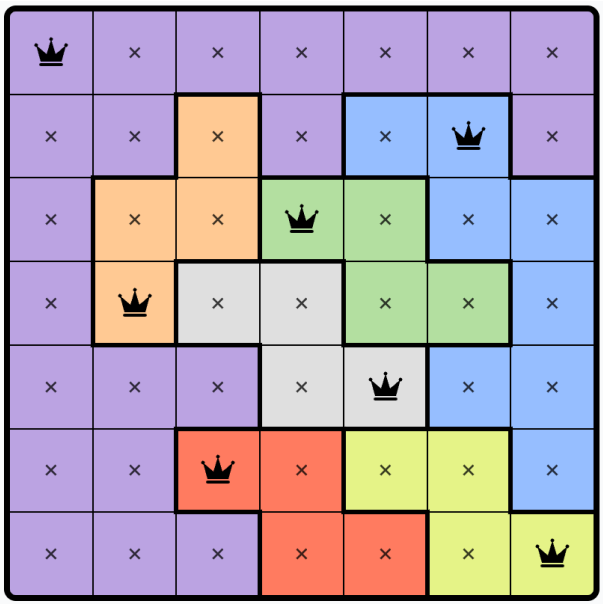
Laporan Tugas Kecil 1

IF2211 Strategi Algoritma

Penyelesaian Permainan *Queens* Linkedin

Semester II Tahun 2025/2026



Oleh:

An-Dafa Anza Avansyah (13524038)

Laboratorium Ilmu dan Rekayasa Komputasi
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

17 Februari 2026

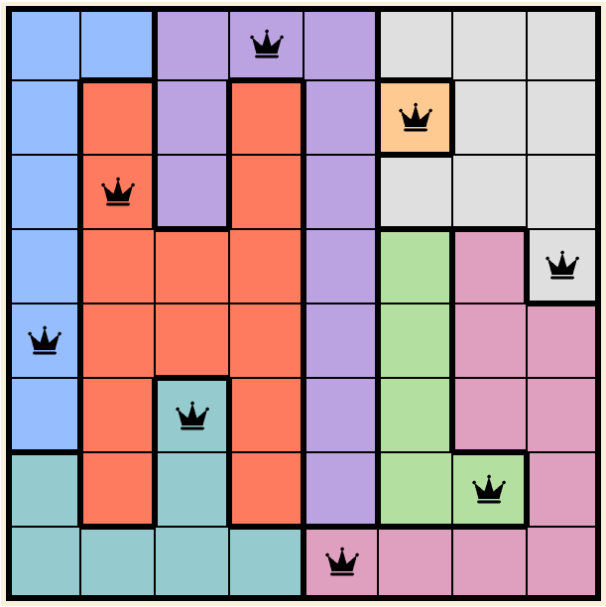
Daftar Isi

Daftar Gambar

Listings

1 Pendahuluan

Salah satu gim *puzzle* yang terkenal adalah *Queens Game* yang dibuat situs jejaring sosial Linkedin. Gim ini adalah gim *puzzle* yang melatih kemampuan intuisi dan pemahaman terhadap ruang spasial yang ditunjukkan oleh petak atau *grid* pada permainannya. Tujuan dari permainan ini adalah meletakkan sejumlah n-buah bidak ratu pada *board* berukuran $n \times n$.



Gambar 1: *Queens Game*

Permainan ini dimulai dari sebuah papan yang di dalam papan tersebut terdapat beberapa daerah atau *region*. Umumnya sebuah papan yang valid memiliki n daerah untuk sebuah petak berukuran $n \times n$. Adapun beberapa aturan dalam permainan ini, yang pertama adalah setiap baris atau kolom hanya boleh terdapat tepat satu bidak ratu, aturan kedua yaitu penempatan ratu tidak boleh saling bertetangga atau berdekatan, baik secara vertikal, horizontal, maupun diagonal, dan aturan ketiga adalah setiap daerah hanya boleh diisi oleh tepat satu ratu.

Secara heuristik permainan ini dapat diselesaikan dengan melakukan percobaan dengan mempertimbangkan keabsahan petak yang akan diletakan sebuah bidak ratu. Dengan demikian, solusi dari permainan ini berhasil didapatkan melalui beberapa kali iterasi *trial and error*. Adapun heuristik lain seperti memulia meletakkan bidak pada daerah dengan luas terkecil, dengan demikian dapat mengurangi kemungkinan peletakkan bidak selanjutnya.

Pada tugas kecil 1 IF2211 Strategi Algoritma, penulis merancang sebuah program yang akan memberikan solusi untuk *Queens Game* ini. Program yang dirancang penulis mengaplikasikan dua pendekatan yaitu dengan menggunakan pendekatan "naif" *brute force* dan *backtracking*. Dengan demikian dapat dibandingkan hasil dari kedua algortima tersebut.

2 Dasar Teori

2.1 Algoritma *Brute Force*

Algoritma *Brute Force* adalah algoritma yang menyelesaikan suatu permasalahan secara lempang (*straightforward*). Algoritma *brute force* memecahkan suatu permasalahan secara sederhana, langsung, dan jelas caranya. Algoritma ini didasarkan pada pernyataan di dalam permasalahan (*problem statement*) dan definisi serta konsep yang dilibatkan di dalam permasalahn tersebut. Beberapa permasalahan yang dapat dipecahkan menggunakan algoritma ini seperti pencarian elemen terbesar atau terkecil dalam larik, pengurutan elemen, menghitung perpangkatan, dan sebagainya.

Algoritma *brute force* seringkali diasosiasikan sebagai algoritma yang tidak "cerdas" dan tidak sangkil karena algoritma ini membutuhkan daya komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Oleh karena itu, algoritma ini seringkali disebut juga algoritma naif (*naive algorithm*). Algoritma *brute force* lebih cocok untuk persoalan yang ukuran masukkannya (n) kecil. Pertimabangan menggunakan algoritma ini adalah sederhana dan implementasinya mudah. Seringkali algoritma *brute force* digunakan sebagai basis pembandung dengan algoritma lain yang lebih sangkil.

