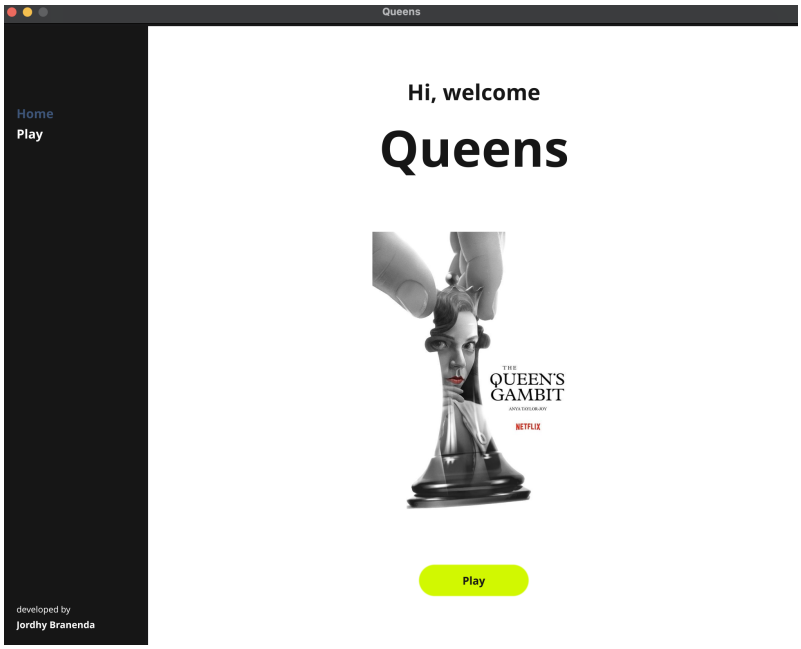
Tugas Kecil 1 IF2211 Strategi Algoritma

Semester II Tahun 2024/2025



Penyelesaian Permainan Queens dengan Algoritma Brute Force

Nama	: Made Branenda Jordhy
NIM	: 13524026
Program Studi	: Teknik Informatika

Institut Teknologi Bandung
2026

Contents

1 Pendahuluan

Queens adalah game logic yang ada pada platform LinkedIn, di mana pemain harus menempatkan queen pada board persegi berwarna dengan aturan tertentu. Each row, column, dan region warna harus berisi tepat satu queen, dan ga boleh ada dua queen yang bersebelahan (termasuk diagonal). Game ini merupakan variasi dari masalah klasik N-Queens yang sudah kita kenal, tapi dengan tambahan constraint berupa region warna yang membuat jadi lebih challenging.

Dalam tugas kecil ini, saya mengimplementasikan solusi menggunakan algoritma pure brute force tanpa optimisasi apapun. Program dibuat dengan bahasa Go dan dilengkapi dengan antarmuka grafis (GUI) menggunakan framework Fyne untuk memvisualisasikan proses pencarian solusi secara realtime. Selain itu, program juga dapat menyimpan hasil solusi dalam bentuk gambar PNG. Tujuan utama adalah memahami cara kerja algoritma brute force dalam menyelesaikan masalah pencarian kombinasi, serta menganalisis performanya pada berbagai ukuran board mulai dari 4x4 hingga 11x11.

2 Dasar Teori

2.1 Algoritma Brute Force

Algoritma brute force adalah cara paling sederhana untuk menyelesaikan masalah "coba semua kemungkinan yang ada sampai ketemu solusinya". Ide dasarnya sangat straightforward kita generate semua kombinasi yang mungkin, lalu cek satu per satu apakah kombinasi tersebut valid atau tidak. Jika valid, yaa itu jawabannya. Kalau tidak, lanjut ke kombinasi berikutnya.

Keuntungan utama dari approach brute force adalah kesederhanaannya. Kita tidak perlu mikir strategi rumit atau trik khusus, kalau kata Mr. Rila sih ini algoritma ga mikir, tinggal exhaustive search aja sampai ketemu. Plus, kalau memang ada solusinya, dijamin pasti ketemu karena kita cek semua kemungkinan. Tapi ya kelemahannya juga jelas, akan lambat jika jumlah kombinasinya banyak, dalam case game ini $N > 9$. Kompleksitas waktunya biasanya eksponensial atau faktorial, jadi tidak cocok untuk problem skala besar.

