Penyelesaian Permainan Teka-Teki KBlackBox dengan *Brute Force* serta *Backtracking*

Fajar Kurniawan - 13523027
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13523027@std.stei.itb.ac.id

***Abstrak*—KBlackBox adalah permainan pada instalasi *Desktop Environment* KDE (Plasma) yang dibuat berdasarkan permainan klasik BlackBox. Inti dari permainan adalah meletakkan bola-bola di posisi tertentu sedemikian sehingga tembakan sinar yang masuk ke papan bisa terkonfigurasi dengan keadaan tertentu (sinar langsung keluar, dibelokkan, atau dipantulkan). Makalah ini membahas penyelesaian permainan KBlackBox menggunakan *Brute Force* dan *Backtracking* serta menentukan seberapa jauh perbedaan waktu eksekusi keduanya.**

Kata kunci—BlackBox; KBlackBox; brute force; backtrack; bola; sinar; teka-teki; puzzle; grid;

I. PENDAHULUAN

KblackBox adalah permainan teka-teki yang merupakan perangkat lunak bawaan pada sistem operasi berbasis GNU/Linux yang menggunakan *desktop environment* bernama Plasma yang dirancang oleh KDE. Pada permainan ini, pemain harus meletakkan sejumlah bola dengan konfigurasi tertentu sehingga sinar-sinar yang masuk ke papan permainan bisa memenuhi tujuan tertentu yang ditentukan sejak awal permainan. Ada tiga macam hal yang dapat terjadi pada sebuah sinar yang masuk. Kemugkinan pertama adalah dipantulkan kembali ke tempat masuk. Kemungkinan kedua adalah dibelokkan arah sinarnya sehingga sinar keluar melewati sisi yang tidak paralel dengan sisi masuknya. Kemungkinan ketiga adalah sinar langsung keluar dari sisi di depan sisi tempat masuknya (sisi yang sejajar).

Ketika jumlah bola relatif sedikit dan ukuran papan permainan relatif kecil, menyelesaikan permainan secara manual memerlukan usaha yang cukup mudah tetapi akan memerlukan waktu yang tidak singkat karena pemain harus mengecek semua kemungkinan yang ada dan mempertimbangkan sinar lain sambil mencari kemungkinan peletakan suatu bola (misal X) yang memenuhi syarat dari perilaku suatu sinar (misal A). Namun, ketika jumlah bola semakin banyak dan ukuran papan permainan relatif besar, penyelesaian teka-teki ini memerlukan waktu yang jauh lebih lama dan harus mempertimbangkan lebih banyak sinar pada waktu yang bersamaan. Maka dari itu digunakan algoritma yang dijalankan pada komputer untuk mempercepat proses pencarian solusi.

Solusi yang akan dicek berupa koordinat masing-masing bola. Konfigurasi peletakan bola akan dicek dengan mensimulasikan masing-masing sinar apakah memenuhi perilaku yang sesuai dengan yang diminta oleh teka-teki. Algoritma *exhaustive search* atau *brute force* dapat digunakan untuk mengecek semua kemungkinan yang ada. Namun, jika teka-teki semakin rumit makan waktu yang diperlukan untuk mengecek semua kemungkinan solusi menjadi sangat panjang, karena itulah digunakan juga algoritma *backtracking* sebagai pembanding. Algortima *backtracking* dapat jauh lebih cepat karena akan mengakhiri atau jalur atau cabang pencarian yang jelas tidak akan mengarahkan ke solusi yang valid. Hal ini memungkinkan pencarian menjadi jauh lebih cepat karena tidak perlu mengecek seluruh kombinasi koordinat bola yang mungkin (kecuali pada kasus terburuk). Backtracking pada dasarnya adalah DFS yang melakukan *pruning* atau pemangkasan di awal jika simpul tidak mengarah ke solusi yang valid.