Meskipun bukan algoritma dengan metode yang sangkil, algoritma *brute force* menjamin solusi permasalahan yang optimum. Semua permasalahan dapat diselesaikan dengan algoritma *brute force* dan sangal sulit menunjukkan permasalah yang tidak dapat diselesaikan dengan algoritma ini.

2.2 *Exhaustive Search*

Exhaustive search atau yang dikenal juga sebagai *brute force search* adalah agoritma pemncarian dengan mencoba kemungkinan solusi yang ada hingga ditemukan solusi yang optimum atau benar. Umumnya *exhaustive search* digunakan untuk memecahkan masalah kombinatorika, yaitu permasalahan yang kemungkinan solusinya merupakan permutasi, kombinasi, atau himpunan bagian dari sekumpulan objek-objek diskrit. Langkah pertama melalukan *exhaustive search* adalah melakukan enumerasi setiap kemungkinan solusi secara sistematis, setelah itu dilanjutkan dengan mengevaluasi kemungkinan solusi satu per satu. Bila pencarian berakhir, umumkan solusi terbaik atau benar. Meskipun *exhaustive search* secara teoritis menghasilkan solusi, namun waktu atau sumber saya komputasi yang dibutuhkan dalam pencarian solusinya juga sangat besar.

3 Desain dan Implementasi

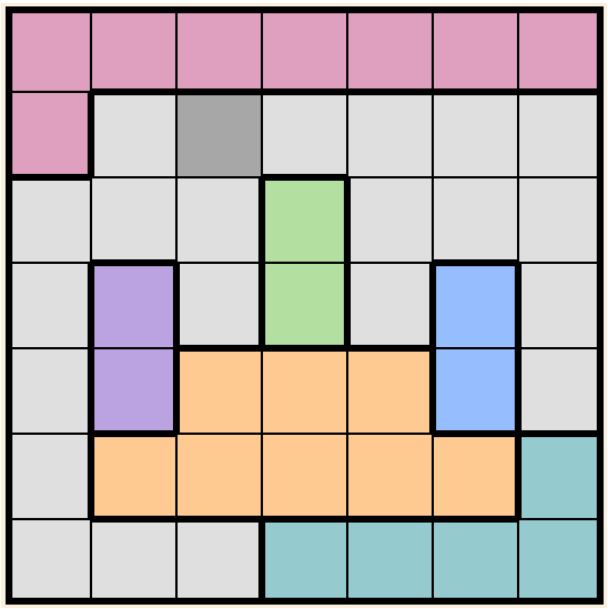
Bab ini membahas perancangan dan implementasi program, yang mencakup algoritma, struktur direktori, *source code*, serta penjelasan kelas-kelas penting yang digunakan dalam program.

3.1 Algoritma

Penulis melakukan dua pendekatan berbeda dalam menyelesaikan program *Queens Game Solver* ini. Pendekatan pertama yang digunakan adalah menggunakan algoritma naif (*brute force*) dan yang kedua adalah menggunakan algoritma *backtracking*. Sejujurnya kedua pendekatan ini dibuat karena penulis tidak memahami cakupan heuristik yang dimaksud di dalam dokumen spesifikasi, oleh karena itu penulis membuat kedua pendekatan ini agar program yang dibuat penulis masih tetap layak dan memenuhi spesifikasi serta dapat membandingkan efisiensi antara kedua algoritma tersebut.

3.1.1 Pendekatan *Brute Force*

Tidak ada persoalan yang tidak dapat diselesaikan menggunakan algoritma *brute force*, begitu pula pada gim ini. Pendekatan brute force yang digunakan oleh penulis adalah algoritma yang benar-benar lempang dan jelas. Penulis melakukan *exhaustive search* untuk mencari semua kemungkinan posisi bidak ratu dalam petak (*grid*) yang kemudian berhenti ketika ditemukan susuna atau kombinasi posisi yang valid dan benar sesuai dengan aturan permainan. Oleh karena itu kekurangan dari algoritma ini adalah daya dan waktu komputasi yang lama.



Gambar 2: Papan Permainan *Queens Game*

Sebagai contoh pada Gambar 2 yang merupakan contoh dari papan permainan *Queens*. Papan tersebut berukuran 7×7 dan terdapat 7 daerah (*regions*) unik. Kita dapat

melakukan enumerasi untuk menghitung luas masing-masing daerah dalam satuan petak. Sehingga dengan mengabaikan aturan permainan didapatkan 56320 kemungkinan berbeda dengan cara melakukan permutasi untuk masing-masing luas daerah dalam papan.

Berikut adalah langkah algoritma *brute force* yang digunakan oleh penulis:

- Mulai dari daerah 1. Pilih satu petak sembarang dan letakkan bidak Ratu.
- Pindah ke daerah 2. Pilih satu petak sembarang lagi dan letakkan bidak Ratu.
- Lakukan iterasi hingga sampai ke daerah terakhir (daerah ke-n).
- Setelah semua bidak Ratu terpasang di papan (ada sebanyak n buah Ratu), maka akan dilakukan pengecekan untuk pemeriksaaan terhadap aturan permainan.
- Cek keabsahan masing-masing posisi ratu berdasarkan aturan permainan. Jika tidak ditemukan pelanggaran maka solusi ditemukan. Namun, jika ada pelanggaran hapus semua posisi Ratu dan ulangi langkah pertama dengan kombinasi berbeda hingga solusi ditemukan.

Pendekatan ini adalah pendekatan yang benar-benar naif tanpa menggunakan heuristik apapun.

3.1.2 Pendekatan *backtracking*

Pada dasarnya algoritma *backtracking* adalah versi algoritma *brute force* yang lebih sangkil. Alih-alih mencari semua kemungkinan posisi bidak Ratu, algoritma ini memungkinkan kita untuk melakukan eliminasi kemungkinan posisi bidak Ratu berdasarkan heuristik bahwa setiap baris atau kolom tepat diisi oleh satu buah bidak Ratu. Dengan demikian dapat dilakukan backtracking bila ditemukan posisi yang tidak absah.

