

PEMBANGUNAN PERANGKAT LUNAK DAN PENYELESAIAN PERMAINAN COLORED QUEENS

ARLO DANTE HANANVYASA—6182201010

1 Data Tugas Akhir

Pembimbing utama/tunggal: **Husnul Hakim, M.T.**

Pembimbing pendamping: -

Kode Topik : **HUH5902ACS**

Topik ini sudah dikerjakan selama : **1** semester

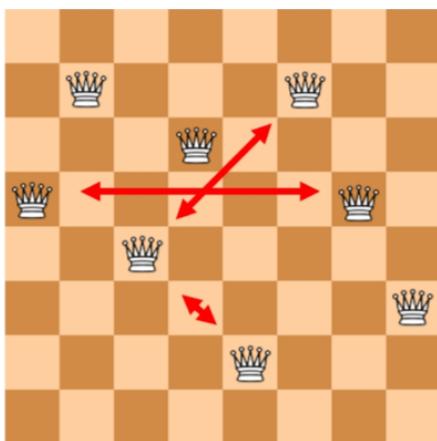
Pengambilan pertama kali topik ini pada : Semester **59** - Ganjil **25/26**

Pengambilan pertama kali topik ini di kuliah : **Tugas Akhir 1**

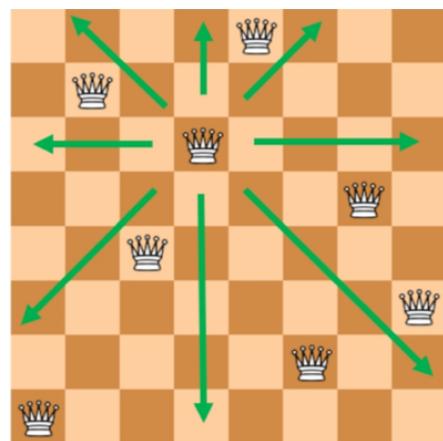
Tipe Laporan : **B** - Dokumen untuk reviewer pada presentasi dan **review Tugas Akhir 1**

2 Latar Belakang

Masalah n-queens merupakan salah satu permasalahan klasik dalam ilmu komputer yang telah dipelajari secara ekstensif sejak abad ke-19.¹ Dalam bentuk standarnya, masalah n-queens memerlukan penempatan n buah bidak menteri pada papan catur berukuran $n \times n$ sedemikian rupa sehingga tidak ada menteri yang dapat menyerang satu sama lain secara horizontal, vertikal, maupun diagonal seperti pada Gambar 1a dan 1b. Sebagai contoh, pada papan berukuran 8×8 , terdapat 92 solusi valid yang memenuhi seluruh kendala tersebut. Masalah ini tidak hanya menarik dari segi teoretis, tetapi juga memiliki aplikasi praktis dalam berbagai bidang seperti penjadwalan, alokasi sumber daya, dan desain sirkuit terpadu, sehingga menjadikannya salah satu tolok ukur penting dalam penelitian algoritma pencarian dan pemodelan berbasis kendala.



Gambar 1(a): Contoh solusi salah masalah N-Queens



Gambar 1(b): Contoh valid permasalahan N-Queens

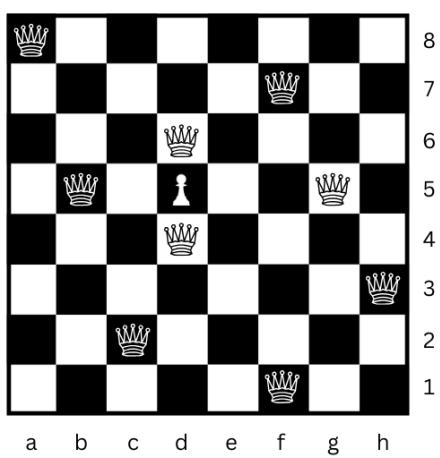
Gambar 1: Perbandingan solusi N-Queens

Sumber: https://www.researchgate.net/figure/N-Queen-problem-explanation-with-8-queens-in-a-chessboard-of-8_fig3_372415823

Seiring perkembangan penelitian, muncul berbagai variasi dari masalah n-queens tradisional yang menawarkan kompleksitas dan tantangan komputasional yang lebih tinggi. Salah satu variasi yang telah diteliti

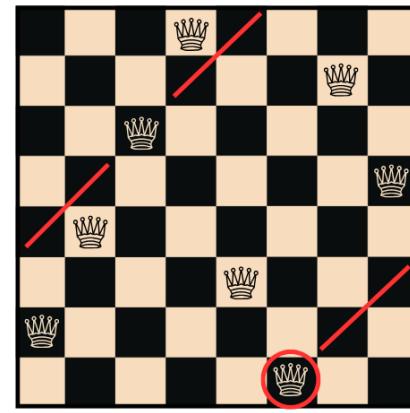
¹I.P. Gent, C. Jefferson, dan P. Nightingale, "Complexity of n-queens completion," *Journal of Artificial Intelligence Research*, vol. 59, hal. 815–848, 2017.

adalah $N+k$ Queens Problem, di mana k buah bidak pion ditempatkan sebagai penghalang sehingga memungkinkan penempatan $N+k$ bidak menteri pada papan berukuran $N \times N$.² Salah satu solusi dari $8+1$ Queens dapat dilihat pada Gambar 2a. Variasi lainnya adalah *Toroidal N-Queens*, di mana papan catur dibentuk menjadi torus dengan menghubungkan sisi-sisi yang berlawanan, sehingga menteri dapat “melingkar” dari satu sisi ke sisi lainnya.³ Penyerangan yang dapat terjadi pada permasalahan *Toroidal Queens* dapat dilihat pada Gambar 2b. Peningkatan kompleksitas pada variasi-variasi ini terlihat jelas; misalnya pada $N+k$ Queens Problem dengan $N = 8$ dan $k = 2$, jumlah kemungkinan konfigurasi yang harus dieksplorasi meningkat secara signifikan dibandingkan masalah standar karena adanya kendala tambahan berupa posisi pion yang tidak dapat dilanggar. Variasi-variasi ini menunjukkan bahwa menambahkan satu atau dua aturan baru saja dapat memperbesar ruang pencarian secara drastis dan mengubah struktur solusi masalah.



Gambar 2(a): Contoh solusi salah masalah $8+1$ Queens

Sumber: Penggambaran ulang dari
<http://www.npluskqueens.info/background.html>



Gambar 2(b): Contoh penyerangan yang dapat terjadi pada masalah *Toroidal Queens*

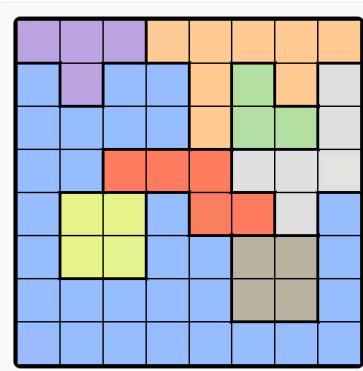
Sumber: Penggambaran ulang dari
<https://www.johndcook.com/blog/2021/08/18/queens-on-a-donut/>

Gambar 2: Beberapa varian dari permasalahan *N-Queens*

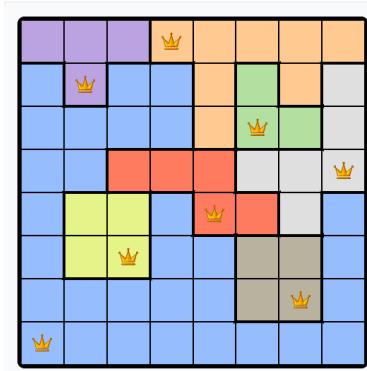
Tugas akhir ini berfokus pada varian *Colored Queens*, sebuah permasalahan yang hingga saat ini memiliki literatur akademis dan penelitian secara formal dalam publikasi ilmiah yang minim. Berbeda dengan masalah n-queens tradisional, permainan *Colored Queens* memiliki aturan yang lebih kompleks: papan permainan dibagi menjadi beberapa sektor berwarna seperti pada Gambar 3a, setiap sektor harus berisi tepat satu bidak menteri, dan tidak ada menteri yang boleh bersebelahan secara langsung, baik horizontal, vertikal, maupun diagonal. Perbedaan fundamental lainnya adalah bahwa bidak menteri pada *Colored Queens* hanya dapat menyerang secara horizontal dan vertikal, sehingga lebih dari satu bidak dapat ditempatkan pada satu garis diagonal yang sama, sehingga salah satu solusi valid untuk contoh permasalahan *Colored Queens* dapat dilihat pada Gambar 3b. Kompleksitas komputasional varian ini lebih tinggi dibandingkan permasalahan n-queens tradisional karena terdapat tiga lapis kendala yang harus dipenuhi secara simultan: kendala partisi warna (setiap warna tepat satu menteri), kendala *adjacency* (tidak boleh bersebelahan), dan kendala serangan (horizontal dan vertikal). Sebagai ilustrasi, pada papan berukuran 6×6 dengan 6 sektor warna, ruang pencarian solusi menjadi jauh lebih sempit dan bergantung pada struktur pembagian sektor, sehingga kompleksitasnya umumnya lebih tinggi dibandingkan 6-queens standar yang memiliki ruang solusi yang lebih teratur. Ketidakteraturan struktur sektor warna juga membuat heuristik dan simetri yang biasa

²R.D. Chatham, "Reflections on the $N+k$ Queens Problem," *The College Mathematics Journal*, vol. 40, no. 3, hal. 204–210, Mei 2009.

³A. García Sánchez, "The n -Queens Problem: An Activity Book," Undergraduate Research Support Scheme, Mathematics Institute, University of Warwick, 2024. Supervised by Dr. Candida Bowtell.



Gambar 3(a): Contoh kondisi awal permainan Colored Queens

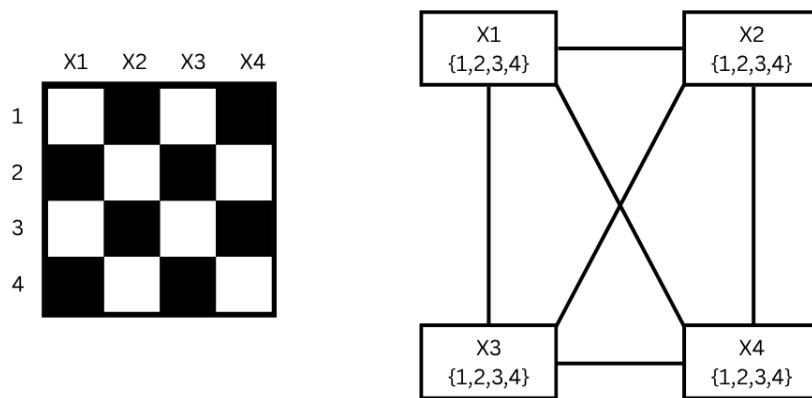


Gambar 3(b): Solusi valid permainan Colored Queens

Gambar 3: Contoh permainan *Colored Queens*

dimanfaatkan pada n-queens standar tidak lagi berlaku, sehingga *Colored Queens* merupakan permasalahan yang jauh lebih sulit.

Masalah n-queens dan seluruh variasinya, termasuk *Colored Queens*, tergolong ke dalam kategori *Constraint Satisfaction Problem* (CSP). CSP adalah jenis permasalahan yang melibatkan pencarian solusi dengan memberikan nilai pada sejumlah variabel sedemikian rupa sehingga memenuhi seperangkat batasan atau kendala tertentu. Secara formal, sebuah CSP didefinisikan oleh tiga komponen: himpunan variabel $X = \{X_1, X_2, \dots, X_n\}$, himpunan domain $D = \{D_1, D_2, \dots, D_n\}$ yang berisi nilai-nilai yang mungkin untuk setiap variabel, dan himpunan kendala C yang membatasi kombinasi nilai yang dapat diberikan pada variabel-variabel tersebut. Dalam konteks *Colored Queens*, variabel-variabelnya adalah posisi menteri untuk setiap sektor warna, domain untuk setiap variabel adalah sel-sel yang tersedia pada sektor tersebut, dan kendala-kendalanya mencakup aturan tidak bersebelahan serta tidak saling menyerang, seperti yang tertera pada Gambar 4. CSP memiliki aplikasi luas dalam penjadwalan, perencanaan, konfigurasi produk, desain jaringan, serta berbagai sistem pengambilan keputusan berbasis kendala.



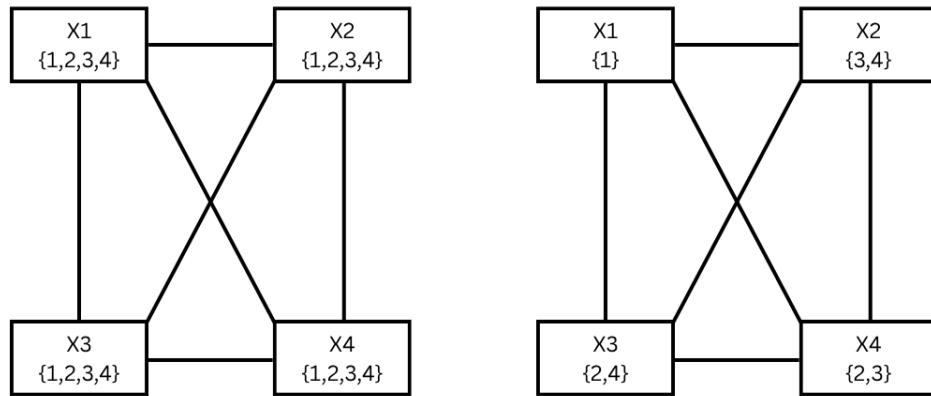
Gambar 4: Contoh permasalahan *4-Queens* yang dimodelkan sebagai CSP, dimana variabel X_1 s.d. X_4 adalah baris bidak menteri pada setiap kolomnya dengan nilai kemungkinan 1,2,3,4 dan kendala yang menghubungkan setiap variabel berupa aturan penyerangan bidak menteri pada permainan catur

Sumber: Penggambaran ulang dari <https://courses.grainger.illinois.edu/cs440/fa2021/lectures/csp1.html>

Untuk menyelesaikan permasalahan CSP seperti *Colored Queens*, salah satu pendekatan klasik yang ter-

bukti efektif adalah algoritma *backtracking*. *Backtracking* bekerja dengan membangun solusi secara bertahap melalui pencarian mendalam, di mana setiap variabel diberi nilai satu per satu sambil memeriksa konsistensi dengan kendala yang ada. Ketika algoritma menemui situasi di mana tidak ada nilai yang valid untuk variabel berikutnya (*dead end*), algoritma akan mundur (*backtrack*) ke langkah sebelumnya dan mencoba alternatif lain. Kekuatan *backtracking* terletak pada sifatnya yang sistematis dan kemampuannya untuk menjamin bahwa solusi akan ditemukan jika memang ada. Namun, pada masalah dengan ruang pencarian yang berkembang secara eksponensial seperti *Colored Queens*, *backtracking* murni dapat menjadi sangat tidak efisien karena harus mengeksplorasi sejumlah besar kemungkinan konfigurasi sebelum menemukan solusi yang valid. Oleh karena itu, diperlukan modifikasi dan teknik tambahan untuk memangkas ruang pencarian agar algoritma tetap praktis digunakan pada instansi masalah *Colored Queens* yang berukuran besar seperti papan 20×20 dan 30×30 .

Untuk meningkatkan efisiensi algoritma backtracking, tugas akhir ini mengintegrasikan teknik *Maintaining Arc Consistency* (MAC) yang mengimplementasikan algoritma *Arc Consistency 3* (AC-3). AC-3 adalah teknik propagasi kendala yang berfungsi untuk menyempitkan ruang pencarian dengan mengeliminasi nilai-nilai dalam domain yang tidak mungkin menghasilkan solusi valid berdasarkan kendala antarvariabel seperti yang dapat dilihat pada contoh Gambar 5. Algoritma ini bekerja dengan memeriksa konsistensi antara pasangan variabel dan secara iteratif menghapus nilai-nilai yang tidak memiliki pasangan yang konsisten pada variabel lain. Dengan menjalankan AC-3 setiap kali backtracking membuat *assignment* baru, banyak konfigurasi yang tidak valid dapat dieliminasi lebih awal sebelum algoritma membuang waktu mengeksploranya. Pada permasalahan seperti *Colored Queens* yang memiliki ketergantungan antarvariabel sangat kuat akibat kendala *adjacency* dan partisi warna, propagasi kendala memberikan dampak reduksi ruang solusi yang signifikan sehingga dapat mengurangi jumlah langkah *backtracking* yang diperlukan secara drastis dan memperpendek waktu yang dibutuhkan untuk mencari solusi yang valid.

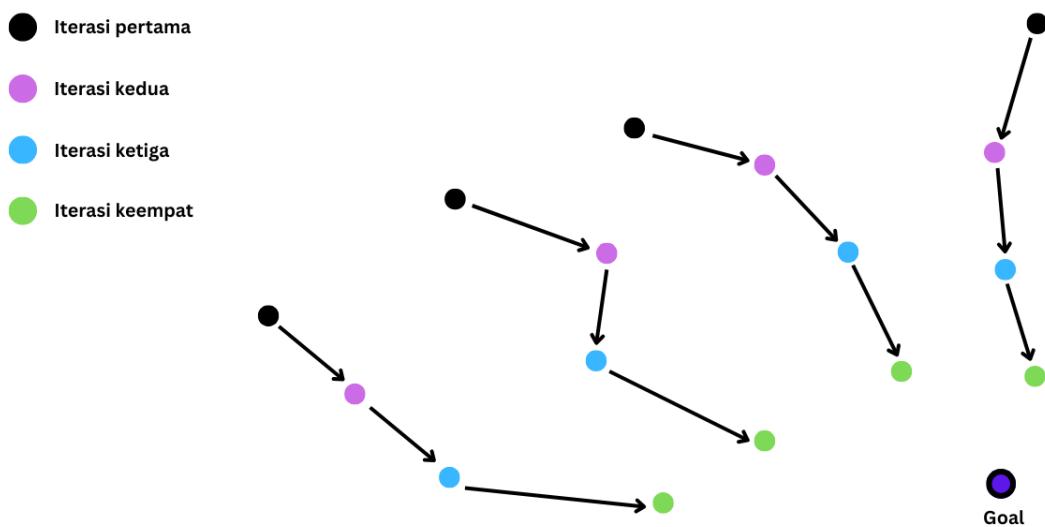


Gambar 5: Contoh ruang solusi *4-Queens* yang dimodelkan sebagai CSP, sebelum(kiri) dan sesudah(kanan) menjalankan AC-3 pada saat menaruh bidak menteri di posisi 1 di kolom X1

Sumber: Diadaptasikan dari <https://courses.grainger.illinois.edu/cs440/fa2021/lectures/csp1.htm>

Sebagai alternatif dari pendekatan deterministik, tugas akhir ini juga mengeksplorasi penggunaan *Particle Swarm Optimization* (PSO), sebuah algoritma metaheuristik yang terinspirasi dari perilaku kolektif kawanan burung atau ikan dalam mencari makanan. Metaheuristik adalah strategi pencarian tingkat tinggi yang memandu proses eksplorasi ruang solusi tanpa menjamin menemukan solusi optimal, namun seringkali dapat menemukan solusi yang cukup baik dalam waktu yang lebih singkat dibandingkan metode eksak seperti backtracking. Gambar 6 menunjukkan bahwa PSO bekerja dengan mensimulasikan sekumpulan partikel

yang bergerak dalam ruang solusi, di mana setiap partikel menyesuaikan posisinya berdasarkan pengalaman terbaiknya sendiri dan pengalaman terbaik kawanannya. Kelebihan PSO terletak pada kemampuannya untuk melakukan eksplorasi ruang solusi secara paralel dan menghindari jebakan solusi lokal dalam beberapa kasus. Namun, karena PSO dirancang untuk ruang solusi kontinu, penerapannya pada *Colored Queens* yang bersifat diskret memerlukan adaptasi khusus. Adaptasi tersebut mencakup representasi setiap partikel sebagai susunan posisi menteri untuk setiap sektor warna pada papan. Konsep velocity dalam PSO juga dimodifikasi menjadi bilangan desimal yang merepresentasikan probabilitas seberapa besar kemungkinan posisi menteri pada suatu sektor akan beralih mengikuti konfigurasi Neighbourhood Best, serta prosedur perbaikan (*repair mechanisms*) untuk menangani solusi yang melanggar kendala hard constraint seperti aturan satu menteri per warna. Adaptasi ini penting agar PSO tetap relevan dan dapat memberikan performa yang kompetitif pada masalah bersifat kombinatorial.