2.2 Kompleksitas untuk Permainan Queens

Dalam permainan Queens, jumlah kombinasi yang harus dicoba sangat bergantung pada ukuran board dan distribusi region-nya. Misalnya untuk board N×N dengan N region, kalau setiap region punya N kotak, maka total kombinasi yang mungkin adalah N^N. Sebagai gambaran:

- Board 4×4: sekitar 256 kombinasi (masih cepat, di bawah 1 milidetik)
- Board 7×7: sekitar 800 ribu kombinasi (beberapa milidetik)
- Board 9×9: bisa mencapai 387 juta kombinasi (beberapa detik)
- Board 11×11 ke atas: miliaran kombinasi (bisa lama banget)

3 Algoritma dan Implementasi

3.1 Cara Kerja Algoritma

Implementasi yang saya buat di sini menggunakan pendekatan Cartesian Product untuk generate semua kombinasi posisi queen yang mungkin, setelah itu baru cek satu per satu validitasnya.

3.1.1 Langkah-Langkah

1. Read dan Validate Input

Pertama-tama, program baca file input yang isinya konfigurasi board. Setiap huruf (A, B, C, dst) merepresentasikan satu region warna. Program akan cek dulu apakah board-nya berbentuk persegi dan jumlah region unique-nya sesuai dengan ukuran board. Kalau ada yang salah, langsung rejected.

2. Generate Semua Kombinasi

Ini bagian yang paling crucial karena konsep brute force ada di sini. Untuk setiap region, ambil semua kotak yang ada di dalamnya. Misalnya region A punya 5 kotak, region B punya 6 kotak, dan seterusnya. Setelah itu kita generate Cartesian Product dari semua region ini. Jadi jika ada 9 region, kita kombinasikan semua kemungkinan. Pilih 1 kotak dari region A, 1 dari region B, sampai 1 dari region I. Total kombinasinya adalah perkalian jumlah kotak di setiap region.

3. Coba Satu per Satu

Setelah punya semua kombinasi, kita coba satu per satu. Untuk setiap kombinasi:

- Cek apakah ada 2 queen di baris yang sama (tidak boleh)
- Cek apakah ada 2 queen di kolom yang sama (tidak boleh)
- Cek apakah ada 2 queen yang bersebelahan, termasuk diagonal (tidak boleh)

Kalau ketemu kombinasi yang valid, langsung stop dan return hasilnya. Kalau tidak, lanjut ke kombinasi berikutnya. Setiap 50 iterasi, program ngirim update ke GUI supaya kita bisa lihat progressnya.

4. Hasil Akhir

Kalau ketemu solusi, program return posisi semua queen beserta statistiknya (berapa iterasi dan berapa lama). Kalau sampai habis semua kombinasi belum ketemu, berarti memang tidak ada solusi untuk konfigurasi board tersebut.

3.1.2 Gambaran Kode

Secara garis besar, implementasinya seperti ini:

Listing 1: Gambaran Algoritma (Pseudocode)

```
function SolveQueens(board):
    regions = ExtractRegions(board)
    allCombinations = GenerateAllCombinations(regions)
    iterations = 0
```

```
    startTime = now()

    for each combination in allCombinations:
        iterations++

        if iterations mod 50 == 0:
            SendUpdateToGUI(combination, iterations)

        if IsValid(combination):
            return Solution(combination, iterations,
               elapsed_time)

    return NoSolution()

function IsValid(queens):
    for i = 0 to n-1:
        for j = i+1 to n-1:
            if queens[i].row == queens[j].row:
                return false
            if queens[i].col == queens[j].col:
                return false
            if abs(queens[i].row - queens[j].row) <= 1 and
               abs(queens[i].col - queens[j].col) <= 1:
                return false

    return true
```

3.2 Struktur Data

Untuk implementasinya di Go, saya pakai beberapa struct:

1. **Cell** - save posisi satu kotak di board

Listing 2: Struct Cell

```
type Cell struct {
    Row int
    Col int
}
```

2. **Region** - save satu region warna lengkap

Listing 3: Struct Region

```
type Region struct {
    Letter rune      // huruf region (A, B, C, dst, bisa tdk
       consecutive)
    Cells  []Cell    // semua kotak yang masuk region ini
}
```

3. **Board** - save seluruh board

Listing 4: Struct Board

```
type Board struct {
    Size    int          //(NxN)
    Grid    [][]rune     //matrix berisi huruf region
    Regions []Region     // list semua region yg ada
}
```

4. **SolverResult** - save result

Listing 5: Struct Solver Result

```
type SolverResult struct {
    Solution      []Cell           //posisi semua queens
    Iterations    int
    ExecutionTime time.Duration
    Found         bool
}
```

3.3 Pure Brute Force tanpa Optimisasi

Sesuai spesifikasi tugas, algoritma ini sengaja dibuat pure brute force tanpa optimisasi apapun. Artinya saya **TIDAK** pakai:

- **Backtracking**: Biasanya langsung stop/block begitu ketemu constraint yang dilanggar
- **Forward Checking**: Mengurangi pilihan sebelum dicoba
- **Constraint Propagation**: Constraint untuk pruning
- **Heuristic**

4 Hasil Pengujian

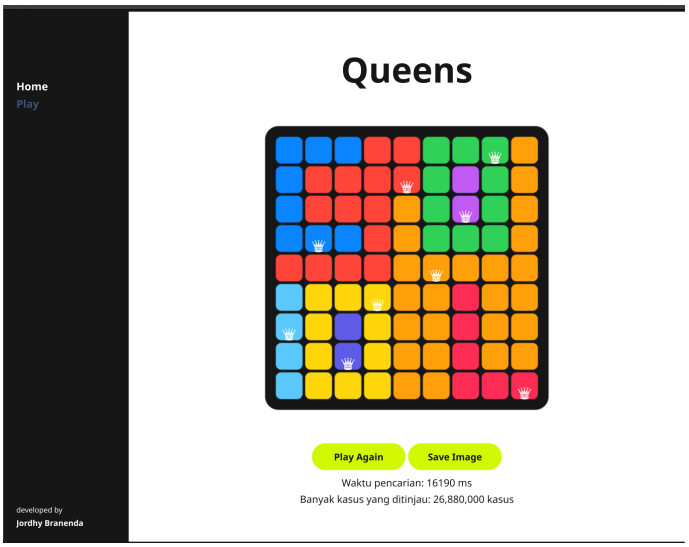
Program telah diuji dengan 10 test cases berbeda dengan ukuran dan kompleksitas yang bervariasi. Setiap test case ada dua screensho yaitu input dan outpit.

4.1 Test Case 1: 9×9 Board

File: `test1_9x9.txt`
Ukuran: 9×9
Region: 9 regions (A-I)
Input:

```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

Output:

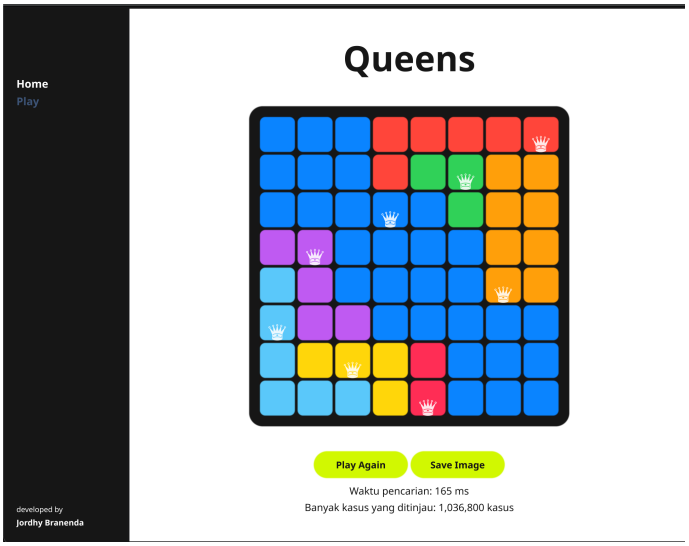


4.2 Test Case 2: 8×8 Board

File: `test2_8x8.txt`
Ukuran: 8×8
Region: 8 regions (A-H)
Input:

```
AAABBBBB
AAABCCDD
AAAAAADD
EEAAAAAD
FEAAAAAD
FEEAAAAA
FGGGHAAA
FFFGHAAA
```

Output:



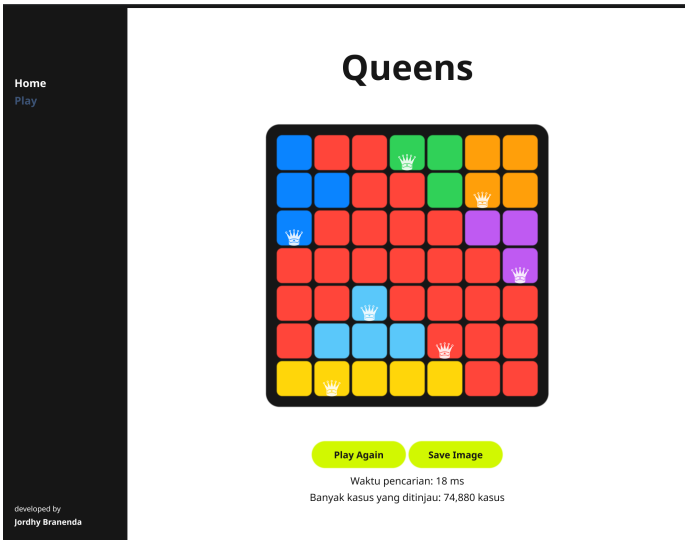
4.3 Test Case 3: 7×7 Board

File: `test3_7x7.txt`
Ukuran: 7×7
Region: 7 regions (A-G)

Input:

```
ABBCCDD
AABBCDD
ABBBBEE
BBBBBBE
BBFBBBB
BFFFBBB
GGGGGBB
```

Output:



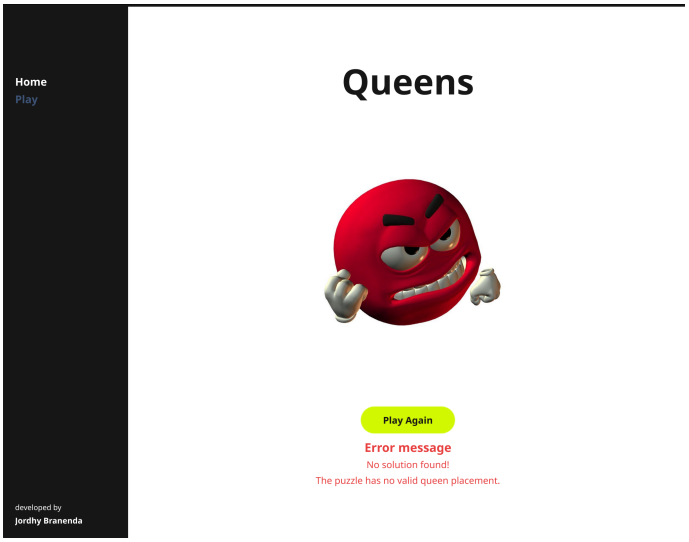
4.4 Test Case 4: 6×6 Board

File: `test4_6x6.txt`
Ukuran: 6×6
Region: 6 regions (A-F)

Input:

```
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
ABCDEF
```

Output:

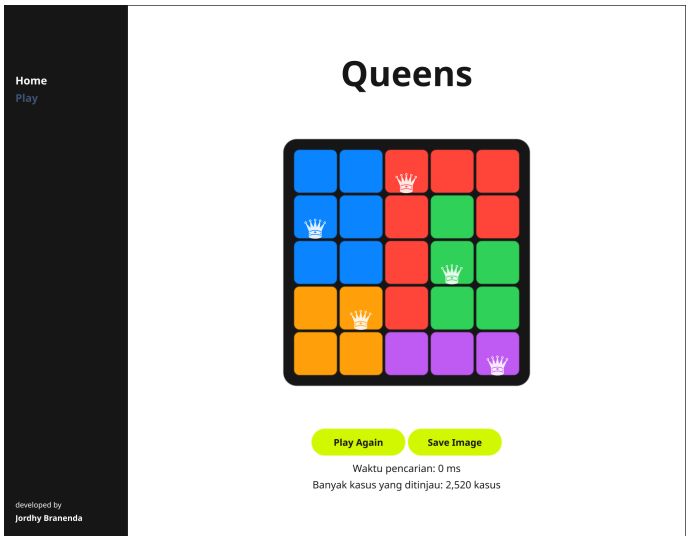


4.5 Test Case 5: 5×5 Board

File: `test5_5x5.txt`
Ukuran: 5×5
Region: 5 regions (A-E)
Input:

```
AABBB
AABCB
AABCC
DDBCC
DDEEE
```

Output:

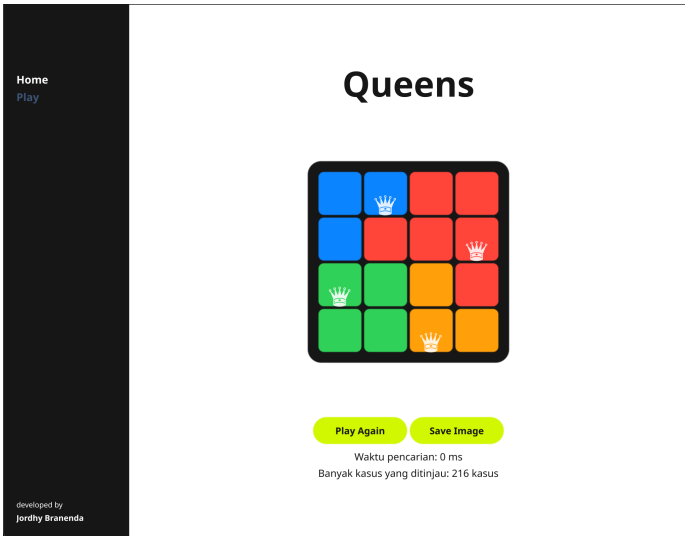


4.6 Test Case 6: 4×4 Board

File: `test6_4x4.txt`
Ukuran: 4×4
Region: 4 regions (A-D)
Input:

```
AABB
ABBB
CCDB
CCDD
```

Output:



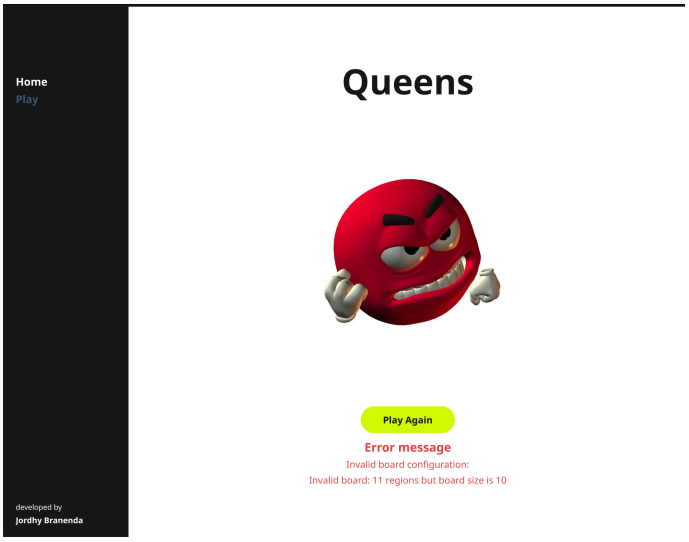
4.7 Test Case 7: 10×10 Board

File: `test7_10x10.txt`
Ukuran: 10×10
Region: 10 regions (A-K)

Input:

```
AABBCCDDEE
AABBCCDDEE
FFBBCCDDGG
FFHHCCIIGG
FFHHJJIIGG
FFHHJJIIGG
FFHHJJIIGG
KKHHJJIIGG
KKHHJJIIGG
KKHHJJIIGG
```

Output:



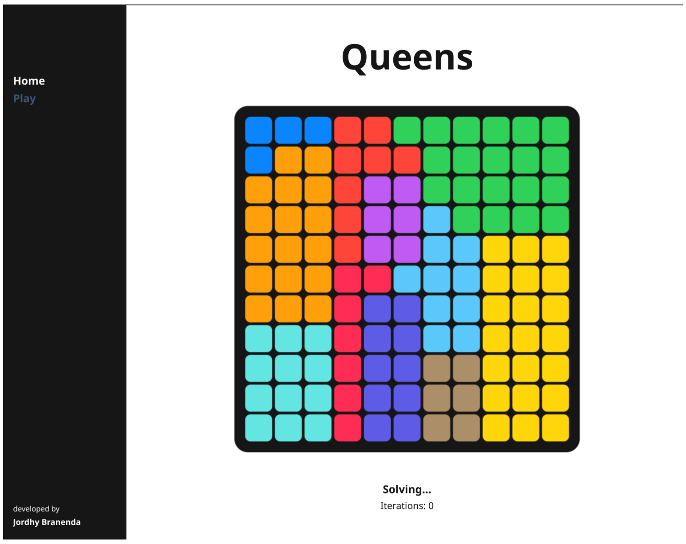
4.8 Test Case 8: 11×11 Board

File: `test8_11x11.txt`
Ukuran: 11×11
Region: 11 regions (A-K)

Input:

```
AAABBCCCCCC
ADDBBBCCCCC
DDDBEECCCCC
DDDBEEFCCCC
DDDBEEFFGGG
DDDHHFFFGGG
DDDHIIFFGGG
JJJHIIFFGGG
JJJHIIKKGGG
JJJHIIKKGGG
JJJHIIKKGGG
```

Output:



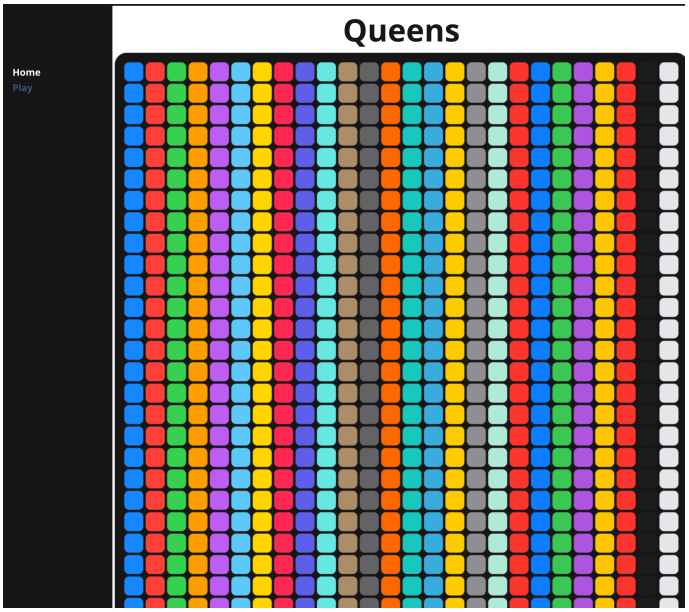
4.9 Test Case 9: 26×26 Board (Extreme)

File: `test9_26x26.txt`
Ukuran: 26×26
Region: 26 regions (A-Z)
Catatan: Test case ekstrem untuk menguji batas algoritma

Input:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Output:



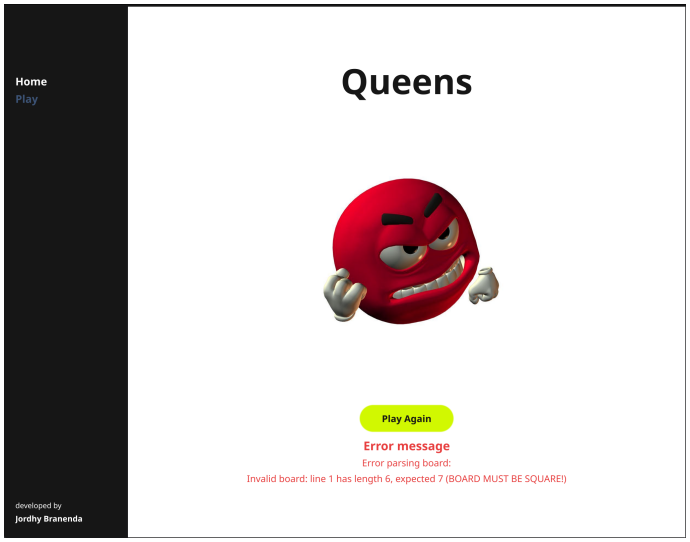
4.10 Test Case 10: 6×7 Board (Non-Square)

File: `test10_6x7.txt`
Ukuran: 6×7 (non-square)
Region: 6 regions (A-F)
Catatan: Test case untuk board non-persegi (akan ditolak oleh validator)

Input:

```
ABCDEF
ABCDEF
ABCDEF
ABCDEF
ABCDEF
ABCDEF
ABCDEF
```

Output:



4.11 Analisis Performa

Test	Ukuran	Status	Iterasi	Waktu (ms)
Test 1	9×9	success	26,880,000	16190 ms
Test 2	8×8	success	1,036,800	165 ms
Test 3	7×7	success	74,880	18 ms
Test 4	6×6	failed	-	-
Test 5	5×5	success	2,520	0 ms (too small)
Test 6	4×4	success	216	0 ms (too small)
Test 7	10×10	failed	-	-
Test 8	11×11	unknown	unknown	unknown (terlalu lama jadi author interrupt)
Test 9	26×26	unknown	unknown	unknown (board melebihi batas screen, unhandled)
Test 10	6×7	failed	-	-

Observasi:

- Waktu eksekusi meningkat drastis seiring bertambahnya ukuran board
- Board kecil (4×4 - 6×6) sangat cepat, dalam hitungan milidetik
- Board menengah (7×7 - 9×9) butuh beberapa detik
- Board besar (10×10 ke atas) bisa sangat lama atau timeout
- Distribusi region sangat berpengaruh pada jumlah kombinasi
- Live update visualization membantu tracking progress

5 Kesimpulan

Setelah mengimplementasikan dan testing game Queens Solver dengan algoritma brute force murni, ada beberapa hal yang bisa saya simpulkan. Pertama, algoritma brute force memang berhasil menyelesaikan permainan Queens. Walaupun caranya sangat straightforward (generate semua kombinasi lalu cek satu-satu), tapi ya... it works. Jika ada solusinya, pasti ketemu. Cuma sangat lambat jika size board (N) besar. Kedua, kompleksitas eksponensialnya bener-bener teras. Board 4×4 bisa selesai dalam beberapa milisecond, tapi begitu naik ke 9×9 udah butuh beberapa detik. Apalagi jika 10×10 ke atas, bisa lama banget (dalam kasus saya seperti test-8) atau bahkan timeout. Ini yang bikin brute force kurang praktis untuk problem skala besar. Ketiga, validasi input itu lumayan crucial. Jika tidak dicek dari awal, bisa-bisa program crash di tengah jalan karena input yang tidak valid. Better fail fast daripada buang-buang waktu compute.

6 Daftar Pustaka

1. Munir, Rinaldi. (2026). *Algoritma Brute Force (Bagian 1)*. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/02-Algoritma-Brute-Force-(2026)-Bag1.pdf

2. Munir, Rinaldi. (2026). *Algoritma Brute Force (Bagian 2)*. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/03-Algoritma-Brute-Force-(2026)-Bag2.pdf

3. LinkedIn. (2024). *Queens - LinkedIn Games*.
https://www.linkedin.com/games/queens/

4. The Go Programming Language. (2024). *Go Documentation*.
https://golang.org/doc/

5. Fyne.io. (2024). *Fyne Toolkit Documentation - Cross Platform GUI Framework*.
https://fyne.io/

7 Lampiran

7.1 Checklist Pemenuhan Kriteria

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

7.2 Repository Source Code

Source code lengkap beserta dokumentasi dan test cases dapat diakses di repository GitHub:

https://github.com/ethj0r/Tucil1_13524026

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

Made Branenda Jordhy
13524026