Berikut adalah langkah algoritma *backtracking* yang digunakan oleh penulis:

- Mulai dari daerah 1. Pilih satu petak yang aman dan letakkan bidak Ratu.
- Pindah ke daerah 2. Pilih satu petak yang aman dari "serangan" Ratu di daerah 1. Jika petaka aman maka taruh bidak Ratu dan kunci petak tersebut sementara dan lanjut ke daerah 3. Jika tidak aman maka coba petak lainnya dalam daerah yang sama.
- Bila ditemukan posisi buntu atau tidak ditemukan posisi yang valid sama sekali untuk meletakkan Ratu, maka mundur ke daerah sebelumnya dan pindahkan bidak Ratu daerah tersebut ke petak berikutnya pada daerah yang sama, setelah posisi di daerah sebelumnya berubah, maju lagi ke daerah sekarang.
- Ulangi proses maju-mundur (*backtracking*) ini sampai semua daerah terisi bidak Ratu dengan posisi aman.

Algoritma ini menggunakan metode *depth-first-search* (DFS) yang tidak memungkinkan untuk menelusuri kombinasi atau posisi yang salah.

3.2 Struktur Program

Program yang penulis buat ditulis dengan bahasa pemrograman C++ dengan menggunakan raylib sebagai sistem *Graphical User Interface* (GUI). Program ini dibangun dengan paradigma pemrograman orientasi objek yang diterapkan pada file `board.h` dan `solve.h` dan prosedural. Berikut adalah struktur *tree* program yang penulis buat:

```
‘-- Tucil1_13524038/
+-- CMakeLists.txt
+-- LICENSE
+-- README.md
+-- doc/
|   ‘-- Tucil1_13524038.pdf
+-- test/
|   +-- testcase/
|   |   ‘-- ...
|   ‘-- solution/
|       +-- output/
|       |   ‘-- ...
|       ‘-- process/
|           ‘-- ...
+-- src/
|   +-- main.cpp
|   +-- utils/
|   |   +-- file_utils.cpp
|   |   +-- file_utils.h
|   |   +-- tinyfiledialogs.c
|   |   ‘-- tinyfiledialogs.h
|   ‘-- core/
|       +-- board.cpp
|       +-- board.h
|       +-- solve.cpp
|       ‘-- solve.h
+-- lib/
|   ‘-- libraylib.a
+-- include/
|   +-- raylib.h
|   ‘-- raymath.h
‘-- bin/
```

3.3 *Source Code*

Pada bab ini penulis akan menampilkan dua kode sumber penting yang menjadi dasar (*backbone*) dari program yang penulis buat, selain itu penulis juga akan menampilkan kode sumber `main.cpp` yang menjadi titik masuk program (GUI) akan dijalankan.

3.3.1 Board

Kelas ini merepresentasikan papan permainan *Queens*. Kelas ini memiliki beberapat atribut *private* diantaranya adalah `rows` yang merepresentasikan ukuran baris papan, `cols` yang merepresentasikan ukuran kolom papan, sebuah *array of string* `orginalGrid` yang merepresentasikan satu baris papan permainan, sebuah matriks boolean `queenGrid` yang merepresentasikan petak yang berisi bidak Ratu, map `regionMap` untuk menandai koordinat masing-masing daerah, *array of char* `uniqueRegions` yang menyimpan "jenis" daerah yang unik (banyak daerah berbeda dari persoalan).

Selain itu, kelas ini juga memiliki beberapa *public method* seperti konstruktor dan selector, `load` untuk mengekstrasi input file.txt menjadi `regionMap` dan `uniqueregions`, `isValidPlacement` untuk menentukan apakah peletakkan bidak Ratu valid atau tidak, `removeQueen` untuk menghapus bidak Ratu dari suatu petak, `print` untuk menampilkan kondisi papan.

1. board.h

```
#ifndef BOARD_H
#define BOARD_H

#include <vector>
#include <string>
#include <map>
#include <iostream>

struct Point {
    int row;
    int col;
};

class Board {
    public:
    // Basically a constructor
    Board();

    // So the idea is parsing the raw text lines that we got
       from txt file into internal grid and regionMap that I
       will expalin it later :D
    void load(const std::vector<std::string>& rawLines);

    // Basically a getter / returns the total number of rows and
        columns
    int getRows() const;
    int getCols() const;

    // Returns a list of all unique region characters (['A', 'B
       ', 'C']) to know how many regions are
    std::vector<char> getUniqueRegions() const;

```

```
29      // Returns all cell coordinates belonging to a specific
            region color, it's the regionMap thing
30      const std::vector<Point>& getCellsForRegion(char regionChar)
            const;
31
32      char getRegionAt(int r, int c) const;
33
34      // Checks if placing a queen at (r, c) is valid or not
35      bool isValidPlacement(int r, int c) const;
36
37      // Modifies the board state to place a queen
38      void placeQueen(int r, int c);
39
40      // Reverses a move or unplace the queen
41      void removeQueen(int r, int c);
42
43      // Returns true if a queen is currently placed at (r, c)
44      bool hasQueen(int r, int c) const;
45
46      // Returns the current state of the grid for the print state
            (spec)
47      std::vector<std::string> getCurrentState() const;
48
49      // Prints the current state to the console (for debugging)
50      void print() const;
51
52      private:
53      int rows;
54      int cols;
55
56      // The static layout of the board (Colors 'A', 'B', et
            cetera)
57      std::vector<std::string> originalGrid;
58
59      // The dynamic state of the board (Where queens are)
60      // true = queen is here, false = empty.
61      std::vector<std::vector<bool>> queenGrid;
62
63      // regionMap['A'] = [{0,0}, {0,1}, {1,0}]
64      std::map<char, std::vector<Point>> regionMap;
65
66      // Stores the unique regions in the order they were found or
            sorted
67      std::vector<char> uniqueRegions;
68  };
69
70  // Hmm I think it's enough, by the way I also uses github online
        references for designing how to store the input, and I came
       up with this idea, using map ;D
```

```
71
72 #endif
```