II. DASAR TEORI

A. *KBlackBox*

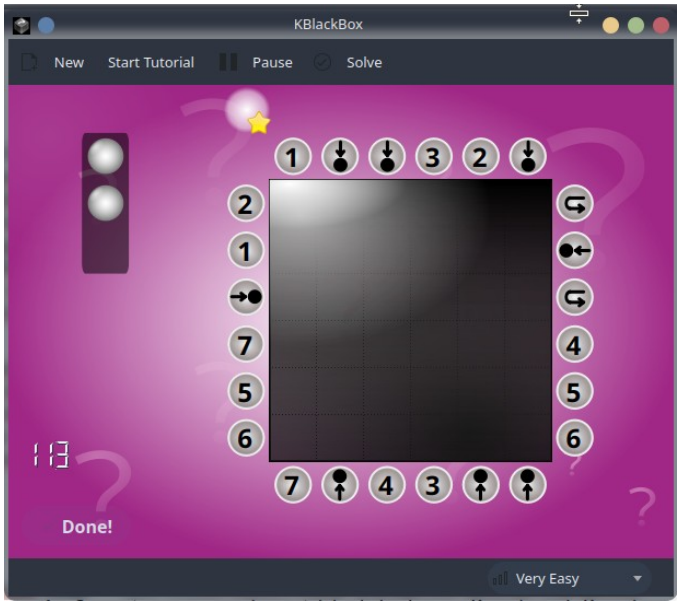
KBlackBox adalah sebuah permainan dengan visual 2D yang dibuat berdasarkan sebuah permainan dengan nama yang mirip yaitu Black Box. Permaina Black Box adalah permainan papan yang dibuat oleh *Parker Brothers* pada akhir tahun 1970-an [1]. Versi digital dari permainan ini memiliki beberapa variasi tetapi KBlackBox tidak menambahkan fitur apapun dan tetap mengikuti permainan Black Box yang asli.

Sesuai namanya, KBlackBox adalah permainan Black Box yang terdapat pada sistem operasi dengan *dekstop environment* KDE Plasma secara bawaan. Huruf ‘K’ di awal merujuk ke KDE seperti huruf ‘K’ pada Kubuntu (varian Ubuntu dengan *desktop environment* KDE dibandingkan GNOME yang merupakaan *desktop environment* bawaan Ubuntu). Pada KBlackBox, pemain menggunakan sinar laser untuk mencoba menentukan penempatan partikel atom (berupa bola) di dalam kotak papan permainan. Dari informasi yang ada, pemain harus menebak penempatan bola yang benar.

Pemain dapat menggunakan *mouse* untuk menggerakan *cursor* dan menekan tombol kiri pada *mouse* untuk menempatkan bola/atom [2]. Tombol kanan *mouse* dapat

digunakan untuk menandai sebuah sel/kotak sebagai tempat yang menurut pemain tidak mungkin terdapat bola di situ. Ini hanya merupakan bantuan visual dan tidaka memengaruhi jalannya permainan. Pemain juga dapat meneken tombol kanan *mouse* pada bola/atom untuk menandainya sebagai "ragu-ragu" jika belum yakin dengan posisi tersebut. Ini juga hanya merupakan bantuan visual. Kemudian selain menekan kotak untuk mengisinya dengan bola/atom, pemain dapat menekan, menahan, dan menarik bola/atom dari tempat awalnya menuju kotak yang ingin diisi.

Ketika pemain sudah yakin bahwa peletakan bolanya benar, pemain bisa menekan tombol "Done!". Pemain akan diberitahu apakah konfigurasi penempatan bola/atomnya benar atau tidak, dan akan memberikan skor. Jika pemain meletakkan bola apa pun pada posisi yang salah, solusi yang benar akan ditampilkan.



***Gambar 1**. Antarmuka permainan KBlackBox, terlihat grid papan permainan dan dua bola yang harus diletakkan (Sumber: dokumentasi pribadi)*

B. *Brute Force*

Algoritma b*rute force* merupakan suatu pendekatan yang lurus atau lempang (*straightforward*) dalam penyelesaian suatu masalah [3]. Algoritma *brute force* memecahkan persoalan dengan sangat sederhana, secara langsung dan jelas karena tidak memerlukan rumus yang rumit sehingga mudah dipahami. Pada dasarnya algoritma ini sama dengan "coba selesaikan saja dengan mencoba semua kemungkinan". Biasanya kita dapat langsung menyusun suatu algoritma *brute force* karena cara berpikirnya atau alur untuk bisa sampai ke algoritma tersebut cukup jelas dan *simple.*

