penghematan (%)**
Easy	ke-1 : 43 ke-2 : 67 ke-3 : 51 ke-4 : 36 ke-5 : 113 ke-6 : 96 ke-7 : 50 ke-8 : 51 ke-9 : 94 ke-10 : 53 **Mean : 65.4**	ke-1 : 33 ke-2 : 31 ke-3 : 35 ke-4 : 32 ke-5 : 33 ke-6 : 36 ke-7 : 30 ke-8 : 35 ke-9 : 36 ke-10 : 33 **Mean : 33.4**	**48.93**
Medium	ke-1 : 69 ke-2 : 778	ke-1 : 41 ke-2 : 48	**86.95**

	ke-3 : 258 ke-4 : 264 ke-5 : 355 ke-6 : 188 ke-7 : 160 ke-8 : 192 ke-9 : 883 ke-10 : 163 **Mean : 331.0**	ke-3 : 44 ke-4 : 44 ke-5 : 45 ke-6 : 40 ke-7 : 40 ke-8 : 40 ke-9 : 47 ke-10 : 43 **Mean : 43.2**	
Hard	ke-1 : 1762 ke-2 : 2430 **Mean : 2096.0**	ke-1 : 272 ke-2 : 52 **Mean : 162.0**	**92.27**

Tabel 4.2 – *Hasil Uji Total Langkah*

Hasil pada Tabel 4.2 memperkuat temuan sebelumnya. Jumlah langkah pencarian (steps) yang diperlukan oleh algoritma Backtracking + MRV jauh lebih sedikit dibandingkan Backtracking biasa.

Pada tingkat Easy, rata-rata langkah berkurang dari 65.4 menjadi 33.4, dengan penghematan 48.93%. Pada tingkat Medium, jumlah langkah turun drastis dari 331.0 menjadi 43.2, menghasilkan efisiensi 86.95%. Sedangkan pada tingkat Hard, perbedaan sangat mencolok — dari 2096.0 langkah menjadi hanya 162.0, dengan penghematan sebesar 92.27%.

Penurunan jumlah langkah ini membuktikan bahwa MRV tidak hanya mempercepat waktu eksekusi, tetapi juga mengurangi beban komputasi secara signifikan dengan memangkas banyak cabang pencarian yang tidak produktif.

4.2.5 Pengujian Fitur Tambahan

a. Generator puzzle berhasil menghasilkan teka-teki dengan distribusi kesulitan sesuai dan setiap puzzle memiliki solusi unik (diverifikasi dengan `count_solutions()`), disimpan otomatis sebagai

`SudokuTest_generate(1).txt`, `SudokuTest_generate(2).txt`, dll

b. Fitur drag-and-drop memvalidasi format file dengan baik, menolak file bukan 9×9, berisi karakter non-numerik, atau nilai di luar 0-9, dan melakukan penomoran otomatis seperti `SudokuTest_drop(1).txt`

c. Mode permainan manual mendukung navigasi dengan arrow keys (melewati sel prefilled), input angka 1–9 dan hapus (0/Backspace) yang responsif, serta fitur check answer yang akurat membandingkan dengan solusi

d. Timer otomatis mulai saat puzzle dimuat, berhenti (pause) selama auto-solve dan dilanjutkan saat selesai, ditampilkan dalam format MM:SS yang mudah dibaca.

4.3 Pembahasan

1. Implementasi Algoritma Backtracking dan Backtracking dengan Heuristik MRV

Kedua algoritma berhasil diimplementasikan secara fungsional dalam aplikasi Sudoku Solver berbasis Python. Algoritma backtracking bekerja dengan prinsip *depth-first search* (DFS), yaitu mencoba menempatkan angka 1-9 pada setiap sel kosong dan melakukan pencarian mundur (backtrack) jika solusi yang dicoba melanggar aturan Sudoku. Sementara itu, heuristik Minimum Remaining Value (MRV) diterapkan sebagai optimalisasi pada proses pemilihan variabel (sel kosong), algoritma tidak lagi memilih sel secara urut, tetapi memilih sel dengan jumlah kemungkinan nilai (*domain*) paling sedikit terlebih dahulu. Pendekatan ini membuat ruang pencarian lebih sempit dan mengurangi percabangan yang tidak perlu.