Listing 1: board.h

2. board.cpp

```
1  #include "board.h"
2  #include <iostream>
3
4  using namespace std;
5
6  Board::Board() : rows(0), cols(0) {}
7
8  void Board::load(const vector<string>& rawLines) {
9      originalGrid = rawLines;
10     rows = rawLines.size();
11
12     if (rows > 0) {
13         cols = rawLines[0].size();
14     }
15     else {
16         cols = 0;
17     }
18
19     queenGrid.assign(rows, vector<bool>(cols, false));
20     regionMap.clear();
21     uniqueRegions.clear();
22
23     for (int r = 0; r < rows; r++) {
24         for (int c = 0; c < cols; c++) {
25             char regionChar = originalGrid[r][c];
26             regionMap[regionChar].push_back({r,c});
27         }
28     }
29
30     for (auto const& [key,val]: regionMap) {
31         uniqueRegions.push_back(key);
32     }
33 }
34
35 int Board::getRows() const {
36     return rows;
37 }
38 int Board::getCols() const {
39     return cols;
40 }
41
42 vector<char> Board::getUniqueRegions() const {
43     return uniqueRegions;
```

```
}

const vector<Point>& Board::getCellsForRegion(char regionChar)
    const {
    return regionMap.at(regionChar);
}

char Board::getRegionAt(int r, int c) const {
    if (r >= 0 && r < rows && c >= 0 && c < cols) {
        return originalGrid[r][c];
    }
    return ' ';
}

bool Board::isValidPlacement(int r, int c) const {
    if (r < 0 || r >= rows || c < 0 || c >= cols) return false;

    for (int i = 0; i < cols; i++) {
        if(queenGrid[r][i]) return false;
    }

    for (int i = 0; i < rows; i++) {
        if(queenGrid[i][c]) return false;
    }

    for (int ar = -1; ar <= 1; ar++) {
        for(int ac = -1; ac <= 1; ac++) {
            int nr = r + ar;
            int nc = c + ac;

            if (nr >= 0 && nr < rows && nc >= 0 && nc < cols) {
                if (queenGrid[nr][nc]) return false;
            }
        }
    }
    return true;
}

void Board::placeQueen(int r, int c) {
    if (r >= 0 && r < rows && c >= 0 && c < cols) {
        queenGrid[r][c] = true;
    }
    return;
}

void Board::removeQueen(int r, int c) {
    if (r >= 0 && r < rows && c >= 0 && c < cols) {
        queenGrid[r][c] = false;
    }
```

```
92      return;
93  }
94
95  bool Board::hasQueen(int r, int c) const {
96      if (r >= 0 && r < rows && c >= 0 && c < cols) {
97          return queenGrid[r][c];
98      }
99      return false;
100 }
101
102 vector<string> Board::getCurrentState() const {
103     vector<string> curGrid = originalGrid;
104
105     for (int r = 0; r < rows; r++) {
106         for (int c = 0; c < cols; c++) {
107             if (queenGrid[r][c]) {
108                 curGrid[r][c] = '#'; // '#' for the queen
109             }
110         }
111     }
112     return curGrid;
113 }
114
115 void Board::print() const {
116     vector<string> curGrid = getCurrentState();
117     for (const string& row: curGrid) {
118         cout << row << '\n';
119     }
120 }
```

Listing 2: board.cpp

3.3.2 Solve

Kelas ini adalah kelas yang merepresentasikan solusi dari permainan *Queens*. Kelas ini memiliki dua atribut yaitu `casesChecked` dan `frequency`. Kelas ini juga memiliki dua metod penyelesaian permainan *Queens* yaitu `solve` yaitu penyelesaian melalui pendekatan *backtracking* dan `solveBruteForce` yaitu penyelesaian melalui pendekatan naif.

1. solve.h

```
1  #ifndef SOLVE_H
2  #define SOLVE_H
3
4  #include "board.h"
5  #include <iostream>
6
7  class Solve {
8      public:
```

```
    // The smart solution I guess :), I don't know if this allowed or not but I made two solution :3
    bool solve(Board& board);

    // The very "Naive" Brute Force, I hope this work with large n-test cases
    bool solveBruteForce(Board& board);

    // For the output (live update)
    void setFrequency(long long k);

    long long getCasesChecked() const {
        return casesChecked;
    }

    private:
    bool solveRecursive(Board& board, int regionIndex);

    bool solveBruteForce(Board& board, int regionIndex, std::vector<Point>& queens);

    bool checkFullBoard(const Board& board, const std::vector<Point>& queens);

    long long casesChecked = 0;

    long long frequency = 0;
};

#endif
```

Listing 3: solve.h

2. solve.cpp

```
#include "solve.h"
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

void Solve::setFrequency(long long k) {
    frequency = k;
}

bool Solve::solve(Board& board) {
    casesChecked = 0;
    bool success = solveRecursive(board, 0);
    cout << "\nCases checked: " << casesChecked << '\n';
```

```
16      return success;
17  }
18
19  bool Solve::solveRecursive(Board& board, int regionIndex) {
20      const vector<char>& regions = board.getUniqueRegions();
21      if (regionIndex >= regions.size()) return true;
22
23      char currentRegionChar = regions[regionIndex];
24      const vector<Point>& candidates = board.getCellsForRegion(
            currentRegionChar);
25
26      for (const Point& p : candidates) {
27          casesChecked++;
28
29          if (board.isValidPlacement(p.row, p.col)) {
30              board.placeQueen(p.row, p.col);
31
32              if (frequency > 0 && casesChecked % frequency == 0)
                    {
33                  cout << "--- Step: " << casesChecked << " ---\n"
                        ;
34                  board.print();
35              }
36              if (solveRecursive(board, regionIndex + 1)) return
                    true;
37              board.removeQueen(p.row, p.col);
38          }
39      }
40      return false;
41  }
42
43  bool Solve::solveBruteForce(Board& board) {
44      casesChecked = 0;
45      vector<Point> tempQueens;
46      bool success = solveBruteForce(board, 0, tempQueens);
47      cout << "\nCases checked: " << casesChecked << '\n';
48      return success;
49  }
50
51  bool Solve::solveBruteForce(Board& board, int regionIndex,
        vector<Point>& queens) {
52      const vector<char>& regions = board.getUniqueRegions();
53
54      if (frequency > 0 && casesChecked > 0 && casesChecked %
            frequency == 0) {
55          for (const Point& p : queens) {
56              board.placeQueen(p.row, p.col);
57          }
58          cout << "--- Step: " << casesChecked << " ---\n";
```

```
59          board.print();
60          for (const Point& p : queens) {
61              board.removeQueen(p.row, p.col);
62          }
63      }
64
65      if (regionIndex >= regions.size()) {
66          casesChecked++;
67          if (checkFullBoard(board, queens)) {
68              for (const Point& p : queens) {
69                  board.placeQueen(p.row, p.col);
70              }
71              return true;
72          }
73          return false;
74      }
75
76      char currentRegionChar = regions[regionIndex];
77      const vector<Point>& candidates = board.getCellsForRegion(
            currentRegionChar);
78
79      for (const Point& p : candidates) {
80          queens.push_back(p);
81          casesChecked++;
82          if (solveBruteForce(board, regionIndex + 1, queens))
                return true;
83          queens.pop_back();
84      }
85      return false;
86  }
87
88  bool Solve::checkFullBoard(const Board& board, const vector<
        Point>& queens) {
89      for (size_t i = 0; i < queens.size(); ++i) {
90          for (size_t j = i + 1; j < queens.size(); ++j) {
91              Point p1 = queens[i];
92              Point p2 = queens[j];
93
94              if (p1.row == p2.row) return false;
95              if (p1.col == p2.col) return false;
96
97              int dr = abs(p1.row - p2.row);
98              int dc = abs(p1.col - p2.col);
99              if (dr <= 1 && dc <= 1) return false;
100         }
101     }
102     return true;
103 }
```