Gambar 6: Contoh pergerakan partikel-partikle ke arah solusi optimal dengan setiap iterasi
Sumber: Modifikasi dari https://www.researchgate.net/figure/sualization-of-the-PSO-Algorithm_fig3_334363118

Dari segi pengembangan perangkat lunak, tugas akhir ini tidak hanya berfokus pada aspek algoritmik, tetapi juga pada penyediaan antarmuka pengguna untuk memainkan dan menyelesaikan berbagai contoh permasalahan *Colored Queens* secara manual. Akan dikembangkan sebuah aplikasi berbasis web yang memungkinkan pengguna untuk berinteraksi langsung dengan permainan *Colored Queens*, memilih tingkat kesulitan berdasarkan ukuran papan, serta memungkinkan pengguna untuk menyelesaikan *puzzle* secara manual. Aplikasi ini akan dilengkapi sistem peringatan visual ketika pengguna menempatkan menteri pada posisi yang melanggar kendala, sehingga membantu pemahaman terhadap aturan permainan. Apabila kesulitan, pengguna dapat meminta bantuan berupa *hint* untuk mempermudah proses penyelesaian *puzzle*. Pengguna juga dapat memanggil *solver* untuk menyelesaikan *puzzle* secara otomatis apabila tidak bisa memecahkan masalah tersebut secara manual.

3 Rumusan Masalah

Sebelum masuk ke pembahasan lebih lanjut, berikut disajikan rumusan masalah yang menjadi dasar tugas akhir ini:

- Bagaimana cara membangun perangkat lunak permainan *Colored Queens*?
- Bagaimana membangun solusi permainan *Colored Queens* menggunakan teknik *Backtracking* dan *Particle Swarm Optimization* (PSO)?

- Bagaimana membangun solver untuk permainan colored queen yang mengimplementasikan *Backtracking* dan PSO yang dapat diintegrasikan dengan perangkat lunak yang dibangun?
- Bagaimana kinerja dari solver yang dibangun dalam mencari solusi permainan *Colored Queens*?

4 Tujuan

Selanjutnya, tujuan dari tugas akhir ini dapat dirumuskan sebagai berikut:

- Membangun perangkat lunak permainan *Colored Queens*.
- Mempelajari cara membangun solusi permainan Colored Queen menggunakan teknik *Backtracking* dan *Particle Swarm Optimization*.
- Membangun solver untuk permainan colored queen yang mengimplementasikan *Backtracking* dan PSO yang dapat diintegrasikan dengan perangkat lunak yang dibangun.
- Melakukan pengujian untuk mengukur kinerja dari solver yang dibangun dalam mencari solusi permainan *Colored Queens*

5 Detail Perkembangan Pengerjaan Tugas Akhir

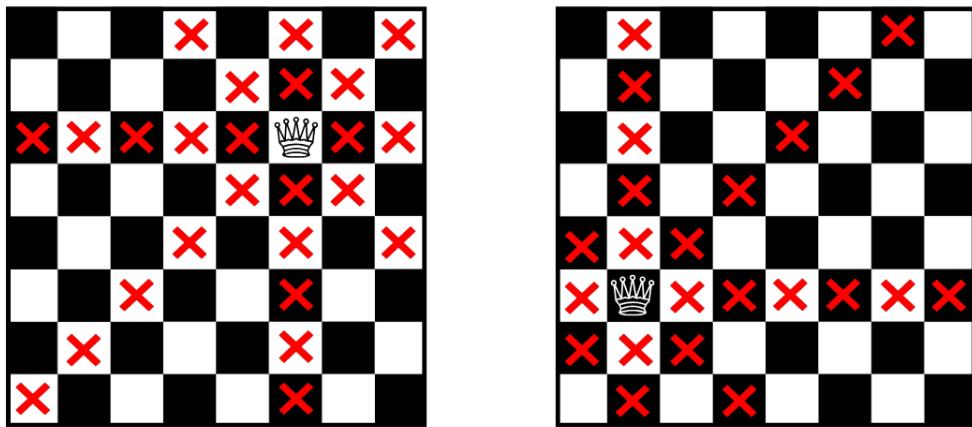
Detail bagian pekerjaan skripsi sesuai dengan rencana kerja/laporan perkembangan terakhir :

5.1 Melakukan studi literatur terkait permasalahan n-queens dan variannya.

Status : Ada sejak rencana kerja skripsi.

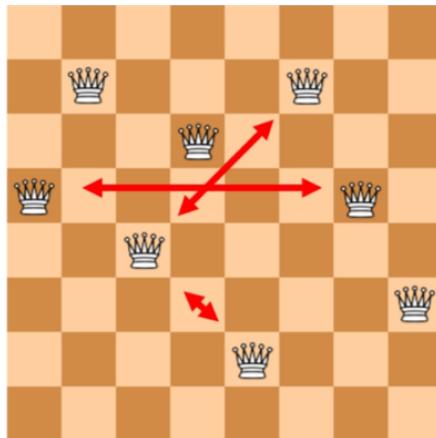
Hasil :

Permasalahan N-Queens merupakan salah satu permasalahan klasik dalam bidang ilmu komputer dan matematika diskrit, dengan sejarah panjang sejak pertama kali diperkenalkan oleh Max Bezzel pada tahun 1848. Pada formulasi dasar permasalahan ini, diberikan sebuah papan catur berukuran $n \times n$ dan tugasnya adalah menempatkan n buah bidak menteri sedemikian rupa sehingga tidak ada dua menteri yang saling menyerang. Sebagaimana dalam aturan permainan catur, sebuah menteri dapat bergerak secara bebas dalam arah horizontal, vertikal, ataupun diagonal dengan jarak tak terbatas, sehingga setiap penempatan menteri harus mempertimbangkan seluruh arah serangan tersebut seperti yang tertera pada Gambar 7.



Gambar 7: Contoh aturan penyerangan bidak menteri pada permainan catur. Silang merah menandakan kotak yang terserang oleh bidak menteri.

Dari perspektif komputasi, permasalahan ini menarik karena memiliki struktur yang sederhana namun ruang solusinya sangat besar dan tumbuh secara eksponensial. Jika setiap baris diasosiasikan dengan satu menteri, maka setiap menteri memiliki n kemungkinan kolom, menghasilkan ruang solusi awal sebesar n^n . Ketika nilai n meningkat, ruang solusi menjadi sangat luas sehingga pencarian solusi tanpa teknik penyempitan ruang menjadi tidak realistik. Sebagai contoh, untuk $n = 20$ saja terdapat 20^{20} kemungkinan konfigurasi awal, jumlah yang jauh melampaui kemampuan komputasi brute-force konvensional.



Gambar 8: Contoh solusi salah masalah N-Queens pada papan berukuran 8×8 (8-Queens)

Sumber: https://www.researchgate.net/figure/N-Queen-problem-explanation-with-8-queens-in-a-chessboard-of-8_fig3_372415823

Secara struktural, permasalahan N-Queens memiliki tiga komponen kendala utama yang harus dipenuhi secara simultan. Pertama, kendala kolom yang memastikan tidak ada dua menteri pada kolom yang sama. Kedua, kendala diagonal positif yang mencegah konflik pada diagonal dengan gradien positif. Ketiga, kendala diagonal negatif yang mencegah konflik pada diagonal dengan gradien negatif. Ketiga kendala ini membentuk sistem pembatasan yang saling berinteraksi, dan pelanggaran terhadap salah satu kendala mengakibatkan konfigurasi menjadi tidak valid. Pada contoh Gambar 8, ketiga kendala dilanggar, sehingga menjadi salah satu solusi yang tidak valid untuk permasalahan N-Queens pada papan berukuran 8×8 (8-Queens). Interaksi kompleks antarkendala inilah yang membuat permasalahan ini dapat dimodelkan menggunakan kerangka kerja formal tertentu, yang akan dibahas lebih lanjut pada bagian berikutnya.

Studi literatur juga menekankan bahwa meskipun solusi untuk N-Queens selalu ada untuk $n \geq 4$, struktur solusinya sangat beragam dan sering kali memiliki pola-pola tertentu yang dapat dimanfaatkan. Misalnya, penelitian yang dilakukan oleh Hoffman, Loessi, dan Moore⁴ yang dikutip dalam survei komprehensif Bell dan Stevens⁵ menjelaskan bahwa terdapat konstruksi aritmatika eksplisit yang dapat menghasilkan solusi dalam waktu linier $O(n)$ tanpa perlu melakukan eksplorasi ruang pencarian (*search space*) sama sekali. Namun, pendekatan konstruktif deterministik semacam ini tidak berlaku secara umum untuk seluruh varian N-Queens—terutama yang memiliki kendala ireguler seperti pada *Colored Queens*—sehingga algoritma pencarian tetap menjadi pendekatan yang paling banyak digunakan. Selain itu, jumlah solusi unik untuk N-Queens tumbuh secara eksponensial seiring bertambahnya n , dengan pola pertumbuhan yang telah menjadi objek kajian dalam teori kombinatorik.

Secara keseluruhan, pemahaman mendalam mengenai sifat dasar N-Queens, karakteristik ruang pencarinya, struktur kendala yang mendasarinya, serta berbagai teknik algoritmik yang digunakan untuk menyelesaikannya memberikan landasan konseptual kuat sebelum mempelajari varian-varian yang lebih kompleks, termasuk Colored Queens yang menjadi fokus tugas akhir ini.

5.1.1 Berbagai Varian N-Queens

Berbagai penelitian telah mengembangkan sejumlah varian dari permasalahan N-Queens untuk mengeksplorasi karakteristik baru dalam struktur constraint, ruang solusi, maupun strategi penyelesaian. Varian-varian ini umumnya memodifikasi jumlah bidak, aturan serangan, atau bahkan topologi papan sehingga menghasilkan dinamika pencarian yang berbeda secara signifikan dari permasalahan N-Queens klasik. Pada bagian ini dibahas tiga varian yang relevan: *N+k Queens*, *Toroidal Queens*, dan *Colored Queens*. Dua varian pertama merupakan topik yang telah diteliti secara formal dalam literatur akademik, sedangkan varian terakhir merupakan permasalahan non-standar yang tidak ditemukan dalam publikasi ilmiah dan menjadi fokus utama tugas akhir ini.

1. Variasi *N + k Queens*

Varian $N+k$ Queens memodifikasi jumlah bidak menteri yang harus ditempatkan pada papan berukuran $n \times n$. Jika pada N-Queens standar terdapat tepat n menteri, pada varian ini jumlah menteri menjadi $n+k$, di mana k dapat bernilai positif maupun negatif. Untuk $k > 0$, algoritma harus menemukan konfigurasi dengan lebih dari n menteri tanpa konflik horizontal, vertikal, maupun diagonal. Sebaliknya, ketika $k < 0$, tidak semua baris perlu diisi sehingga proses pencarian mencakup pemilihan subset baris yang digunakan.

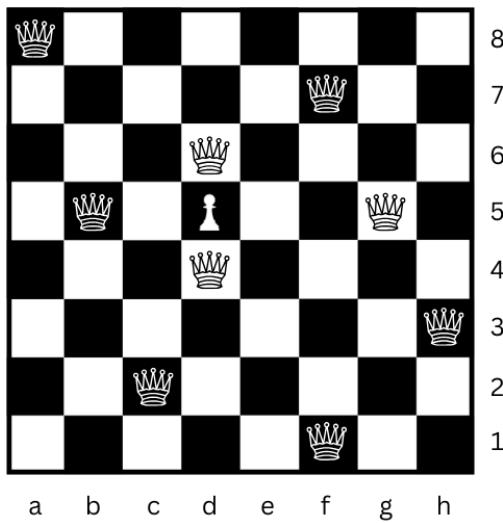
Struktur constraint pada $N+k$ Queens lebih kompleks karena hubungan antara jumlah variabel dan domainnya tidak lagi linear seperti pada N-Queens standar. Penambahan jumlah menteri secara langsung meningkatkan densitas konflik, sehingga ruang pencarian menjadi lebih padat dan heuristik penyempitan domain menjadi jauh lebih penting. Varian ini sering digunakan untuk menguji ketahanan algoritma metaheuristik dan adaptasi CSP, sebab perubahan k dapat menghasilkan tipe solusi yang berbeda, termasuk konfigurasi yang tidak mungkin muncul pada permasalahan dasar.

2. Variasi *Toroidal Queens*

Varian toroidal mengubah struktur papan menjadi sebuah torus, di mana sisi kiri dan kanan papan terhubung, demikian pula sisi atas dan bawah. Perubahan topologi ini mengubah definisi serangan menteri karena pergerakannya bersifat *wrap-around*: sebuah garis serangan tidak berhenti pada tepi papan tetapi terus berlanjut dari sisi berlawanan. Dengan demikian, konflik horizontal, vertikal, dan diagonal harus diperiksa dengan mempertimbangkan operasi modulo terhadap ukuran papan.

⁴E. J. Hoffman, J. C. Loessi, and R. C. Moore, “Constructions for the Solution of the m Queens Problem,” *Mathematics Magazine*, vol. 42, no. 2, hal. 66–72, 1969.

⁵J. Bell and B. Stevens, “A survey of known results and research areas for n -queens,” *Discrete Mathematics*, vol. 309, no. 1, hal. 1–31, 2009.



Gambar 9: Contoh solusi dari *8+1 Queens* dimana terdapat 1 bidak pion pada posisi D5 mencegah penyebrangan antara bidak menteri D4 dengan D6 dan B5 dengan G5.

Sumber: Penggambaran ulang dari <http://www.npluskqueens.info/background.html>

Untuk memahami kompleksitas varian toroidal secara bertahap, dapat dipertimbangkan konsep *semi-queen*, yaitu bidak imajiner yang dapat bergerak sepanjang baris, kolom, dan diagonal jumlah, tetapi tidak pada diagonal selisih. Analisis terhadap toroidal semi-queens menunjukkan bahwa solusi hanya ada ketika n ganjil.⁶

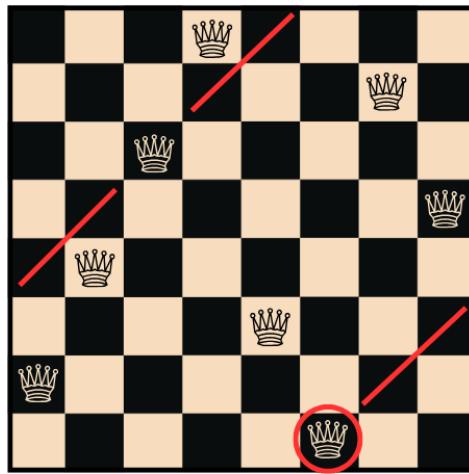
Penelitian teoretis menunjukkan bahwa tidak semua ukuran papan memiliki solusi pada varian toroidal penuh. Hal ini berbeda jauh dari N-Queens standar yang memiliki solusi untuk semua $n \geq 4$. Dari perspektif CSP, varian toroidal menghasilkan grafik constraint yang lebih padat karena hubungan antarvariabel bersifat siklik, sehingga banyak teknik heuristik tradisional harus disesuaikan agar mempertimbangkan topologi torus. Studi varian ini memberikan wawasan mendalam mengenai bagaimana perubahan struktural pada ruang masalah dapat memengaruhi pola solusi dan kompleksitas komputasional.

3. Variasi *Colored Queens*

Berbeda dengan dua varian sebelumnya, *Colored Queens* merupakan varian non-standar yang tidak ditemukan dalam literatur akademik formal. Hingga saat penulisan laporan ini, tidak terdapat publikasi ilmiah—baik jurnal maupun prosiding—yang membahas permasalahan Colored Queens secara eksplisit. Permasalahan ini muncul dalam konteks permainan logika komersial berbasis teka-teki, bukan dalam riset teoretis N-Queens. Oleh karena itu, struktur formal, representasi CSP, serta analisis algoritmik untuk varian ini harus dikonstruksi sendiri sebagai bagian dari penelitian.

Pada varian Colored Queens, papan permainan dibagi ke dalam sejumlah kawasan atau sektor yang masing-masing diberi warna berbeda. Setiap sektor harus berisi tepat satu menteri, sehingga constraint warna menjadi komponen tambahan selain constraint klasik (baris, kolom, dan—pada Colored Queens—hanya sebagian aspek diagonal). Salah satu perbedaan paling signifikan dari N-Queens tradisional adalah bahwa pada Colored Queens, menteri tidak menyerang secara diagonal, melainkan hanya secara horizontal dan vertikal. Akibatnya, lebih dari satu menteri dapat berada dalam satu diagonal tanpa menyebabkan konflik, berbeda dengan aturan standar catur. Namun, varian ini menambahkan

⁶A. García Sánchez, "The n-Queens Problem: An Activity Book," Undergraduate Research Support Scheme, Mathematics Institute, University of Warwick, 2024. Supervised by Dr. Candida Bowtell.



Gambar 10: Ilustrasi serangan menteri pada papan toroidal. Garis merah menunjukkan serangan yang melewati batas papan dan berlanjut dari sisi berlawanan (*wrap-around*). Menteri yang dilingkari merah menyerang menteri lain melalui koneksi toroidal.

Sumber: Penggambaran ulang dari <https://www.johndcook.com/blog/2021/08/18/queens-on-a-donut/>

aturan baru berupa larangan menempatkan dua menteri pada sel yang saling bersebelahan baik secara horizontal, vertikal, maupun diagonal. Hal ini menjadikan struktur constraint unik: sebagian diagonal diabaikan (untuk serangan), tetapi sebagian diagonal diperhitungkan (untuk adjacency constraint).

Dibandingkan varian-varian formal lainnya, Colored Queens memiliki dua karakteristik yang membuatnya menarik untuk dikaji sebagai CSP dan sebagai objek analisis algoritmik:

- (a) *Constraint warna* menjadikan setiap sektor bertindak sebagai variabel tingkat tinggi dengan domain yang harus memastikan pemilihan tepat satu sel dalam area tertentu.
- (b) *Aturan adjacency* menciptakan constraint jarak pendek yang meningkatkan kepadatan konflik lokal, namun tidak mempengaruhi konflik jarak jauh seperti pada N-Queens standar.

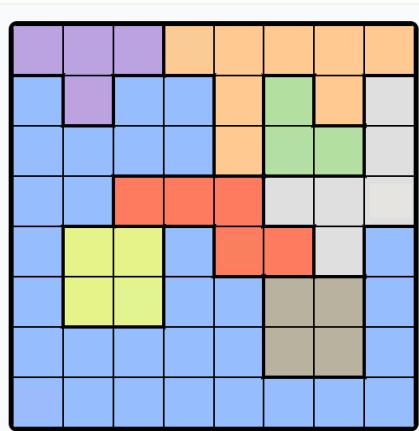
Sifat-sifat ini menjadikan Colored Queens sebagai permasalahan yang secara struktural berbeda dari varian N-Queens mana pun dalam literatur. Akibat ketiadaan referensi formal, tugas akhir ini perlu menyusun sendiri definisi matematis, model CSP, serta strategi algoritmik berbasis Backtracking dengan AC-3 dan Particle Swarm Optimization untuk memecahkan permasalahan tersebut. Dengan demikian, studi Colored Queens tidak hanya mengimplementasikan algoritma, tetapi juga berkontribusi dalam formalisasi permasalahan baru, pembangunan model CSP yang sesuai, dan analisis komparatif strategi algoritmik berbasis Backtracking dengan AC-3 dan Particle Swarm Optimization pada domain yang belum pernah dikaji secara akademik sebelumnya.

5.2 Melakukan studi literatur terkait Constraint Satisfaction Problem (CSP).

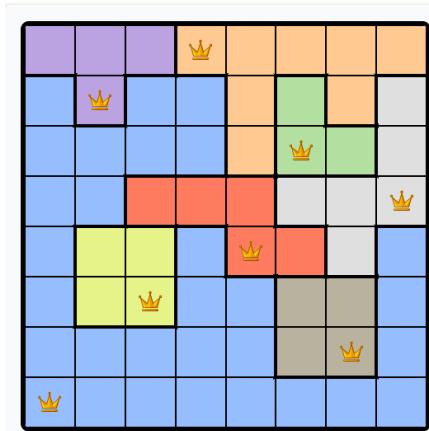
Status : Ada sejak rencana kerja skripsi.