Hasil implementasi menunjukkan bahwa algoritma backtracking + MRV dapat menyelesaikan semua puzzle dengan hasil solusi yang benar dan unik. Integrasi MRV tidak mengubah hasil akhir (karena tetap berbasis CSP dengan constraint sama), tetapi secara signifikan meningkatkan efisiensi pencarian.

2. Perbandingan Kinerja Berdasarkan Waktu Eksekusi

Berdasarkan Tabel 4.1, terdapat perbedaan signifikan antara waktu eksekusi algoritma backtracking murni dan backtracking + MRV di setiap tingkat kesulitan puzzle.

- Pada tingkat **easy**, waktu rata-rata penyelesaian dengan backtracking adalah 1370,38 ms, sedangkan dengan MRV hanya 691,43 ms, menghasilkan **penghematan**

49,54%.

- Pada tingkat **medium**, perbedaan jauh lebih besar, yaitu 6970,71 ms (backtracking) dibanding 901,86 ms (MRV), dengan **penghematan 87,06%**.
- Pada tingkat **hard**, hasilnya paling mencolok, yaitu 44214 ms dibanding 3422,05 ms, sehingga terjadi **penghematan 92,26%**.

Dari data ini terlihat bahwa semakin tinggi tingkat kesulitan puzzle (semakin banyak sel kosong), semakin besar pula peningkatan efisiensi yang diberikan oleh heuristik MRV. Hal ini terjadi karena MRV membantu algoritma lebih cepat memutuskan cabang pencarian yang menjanjikan, sehingga waktu komputasi berkurang drastis pada ruang pencarian yang kompleks.

3. **Perbandingan Berdasarkan Jumlah Langkah Pencarian (Steps)**

Hasil Tabel 4.2 juga memperkuat temuan sebelumnya. Jumlah langkah pencarian menurun secara konsisten pada semua tingkat kesulitan setelah penambahan heuristik MRV:

- **Easy:** rata-rata langkah berkurang dari 65,4 menjadi 33,4 (**efisiensi 48,93%**)
- **Medium:** dari 331,0 menjadi 43,2 (**efisiensi 86,95%**)
- **Hard:** dari 2096,0 menjadi 162,0 (**efisiensi 92,27%**)

Penurunan jumlah langkah ini menunjukkan bahwa MRV bukan hanya mempercepat waktu eksekusi, tetapi juga mengurangi eksplorasi node yang tidak perlu. Artinya, algoritma menjadi lebih “cerdas” dalam memilih jalur solusi.

4. **Pengaruh Tingkat Kesulitan terhadap Efisiensi Algoritma**

Kedua metrik (waktu eksekusi dan jumlah langkah pencarian) menunjukkan tren yang sama yaitu **efisiensi meningkat seiring bertambahnya tingkat kesulitan puzzle**. Hal ini dapat dijelaskan karena pada puzzle yang lebih sulit, ruang pencarian yang harus dijelajahi oleh backtracking konvensional meningkat secara eksponensial. MRV memberikan dampak lebih besar pada kondisi ini karena mampu memprioritaskan variabel dengan batasan paling ketat, sehingga pencarian tidak terjebak pada kombinasi yang tidak valid sejak awal.