Listing 4: solve.cpp

3.3.3 Main.cpp

```cpp
#include "raylib.h"
#include <vector>
#include <string>
#include <iostream>
#include <fstream>
#include <chrono>

#include "core/board.h"
#include "core/solve.h"
#include "utils/file_utils.h"
#include "utils/tinyfiledialogs.h"

using namespace std;

#define PANEL_WIDTH 280
#define BG_COLOR RAYWHITE
#define PANEL_COLOR (Color){230, 230, 230, 255}

Color GetRegionColor(char region) {
    if (region == ' ') return LIGHTGRAY;
    int hash = (int)region * 11423;
    return (Color){
        (unsigned char)((hash % 100) + 155),
        (unsigned char)(((hash / 100) % 100) + 155),
        (unsigned char)(((hash / 10000) % 100) + 155),
        255
    };
}

void SolveWithRedirection(Solve& solver, Board& board, bool
   bruteForce, long long freq, string filename) {
    streambuf* originalCout = cout.rdbuf();
    ofstream file(filename);
    if (file.is_open()) {
        cout.rdbuf(file.rdbuf());
    }

    solver.setFrequency(freq);
    if (bruteForce) solver.solveBruteForce(board);
    else solver.solve(board);

    cout.rdbuf(originalCout);
    file.close();
```

```
}

bool DrawButton(Rectangle bounds, const char* text, bool active = true) {
    if (!active) {
        DrawRectangleRec(bounds, Fade(LIGHTGRAY, 0.5f));
        DrawRectangleLinesEx(bounds, 1, GRAY);
        DrawText(text, bounds.x + 10, bounds.y + 10, 20, Fade(GRAY, 0.5f));
        return false;
    }
    Vector2 mousePoint = GetMousePosition();
    bool isHovered = CheckCollisionPointRec(mousePoint, bounds);
    DrawRectangleRec(bounds, isHovered ? SKYBLUE : WHITE);
    DrawRectangleLinesEx(bounds, 2, isHovered ? BLUE : DARKGRAY);
    int textWidth = MeasureText(text, 20);
    DrawText(text, bounds.x + (bounds.width - textWidth)/2, bounds.y + 10, 20, BLACK);
    return (isHovered && IsMouseButtonReleased(MOUSE_LEFT_BUTTON));
}

int main() {
    InitWindow(1000, 700, "Queens Solver");
    SetTargetFPS(60);

    Board board;
    Solve solver;

    bool fileLoaded = false;
    bool isSolved = false;
    string currentFile = "None";
    string statusMsg = "Please Load a File";
    string lastAlgo = "";

    long long timeTaken = 0;
    long long casesChecked = 0;
    long long frequency = 1000;
    bool saveProcess = false;

    while (!WindowShouldClose()) {
        BeginDrawing();
        ClearBackground(BG_COLOR);

        DrawRectangle(0, 0, PANEL_WIDTH, 700, PANEL_COLOR);
        DrawLine(PANEL_WIDTH, 0, PANEL_WIDTH, 700, GRAY);

        int y = 20;
        DrawText("QUEEN SOLVER", 20, y, 30, DARKGRAY); y += 50;