Hasil :

Constraint Satisfaction Problem (CSP) adalah kerangka kerja umum untuk merepresentasikan dan menyelesaikan berbagai permasalahan kombinatorial. Secara formal, CSP terdiri dari tiga komponen utama: (1) sekumpulan variabel $X = \{X_1, \dots, X_n\}$, (2) untuk setiap variabel X_i , sebuah himpunan domain D_i yang berisi nilai-nilai yang mungkin dapat diberikan kepada variabel tersebut, dan (3) sekumpulan *constraint* atau



Gambar 11(a): Contoh kondisi awal permainan Colored Queens



Gambar 11(b): Solusi valid permainan Colored Queens

Gambar 11: Contoh permainan Colored Queens

batasan yang membatasi kombinasi nilai yang dapat diberikan kepada variabel-variabel secara bersamaan.⁷ Solusi dari CSP adalah pemberian nilai kepada setiap variabel dari domain masing-masing sedemikian rupa sehingga semua batasan terpenuhi.

Keunggulan pendekatan CSP terletak pada fleksibilitasnya dalam menangani berbagai jenis batasan, tidak terbatas pada batasan linear seperti dalam *integer programming*. Algoritma penyelesaian CSP umumnya menggunakan pencarian pohon yang dikombinasikan dengan teknik *backtracking* dan *consistency checking*, di mana pemberian nilai pada satu variabel dapat memicu propagasi batasan yang mengurangi domain variabel-variabel lainnya, sehingga mempercepat pencarian solusi.

Klasifikasi *constraint* dalam CSP umumnya dibedakan berdasarkan aritas atau jumlah variabel yang terlibat dalam batasan tersebut. Batasan paling sederhana adalah *unary constraint* yang membatasi nilai domain variabel tunggal. Batasan yang melibatkan dua variabel disebut *binary constraint* dan sering direpresentasikan sebagai sisi (*edge*) dalam graf kendala (*constraint graph*). Selain itu, terdapat *global constraint* yang melibatkan jumlah variabel arbitrer, seperti *Alldifferent*, yang mengharuskan semua variabel dalam himpunan tertentu memiliki nilai yang berbeda.⁸ Penggunaan *global constraint* memungkinkan algoritma penyelesaian untuk melakukan pemangkasan (*pruning*) ruang pencarian secara lebih efisien dibandingkan dengan hanya menggunakan sekumpulan *binary constraint*.

Salah satu konsep fundamental dalam reduksi ruang pencarian CSP adalah konsistensi lokal, khususnya *Arc Consistency*. Sebuah variabel X_i dikatakan *arc-consistent* terhadap variabel X_j jika untuk setiap nilai dalam domain D_i , terdapat setidaknya satu nilai dalam domain D_j yang memenuhi batasan biner antara keduanya. Algoritma seperti AC-3 bekerja dengan menegakkan properti ini secara iteratif; jika suatu nilai dalam D_i tidak memiliki dukungan (*support*) di D_j , maka nilai tersebut dihapus dari domain.⁹ Proses ini mencegah algoritma pencarian mengeksplorasi cabang pohon yang sudah pasti akan gagal, sehingga mengurangi upaya komputasi secara signifikan.

Selain propagasi batasan, efisiensi penyelesaian CSP juga sangat bergantung pada urutan instansiasi variabel dan pemilihan nilai. Heuristik *Variable Ordering* yang umum digunakan adalah *Minimum Remaining Values* (MRV), yang memprioritaskan variabel dengan domain terkecil untuk dievaluasi lebih awal guna mendeteksi kegagalan secepat mungkin (*fail-first principle*). Sebaliknya, untuk pemilihan nilai, heuristik *Least Constraining Value* (LCV) sering diterapkan dengan memilih nilai yang paling sedikit membatasi pilihan

⁷S.C. Brailsford, C.N. Potts, dan B.M. Smith, "Constraint satisfaction problems: Algorithms and applications," *European Journal of Operational Research*, vol. 119, no. 3, hal. 557-581, 1999.

⁸F. Rossi, P. van Beek, dan T. Walsh, *Handbook of Constraint Programming*, Elsevier, 2006, hal. 13-15.

⁹A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, vol. 8, no. 1, hal. 99-118, 1977.

variabel lain di masa depan, dengan tujuan menemukan solusi valid lebih cepat.¹⁰

5.3 Melakukan studi literatur terkait algoritma pencarian Backtracking.

Status : Ada sejak rencana kerja skripsi.

Hasil :

Algoritma backtracking merupakan salah satu dari tiga teknik algoritma utama untuk menyelesaikan CSP, bersama dengan *local search* dan *dynamic programming*.¹¹ Algoritma backtracking tergolong sebagai algoritma *complete* atau sistematis, yang berarti algoritma ini dijamin menemukan solusi jika solusi tersebut ada, dan dapat membuktikan bahwa suatu CSP tidak memiliki solusi. Berbeda dengan dynamic programming yang memerlukan waktu dan ruang eksponensial serta menemukan semua solusi sekaligus, algoritma backtracking bekerja pada satu solusi dalam satu waktu sehingga hanya memerlukan ruang polinomial. Prinsip dasar backtracking adalah melakukan penelusuran depth-first pada pohon pencarian, di mana pohon pencarian dihasilkan secara bertahap selama pencarian berlangsung dan merepresentasikan pilihan-pilihan alternatif yang mungkin perlu diperiksa untuk menemukan solusi.

Mekanisme fundamental yang membedakan backtracking dari pencarian *brute-force* murni adalah penetrasi fungsi pembatas (*bounding function*) untuk memangkas ruang pencarian. Dalam terminologi pohon ruang status (*state space tree*), setiap simpul yang dikunjungi dievaluasi apakah "menjanjikan" (*promising*) atau tidak. Sebuah simpul dikatakan menjanjikan jika simpul tersebut dapat mengarah pada solusi parsial yang valid dengan memenuhi semua batasan yang ada saat ini. Sebaliknya, jika sebuah simpul melanggar batasan, simpul tersebut dinyatakan *non-promising* atau "mati". Konsep pemangkasan simpul yang tidak menjanjikan ini diilustrasikan pada Gambar 12.

Ketika algoritma menemukan simpul yang tidak menjanjikan, algoritma tidak akan melanjutkan eksplorasi ke anak-anak simpul tersebut (*pruning* atau pemangkasan sub-pohon), melainkan segera melakukan langkah mundur (*backtrack*) ke simpul induk untuk mencoba alternatif nilai lain.¹² Strategi ini memungkinkan algoritma menghindari eksplorasi jutaan status yang tidak relevan, yang secara signifikan mengurangi waktu komputasi dibandingkan dengan metode pembangkitan dan pengujian (*generate-and-test*) yang naif.

Meskipun backtracking standar menjamin ditemukannya solusi yang valid, kinerjanya sangat bergantung pada ukuran domain dan urutan penugasan variabel. Oleh karena itu, dalam aplikasi CSP modern, backtracking jarang digunakan dalam bentuk murninya (*naive backtracking*). Algoritma ini umumnya ditingkatkan dengan teknik "look-ahead" seperti *Forward Checking* atau *Maintaining Arc Consistency* (MAC), yang bertujuan untuk mendeteksi jalan buntu (*dead-end*) lebih awal dengan menyaring domain variabel masa depan setiap kali sebuah keputusan dibuat.¹³

5.4 Melakukan studi literatur terkait teknik metaheuristik Particle Swarm Optimization.

Status : Ada sejak rencana kerja skripsi.

Hasil :

Particle Swarm Optimization (PSO) adalah algoritma optimasi berbasis populasi yang diperkenalkan oleh Kennedy dan Eberhart pada tahun 1995.¹⁴ Algoritma ini terinspirasi dari perilaku sosial kawanan burung (*bird flocking*) atau gerombolan ikan (*fish schooling*) dalam mencari sumber makanan. Dalam alam, ketika seekor burung menemukan jalur yang efisien menuju makanan, anggota kawanan lainnya dengan cepat

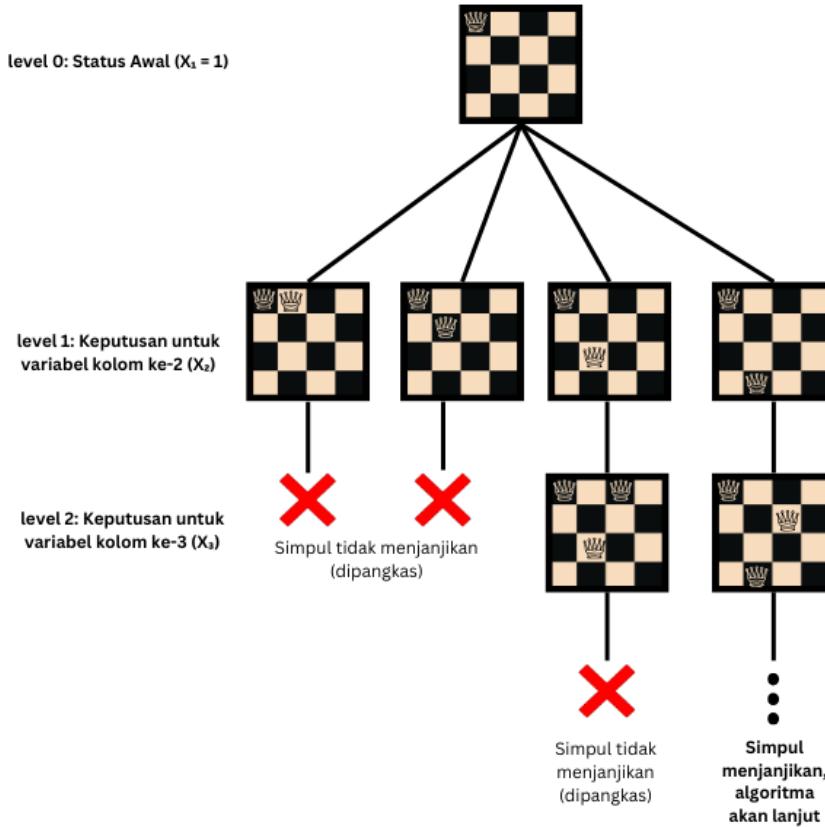
¹⁰R. M. Haralick dan G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, no. 3, hal. 263-313, 1980.

¹¹P. van Beek, "Backtracking Search Algorithms," dalam *Handbook of Constraint Programming*, F. Rossi, P. van Beek, dan T. Walsh, Ed., Elsevier, 2006, hal. 85-133.

¹²A. Levitin, *Introduction to the Design and Analysis of Algorithms*, 3rd ed., Pearson Education, 2012, hal. 423.

¹³S. Russell dan P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed., Pearson, 2020, hal. 215-217.

¹⁴J. Kennedy dan R. Eberhart, "Particle swarm optimization," *Proceedings of IEEE International Conference on Neural Networks*, Perth, Australia, pp. 1942-1948, 1995.



Gambar 12: Visualisasi pohon ruang status (*State Space Tree*) untuk masalah 4-Queens. Simpul yang melanggar batasan (bertanda silang merah) diidentifikasi sebagai *non-promising* dan dipangkas (*pruned*), sehingga algoritma tidak perlu mengeksplorasi cabang di bawahnya.

menyesuaikan jalur mereka untuk mengikutiinya, terlepas dari posisi awal mereka dalam kelompok.¹⁵

Konsep Dasar PSO

Dalam PSO, setiap solusi potensial disebut sebagai *particle* (partikel), dan kumpulan dari partikel-partikel tersebut disebut *swarm* (kawanan). Setiap partikel bernavigasi melalui ruang solusi dengan kecepatan (*velocity*) yang disesuaikan berdasarkan pengalaman mereka sendiri dan pengalaman kawan secara keseluruhan. Algoritma PSO dimulai dengan populasi acak yang disebut partikel, dan setiap partikel "terbang" melalui ruang masalah dengan mengikuti partikel-partikel yang berkinerja lebih baik.¹⁶

Setiap partikel memiliki dua atribut utama:

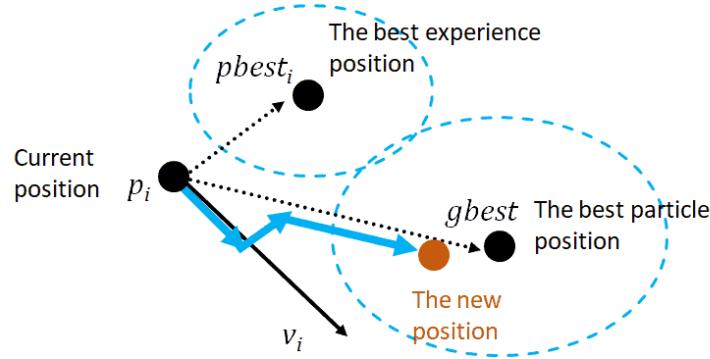
1. **Posisi** (X_i): merepresentasikan solusi kandidat dalam ruang pencarian
2. **Velocity** (V_i): menentukan arah dan jarak perpindahan partikel

Selain itu, setiap partikel melacak dua nilai penting:

- **Personal Best** (*pBest*): posisi terbaik yang pernah dicapai oleh partikel tersebut selama proses pencarian
- **Global Best** (*gBest*) atau **Neighborhood Best** (*nBest*): posisi terbaik yang dicapai oleh seluruh kawan (global) atau oleh tetangga-tetangga terdekat (lokal)

¹⁵Okto Suprianto, Mariatul Kiftiah, dan Hendra Perdana, "Penerapan Particle Swarm Optimization dalam Penyusunan Jadwal Mata Kuliah di Prodi Matematika Universitas Tanjungpura," *Bimaster: Buletin Ilmiah Matematika, Statistika dan Terapannya*, Volume 13, No 05, hal 609-618, 2024.

¹⁶Xiaohui Hu, Russell C. Eberhart, dan Yuhui Shi, "Swarm Intelligence for Permutation Optimization: A Case Study of n-Queens Problem," *Proceedings of IEEE Swarm Intelligence Symposium*, pp. 243-246, 2003.



Gambar 13: Ilustrasi perpindahan satu partikel berdaskarkan *Velocity* baru yang dipengaruhi oleh *personal best* dan *global best*

Sumber: <https://www.baeldung.com/cs/pso>

Persamaan Update PSO

Pada setiap iterasi, velocity dan posisi setiap partikel diperbarui menggunakan persamaan berikut:

$$V_i^{k+1} = \omega \cdot V_i^k + c_1 \cdot r_1 \cdot (pBest_i - X_i^k) + c_2 \cdot r_2 \cdot (gBest - X_i^k) \quad (1)$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \quad (2)$$

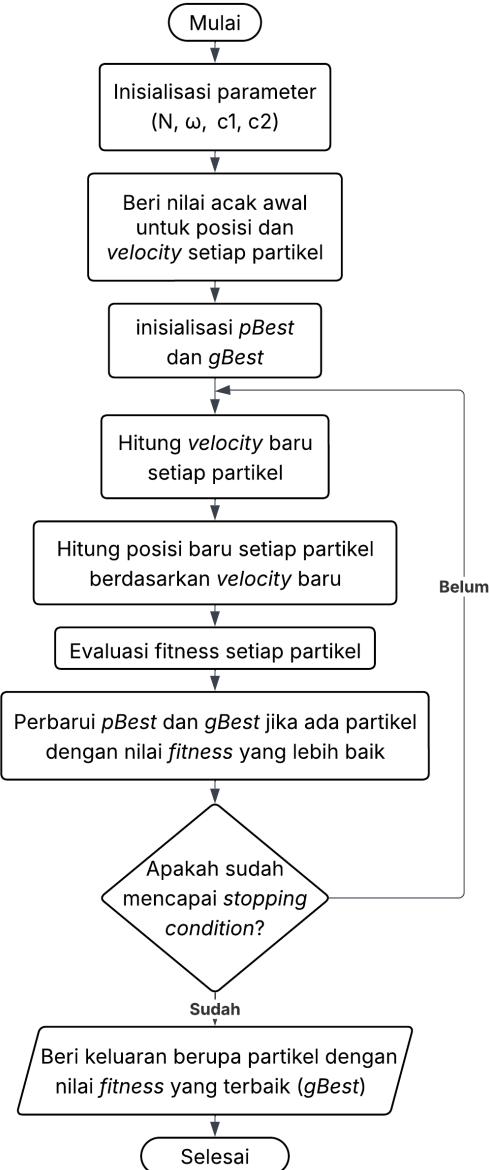
di mana:

- V_i^k adalah velocity partikel i pada iterasi ke- k
- X_i^k adalah posisi partikel i pada iterasi ke- k
- ω adalah *inertia weight* yang mengontrol momentum partikel (biasanya 0.7-0.9)
- c_1 adalah *cognitive coefficient* yang mengontrol pengaruh *personal best* (biasanya 1.5-2.0)
- c_2 adalah *social coefficient* yang mengontrol pengaruh *global best* (biasanya 1.5-2.0)
- r_1, r_2 adalah bilangan acak dalam rentang $[0,1]$

Secara konseptual, persamaan (1) menggabungkan tiga komponen pergerakan yang diilustrasikan pada Gambar 13:

1. **Komponen Inersia ($\omega \cdot V_i^k$)**: Kecenderungan partikel untuk terus bergerak ke arah yang sama dengan langkah sebelumnya (digambarkan sebagai vektor v_i).
2. **Komponen Kognitif ($c_1 \cdot r_1 \cdot (pBest_i - X_i^k)$)**: Kecenderungan partikel untuk kembali ke posisi terbaik yang pernah ia temukan sendiri (vektor putus-putus menuju $pbest_i$).
3. **Komponen Sosial ($c_2 \cdot r_2 \cdot (gBest - X_i^k)$)**: Kecenderungan partikel untuk bergerak menuju posisi terbaik yang ditemukan oleh kawan/kelompok (vektor putus-putus menuju $gbest$).

Interaksi ketiga vektor gaya inilah yang menghasilkan posisi baru (lingkaran oranye pada gambar) yang diharapkan lebih dekat dengan solusi optimal.



Gambar 14: Diagram alir PSO yang menggambarkan inisialisasi partikel, pembaruan *velocity* dan posisi, evaluasi fitness, serta mekanisme pembaruan pBest dan gBest.

Sumber: Terjemahan dan penggambaran ulang dari <https://www.elsevier.es/en-revista-journal-applied-research-technology-jart-81-articulo-a-psos-procedure-for-coordinated-S1665642313715748>

Langkah-langkah algoritma PSO (mengacu pada diagram alir 14)

1. Inisialisasi parameter

Tentukan nilai-nilai untuk ukuran populasi N , inertia ω , konstanta c_1 dan c_2 , serta kriteria penghentian (mis. jumlah iterasi maksimum atau toleransi perubahan fitness).

2. Beri nilai acak awal untuk posisi dan kecepatan setiap partikel.

Untuk setiap partikel $i = 1 \dots N$, inisialisasi X_i^0 (mis. di ruang solusi) dan V_i^0 (mis. kecil dan acak).

3. Inisialisasi pBest dan gBest.

Hitung fitness pada posisi awal tiap partikel. Set $pBest_i = X_i^0$ untuk setiap partikel. Tentukan $gBest$ sebagai posisi dengan nilai fitness terbaik di antara semua $pBest_i$.

4. Perulangan hingga memenuhi *stopping condition*:

(a) **Hitung velocity baru setiap partikel**

Untuk tiap partikel i , hitung V_i^{k+1} sesuai persamaan (1).

(b) **Hitung posisi baru setiap partikel**

Untuk tiap partikel i , perbarui posisi menggunakan persamaan (2)

(c) **Evaluasi fitness setiap partikel.**

Hitung nilai fungsi objektif pada X_i^{k+1} untuk tiap partikel.

(d) **Perbarui $pBest$ dan $gBest$ jika ada perbaikan.**

Jika $\text{fitness}(X_i^{k+1})$ lebih baik daripada $\text{fitness}(pBest_i)$, maka $pBest_i = X_i^{k+1}$. Jika ada $pBest_i$ yang lebih baik daripada $\text{fitness}(gBest)$, maka $gBest = pBest_i$.

(e) **Cek stopping condition.**

Jika kondisi penghentian tercapai, bisa dalam bentuk jumlah iterasi maksimum atau tidak ada peningkatan kualitas solusi yang signifikan, keluarkan hasil; jika belum, lanjutkan ke iterasi berikutnya dan kembali ke langkah menghitung *velocity*.

5. **Keluaran** Berikan posisi partikel dengan nilai fitness terbaik yaitu $gBest$ sebagai solusi akhir.