Untuk lebih jelasnya, contoh dari algoritma *brute force* adalah pencarian beruntun pada senarai yang berisi N buah bilangan bulat (X1, X2 , …, Xn). Program akan mencari nilai X di dalam senarai tersebut. Algoritma *brute force* yang dilakukan adalah dengan membandingkan setiap elemen senarai dengan X sampai X ditemukan atau seluruh elemen senarai sudah habis diperiksa. Ini menunjukkan bahwa program akan benar-benar mencoba seluruh kemungkinan yang ada sehingga kurang optimal khususnya pada masalah yang ukurannya (dalam kasus ini nilai N) sangat besar.

```
function pangkat(a : real, n : integer) → real
{ Menghitung a^n }

Deklarasi
  i : integer
  hasil : real

Algoritma:
  hasil ← 1
  for i ← 1 to n do
    hasil ← hasil * a
  endfor
  return hasil
```

***Gambar 2**. Algoritma brute force untuk mencari nilai suatu eksponen (Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf)*

Meski begitu, salah satu kelebihan algoritma *brute force* adalah sebuah solusi pasti ditemukan (jika persoalan memang memiliki solusi). Hal ini dapat terjadi karena algoritma mengecek seluruh kemungkinan solusi yang ada.

Algoritma *brute force* akan membangkitkan semua kemungkinan dan hanya akan berhenti jika sebuah solusi sudah ditemukan pada masalah satu solusi. Namun, pada masalah minimasi dan maksimasi, ia tidak akan berhenti sampai benar-benar semua kemungkinan dibangkitkan dan dicek. Jika ada banyak bagian dari kemungkinan solusi yang salah maka kemungkinan lain dengan modifikasi minimal pun akan tetap dicoba meskipun ada satu bagian dari kemungkinan solusi yang jelas-jelas salah.

C. *Backtracking*

Algoritma *backtracking* dapat dipandang sebagai sebuah fase di dalam algoritma DFS maupun sebagai sebuah metode pemecahan masalah yang sangkil, terstruktur, dan sistematis untuk persoalan optimasi dan non-optimasi [4]. Optimasi artinya digunakan dalam penyelesaian permasalahan optimasi, artinya algoritma ini digunakan untuk mencari solusi yang paling optimal dari semua kemungkinan solusi yang ada. Algoritma *backtracking* pertama kali diperkenalkan oleh D. H. Lehmer tahun 1950 lalu diuraikan secara umum oleh R.J Walker, Golomb, dan Baumert. Algoritma ini lebih optimal dan lebih cepat dari algoritma *brute force* karena tidak mencari seluruh kemungkinan yang ada dan melakukan *back* atau "mundur" dari simpul pencarian jika jelas bahwa simpul tersebut mengandung kesalahan yang menyebabkan tidak bisa lanjut ke anak-anak dari simpul tersebut atau memang mustahil untuk menjadi solusi. Ini membuat *backtracking* menjadi lebih efektif daripada *brute force* karena kemungkinan solusi yang sejak awal mustahil tidak akan dilanjutkan untuk dibangun sehingga tidak menyia-nyiakan waktu dan komputasi untuk pencarian yang jelas salah.

Algoritma backtracking bekerja dengan melakukan pencarian secara rekursif untuk memperoleh semua solusi yang mungkin untuk suatu masalah (atau berhenti ketika menemukan solusi jika hanya diminta satu solusi). Algoritma ini dimulai dengan memilih solusi awal (belum lengkap) kemudian mengeksplorasi semua kemungkinan perluasan dari solusi tersebut. Jika sebuah perluasan dari simpul mengarah ke solusi yang lengkap dan valid, algoritma akan mengembalikan solusi tersebut. Jika sebuah perluasan tidak mengarah pada solusi, algoritma akan kembali ke solusi parsial (simpul) sebelumnya dan mencoba perluasan yang berbeda.

Algoritma *backtracking* bekerja dengan mekanisme [5]:

1. Memilih solusi awal.
2. Menjelajahi semua kemungkinan perluasan dari solusi saat ini.
3. Jika sebuah perluasan mengarah ke sebuah solusi, solusi tersebut dikembalikan.
4. Jika perluasan tidak mengarah ke solusi, maka kembali (*backtrack*) ke solusi sebelumnya untuk mencoba perluasan yang berbeda.
5. Mengulangi langkah 2-4 sampai semua solusi yang mungkin telah dieksplorasi.

Berikut merupakan sebuah contoh algoritma *backtracking* untuk menemukan jalur terpendek melalui sebuah labirin. Masukan (*input*) program merupakan sebuah labirin yang direpresentasikan sebagai larik 2D alias sebuah matriks dengan nilai 0 merepresentasikan ruang terbuka dan 1 merepresentasikan dinding.