```

```
89          if (DrawButton((Rectangle){20, (float)y, 240, 40}, "Load
               Input File...")) {
90              const char *filterPatterns[1] = { "*.txt" };
91              const char *filePath = tinyfd_openFileDialog("Select
                   Input File", "", 1, filterPatterns, "Text Files", 0);
92
93              if (filePath) {
94                  try {
95                      vector<string> lines = readInputFile(filePath);
96                      board.load(lines);
97
98                      fileLoaded = true;
99                      isSolved = false;
100                     timeTaken = 0;
101                     casesChecked = 0;
102                     string fullPath = filePath;
103                     size_t lastSlash = fullPath.find_last_of("/\\");
104                     currentFile = (lastSlash == string::npos) ?
                           fullPath : fullPath.substr(lastSlash + 1);
105                     statusMsg = "File Loaded.";
106                 } catch (...) {
107                     statusMsg = "Error: Invalid File";
108                 }
109             }
110         }
111         y += 50;
112
113         DrawText(("File: " + currentFile).c_str(), 20, y, 20,
               fileLoaded ? BLACK : RED);
114         y += 40;
115
116         DrawText("OUTPUT OPTIONS", 20, y, 20, DARKGRAY); y += 25;
117
118         DrawRectangle(20, y, 20, 20, saveProcess ? BLUE : WHITE);
119         DrawRectangleLines(20, y, 20, 20, BLACK);
120         DrawText("Save Process?", 50, y, 20, DARKGRAY);
121         if (CheckCollisionPointRec(GetMousePosition(), (Rectangle)
               {20, (float)y, 200, 20}) && IsMouseButtonReleased(
               MOUSE_LEFT_BUTTON)) {
122             saveProcess = !saveProcess;
123         }
124         y += 30;
125
126         if (saveProcess) {
127             DrawText("Freq:", 20, y+5, 20, DARKGRAY);
128             if (DrawButton((Rectangle){80, (float)y, 50, 30}, "100"))
                   frequency = 100;
129             if (DrawButton((Rectangle){140, (float)y, 50, 30}, "1000"
                   )) frequency = 1000;
```

```
            y += 40;
        } else {
            y += 40;
        }

        if (DrawButton((Rectangle){20, (float)y, 240, 45}, "Solve (Backtrack)", fileLoaded)) {
            statusMsg = "Solving...";
            EndDrawing();
            BeginDrawing();

            auto start = chrono::high_resolution_clock::now();

            if (saveProcess) {
                const char *lFilter[1] = { "*.txt" };
                const char *logPath = tinyfd_saveFileDialog("Save Process Log", ".txt", 1, lFilter, "Text File");

                if (logPath) SolveWithRedirection(solver, board, false, frequency, logPath);
                else solver.solve(board);
            } else {
                solver.solve(board);
            }

            auto stop = chrono::high_resolution_clock::now();
            timeTaken = chrono::duration_cast<chrono::milliseconds>(stop - start).count();
            casesChecked = solver.getCasesChecked();
            lastAlgo = "Backtracking";
            isSolved = true;
            statusMsg = "Solved!";
        }
        y += 55;

        if (DrawButton((Rectangle){20, (float)y, 240, 45}, "Solve (BruteForce)", fileLoaded)) {
            statusMsg = "Solving...";
            EndDrawing();
            BeginDrawing();

            auto start = chrono::high_resolution_clock::now();

            if (saveProcess) {
                const char *lFilter[1] = { "*.txt" };
                const char *logPath = tinyfd_saveFileDialog("Save Process Log", "process.txt", 1, lFilter, "Text File");
```

```
171                 if (logPath) SolveWithRedirection(solver, board, true
                        , frequency, logPath);
172                 else solver.solveBruteForce(board);
173             } else {
174                 solver.solveBruteForce(board);
175             }
176 
177             auto stop = chrono::high_resolution_clock::now();
178             timeTaken = chrono::duration_cast<chrono::milliseconds>(
                    stop - start).count();
179             casesChecked = solver.getCasesChecked();
180             lastAlgo = "Brute Force";
181             isSolved = true;
182             statusMsg = "Solved!";
183         }
184         y += 60;
185         DrawText(TextFormat("Time: %lld ms", timeTaken), 20, y, 20,
                DARKBLUE); y += 25;
186         DrawText(TextFormat("Cases: %lld", casesChecked), 20, y, 20,
                DARKPURPLE); y += 35;
187         DrawText(statusMsg.c_str(), 20, y, 20, (statusMsg.find("Error
                ") != string::npos) ? RED : DARKGREEN);
188         y += 35;
189 
190         if (isSolved) {
191             if (DrawButton((Rectangle){20, (float)y, 240, 45}, "Save
                    Final Result", true)) {
192                 const char *filterPatterns[1] = { "*.txt" };
193                 const char *savePath = tinyfd_saveFileDialog("Save
                        Solution As...", ("solution_" + currentFile).c_str
                        (), 1, filterPatterns, "Text File");
194                 if (savePath) {
195                     saveSolution(savePath, board.getCurrentState(),
                            timeTaken, casesChecked, lastAlgo);
196                     statusMsg = "File Saved!";
197                 }
198             }
199         }
200 
201         if (fileLoaded) {
202             int rows = board.getRows();
203             int cols = board.getCols();
204             int availableWidth = 1000 - PANEL_WIDTH - 40;
205             int availableHeight = 700 - 40;
206             int maxDim = max(rows, cols);
207             int cellSize = min(availableWidth, availableHeight) /
                    maxDim;
208             int startX = PANEL_WIDTH + 20 + (availableWidth - (cols *
                    cellSize)) / 2;
```

```
209             int startY = 20 + (availableHeight - (rows * cellSize)) /
                    2;
210
211             vector<string> grid = board.getCurrentState();
212
213             for (int r = 0; r < rows; r++) {
214                 for (int c = 0; c < cols; c++) {
215                     int x = startX + c * cellSize;
216                     int py = startY + r * cellSize;
217
218                     DrawRectangle(x, py, cellSize, cellSize,
                           GetRegionColor(board.getRegionAt(r, c)));
219                     DrawRectangleLines(x, py, cellSize, cellSize,
                           BLACK);
220
221                     if (grid[r][c] == '#') {
222                         DrawCircle(x + cellSize/2, py + cellSize/2,
                               cellSize/2.5f, BLACK);
223                         DrawText("Q", x + cellSize/2 - 6, py +
                               cellSize/2 - 10, cellSize/2, WHITE);
224                     } else {
225                         char s[2] = {board.getRegionAt(r, c), '\0'};
226                         DrawText(s, x + 5, py + 5, cellSize/3, Fade(
                               BLACK, 0.2f));
227                     }
228                 }
229             }
230         } else {
231             DrawText("Please Load an Input File", PANEL_WIDTH + 200,
                    350, 30, LIGHTGRAY);
232         }
233         EndDrawing();
234     }
235     CloseWindow();
236     return 0;
237 }
```