Keunggulan PSO

PSO memiliki beberapa keunggulan yang menjadikannya populer di kalangan peneliti dan praktisi:

1. **Kesederhanaan implementasi:** PSO memiliki struktur algoritma yang sederhana dengan parameter yang sedikit, sehingga mudah dipahami dan diimplementasikan.¹⁷ Tidak seperti Genetic Algorithm (GA), PSO tidak memerlukan operator evolusi seperti *crossover* dan *mutation*.
2. **Konvergensi cepat:** PSO cenderung konvergen lebih cepat dibandingkan algoritma evolusioner lainnya, terutama pada tahap awal pencarian. Hal ini karena informasi *global best* langsung dibagikan ke seluruh partikel dalam kawanan.
3. **Fleksibilitas:** PSO dapat disesuaikan untuk berbagai jenis masalah optimasi, baik kontinu maupun diskret, tanpa perubahan mendasar pada struktur algoritma.
4. **Keseimbangan eksplorasi-eksplotasi:** Melalui parameter ω , c_1 , dan c_2 , PSO dapat menyeimbangkan antara eksplorasi (pencarian area baru) dan eksplotasi (pencarian intensif di area yang menjanjikan).
5. **Paralelisasi mudah:** Karena setiap partikel dapat dievaluasi secara independen, PSO sangat cocok untuk implementasi paralel, yang dapat mempercepat proses komputasi secara signifikan.

Keberhasilan PSO di berbagai domain telah memicu banyak penelitian untuk memodifikasi dan meningkatkan performanya, seperti penambahan mutasi, hibridisasi dengan algoritma lain, dan adaptasi untuk masalah multiobjektif atau masalah diskret kombinatorika.¹⁸

5.5 Melakukan studi literatur terkait metode propagasi kendala *Maintaining Arc Consistency* (MAC), terutama algoritma *Arc Consistency 3* (AC-3).

Status : Ada sejak rencana kerja skripsi.

Hasil :

Maintaining Arc Consistency (MAC) adalah teknik propagasi constraint yang digunakan selama proses

¹⁷N. K. Jain, Uma Nangia, dan Jyoti Jain, "A Review of Particle Swarm Optimization," *Journal of The Institution of Engineers (India): Series B*, 2018.

¹⁸Y. Zhang, S. Wang, dan G. Ji, "A comprehensive survey on particle swarm optimization algorithm and its applications," *Mathematical Problems in Engineering*, 2015.

pencarian solusi dalam *Constraint Satisfaction Problem* (CSP). Untuk memahami MAC, pertama-tama kita perlu memahami konsep *arc* dalam CSP.

Arc dalam CSP: Dalam jaringan constraint biner yang ternormalisasi, sebuah *arc* adalah pasangan terurut (x_i, c_{ij}) yang menunjukkan bahwa variabel x_i harus konsisten dengan constraint c_{ij} yang menghubungkan x_i dengan variabel lain x_j .¹⁹ Sebuah nilai $v_i \in D(x_i)$ dikatakan konsisten dengan constraint c_{ij} jika terdapat setidaknya satu nilai $v_j \in D(x_j)$ sedemikian sehingga pasangan (v_i, v_j) memenuhi constraint c_{ij} . Nilai v_j ini disebut sebagai *support* untuk v_i pada constraint c_{ij} .

Peran MAC: MAC bekerja dengan menjaga *arc consistency* setelah setiap assignment variabel selama pencarian solusi.²⁰ Ketika sebuah variabel diberi nilai, MAC akan melakukan propagasi untuk menghilangkan nilai-nilai yang tidak konsisten dari domain variabel-variabel lain yang terhubung. Hal ini membantu mendekripsi inkonsistensi lebih awal dalam proses pencarian dan mengurangi ruang pencarian secara signifikan.

Algoritma AC-3: AC-3 adalah algoritma yang paling terkenal untuk menegakkan *arc consistency*.²¹ Algoritma ini menggunakan pendekatan berbasis antrian (*queue*) untuk memproses arc yang perlu direvisi. Berikut adalah pseudocode algoritma AC-3:

Listing 1: Algoritma AC-3

```

1  function AC3(X: set): Boolean;
2  begin
3    /* inisialisasi */
4    Q ← {(xi, c) | c ∈ C, xi ∈ X(c)};
5
6    /* propagasi */
7    while Q ≠ ∅ do
8      pilih dan hapus (xi, c) dari Q;
9      if Revise(xi, c) then
10        if D(xi) = ∅ then return false;
11        else Q ← Q ∪ {(xj, c') | c' ∈ C ∧
12                      c' ≠ c ∧ xi, xj ∈ X(c') ∧ j ≠ i};
13    return true;
14  end
15
16  function Revise(xi: variable; c: constraint): Boolean;
17  begin
18    CHANGE ← false;
19    foreach vi ∈ D(xi) do
20      if #τ ∈ c ∩ πX(c)(D) with τ[xi] = vi then
21        hapus vi dari D(xi);
22        CHANGE ← true;
23    return CHANGE;
24  end

```

Penjelasan notasi dan cara kerja:

- X : himpunan semua variabel dalam CSP
- $D(x_i)$: domain dari variabel x_i , yaitu himpunan nilai-nilai yang mungkin untuk x_i
- c : sebuah constraint dalam jaringan
- $X(c)$: skema constraint c , yaitu urutan variabel yang terlibat dalam constraint c
- Q : antrian yang berisi pasangan (x_i, c) yang perlu direvisi
- $π_{X(c)}(D)$: proyeksi domain D pada variabel-variabel dalam $X(c)$
- $τ[x_i]$: nilai yang diberikan pada variabel x_i dalam tuple $τ$

Algoritma dimulai dengan memasukkan semua pasangan (x_i, c) ke dalam antrian Q (baris 3). Kemudian, selama Q tidak kosong, algoritma mengambil sebuah pasangan (x_i, c) dan memanggil fungsi *Revise* untuk

¹⁹Bessiere, C. (2006). Constraint Propagation. *Handbook of Constraint Programming*, hal. 39.

²⁰Thorisson, H. (2017). Arc Consistency and Domain Splitting. Tersedia di: <https://www.youtube.com/watch?v=4cCS8rrYT14>

²¹Bessiere, C. (2006). Constraint Propagation. *Handbook of Constraint Programming*, hal. 38-40.

memeriksa apakah setiap nilai dalam $D(x_i)$ memiliki *support* pada constraint c (baris 6-7). Fungsi `Revise` mengembalikan `true` jika ada nilai yang dihapus dari $D(x_i)$.

Jika domain $D(x_i)$ menjadi kosong setelah revisi, algoritma mengembalikan `false` yang menandakan bahwa tidak ada solusi (baris 8). Jika ada nilai yang dihapus tetapi domain tidak kosong, maka semua arc (x_j, c') yang melibatkan x_i ditambahkan kembali ke Q karena perubahan pada $D(x_i)$ mungkin mempengaruhi konsistensi variabel lain (baris 9). Algoritma berakhir dan mengembalikan `true` ketika Q kosong, yang berarti semua arc telah konsisten.

5.6 Mengumpulkan dan menyusun berbagai skenario permasalahan Colored Queens yang akan digunakan sebagai basis pengujian algoritma serta sebagai pilihan tingkat kesulitan bagi pengguna.

Status : Ada sejak rencana kerja skripsi.

Hasil :

Sebagai bagian dari tahap pengumpulan dan penyusunan skenario permasalahan Colored Queens, saya membangun sebuah web scraper Python yang menggunakan library Selenium, seperti pada listing 3, untuk mengekstraksi beragam konfigurasi papan dari situs *Play Queens Game*.²² Data ini diperlukan sebagai dasar pengujian untuk algoritma Backtracking dan Particle Swarm Optimization (PSO), sekaligus menjadi kumpulan level yang dapat dipilih pengguna berdasarkan tingkat kesulitan. Skrip yang dibuat mengotomatisasi proses pengambilan puzzle untuk berbagai ukuran papan—mulai dari 7×7 hingga 11×11 —dengan jumlah level yang bervariasi pada setiap ukuran. Setiap puzzle diakses melalui URL tertentu, kemudian Selenium membaca atribut setiap sel (baris, kolom, dan warna) untuk membentuk representasi papan yang terstruktur. Inti dari *web scraper* ini, yang dapat dilihat pada potongan kode listing 4, adalah pengambilan data sel papan permainan dengan menargetkan elemen HTML `<div>` yang memiliki atribut kustom `data-row` dan `data-col`, lalu mengekstraksi koordinat serta nilai warna RGB dari properti CSS `background-color`.

Listing 2 merupakan cuplikan dari hasil ekstraksi disimpan dalam bentuk berkas JSON yang menyimpan posisi baris, kolom, serta warna dari setiap kotak di dalam papan, agar mudah diproses kembali oleh *solver* maupun antarmuka pengguna. Pendekatan ini memastikan bahwa dataset berisi skenario yang beragam, valid, dan sesuai dengan mekanisme permainan asli. Dengan adanya kumpulan skenario ini, pengujian algoritma dapat dilakukan pada berbagai tingkat kesulitan, dan pengguna aplikasi dapat memilih puzzle berdasarkan ukuran papan.

Selain puzzle yang diambil dari situs daring tersebut, dua papan vaild tambahan berukuran 20×20 dan 30×30 telah dibuat untuk pengujian algoritma yang akan diimplementasikan pada tugas akhir ini dan dapat dilihat pada listing 5 dan listing 6. Pembuatan papan-papan baru ini dimulai dengan sebuah solusi valid permasalahan N-Queens lalu menambahkan sektor-sektor warna sedemikian rupa agar hanya terdapat satu bidak menteri di dalam satu sektor warna.

Listing 2: Cuplikan file JSON

```

1  [
2    {
3      "row": 0, // posisi baris
4      "col": 0, // posisi kolom
5      "color": "rgba(253, 224, 71, 1)" // warna
6    },
7  ]

```

Listing 3: Keseluruhan Scraper menggunakan Python

```

1  from selenium import webdriver
2  from selenium.webdriver.common.by import By
3  import chromedriver_autoinstaller
4  import json
5  import time

```

²²Situs daring dapat diakses melalui <https://www.playqueengame.com/>

```

6 import os
7
8 # Auto-install ChromeDriver
9 chromedriver_autoinstaller.install()
10
11 # Base URL
12 BASE_URL = "https://www.playqueensgame.com/puzzles/{size}x{size}/{level}"
13
14 # Define board sizes and number of levels
15 board_levels = {
16     7: 50,
17     8: 130,
18     9: 110,
19     10: 60,
20     11: 50
21 }
22
23 # Create output folder
24 os.makedirs("boards", exist_ok=True)
25
26 # Launch browser
27 driver = webdriver.Chrome()
28
29 for size, max_level in board_levels.items():
30     for level in range(1, max_level + 1):
31         url = BASE_URL.format(size=size, level=level)
32         driver.get(url)
33         time.sleep(2) # allow page to load
34
35     # Extract cells
36     cells = driver.find_elements(By.CSS_SELECTOR, "div[data-row][data-col]")
37     board = []
38     for cell in cells:
39         row = int(cell.get_attribute("data-row"))
40         col = int(cell.get_attribute("data-col"))
41         color = cell.value_of_css_property("background-color")
42         board.append({
43             "row": row,
44             "col": col,
45             "color": color
46         })
47
48     # Sort for consistency
49     board.sort(key=lambda c: (c["row"], c["col"]))
50
51     # Save JSON
52     filename = f"boards/{size}x{size}-level{level}.json"
53     with open(filename, "w") as f:
54         json.dump(board, f, indent=2)
55
56     print(f"Saved_{filename}")
57
58 driver.quit()
59 print("All_boards_extracted!")

```

Listing 4: Cuplikan fungsi inti program Python

```

1 for cell in cells: # untuk setiap cell/kotak dalam papan:
2     row = int(cell.get_attribute("data-row")) # ambil posisi baris
3     col = int(cell.get_attribute("data-col")) # ambil posisi kolom
4     color = cell.value_of_css_property("background-color") # ambil warna kotak tersebut
5     # menambahkan file JSON sesuai format
6     board.append({
7         "row": row,
8         "col": col,
9         "color": color
10    })

```

Listing 5: Papan 20×20 valid (setiap warna dilambangkan sebagai sebuah karakter yang dimulai dari kode ASCII 'A')

```

1 J J J J J J J G G G G G E E E E I I I I
2 J J J J J J J G G G G E E E E I I I I
3 J J J J J J J G G G G E E E E E I I I I
4 D D D J J J J G G E E E E E R R R I I I
5 D D D D D D D D G H E E R R R R R I I I
6 D D D D D D D U U U H H H P P R R R O I I
7 D D D D D U U U U H H H H P P P P P O O I
8 D D D D U U U U U H H H H H P P O O O O O
9 D D D D S U U U U U H H H H P P O O O O O
10 M D S S S U U U U U H H H P K K K O O O
11 M M S S S S S U U U F H H H K K K K K K K
12 M S S S S S S S F F F L H K K K K K K K K
13 M M M S S S S F F F L L L K K K K K K K K
14 M M M S S T F F F L L L L L K K K K K C C
15 M B B B T T T L L L L L L L K K C C C C
16 B B B B T T T T L L L L L A A C C C C C
17 B B B B T T T T T N L L L L A A A C C C C
18 B B B B T T T T N N N N N A A A A C C C C

```

19 B B B T T T N N N N N N A A A A A C C C
20 B B B T T T N N N N N N A A A A A A C C C

Listing 6: Papan 30×30 valid (setiap warna dilambangkan sebagai sebuah karakter yang dimulai dari kode ASCII 'A')

1 I I I I I I I ^ ^ ^ ^ F F F W W W W W W W V V V V V V V V
2 I I I I I I I ^ ^ ^ ^ F F F F W W W W V V V V V V V V V V V V
3 I I I I - - - ^ ^ ^ ^ F F F F F W W V V V V V V V V V V V V
4 I I I I - - - ^ ^ ^ ^ F F F F F W W V V V V V V V V V V V V
5 I I I I - - - ^ ^ ^ ^ F F 0 0 0 0 V V V V V V V V V Z Z
6 I I - - - - - ^ ^ ^ ^ 0 0 0 0 0 0 0 0 V V V V V V V Z Z Z Z
7 I - - - - - ^ ^ D [[0 0 C C C C C V V V Z Z Z Z Z Z
8 I - - - - - ^ ^ D [[C C C C C C V R Z Z Z Z Z Z
9 - - - - - D D D [[C C C C C C R R R Z Z Z Z Z Z
10 \ \ G G G G G D D D [[C C C C U R R R R Z Z Z Z Z Z
11 \ \ G G G G G G D D D [[C C C U U R R R R Z Z Z Z Z Z
12 \ \ G G G G G G D D D [[C C U U U R R R R P Z Z Z Z
13 \ \ \ \ G G G G G D D [[U U U U U R R R R P P Z Z Z
14 \ \ \ \ \ G G G G G D D [[U U U U U U R R R P P P Z Z Z
15 \ \ \ \ \ \ G G G G D D [[U U U U U U R R P P P P P P
16 \ \ \ \ \ \ G G G B D D [[U U U U U U R R P P P P P P
17 \ \ \ \ \ \ B B B T T T U Y Y Y U Y S S L L L J J J J
18 M M \ \ \ \ B B B B T T T Y Y Y Y Y S S S S L L L J J J J
19 M M M M M M B B B B T T T Y Y Y Y Y S S S S L L L J J J J
20 M M M M M M B B B B T T T Y Y Y Y Y S S S S L L L J J H H
21 M M M M B B B B B B T T T Y Y Y Y Y S S S S L L L J H H H
22 M M X X X B B B B B T T T T Y Y Y Y Y Y A A L H H H H
23 M M X X X X B B B B T T T T T Y Y Y Y Y Y A A A H H H H
24 X X X X X X B B B T T T T N N N Y Y Y A A A H H H H
25 X X X X X X B B T T T T N N N N Y A A A A A A H H H H
26 X X X X X X X J B T T T T N N N N N A A A A A A A K K K
27 X X X E E J J J T T N N N N N N N N A A A A A A K K K
28 E E E E E E J J J J N N N N N N N N N A A A A A A K K K
29 E E E E E E J J J J J N N N N N N N N N N A A A A A A K K K
30 E E E E E J J J J J J N N N N N N N N N A A A A A A K K K

5.7 Melakukan pemodelan masalah Colored Queens ke dalam bentuk CSP agar dapat diproses oleh algoritma pencarian.

Status : Ada sejak rencana kerja skripsi.

Hasil :

Pemodelan permainan Colored Queens sebagai *Constraint Satisfaction Problem* (CSP) dilakukan untuk memastikan bahwa aturan permainan dapat diproses secara konsisten oleh algoritma Backtracking maupun Particle Swarm Optimization (PSO). Pemodelan ini dilakukan dengan mengidentifikasi tiga komponen utama dalam CSP, yaitu variabel, domain atau ruang solusi, dan kendala sebagai berikut:

1. Variabel

Setiap warna pada papan diperlakukan sebagai sebuah variabel. Masing-masing variabel harus ditempatkan tepat satu menteri pada salah satu sel yang termasuk ke dalam sektor warna tersebut.

2. Ruang Solusi

Ruang solusi untuk setiap variabel merupakan himpunan seluruh sel yang memiliki warna yang sesuai dengan variabel tersebut. Dengan demikian, setiap variabel memiliki domain yang berbeda tergantung pada posisi warna di papan.

3. Kendala (Constraints)

Aturan permainan diterjemahkan menjadi sejumlah kendala yang harus dipenuhi secara bersamaan:

- Tidak boleh ada dua menteri yang bersebelahan secara horizontal, vertikal, maupun diagonal.
 - Setiap warna hanya boleh berisi satu menteri.
 - Serangan diagonal tidak dihitung, sehingga dua menteri dapat berada pada garis diagonal yang sama selama tidak bersebelahan.
 - Serangan horizontal dan vertikal tetap berlaku, sehingga dua menteri tidak boleh berada pada baris atau kolom yang sama.

Contoh sederhana:

Misalkan papan 4x4 dengan 4 warna berbeda (R, B, G, Y) dan setiap warna membentuk sektor yang tersebar di papan seperti pada ⁷

Listing 7: papan 4x4 dengan 4 warna berbeda (R, B, G, Y)

1	R	B	B	B
2	R	R	B	Y
3	G	Y	Y	Y
4	G	G	G	Y

Papan *Colored Queens* ⁷ dapat direpresentasikan sebagai CSP dengan ketentuan berikut:

- Variabel: R, B, G, Y
- Domain variabel: Semua sel dengan warna yang sesuai
 - R: {(1,1), (2,1), (2,2)}
 - B: {(1,2), (1,3), (1,4), (2,3)}
 - G: {(3,1), (4,1), (4,2), (4,3)}
 - Y: {(2,4), (3,2), (3,3), (3,4), (4,4)}
- Tujuan: Menempatkan satu menteri per warna tanpa melanggar kendala

Listing 8: Contoh solusi valid dari papan di listing 7

1	-	-	Q	-
2	Q	-	-	-
3	-	-	-	Q
4	-	Q	-	-

Keterangan:

- 'Q' menandakan posisi menteri pada papan.
- Setiap menteri berada di sel dengan warna yang berbeda (satu per variabel/warna).
- Tidak ada dua menteri yang bersebelahan atau berada pada baris/kolom yang sama.

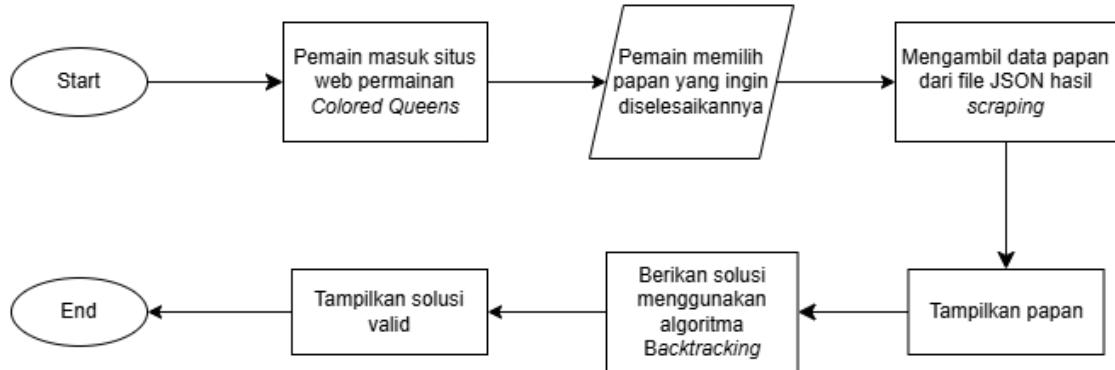
Dengan pemodelan CSP tersebut, seluruh aturan permainan dapat dinyatakan secara terstruktur sehingga solver dapat melakukan pemeriksaan dan pencarian solusi dengan cara yang konsisten dan sistematis.