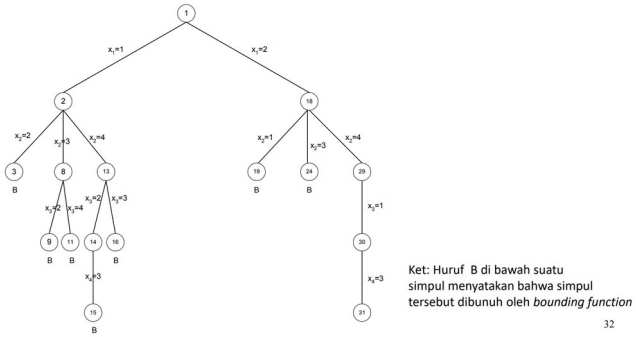
Algoritma program akan mulai dari titik awal. Kemudian untuk masing-masing dari empat arah yang memungkinkan (atas, bawah, kiri, kanan), dicoba pergerakan ke arah tersebut. Jika pergerakan ke arah tersebut mengarahkan ke simpul tujuan, maka dilanjutkan pencarian melalui jalur yang diambil tersebut. Namun, jika bergerak ke arah tersebut tidak mengarah ke simpul tujuan, maka dikembalikan ke posisi sebelumnya dan dicoba ke arah lain. Langkah-langkah tersebut diulangi hingga titik akhir tercapai atau semua jalur yang mungkin telah dijelajahi.

Algoritma *backtracking* paling baik digunakan untuk memecahkan masalah yang memiliki karakteristik berikut:

1. Ada beberapa kemungkinan solusi untuk masalah tersebut.
2. Masalahnya dapat dipecah menjadi submasalah yang lebih kecil.
3. Submasalah dapat diselesaikan secara independen.

Poin kedua dan ketiga saling berhubungan dan merupakan penyebab algoritma *backtracking* bisa “mundur” karena ada submasalah pada simpul pohon pencarian yang bisa dicek kebenarannya, jika benar maka dilanjutkan, jika salah maka mundur.

Pada contoh algoritma pencarian dengan *backtracking* pada Gambar 3, terlihat bahwa pada simpul nomor 3, program mendeteksi bahwa itu tidak akan mengarahkan ke solusi karena melanggar fungsi pembatas, maka dari itu pencarian melakukan *backtrack* yaitu mundur ke simpul ayah untuk mencoba anak yang lain, tetapi jika tidak ada anak yang lain maka naik lagi ke simpul ayah hingga ada simpul yang memiliki anak yang dapat ditelusuri. Alur pencarian tersebut terdengar seperti pencarian dengan DFS hanya saja lebih efisien.



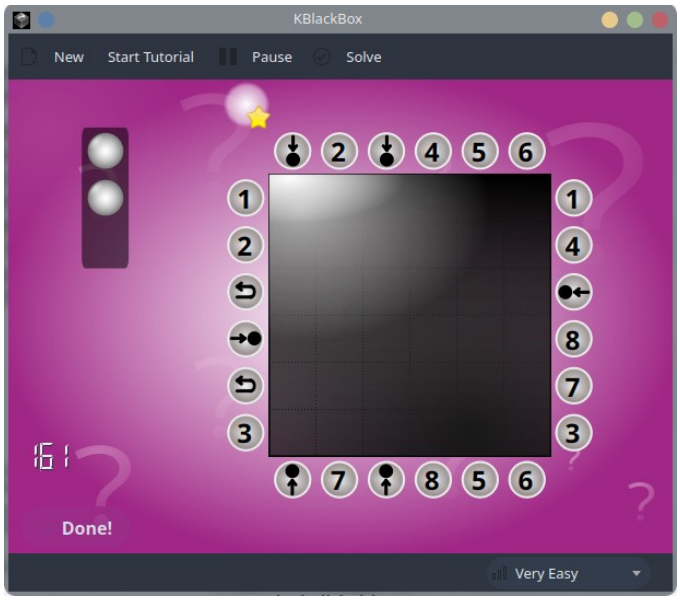
***Gambar 3**. Pohon pencarian permasalahan N buah ratu dengan algoritma backtracking*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/15-Algoritma-backtracking-(2025)-Bagian1.pdf)

Perbedaannya dengan *brute force* adalah pada *brute force*, program akan mengecek semua kemungkinan walaupun salah satu bagian dari kemungkinan solusi yang sedang dicek jelas-jelas menyebabkan kemungkinan solusi menjadi tidak valid atau salah. Namun, pada *backtracking*, kita dapat mendeteksi salah satu bagian yang menyebabkan kesalahan tersebut dan kita tidak akan melanjutkan mencari anak-anak dari kemungkinan yang sudah jelas salah tadi. Ini dapat menyebabkan pencarian menjadi jauh lebih sedikit.