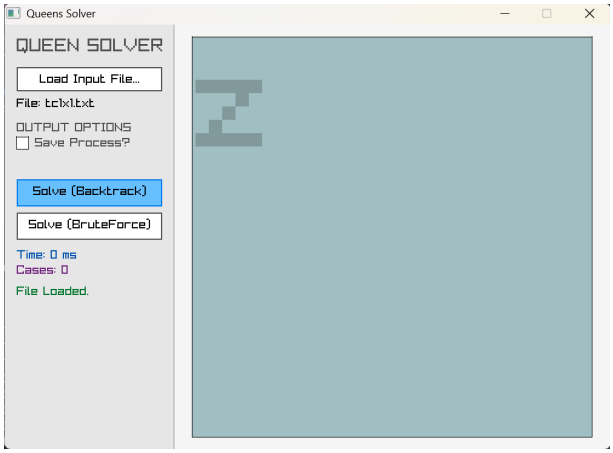
Listing 5: main.cpp

4 Eksperimen

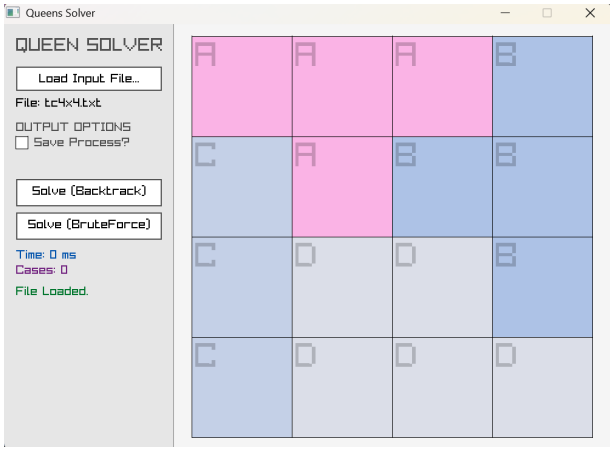
Bab ini memaparkan hasil pengujian fungsionalitas dan kinerja program yang telah dibuat. Eksperimen dilakukan dengan menguji program terhadap beberapa file tes uji yang telah disiapkan sebelumnya.