5.8 Melakukan analisis kebutuhan perangkat lunak, baik fungsional maupun non-fungsional, termasuk kebutuhan solver dan antarmuka pengguna.

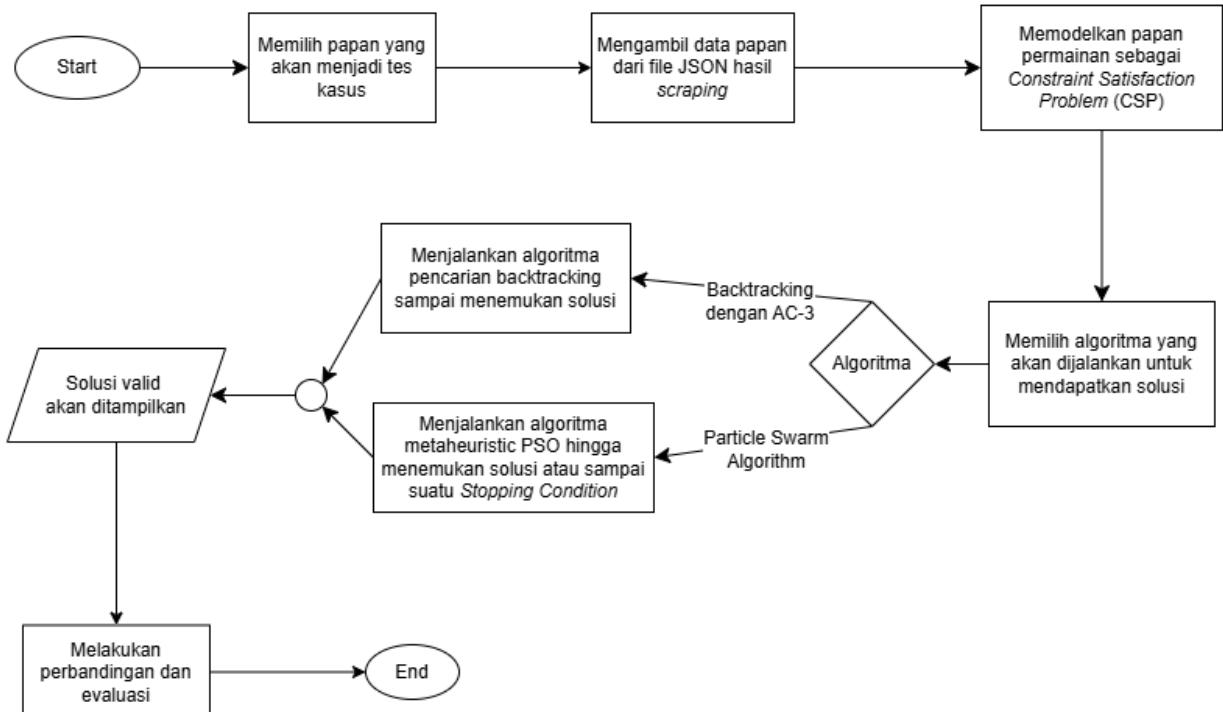
Status : Ada sejak rencana kerja skripsi.

Hasil :

Berdasarkan hasil analisis dan diskusi mengenai proses bisnis tugas akhir ini, dihasilkan dua *flowchart* yang menggambarkan fungsi-fungsi utama dari sistem yang akan dibangun. Sebuah aplikasi berbasis web akan dikembangkan dengan fungsi utama menyediakan sarana untuk memainkan permainan *Colored Queens* dengan fitur *solver* otomatis yang dapat memberikan solusi valid ketika pemain mengalami kesulitan, seperti yang dipaparkan pada Gambar 15. Sistem juga menyediakan fitur perbandingan kinerja antara algoritma Backtracking dan PSO, dengan alur seperti yang ditunjukkan pada Gambar 16.



Gambar 15: *Flowchart* dari proses bisnis pertama, yaitu permainan *Colored Queens* yang berbasis web dan dapat memberikan solusi yang valid kepada pemain dengan memanggil *solver* yang memanfaatkan algoritma backtracking



Gambar 16: *Flowchart* dari proses bisnis kedua, yaitu membandingkan performa dua algoritma berbeda, Backtracking dan PSO, dalam menyelesaikan masalah *Colored Queens*.

1. Kebutuhan Fungsional Sistem

Untuk membangun permainan *Colored Queens*, sistem akan dibangun dengan arsitektur berbasis web, menggunakan *frontend* React dan *backend* Spring Boot. *Frontend* bertugas menampilkan papan permainan, menerima input pengguna, memberikan umpan balik visual secara real-time, dan menyediakan antarmuka interaktif untuk bermain atau menguji solusi. *Backend* menjalankan *solver* Java yang mengimplementasikan algoritma Backtracking dengan AC-3 dan Particle Swarm Optimization (PSO), serta menangani logika validasi.

2. Situs Web dan Pemilihan Papan

Aplikasi akan mengelompokkan papan-papan yang dapat dimainkan berdasarkan ukuran agar mudah dinavigasikan. Pemain juga dapat memilih papan secara acak dari daftar tersebut dan halaman web akan menampilkan papan yang sesuai dengan pilihan pemain.

3. Mengambil data papan dari JSON

Ketika pemain memilih salah satu papan untuk dimainkan, *Frontend* aplikasi web akan mengambil keseluruhan data papan dari file JSON yang tersedia setelah menjalankan *web scraper*, seperti yang dijelaskan di bagian 3, lalu memprosesnya menjadi papan berwarna yang dapat ditampilkan dan dimainkan.

4. Adaptasi Algoritma Backtracking dengan AC-3

Algoritma Backtracking standar untuk Colored Queens menggunakan pendekatan rekursif untuk menempatkan setiap menteri berdasarkan warna pada papan. Agar lebih efisien, algoritma ini akan digabungkan dengan *Arc Consistency Algorithm 3* (AC-3) untuk menyempitkan ruang solusi sebelum dan selama pencarian.

Proses Backtracking dengan AC-3:

- (a) **Inisialisasi:** Setiap warna direpresentasikan sebagai variabel, dan domainnya adalah semua posisi yang valid dalam warna tersebut.
- (b) **Preprocessing AC-3:** Jalankan AC-3 untuk mengeliminasi posisi yang jelas tidak mungkin, misalnya karena konflik adjacency atau serangan langsung.
- (c) **Rekursi Backtracking:**
 - Pilih variabel warna berikutnya.
 - Coba setiap posisi dalam domain yang tersisa.
 - Jika posisi valid (tidak melanggar adjacency atau aturan serangan):
 - Tempatkan menteri pada posisi tersebut.
 - Jalankan AC-3 lagi untuk memperbarui domain variabel lain.
 - Lanjutkan ke variabel warna berikutnya secara rekursif.
 - Jika tidak ada posisi valid, mundur (*backtrack*) ke variabel sebelumnya dan coba posisi lain.
- (d) **Kondisi berhenti:** Semua warna telah ditempatkan tanpa konflik \rightarrow solusi valid ditemukan.

5. Adaptasi Algoritma PSO

PSO biasanya digunakan untuk optimasi kontinu. Agar dapat diterapkan pada masalah diskret seperti Colored Queens, beberapa modifikasi dilakukan:

- (a) **Representasi Partikel:** Setiap partikel adalah solusi kandidat, direpresentasikan sebagai array integer di mana elemen ke-*i* menunjukkan posisi ke-*n* dari warna *i*.
- (b) **Velocity Diskret:** Velocity yang semula berupa jarak perpindahan kontinu diubah menjadi probabilitas pertukaran posisi dengan *neighborhood best* (nBest). Misalnya, partikel [2, 2, 4, 1] dengan velocity [0.3, 0.5, 0.4, 0.6] berarti elemen ke-1 yang bernilai 2 memiliki kemungkinan 0.3 untuk diubah menjadi nilai yang sesuai di nBest.
- (c) **Update Velocity:** Velocity diperbarui menggunakan rumus standar PSO:

$$v[i] = \omega \cdot v[i] + c_1 \cdot r_1 \cdot \Delta_p[i] + c_2 \cdot r_2 \cdot \Delta_n[i]$$

di mana $\Delta_p[i]$ dan $\Delta_n[i]$ adalah nilai boolean yang menunjukkan apakah posisi saat ini berbeda dari pBest/nBest, dan c_1, c_2, r_1, r_2 mengikuti nilai konvensional PSO.

- (d) **Update Partikel:** Setiap partikel akan diperbarui berdasarkan *velocity* barunya yang sudah dihitung dengan cara mengambil nilai acak untuk setiap elemen *i*. Apabila nilai acak $< velocity$, elemen tersebut akan mengubah nilainya sesuai dengan elemen pada posisi yang sama di nBest. Apabila nilai acak $> velocity$, elemen tidak akan mengalami perubahan.

- (e) **Fungsi Fitness:** *Fitness* dihitung untuk meminimalkan pelanggaran adjacency dan serangan dengan:

$$\text{fitness} = w_1 \cdot \text{adjacencyViolation} + w_2 \cdot \text{attackingViolation}$$

dimana nilai *Fitness* yang optimal adalah 0.

Contoh iterasi PSO

Contoh iterasi ini menggunakan papan pada listing 7 dengan parameter $\omega = 0.7$, $c_1 = 2$, $c_2 = 2$, $r_1 = 0.2$, dan $r_2 = 0.2$. nBest juga ditetapkan sebagai $[2, 1, 3, 4]$ dan pBest sebagai $[2, 3, 3, 2]$. Bobot dari pelanggaran ditetapkan sebagai $w_1(\text{adjacency}) = 2$ dan $w_2(\text{attacking}) = 1$

1. Inisialisasi partikel secara acak, sebagai contoh $[2, 4, 3, 3]$ yang dapat dibaca sebagai:

- R → posisi ke-2 = $(2, 1)$
- B → posisi ke-1 = $(2, 3)$
- G → posisi ke-3 = $(4, 2)$
- Y → posisi ke-3 = $(3, 3)$

dengan *velocity* awal sebagai $[0.3, 0.5, 0.4, 0.6]$

2. Hitung *velocity* baru dengan menghitung:

- $0.7 \cdot 0.3 + 2.0 \cdot 0.2 \cdot 0 + 2.0 \cdot 0.2 \cdot 0 = 0.21$
- $0.7 \cdot 0.5 + 2.0 \cdot 0.2 \cdot 0 + 2.0 \cdot 0.2 \cdot 1 = 0.75$
- $0.7 \cdot 0.4 + 2.0 \cdot 0.2 \cdot 0 + 2.0 \cdot 0.2 \cdot 0 = 0.28$
- $0.7 \cdot 0.6 + 2.0 \cdot 0.2 \cdot 1 + 2.0 \cdot 0.2 \cdot 0 = 0.82$

Maka $(v + 1) = [0.21, 0.75, 0.28, 0.82]$

3. Hitung posisi baru partikel tersebut dengan cara mengambil nilai acak untuk setiap elemen:

- Elemen 1 → $0.3 > 0.21 \therefore$ tidak terjadi perubahan.
- Elemen 2 → $0.78 > 0.75 \therefore$ tidak terjadi perubahan
- Elemen 3 → $0.4 > 0.28 \therefore$ tidak terjadi perubahan.
- Elemen 4 → $0.45 < 0.82 \therefore$ nilai berubah sesuai dengan nBest (4).

Maka $x + (v + 1) = [2, 4, 3, 4]$

4. Hitung fitness untuk kondisi partikel sekarang berdasarkan posisi bidak menteri:

- Bidak 1 → $(2, 1)$
Adjacency Violation dengan Bidak 2
- Bidak 2 → $(1, 2)$
Adjacency Violation dengan Bidak 1
Attacking Violation dengan bidak 3
- Bidak 3 → $(4, 2)$
Attacking Violation dengan bidak 2
- Bidak 4 → $(3, 4)$
Maka fitness partikel ini adalah:

$$\text{fitness} = 2 \cdot 2 + 1 \cdot 2 = 6$$

5. Evaluasi pBest dan nBest

- Fitness pBest = 10
- Fitness nBest = 8

- Current Fitness = 6

Karena current fitness < nbest, kedua pBest dan nBest ditukar dengan partikel sekarang. Sehingga nBest = pBest = [0.21, 0.75, 0.28, 0.82] dan Fitness nBest = Fitness pBest = 6

6. Optimasi menggunakan *BitSet*

- Untuk meningkatkan efisiensi, baik Backtracking maupun PSO memanfaatkan *bitset* untuk memantau baris, kolom, dan warna yang sudah ditempati.
- Pengecekan konflik dapat dilakukan dalam waktu konstan $O(1)$ menggunakan operasi logika bitwise.
- *Bitset* juga memungkinkan perhitungan adjacency violation secara cepat dengan operasi AND/OR pada *mask* bit.

7. Integrasi Web dan Solver

Pada sistem ini, papan permainan Colored Queens sepenuhnya di-render oleh React di sisi klien. Pengguna dapat menempatkan menteri, memindahkan bidak, dan menerima umpan balik secara real-time mengenai pelanggaran aturan atau saran langkah, tanpa memerlukan komunikasi dengan *backend*. Hal ini memastikan pengalaman bermain yang interaktif dan responsif.

Fitur *solver* (menggunakan Backtracking atau PSO) hanya akan dipanggil melalui *backend* Spring Boot ketika pengguna menekan tombol *Solve* untuk papan berukuran hingga 11×11 . Backend akan memproses perhitungan, mengembalikan solusi beserta informasi tambahan seperti langkah-langkah penempatan menteri dan status validasi. React kemudian akan menampilkan solusi ini pada papan secara visual.

Untuk papan yang lebih besar (20×20 dan 30×30), *solver* tidak dijalankan secara real-time. Sebagai gantinya, solusi telah disiapkan sebelumnya dan hanya ditampilkan oleh frontend, sehingga pengguna tetap dapat melihat hasil tanpa menunggu proses komputasi yang berat.

Dengan arsitektur ini, interaksi normal pengguna tetap cepat dan sepenuhnya client-side, sedangkan operasi komputasi berat dialihkan ke server hanya ketika diperlukan.

8. Perbandingan Kinerja Algoritma

Agar proses penelitian lebih cepat dan efisien, perbandingan kinerja algoritma tidak dilakukan melalui antarmuka web, melainkan menggunakan *Command Line Interface* (CLI). Pendekatan ini memungkinkan perubahan *hyperparameter* dengan mudah dan eksekusi pengujian secara otomatis tanpa *overhead rendering* antarmuka grafis, sehingga waktu eksekusi murni dari algoritma dapat diukur dengan lebih akurat.

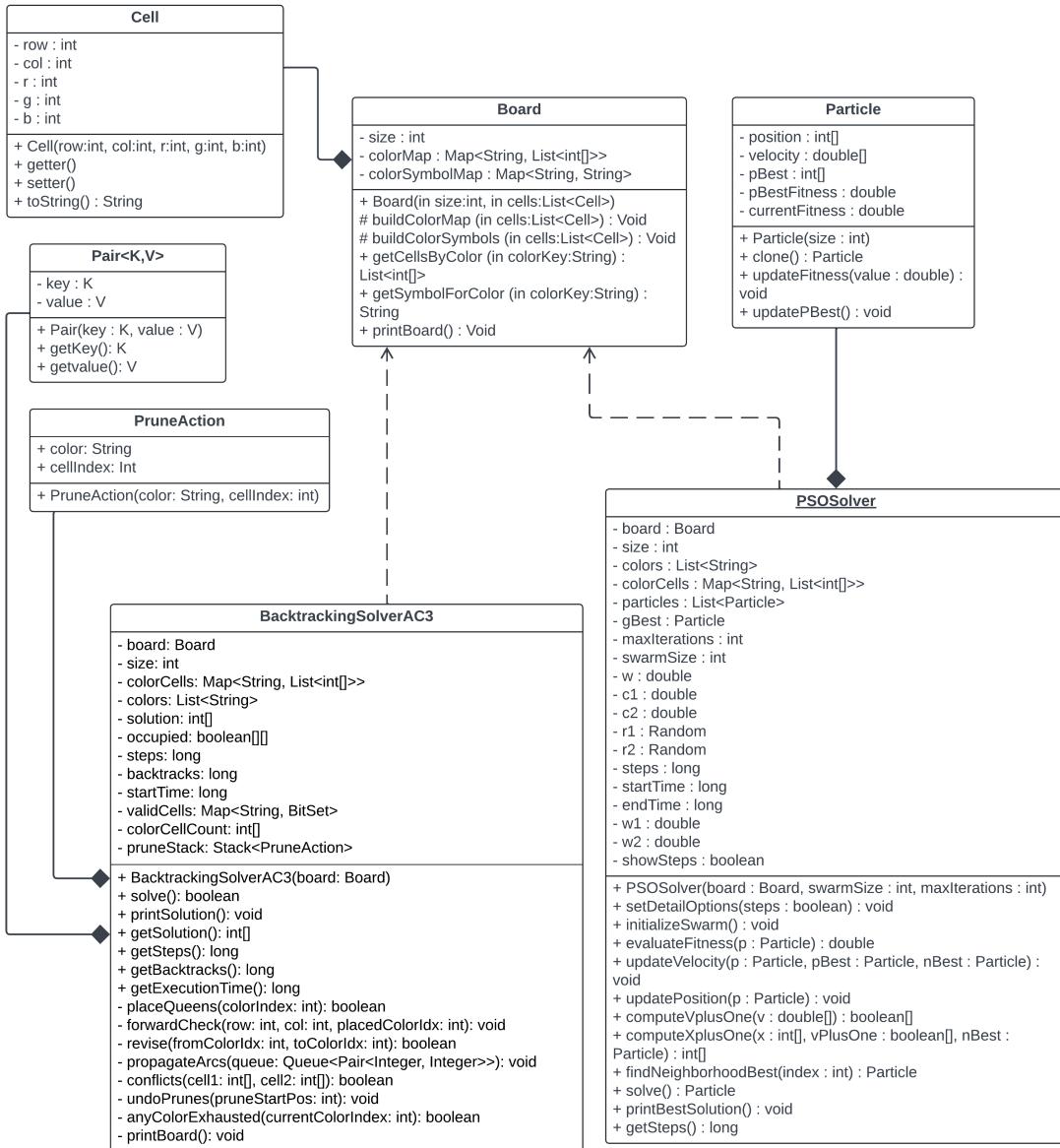
5.9 Merancang arsitektur sistem serta antarmuka pengguna untuk aplikasi Colored Queens Solver.

Status : Ada sejak rencana kerja skripsi.

Hasil :

5.9.1 Arsitektur sistem dalam bentuk *Class Diagram*

Gambar 17 menunjukkan diagram kelas utama yang digunakan dalam implementasi sistem *Colored Queens* beserta dua *solver* yang dikembangkan. Diagram ini memberikan gambaran umum mengenai struktur kelas, atribut penting, metode inti, serta relasi antarkomponen yang menyusun arsitektur sistem.

Gambar 17: *Class Diagram* dari *solver Colored Queens* yang akan dibangun pada tugas akhir ini

Secara keseluruhan, terdapat empat kelas utama, yaitu `Cell`, `Board`, `BacktrackingSolverAC3`, dan `PSOSolver`, dengan satu kelas internal tambahan yaitu `Particle` yang merupakan bagian dari `PSOSolver`. Berikut adalah penjelasan tiap elemen dan hubungan antarkelas dalam diagram.

- **Cell** Kelas ini merepresentasikan sebuah sel pada papan. Setiap sel memiliki koordinat baris dan kolom, serta nilai warna dalam format RGB. Kelas ini berfungsi sebagai unit data dasar yang kemudian dikelompokkan oleh kelas `Board`.

Metode utama:

- `Cell(row, col, r, g, b)`: Konstruktor untuk inisialisasi sel dengan koordinat dan nilai warna RGB
- `getter()` dan `setter()`: Metode akses untuk membaca dan mengubah atribut sel
- `toString()`: Menghasilkan representasi string dari sel dalam format `Cell[row=x, col=y, rgb=(r,g,b)]`

- **Board** Kelas ini mengelola struktur papan dan memetakan setiap warna ke daftar sel yang ditempati. `Board` membangun `colorMap` dan `colorSymbolMap`, serta menyediakan metode untuk memperoleh sel berdasarkan warna dan menampilkan papan. Kelas ini menjadi dependensi utama bagi kedua algoritma `solver`.