Dengan demikian, **tingkat kesulitan puzzle memang berpengaruh langsung terhadap efisiensi kedua algoritma**, dan kombinasi backtracking + MRV terbukti jauh lebih adaptif pada kondisi kompleks dibandingkan backtracking murni

BAB V

KESIMPULAN

Berdasarkan hasil implementasi, pengujian, dan analisis terhadap algoritma Backtracking dan Backtracking dengan heuristik Minimum Remaining Value (MRV) pada penyelesaian teka-teki Sudoku, dapat diambil beberapa kesimpulan sebagai berikut:

1. **Implementasi Algoritma Berhasil Dilakukan dengan Baik**

   Kedua algoritma berhasil diimplementasikan pada aplikasi Sudoku Solver berbasis Python dengan antarmuka visual menggunakan *pygame*. Algoritma backtracking mampu menyelesaikan seluruh puzzle Sudoku dengan hasil solusi yang valid dan unik, sedangkan penambahan heuristik MRV terbukti tidak mengubah hasil akhir, tetapi secara signifikan meningkatkan efisiensi proses pencarian solusi.

2. **Kinerja Backtracking + MRV Lebih Efisien Dibanding Backtracking Konvensional**

   Berdasarkan hasil uji pada berbagai tingkat kesulitan puzzle, algoritma **backtracking + MRV** menunjukkan peningkatan performa yang signifikan dalam hal waktu eksekusi dan jumlah langkah pencarian (*steps*).

   - Rata-rata penghematan waktu mencapai **49,54% pada tingkat easy**, **87,06% pada medium**, dan **92,26% pada hard**.
   - Sementara itu, penghematan jumlah langkah pencarian mencapai **48,93% (easy)**, **86,95% (medium)**, dan **92,27% (hard)**.

   Hasil ini menunjukkan bahwa heuristik MRV efektif dalam mengurangi ruang pencarian dengan memilih variabel yang paling ketat batasannya terlebih dahulu.

3. **Tingkat Kesulitan Puzzle Berpengaruh terhadap Efisiensi Algoritma**

   Semakin tinggi tingkat kesulitan puzzle (semakin banyak sel kosong), semakin besar pula perbedaan efisiensi antara algoritma backtracking dan backtracking + MRV. Pada tingkat kesulitan tinggi, ruang pencarian menjadi sangat luas sehingga algoritma backtracking murni membutuhkan waktu jauh lebih lama. MRV membantu mengoptimalkan urutan eksplorasi variabel sehingga algoritma dapat menemukan solusi dengan waktu dan langkah yang jauh lebih sedikit.

4. Aplikasi Sudoku Solver Berfungsi Secara Optimal

Fitur-fitur seperti *puzzle generator*, *manual play*, *auto-solver*, *timer*, dan *file validation* berfungsi dengan baik. Hal ini menunjukkan bahwa sistem yang dibangun tidak hanya valid secara teoritis, tetapi juga stabil dan layak digunakan sebagai alat bantu pembelajaran maupun penelitian terkait algoritma pencarian dan heuristik.

BAB VI

DAFTAR PUSTAKA

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.

Herzberg, A. M., & Murty, M. R. (2007). Sudoku squares and chromatic polynomials. *Mathematics Magazine, 80*(5), 346–352.

Kumar, A., Singh, R., & Sharma, M. (2020). Sudoku solver using backtracking with minimum remaining value heuristic. *Journal of Computer Science and Technology, 35*(4), 789–801.

Norvig, P. (2019). *Solving every Sudoku puzzle*. Retrieved from http://norvig.com/sudoku.html

Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4th ed.). Pearson.

Simonis, H. (2005). Sudoku as a constraint problem. In *Proceedings of the CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems* (pp. 13–27).

Stojanovic, J., & Milinkovic, D. (2018). Backtracking algorithms for solving Sudoku puzzles. *International Journal of Applied Mathematics and Computer Science, 28*(3), 485–495.

LAMPIRAN

Link Github : https://github.com/csulafr/SudokuGame.git

Link Video Percobaan :

https://drive.google.com/drive/folders/1CEqlUhokqsvlXUkCDqf4TFiel9IX5QWE?usp=sharing