4.1 Testcases

1. tc1x1.txt

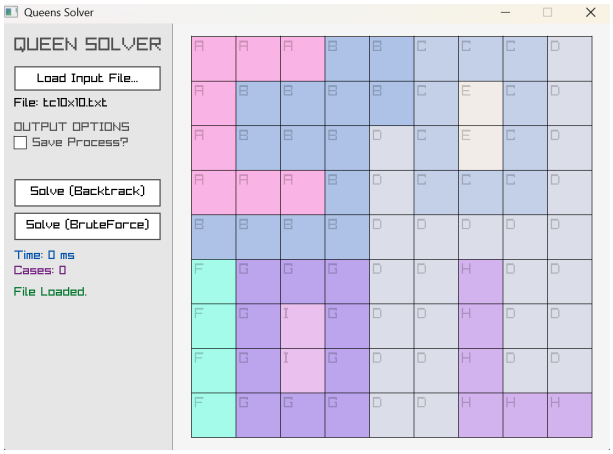


2. tc4x4.txt



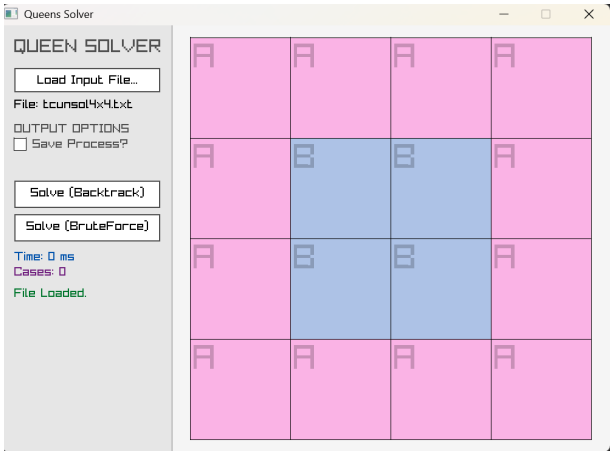
3. tc5x5.txt

4. tc10x10.txt

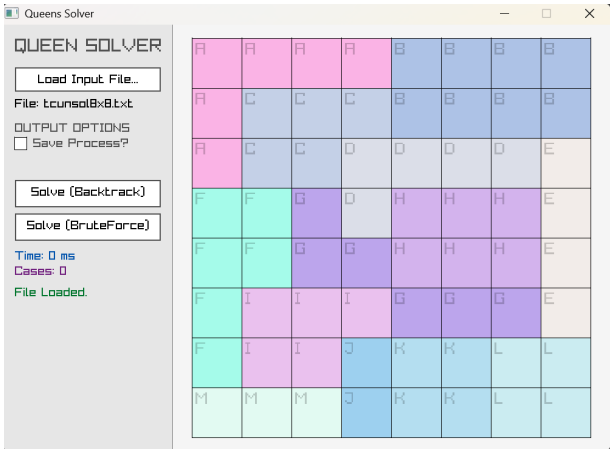


5. tcunsol3x3.txt

6. tcunsol4x4.txt

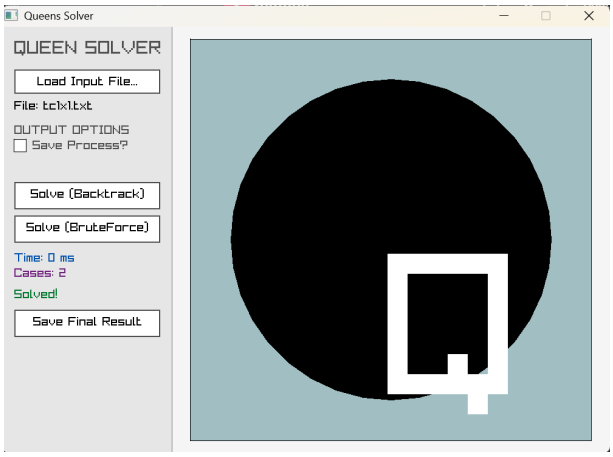


7. tcunsol8x8.txt

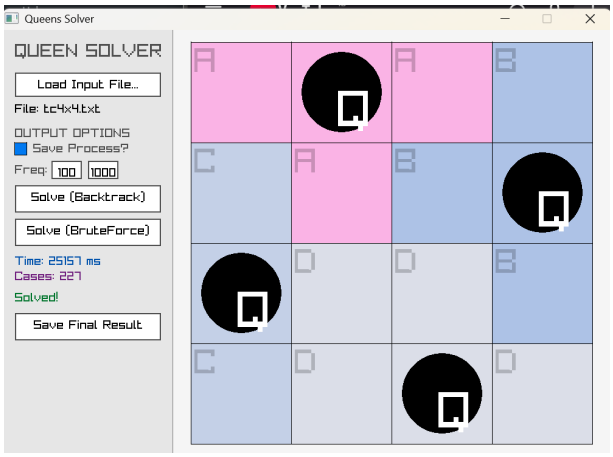


4.2 Hasil

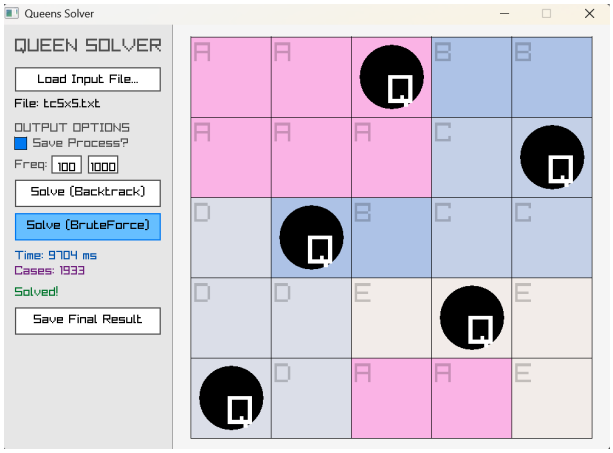
1. tc1x1.txt



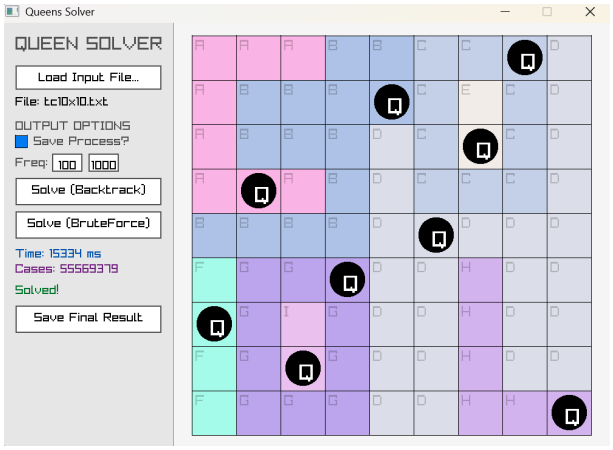
2. tc4x4.txt



3. tc5x5.txt

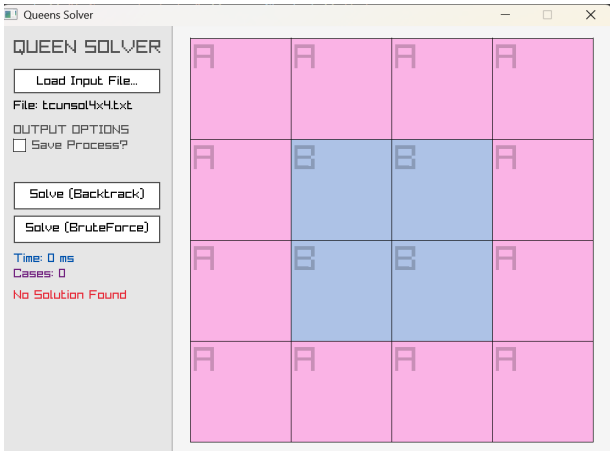


4. tc10x10.txt

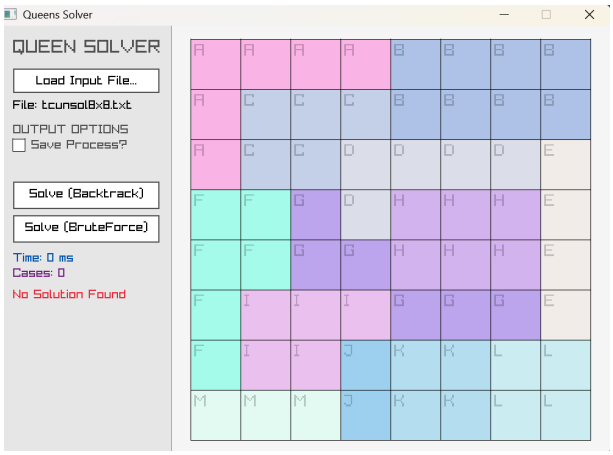


5. tcunsol3x3.txt

6. tcunsol4x4.txt



7. tcunsol8x8.txt



5 Penutup

5.1 Kesimpulan

Berdasarkan hasil implementasi serta pengujian program, dapat ditarik kesimpulan sebagai berikut:

1. Algoritma *brute force* bukanlah algoritma yang sangkil, namun hasil atau solusi yang diberikan adalah hasil paling optimum.
2. Permainan *Queens* memiliki banyak heuristik untuk mencari solusinya, salah satu heuristik yang penulis gunakan adalah *backtracking* dengan pengecekan kolom dan baris yang absah.
3. Pendekatan *backtracking* yang dilakukan penulis terbukti lebih efisien dibandingkan dengan algoritma naif (*brute force*). Hal ini ditunjukkan oleh waktu pemrosesan yang lebih singkat pada program.

5.2 Surat Cinta dari Waifu

Changli: Tucil 2 dan 3 jangan pakai C++ ya!!!

Referensi

1. github.com/raysan5/raygui
2. informatika.stei.itb.ac.id/ rinaldi.munir/Stmik/2025-2026/02-Algoritma-Brute-Force-(2026)-Bag1.pdf
3. www.raylib.com
4. github.com/native-toolkit/libtinyfiledialogs
5. stackoverflow.com/questions/8365013/reading-line-from-text-file-and-putting-the-strings-into-a-vector

Lampiran

- Tautan Repositori Github: github.com/An-Dafa/Tucil1-13524038

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	V	
2	Program berhasil di jalankan	V	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	V	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	V	
5	Program memiliki Graphical User Interface (GUI)	V	
6	Program dapat menyimpan solusi dalam bentuk file gambar		V

Tugas ini disusun sepenuhnya tanpa bantuan kecerdasan buatan (Generative AI), melainkan hasil pemikiran dan analisis mandiri.

An-Dafa Anza Avansyah