Metode utama:

- `Board(size, cells)`: Konstruktor yang membangun papan dengan ukuran tertentu dan daftar sel, kemudian memanggil `buildColorMap()` dan `buildColorSymbols()`
- `buildColorMap(cells)`: Memetakan setiap warna RGB unik ke daftar posisi sel yang memiliki warna tersebut
- `buildColorSymbols(cells)`: Memberikan simbol huruf (A, B, C, ...) untuk setiap warna unik, melewati 'Q' yang digunakan untuk menteri
- `getCellsByColor(colorKey)`: Mengembalikan daftar posisi sel untuk warna tertentu
- `getSymbolForColor(colorKey)`: Mengembalikan simbol huruf yang merepresentasikan warna tertentu
- `printBoard()`: Mencetak papan dalam format grid dengan simbol warna

- **BacktrackingSolverAC3** Kelas ini mengimplementasikan algoritma `Backtracking` yang diperkaya dengan propagasi kendala AC-3. Di dalamnya terdapat struktur optimisasi seperti `bitset`, `prune stack`, dan matriks okupansi. Kelas ini bergantung pada `Board` untuk memperoleh domain warna dan posisi sel, lalu melakukan pencarian solusi secara deterministik.

Metode utama:

- `BacktrackingSolverAC3(board)`: Konstruktor yang menginisialisasi solver, memetakan sel ke dalam `BitSet` untuk pelacakan domain valid, dan menyiapkan stack untuk merekam aksi pemangkasan (`pruning`)
- `solve()`: *Entry point* utama yang memulai pencarian solusi dan menghitung statistik waktu eksekusi
- `placeQueens(colorIndex)`: Metode rekursif utama yang mencoba menempatkan menteri pada posisi valid, lalu memicu `forward checking` dan propagasi AC-3 sebelum lanjut ke warna berikutnya
- `forwardCheck(row, col, placedColorIndex)`: Melakukan penyaringan awal (`filtering`) dengan menghapus nilai-nilai pada domain warna masa depan yang berkonflik langsung dengan menteri yang baru ditempatkan menggunakan perhitungan konflik `inline`

- `propagateArcs(queue)`: Mengelola antrean (`queue`) pasangan variabel untuk algoritma AC-3, memastikan bahwa setiap pengurangan domain dipropagasi ke variabel tetangga yang relevan
 - `revise(fromColorIdx, toColorIdx)`: Inti dari logika AC-3; memeriksa apakah setiap nilai dalam domain tujuan (`toColor`) memiliki dukungan (`support`) yang valid di domain asal (`fromColor`), dan memangkas nilai yang tidak memiliki dukungan
 - `conflicts(cell1, cell2)`: Metode utilitas untuk memeriksa apakah dua sel saling menyerang (baris sama, kolom sama, atau bersebelahan/tetangga 8 arah)
 - `undoPrunes(pruneStartPos)`: Mengembalikan nilai-nilai ke dalam domain (`BitSet`) yang sebelumnya dipangkas, dipanggil saat algoritma melakukan *backtrack*
 - `anyColorExhausted(currentColorIndex)`: Memeriksa apakah terjadi *domain wipeout* (domain kosong) pada warna apa pun di masa depan, yang menandakan jalur solusi saat ini gagal
 - `getSolution()`, `getSteps()`, `getExecutionTime()`: Mengembalikan hasil solusi akhir dan statistik kinerja algoritma
- **PSOSolver** Kelas ini mengimplementasikan algoritma *Particle Swarm Optimization* (PSO) yang telah dimodifikasi untuk permasalahan diskret Colored Queens. PSO bekerja dengan membangkitkan populasi partikel, memperbarui *velocity*, memindahkan posisi partikel, menghitung *fitness*, serta menentukan *global best*. Kelas ini juga bergantung pada `Board` untuk mendapatkan domain posisi tiap warna.

Metode utama:

- `PSOSolver(board, swarmSize, maxIterations)`: Konstruktor yang menginisialisasi solver PSO dengan board, ukuran *swarm*, dan jumlah iterasi maksimum
- `setParameters(w, c1, c2, r1, r2, w1, w2)`: Mengatur parameter PSO (*inertia weight, cognitive/social coefficients*, faktor random, bobot fitness)
- `solve()`: *Entry point* utama PSO yang menginisialisasi *swarm*, melakukan iterasi update, dan mengembalikan solusi *global best*
- `initializeSwarm()`: Membangkitkan populasi awal partikel dengan posisi dan *velocity* acak dalam domain yang valid
- `evaluateFitness(particle)`: Menghitung fitness satu partikel menggunakan formula: $w1 \times \text{adjacencyViolations} + w2 \times \text{attackingViolations}$
- `updateVelocity(particle, pBest, nBest)`: Memperbarui *velocity* partikel menggunakan formula PSO yang diadaptasi untuk ruang diskret: $v = \omega \cdot v + c_1 \cdot r_1 \cdot (pBest - x) + c_2 \cdot r_2 \cdot (nBest - x)$
- `updatePosition(particle)`: Memperbarui posisi partikel berdasarkan *velocity* dengan mekanisme *swap* probabilistik
- `computeXplusOne(x, vPlusOne, boolean[], nBest)`: Menghitung posisi baru partikel berdasarkan *velocity* dan *neighborhood best*
- `findNeighborhoodBest(index)`: Mencari partikel dengan fitness terbaik dalam *neighborhood* (topologi ring atau global)
- `getSteps()`: Mengembalikan jumlah iterasi yang dilakukan

- **Particle** (kelas internal dari `PSOSolver`) Kelas ini mewakili satu kandidat solusi dalam PSO. Setiap partikel memiliki posisi, *velocity*, nilai fitness, dan *personal best*. Karena partikel sepenuhnya dikelola oleh `PSOSolver`, relasinya ditunjukkan dengan komposisi (*black diamond*).

Metode utama:

- `Particle(size)`: Konstruktor yang menginisialisasi partikel dengan array `position` dan `velocity` sesuai jumlah warna
- `getPosition()`, `setPosition()`: Mengakses dan mengubah posisi partikel (array indeks posisi untuk setiap warna)
- `getVelocity()`, `setVelocity()`: Mengakses dan mengubah `velocity` partikel (array probabilitas pertukaran)
- `getPBest()`, `setPBest()`: Mengakses dan mengubah posisi *personal best*
- `updateFitness(value)`: Memperbarui nilai fitness saat ini
- `updatePBest()`: Memperbarui *personal best* jika fitness saat ini lebih baik
- `clone()`: Membuat salinan independen dari partikel untuk keperluan komparasi atau backup

Relasi antarkelas dalam diagram dapat diringkas sebagai berikut:

- **Board** mengandung banyak **Cell**.
- **BacktrackingSolverAC3** bergantung pada **Board** untuk membaca struktur domain.
- **PSOSolver** juga bergantung pada **Board** karena menggunakan informasi warna dan posisi sel.
- **PSOSolver memiliki komposisi** terhadap kelas **Particle**, yang menunjukkan bahwa **Particle** tidak berdiri sendiri dan hanya ada sebagai bagian internal dari proses PSO.

Dengan demikian, Gambar 17 menggambarkan arsitektur lengkap sistem Colored Queens beserta algoritma pencarian yang digunakan, serta menunjukkan pembagian tanggung jawab masing-masing kelas dalam proses penyelesaian masalah.

5.9.2 *Low-fidelity mockup* dari antarmuka aplikasi web

Bagian ini menyajikan rancangan antarmuka awal dari sistem *Colored Queens*. *Mockup* ini bertujuan untuk menggambarkan struktur halaman, letak komponen utama, serta alur interaksi pengguna secara sederhana tanpa mempertimbangkan aspek visual akhir. Desain dapat berubah pada implementasi *high-fidelity* atau tahap pengembangan berikutnya.

1. Halaman Utama

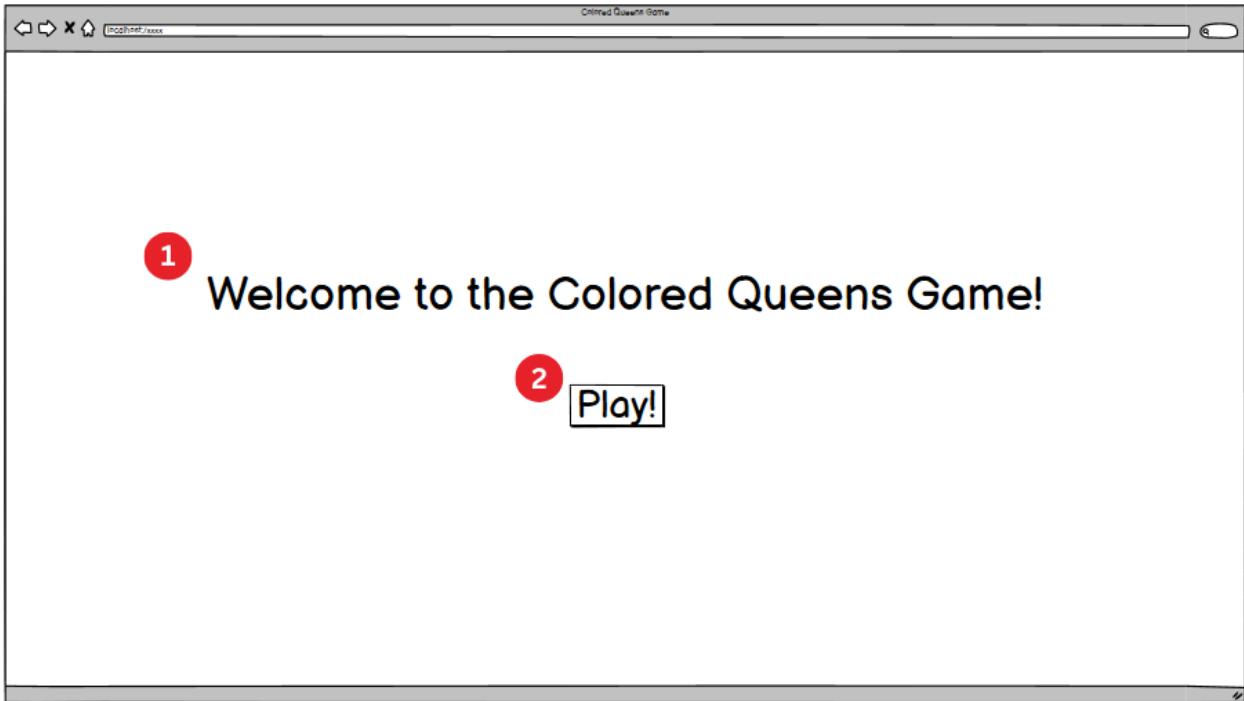
Gambar 18 menunjukkan halaman utama yang merupakan titik masuk pertama ketika pengguna mengunjungi situs *Colored Queens*. Pengguna diperkenalkan dengan tampilan sederhana yang berfokus pada tombol utama untuk memulai permainan.

Komponen-komponen:

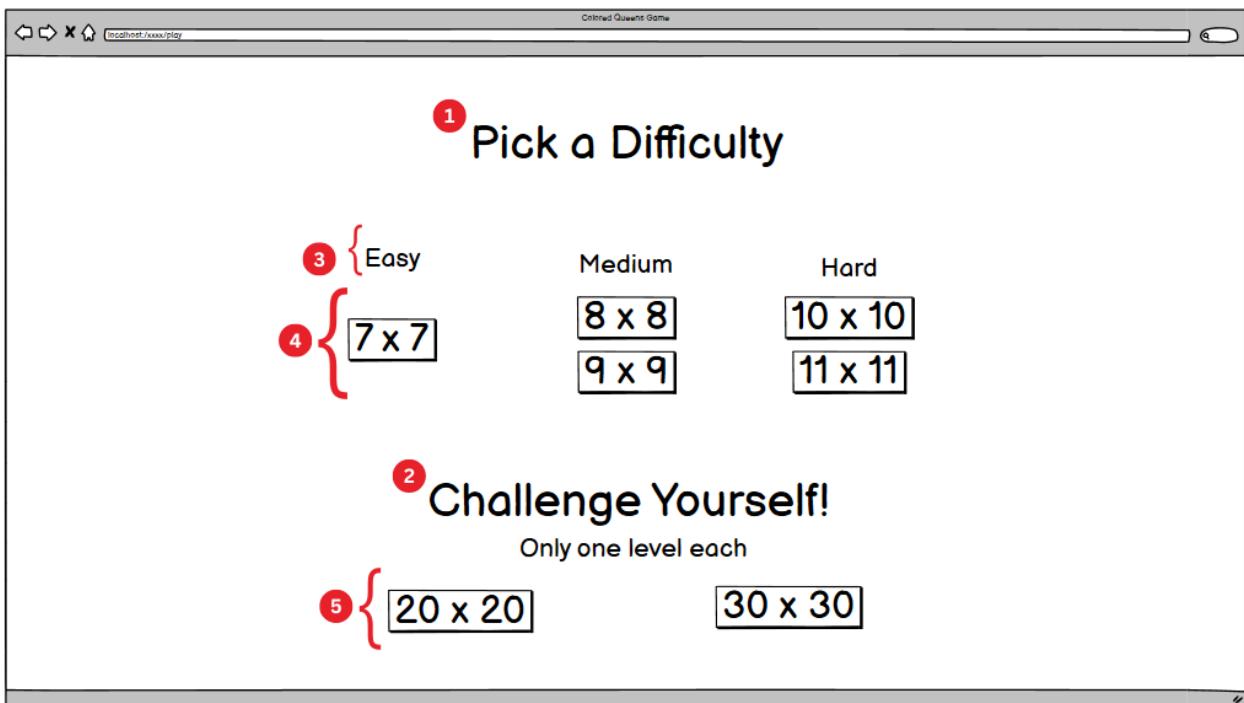
1. **Kata Sambutan:** Teks sapaan atau judul permainan yang berfungsi menyambut pengguna serta menampilkan identitas aplikasi *Colored Queens*.
2. **Tombol mulai permainan/*Play!*:** Tombol interaksi utama yang mengarahkan pengguna menuju halaman pemilihan tingkat kesulitan (*Level Selector*) untuk memulai sesi permainan.

2. Pemilihan Tingkat Kesulitan

Setelah menekan tombol *Play*, pengguna diarahkan menuju halaman pemilihan tingkat kesulitan seperti terlihat pada Gambar 19.



Gambar 18: Tampilan halaman utama ketika pengguna pertama kali mengunjungi situs *Colored Queens*. Pengguna dapat mengklik tombol "Play!" untuk memulai permainan.



Gambar 19: Halaman pemilihan tingkat kesulitan yang membagi level berdasarkan ukuran grid.

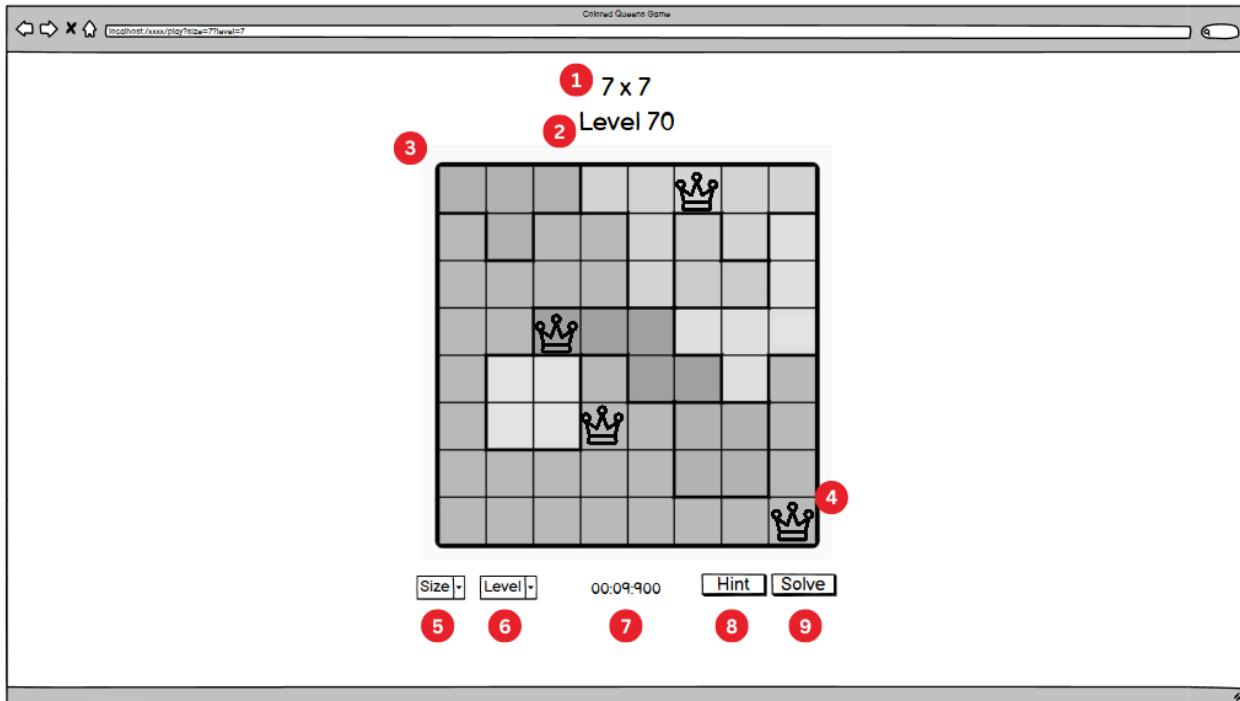
Komponen-komponen:

1. **Judul Utama:** Instruksi untuk memilih tingkat kesulitan.
2. **Judul Kategori Tantangan:** Bagian khusus untuk level dengan ukuran papan ekstrem.
3. **Label Kategori:** Penanda tingkat kesulitan (*Easy*, *Medium*, *Hard*).
4. **Tombol Level Standar:** Tombol untuk memilih ukuran papan umum (7×7 hingga 11×11).

5. **Tombol Level Tantangan:** Tombol untuk memilih ukuran papan besar (20×20 dan 30×30).

3. Halaman Level / Papan Permainan

Pada halaman 20, pengguna berinteraksi langsung dengan papan permainan. Detail antarmuka dapat dilihat pada gambar berikut:



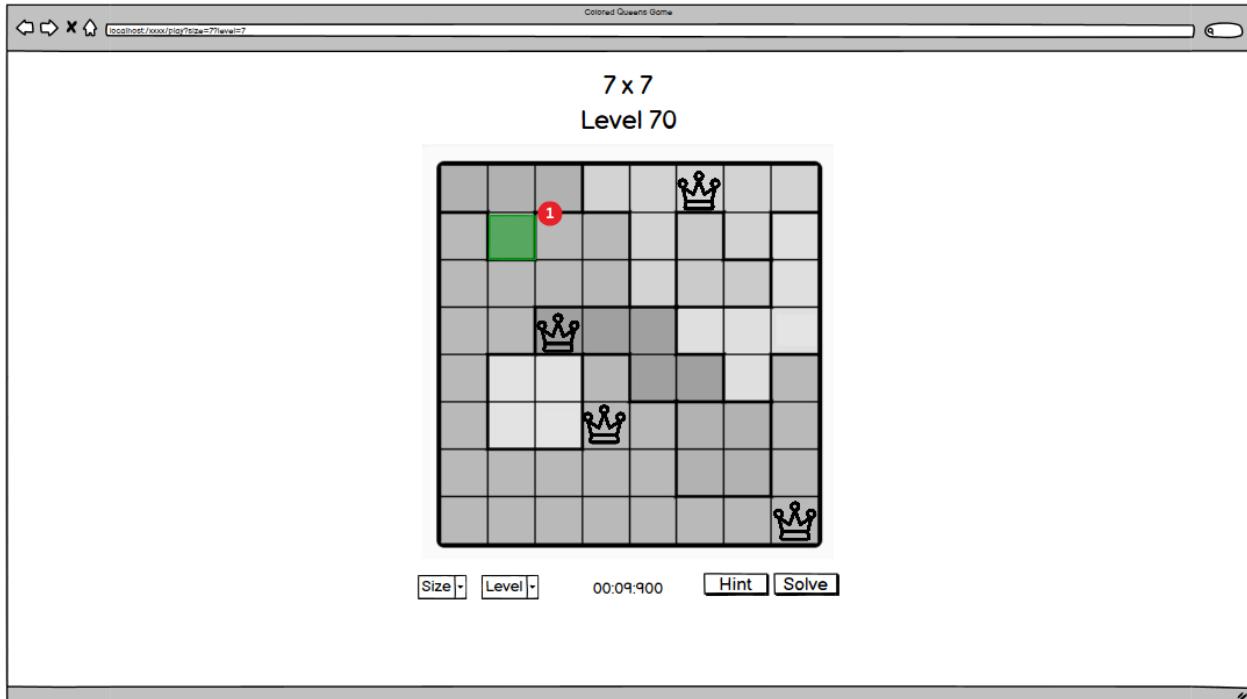
Gambar 20: Tampilan antarmuka permainan utama beserta instrumen pendukungnya.

Komponen-komponen:

1. **Indikator Ukuran:** Menampilkan dimensi papan yang sedang dimainkan.
2. **Indikator Level:** Menampilkan nomor level saat ini.
3. **Area Papan Permainan:** Grid tempat permainan berlangsung.
4. **Bidak Menteri (Queen):** Objek permainan yang dapat ditempatkan oleh pemain.
5. **Dropdown Ukuran:** Navigasi cepat untuk mengganti ukuran papan.
6. **Dropdown Level:** Navigasi cepat untuk berpindah level dalam ukuran yang sama.
7. **Timer:** Penghitung durasi permainan yang dimulai saat pemain menempatkan bidak menteri pertamanya.
8. **Tombol Hint:** Tombol untuk meminta bantuan sistem.
9. **Tombol Solve:** Tombol untuk menyerah dan melihat solusi otomatis. Tombol ini akan memanggil fungsi/algoritma pencarian untuk menemukan solusi valid untuk papan yang sedang ditampilkan.

4. Hint / Petunjuk

Fitur *hint* memberikan bantuan visual dalam tiga bentuk. Pertama, rekomendasi langkah valid (Gambar 21).

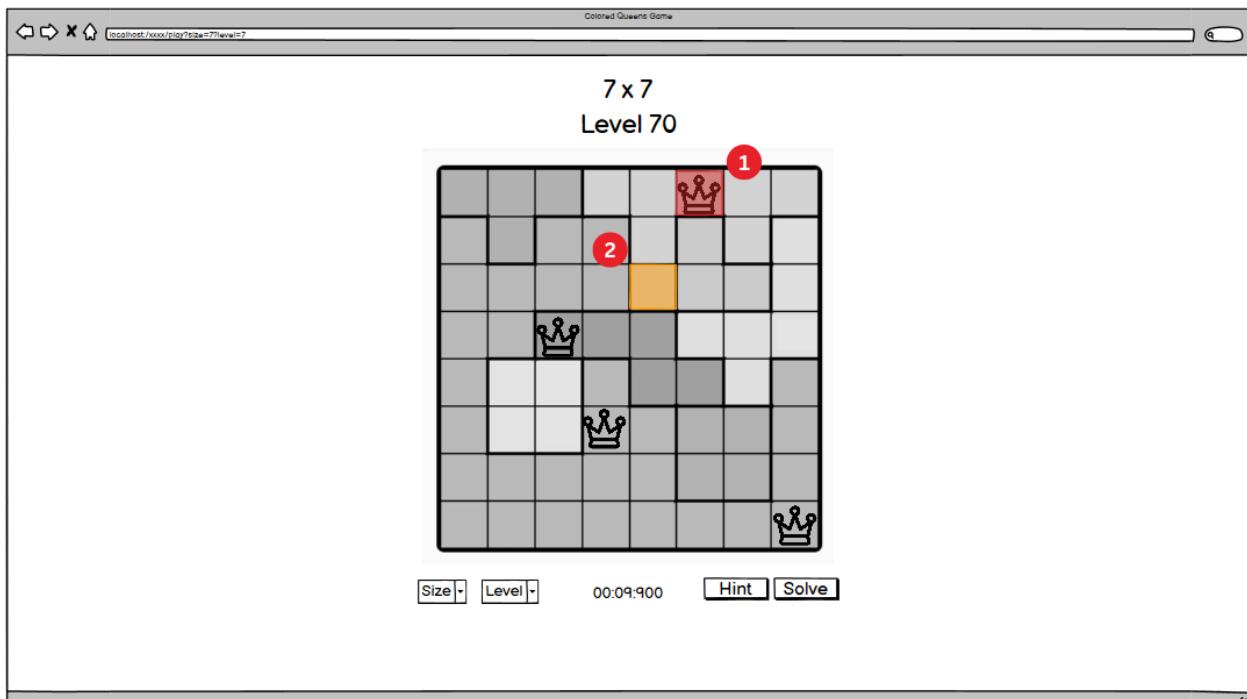


Gambar 21: Visualisasi *hint* rekomendasi langkah.

Komponen-komponen:

1. **Highlight Hijau:** Menandakan sel kosong yang aman dan disarankan untuk ditempati menteri.

Kedua, koreksi kesalahan (Gambar 22) jika pemain menempatkan bidak di posisi yang salah di suatu sektor warna.

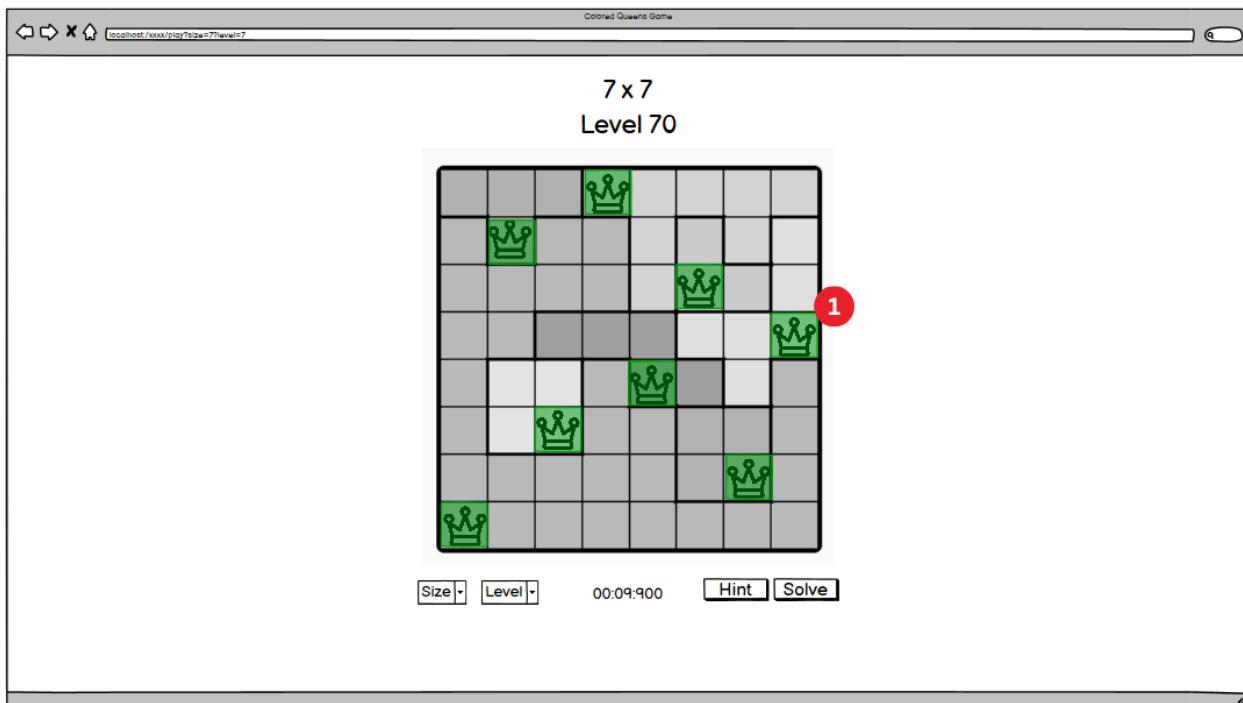


Gambar 22: Visualisasi *hint* korektif.

Komponen-komponen:

1. **Highlight Merah:** Menandakan bidak menteri yang posisinya salah pada suatu warna dan harus dipindahkan ke posisi lain pada sektor warna yang sama.
2. **Highlight Oranye:** Menandakan sel tujuan yang benar untuk memindahkan bidak yang salah tersebut.

Ketiga, fitur *Auto-solve* (Gambar 23) yang akan memanggil algoritma pencarian untuk menyelesaikan masalah (kecuali untuk papan ukuran besar).

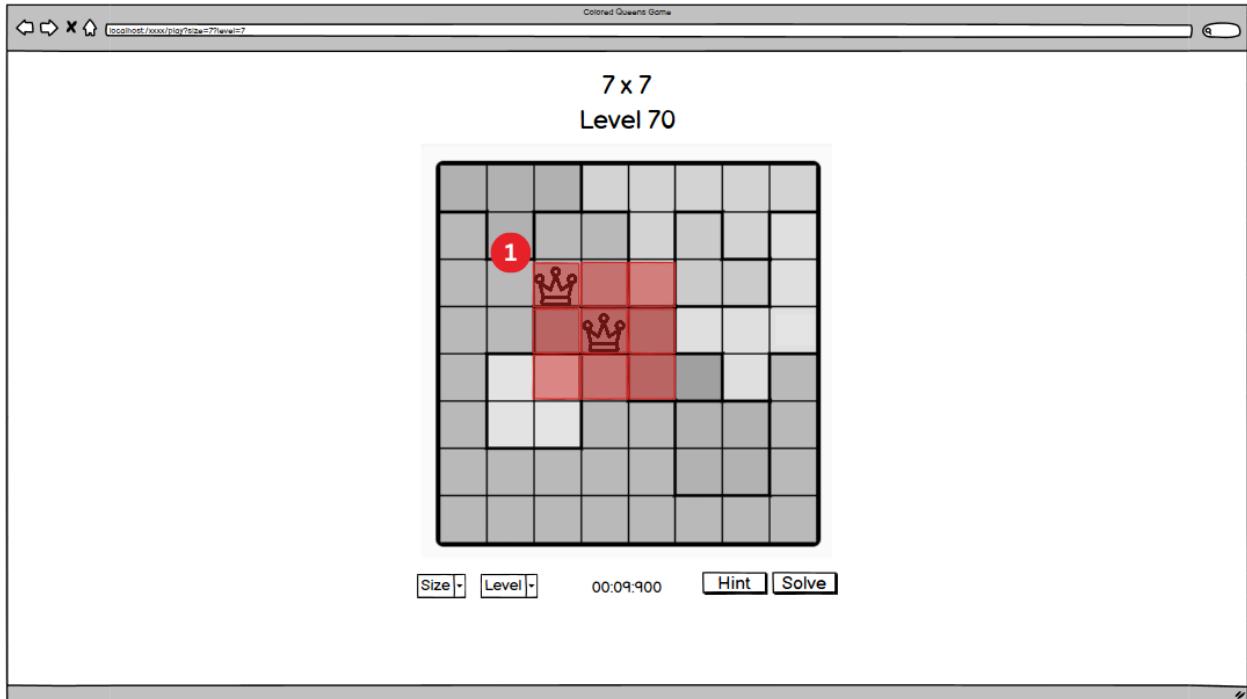


Gambar 23: Kondisi papan setelah fitur *Solve* diaktifkan.

Komponen-komponen:

1. **Bidak Hijau:** Menandakan seluruh menteri telah ditempatkan di posisi solusi yang benar oleh sistem.
2. **Highlight Biru:** Menandakan seluruh menteri yang posisinya salah pada suatu warna dan harus dipindahkan ke posisi lain pada sektor warna yang sama.
3. **Highlight Orange:** Menandakan sel tujuan yang benar untuk memindahkan bidak yang salah tersebut.
4. **Highlight Merah:** Menandakan bidak menteri yang posisinya salah pada suatu warna dan harus dipindahkan ke posisi lain pada sektor warna yang sama.
5. **Warning / Peringatan**

Sistem memberikan peringatan visual merah jika terjadi pelanggaran aturan. Contoh pelanggaran *adjacency* (bersebelahan) terlihat pada Gambar 24.

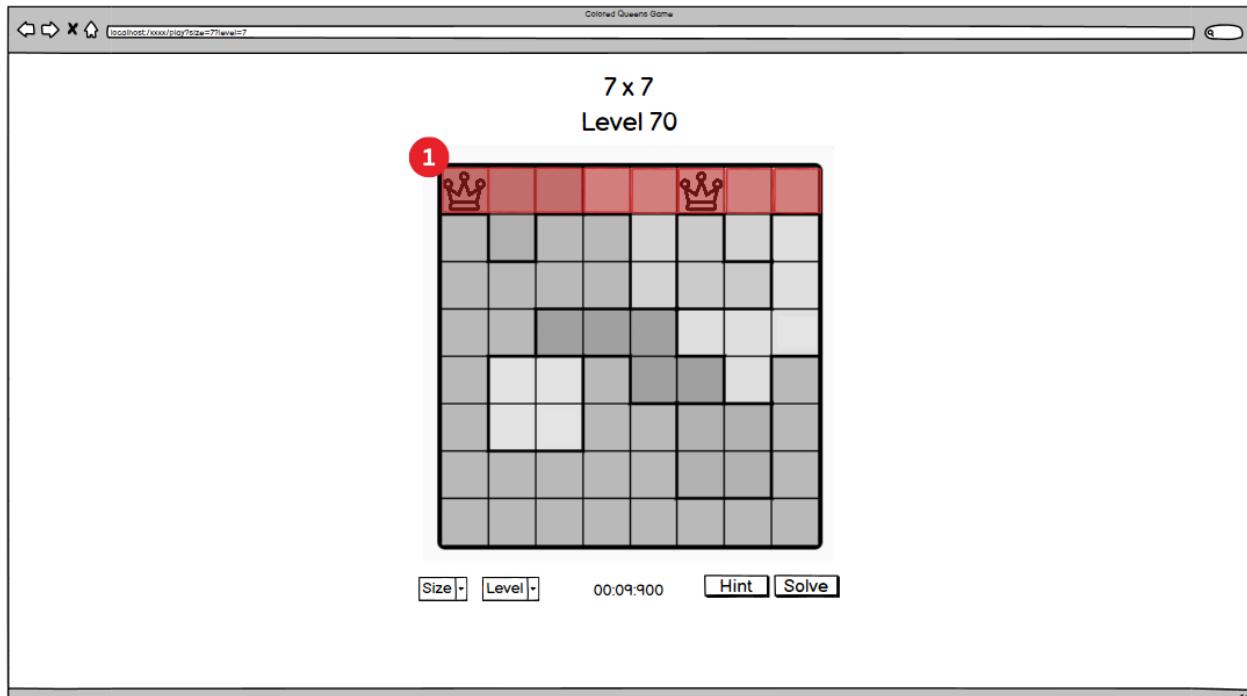


Gambar 24: Indikator kesalahan aturan *adjacency*.

Komponen-komponen:

- Area Merah (Kotak 3x3):** Menandakan konflik karena dua menteri berada terlalu dekat (radius 1 petak).

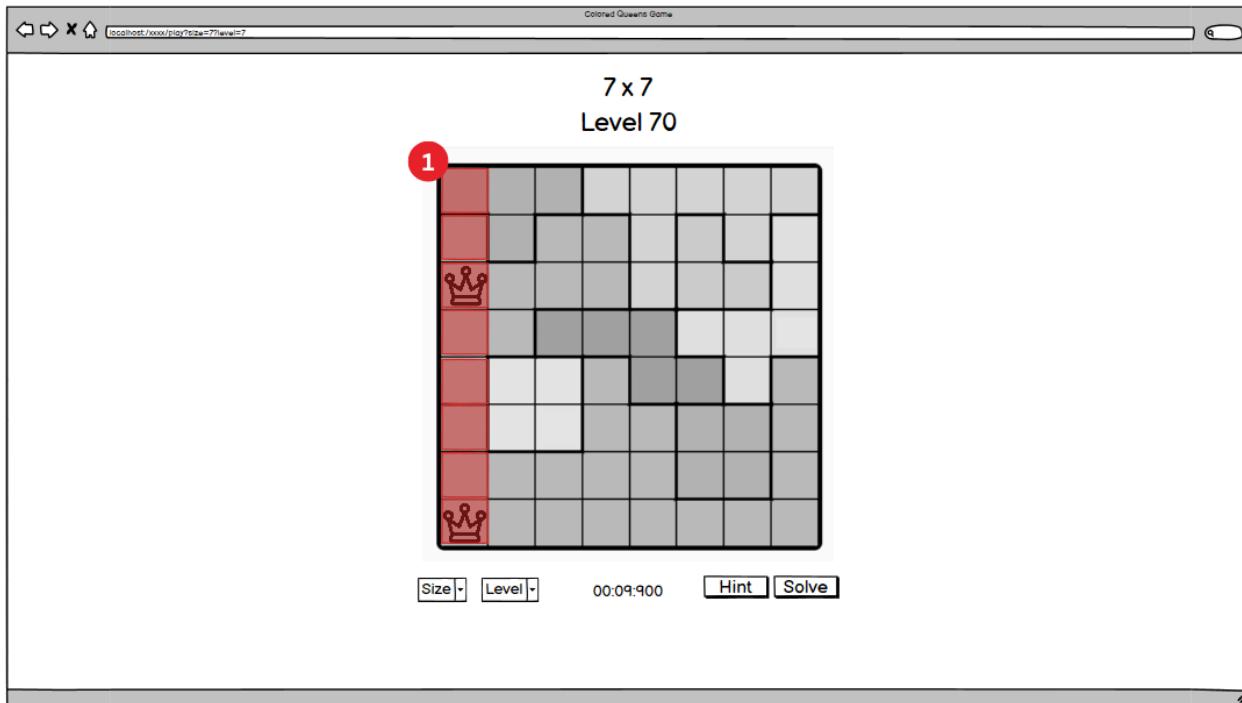
Pelanggaran pada baris atau kolom ditunjukkan pada Gambar 25 dan 26.



Gambar 25: Visualisasi konflik horizontal.

Komponen-komponen:

- Highlight Baris Merah:** Menandakan terdapat lebih dari satu menteri pada baris yang sama.

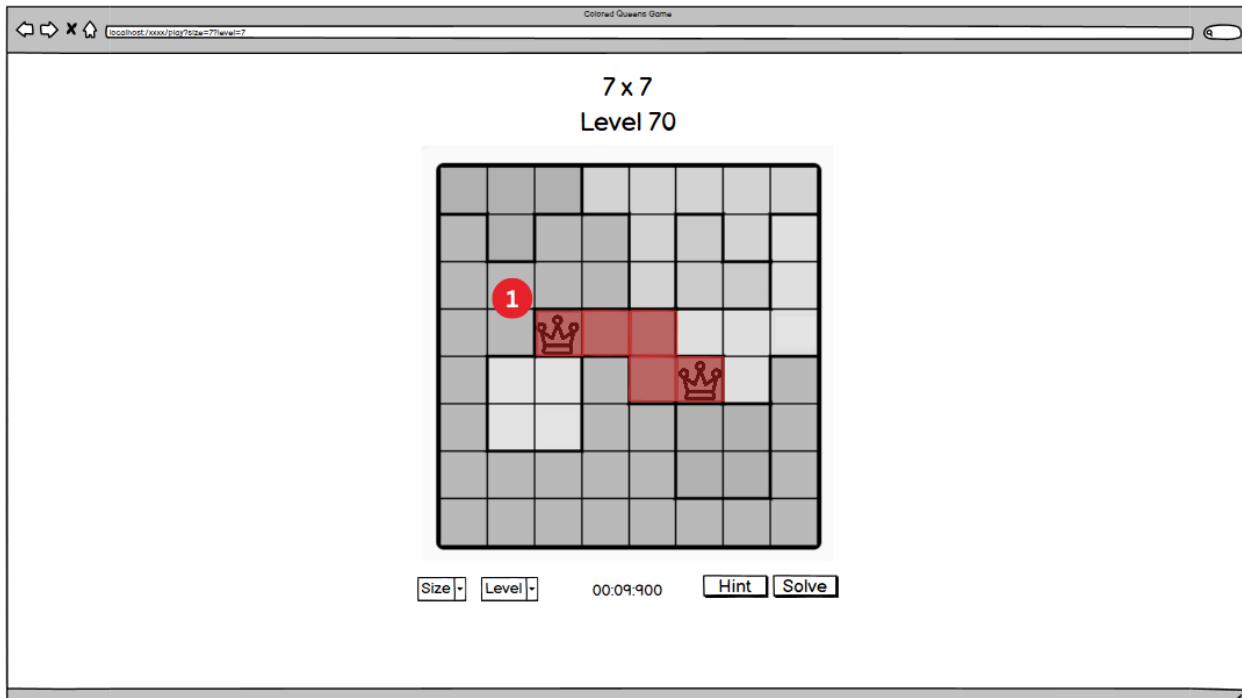


Gambar 26: Visualisasi konflik vertikal.

Komponen-komponen:

- Highlight Kolom Merah:** Menandakan terdapat lebih dari satu menteri pada kolom yang sama.

Pelanggaran aturan warna (Gambar 27).



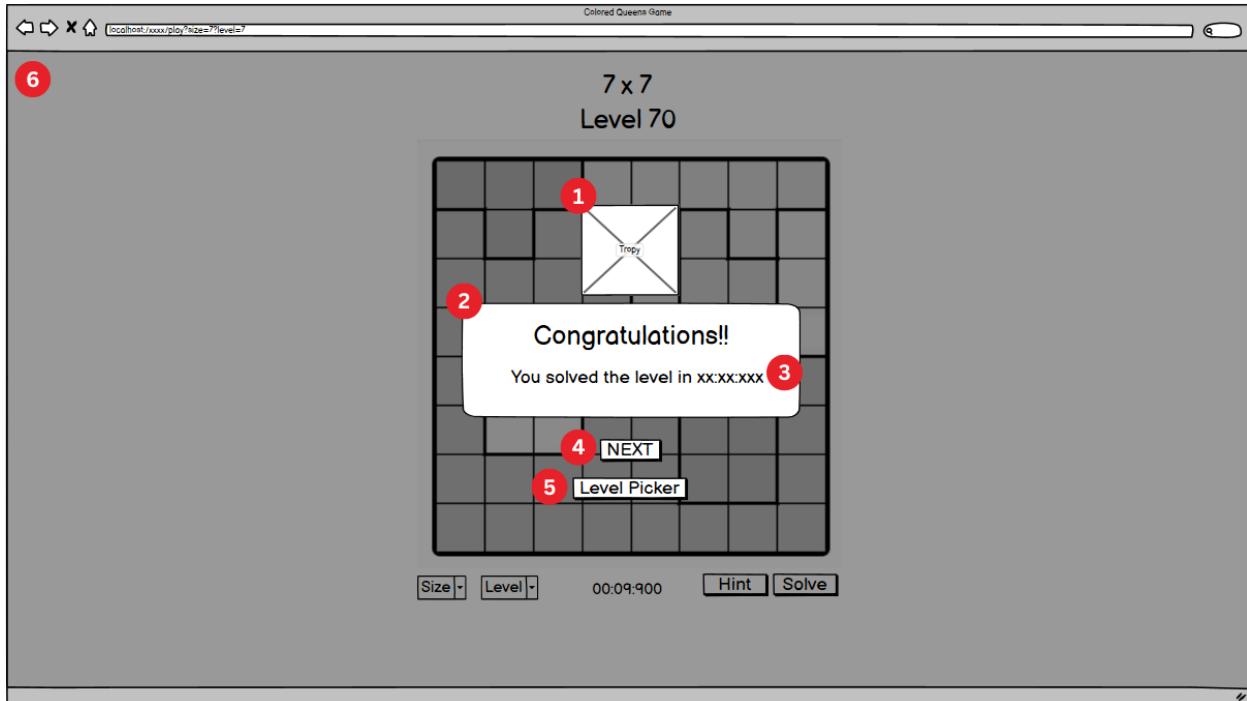
Gambar 27: Peringatan spesifik wilayah warna.

Komponen-komponen:

- Highlight Area Warna Merah:** Menandakan satu wilayah warna telah diisi oleh lebih dari satu menteri.

6. Halaman Menang

Setelah pemain menyelesaikan puzzle, halaman kemenangan ditampilkan (Gambar 28).



Gambar 28: Tampilan layar kemenangan.

Komponen-komponen:

- Ikon Trofi:** Ilustrasi visual keberhasilan.
- Modal Box:** Jendela *pop-up* yang memuat informasi kemenangan.
- Informasi Waktu:** Menampilkan durasi penyelesaian level.
- Tombol Next:** Tombol untuk langsung melanjutkan ke level berikutnya.
- Tombol Level Picker:** Tombol untuk kembali ke menu pemilihan level.
- Overlay Latar Belakang:** Lapisan semi-transparan yang menutupi papan permainan untuk memfokuskan perhatian pemain pada pesan kemenangan.

5.10 Menyusun dokumentasi tugas akhir untuk tahap TA 1.

Status : Ada sejak rencana kerja skripsi.

Hasil :

Bagian latar belakang, dasar teori dan analisis dapat diambil dari dokumen ini. Dasar teori dapat diambil dari bagian ?? sedangkan analisis dapat diambil dari bagian ?? dan ??.

5.11 Mengimplementasikan Solver menggunakan algoritma Backtracking murni yang tidak terintegrasi dengan algoritma AC-3 sebagai tolok ukur

Status : Ditambah di semester ini.

Hasil :

Untuk merepresentasikan *puzzle Colored Queens* secara terstruktur, sistem dilengkapi dengan kelas Board dan Cell yang digunakan untuk menyimpan konfigurasi papan dan properti setiap sel. Pemisahan model ini mempermudah proses validasi kendala, pembacaan pola warna, serta integrasi *solver* dengan antarmuka pengguna. Keduanya berperan dalam jalannya algoritma pencarian, serta membangun dan mengevaluasi solusi.

Listing 9: Kelas Board untuk merepresentasikan papan *Colored Queens*

```

1  public class Board {
2      private int size;
3      private Map<String, List<int[]>> colorMap = new HashMap<>();
4      private Map<String, String> colorSymbolMap = new HashMap<>();
5
6      public Board(int size, List<Cell> cells) {
7          this.size = size;
8          buildColorMap(cells);
9          buildColorSymbols();
10     }
11
12     //memetakan setiap posisi warna ke dalam map
13     private void buildColorMap(List<Cell> cells) {
14         for (Cell cell : cells) {
15             String colorKey = String.format("%d,%d,%d", cell.getR(), cell.getG(), cell.getB());
16             colorMap.computeIfAbsent(colorKey, k -> new ArrayList<>())
17                 .add(new int[]{cell.getRow(), cell.getCol()});
18         }
19     }
20
21     //menggunakan huruf (A,B,C,D, dst.) untuk memodelkan warna RGB yang unik
22     private void buildColorSymbols() {
23         char symbol = 'A';
24         for (String color : colorMap.keySet()) {
25             if (symbol == 'Q') {
26                 symbol++;
27             }
28             colorSymbolMap.put(color, String.valueOf(symbol));
29             symbol++;
30         }
31     }
32
33     public int getSize() {
34         return size;
35     }
36
37     public Map<String, List<int[]>> getColorMap() {
38         return colorMap;
39     }
40
41     public Map<String, String> getColorSymbolMap() {
42         return colorSymbolMap;
43     }
44
45     //mengembalikan semua cell yang merupakan suatu warna
46     public List<int[]> getCellsByColor(String colorKey) {
47         return colorMap.getOrDefault(colorKey, Collections.emptyList());
48     }
49
50     //mengembalikan huruf yang merepresentasikan suatu key RGB
51     public String getSymbolForColor(String colorKey) {
52         return colorSymbolMap.getOrDefault(colorKey, "?");
53     }
54
55     //print dalam format huruf
56     public void printSymbolBoard() {
57         String[][] grid = new String[size][size];
58         for (Map.Entry<String, List<int[]>> entry : colorMap.entrySet()) {
59             String symbol = getSymbolForColor(entry.getKey());
60             for (int[] cell : entry.getValue()) {
61                 grid[cell[0]][cell[1]] = symbol;
62             }
63         }
64
65         for (int r = 0; r < size; r++) {
66             for (int c = 0; c < size; c++) {
67                 System.out.print((grid[r][c] != null ? grid[r][c] : ".") + " ");
68             }
69             System.out.println();
70         }
71     }
72
73     //logging
74     public void printColorSummary() {
75         System.out.println("Board_color_distribution:");
76         for (String color : colorMap.keySet()) {
77             String symbol = getSymbolForColor(color);
78             System.out.printf("Color_%s_(%s)_->_%d_cells%\n",
79                             color, symbol, colorMap.get(color).size());
80         }
81         System.out.println("Board_size:_ " + size);
82         System.out.println("Unique_colors:_ " + colorMap.size());

```

```
83    }
84}
```

Listing 10: Kelas `Cell` untuk merepresentasikan satu kotak di dalam papan *Colored Queens* yang menyimpan posisi dan warna kotak tersebut

```
1 //Merepresentasikan 1 cell/kotak dalam permainan colored queens untuk memudahkan memasukkan data ke dalam Map di dalam object Board
2 public class Cell {
3     //menyimpan posisi dan warna dari kotak
4     private int row;
5     private int col;
6     private int r;
7     private int g;
8     private int b;
9
10    public Cell(int row, int col, int r, int g, int b) {
11        this.row = row;
12        this.col = col;
13        this.r = r;
14        this.g = g;
15        this.b = b;
16    }
17
18    // Getter dan setter
19    public int getRow() {
20        return row;
21    }
22
23    public void setRow(int row) {
24        this.row = row;
25    }
26
27    public int getCol() {
28        return col;
29    }
30
31    public void setCol(int col) {
32        this.col = col;
33    }
34
35    public int getR() {
36        return r;
37    }
38
39    public void setR(int r) {
40        this.r = r;
41    }
42
43    public int getG() {
44        return g;
45    }
46
47    public void setG(int g) {
48        this.g = g;
49    }
50
51    public int getB() {
52        return b;
53    }
54    public void setB(int b) {
55        this.b = b;
56    }
57
58    @Override
59    public String toString() {
60        return String.format("Cell[row=%d, col=%d, rbg=(%d,%d,%d)]", row, col, r, g, b);
61    }
62}
```

Sebelum mengintegrasikan AC-3 sebagai mekanisme penyempitan ruang solusi ataupun optimasi menggunakan `BitSet`, sebuah solver Backtracking murni untuk dijadikan *baseline* dalam pengukuran kinerja. Implementasi awal ini menjalankan proses pencarian secara eksploratif tanpa bantuan propagasi kendala tambahan, sehingga karakteristik performanya dapat menjadi pembanding langsung terhadap versi yang telah diperkuat dengan AC-3.

Listing 11: Solver *backtracking* murni yang diimplementasikan tanpa optimasi apapun, termasuk propagasi kendala atau penggunaan `BitSet`

```
1 public class BacktrackingSolver {
2     private Board board;
3     private int size;
4     private Map<String, List<int[]>> colorCells;
5     private List<String> colors;
6     private int[] solution;           //solution[colorIndex] = index in color's cell list
7     private boolean[][] occupied;   //tracks queen positions
```

```

8   private long steps;
9   private long backtracks;
10  private long startTime;
11
12  public BacktrackingSolver(Board board) {
13      this.board = board;
14      this.size = board.getSize();
15      this.colorCells = board.getColorMap();
16      this.colors = new ArrayList<>(colorCells.keySet());
17      this.solution = new int[colors.size()];
18      Arrays.fill(solution, -1);
19      this.occupied = new boolean[size][size];
20      this.steps = 0;
21      this.backtracks = 0;
22  }
23
24 //memanggil fungsi rekursif
25  public boolean solve() {
26      System.out.println("Starting solver for " + size + "x" + size + " board with " + colors.size() + " colors.");
27      startTime = System.currentTimeMillis();
28      boolean result = placeQueens(0); //hasil fungsi rekursif disimpan di sini
29      long endTime = System.currentTimeMillis();
30
31      System.out.println("\nSolver stats:");
32      System.out.println("Steps: " + steps);
33      System.out.println("Backtracks: " + backtracks);
34      System.out.println("Time: " + (endTime - startTime) + " ms");
35      System.out.println("Solution found: " + result);
36
37      return result;
38  }
39
40 //fungsi backtracking utama
41  private boolean placeQueens(int colorIndex) {
42      if (colorIndex == colors.size()) {
43          return true; //base case
44      }
45
46      //berbeda dengan N-Queens, domain 1 queen merupakan daerah warna itu sendiri, bukan satu kolom/baris agar tidak perlu mengecek "color
47      //constraint" di setiap step
48      String color = colors.get(colorIndex);
49      List<int[]> cells = colorCells.get(color);
50
51      steps++;
52
53      //mencoba setiap cell yang berwarna sama
54      for (int i = 0; i < cells.size(); i++) {
55          int row = cells.get(i)[0];
56          int col = cells.get(i)[1];
57
58          //jika posisi valid
59          if (isValid(row, col)) {
60              solution[colorIndex] = i;
61              occupied[row][col] = true;
62
63              //lanjutkan untuk warna sebelumnya
64              if (placeQueens(colorIndex + 1)) {
65                  return true;
66              }
67
68              backtracks++;
69              solution[colorIndex] = -1;
70              occupied[row][col] = false;
71          }
72      }
73
74      return false;
75  }
76
77  private boolean isValid(int row, int col) {
78      //mengecek apakah cell tersebut sudah ada menteri atau belum
79      if (occupied[row][col]) {
80          return false;
81      }
82
83      //mengecek semua cell horizontal dan vertikal
84      for (int i = 0; i < size; i++) {
85          if (occupied[row][i] || occupied[i][col]) {
86              return false;
87          }
88      }
89
90
91      //mengecek apakah ada bidak yang bersebelahan (8 mata angin)
92      int[][] dirs = {{-1,-1},{-1,0},{-1,1},{0,-1},{0,1},{1,-1},{1,0},{1,1}};
93      for (int[] d : dirs) {
94          int r = row + d[0];
95          int c = col + d[1];
96          if (r >= 0 && r < size && c >= 0 && c < size && occupied[r][c]) {
97              return false;
98          }
99      }
100 }

```

```

102         return true;
103     }
104
105     //print hasil akhir papan dengan Q menandakan Bidak menteri
106     private void printBoard() {
107         System.out.println("Board_state:");
108         String[][] grid = new String[size][size];
109
110         for (Map.Entry<String, List<int[]>> entry : colorCells.entrySet()) {
111             String symbol = board.getSymbolForColor(entry.getKey());
112             for (int[] cell : entry.getValue()) {
113                 grid[cell[0]][cell[1]] = symbol;
114             }
115         }
116
117         for (int r = 0; r < size; r++) {
118             for (int c = 0; c < size; c++) {
119                 if (occupied[r][c]) {
120                     grid[r][c] = "Q";
121                 }
122             }
123         }
124
125         System.out.print("  ");
126         for (int c = 0; c < size; c++) {
127             System.out.print(c + " ");
128         }
129         System.out.println();
130
131         for (int r = 0; r < size; r++) {
132             System.out.print(r + "  ");
133             for (int c = 0; c < size; c++) {
134                 System.out.print((grid[r][c] != null ? grid[r][c] : ".") + " ");
135             }
136             System.out.println();
137         }
138         System.out.println();
139     }
140
141     //Print posisi akhir semua bidak menteri dalam bentuk per baris
142     public void printSolution() {
143         if (solution[0] == -1) {
144             System.out.println("No_solution_found!");
145             return;
146         }
147
148         System.out.println("Final_solution:");
149         for (int i = 0; i < colors.size(); i++) {
150             String color = colors.get(i);
151             String symbol = board.getSymbolForColor(color);
152             int[] cell = colorCells.get(color).get(solution[i]);
153             System.out.println("Color " + symbol + " at [" + cell[0] + "," + cell[1] + "]");
154         }
155
156         printBoard();
157     }
158
159     public long getSteps() {
160         return steps;
161     }
162     public long getBacktracks() {
163         return backtracks;
164     }
165 }
```

Setelah implementasi solver Backtracking murni selesai, dilakukan pengujian khusus pada papan berukuran besar, yaitu 20×20 , seperti pada listing 12, dan 30×30 pada listing 13, untuk mengevaluasi batas kemampuan pendekatan ini tanpa bantuan teknik penyempitan ruang solusi. Pengujian ini bertujuan untuk menunjukkan bagaimana kompleksitas eksponensial backtracking berdampak langsung pada waktu eksekusi ketika ukuran papan meningkat secara signifikan.

Listing 12: Tolok ukur untuk papan 20×20

```

1 Solver stats:
2 Steps: 118401240766
3 Backtracks: 118401240746
4 Time: 24638023 ms
5 Solution found: true
6 Final solution:
7 Color A at [15,14]
8 Color B at [14,1]
9 Color C at [17,16]
10 Color D at [3,0]
11 Color E at [0,12]
12 Color F at [10,9]
```

```

13 Color G at [1,7]
14 Color H at [5,8]
15 Color I at [6,19]
16 Color J at [2,3]
17 Color K at [11,17]
18 Color L at [16,10]
19 Color M at [12,2]
20 Color N at [18,11]
21 Color O at [8,18]
22 Color P at [7,13]
23 Color R at [4,15]
24 Color S at [13,4]
25 Color T at [19,5]
26 Color U at [9,6]
27 Board state:
28 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
29 0 J J J J J G G G G Q E E E I I I I
30 1 J J J J J Q G G G E E E E I I I I
31 2 J J Q J J G G G G E E E E I I I I
32 3 Q D D J J J G G E E E E E R R R R I I I
33 4 D D D D D D D G H E R R R R Q R I I I
34 5 D D D D D D U U Q H H H P P R R R R O I I
35 6 D D D D D U U U H H H H P P P P P P O O Q
36 7 D D D D U U U U H H H H H Q P O O O O O
37 8 D D D D S U U U U H H H H P P P O O O Q O
38 9 M D S S S U Q U U U H H H P K K K O O O
39 10 M M S S S S S U U U Q H H H K K K K K K K K
40 11 M S S S S S S F F F L H K K K K Q K K
41 12 M M Q S S S S F F F F L L L K K K K K K K
42 13 M M M S Q T F F F L L L L L K K K K C C
43 14 M Q B B T T L L L L L L L K K C C C
44 15 B B B B T T T L L L L L L Q A C C C C
45 16 B B B T T T T T N L Q L L A A A A C C C C
46 17 B B B T T T T N N N N N A A A A A Q C C C
47 18 B B B T T T N N N N N Q A A A A A C C C C
48 19 B B B T T Q N N N N N A A A A A A A A C C
49
50
51 BUILD SUCCESSFUL in 6h 50m 39s

```

Listing 13: Tolok ukur untuk papan 30×30

```

1 Solver stats:
2 Steps: 429459387595
3 Backtracks: 429459387565
4 Time: 175308806 ms
5 Solution found: true
6 Final solution:
7 Color A at [21,22]
8 Color B at [15,9]
9 Color C at [6,15]
10 Color D at [7,10]
11 Color E at [26,4]
12 Color F at [0,11]
13 Color G at [9,8]
14 Color H at [19,28]
15 Color I at [1,0]
16 Color J at [27,7]
17 Color K at [28,26]
18 Color L at [18,23]
19 Color M at [22,1]
20 Color N at [29,13]
21 Color O at [4,16]
22 Color P at [11,25]
23 Color R at [10,21]
24 Color S at [20,20]
25 Color T at [23,12]
26 Color U at [14,19]
27 Color V at [3,24]
28 Color W at [2,17]
29 Color X at [25,2]

```

6 Pencapaian Rencana Kerja

Langkah-langkah kerja yang berhasil diselesaikan dalam Tugas Akhir 1 ini adalah sebagai berikut:

1. Melakukan studi literatur terkait permasalahan n-queens dan variannya, Constraint Satisfaction Problem (CSP), algoritma pencarian Backtracking, teknik metaheuristik Particle Swarm Optimization, serta metode propagasi kendala AC-3.
 2. Mengumpulkan dan menyusun berbagai skenario permasalahan Colored Queens yang akan digunakan sebagai basis pengujian algoritma serta sebagai pilihan tingkat kesulitan bagi pengguna.
 3. Melakukan pemodelan masalah Colored Queens ke dalam bentuk CSP agar dapat diproses oleh algoritma pencarian.
 4. Melakukan analisis kebutuhan perangkat lunak, baik fungsional maupun non-fungsional, termasuk kebutuhan solver dan antarmuka pengguna.
 5. Merancang arsitektur sistem serta antarmuka pengguna untuk aplikasi Colored Queens Solver.
 6. Menyusun dokumentasi tugas akhir untuk tahap TA 1.

Bandung, 11/12/2025

Arlo Dante Hananvyasa

Menyetujui,

Nama: Husnul Hakim, M.T.
Pembimbing Tunggal