III. IMPLEMENTASI

Untuk mengimplementasikan program yang dapat menyelesaikan (solver) teka-teki KBlackBox, penulis menggunakan program sederhana dengan bahasa Python karena visualisasi solusi akan lebih mudah dengan *library* matplotlib.

Cara kerja dari *solver* dijelaskan di bawah sebagai berikut. Misal kita memiliki papan permainan dengan keadaan seperti gambar berikut. Di sebelah kanan terdapat papan permainan berupa matriks/grid sedangkan di sebelah kiri terdapat bola/atom yang dapat diletakkan pada papan. Di sebelah atas, kanan, bawah, dan kiri papan terdapat sinar-sinar laser yang harus dikonfigurasikan dengan permintaan puzzle.



***Gambar 4**. Contoh keadaan awal teka-teki*
(Sumber: Dokumentasi pribadi)

Gambar tersebut menunjukkan papan permainan dengan konfigurasi sedemikian sehingga sinar nomor satu masuk dari sisi kiri papan dan langsung keluar di sisi kanan papan bagian paling atas. Hal ini berarti sinar tidak terpengaruh oleh bola/atom sama sekali sehingga tidak ada bola di depan atau disamping sinar dari masuk hingga keluar. Sedangkan sinar kedua masuk dari atas papan dan keluar dari bagian kiri papan, ini berarti pada suatu titik (kemungkinan baris kedua dan kolom kedua dari kiri atas (2,2)) sinar menemukan ada bola di diagonal depan kirinya sehingga belok ke kanan 90 derajat atau dari perspektif gambar di atas maka ia berbelok dengan berputar searah jarum jam. Meski begitu, bisa terdapat kemungkinan lain. Bisa saja terletak di sisi lain sehingga sinar dibelokkan beberapa kali sehingga tetap keluar pada kotak keuda dari atas pada sisi kiri papan. Kemudian untuk sinar paling kiri pada yang masuk dari sisi atas, ia terhalangi oleh sebuah bola/atom yang ditunjukkan oleh logo panah menabrak sebuah lingkaran. Terakhir, sinar ketiga dari atas yang masuk dari sisi kiri papan berbalik karena dipantulkan sehingga ia keluar dari tempat yang sama seperti tempat masuknya, itu ditunjukkan dengan logo panah yang bengkok dan seperti putar balik ke kanan. Selebihnya untuk sinar sisanya seperti penjesalan yang sudah digunakan untuk mendeskripsikan sinar-sinar yang sudah disebutkan sesuai logonya. Angka pada lingkaran di tepi papan berarti sinar masuk dan keluar dari dan/atau ke nomor yang sama.

A. Brute force

Pada program *solver* ini, program akan membangkitkan sebuah senarai yang berisi semua kombinasi koordinat N buah bola/atom. Jika ada 2 bola maka senarai berisi dua elemen yang merupakan set dari koordinat y dan x (nilai x membesar ke kanan dan y membesar ke bawah). Program selanjutnya akan mengambil salah satu kombinasi dan dilakukan pengecekan apakah keadaan akhir dari semua sinar memenuhi permintaan puzzle. Jika iya maka program selesai melakukan *brute force.* Namun jika tidak maka program akan melanjutkan dengan kombinasi yang lain hingga diperoleh solusi yang valid.

Algoritma ini sangat sederhana dan jelas, atau bisa dikatakan *straightforward*. Tidak ada teknik khusus dan program hanya memaksakan semua kemungkinan yang ada meskipun tidak masuk akal, itulah mengapa disebut *brute force*.

B. Backtracking

Kemudian untuk algoritma *backtracking*, program akan membangkitkan juga semua koordinat letak atom/bola yang mungkin dari ukuran papan. Hanya saja program tidak akan mencoba seluruh kemungkinan tetapi sedikit lebih cerdas. Program akan menempatkan semua bola dengan posisi berbeda dan melakukan pengecekan apakah peletakan tadi valid. Jika tidak valid maka kemungkinan peletakan yang terakhir dilakukan dimodifikasi dengan mengubah posisi bola yang terakhir diletakkan. Jika sudah semua kemungkinan dicoba maka bola sebelumnya lagi yang diubah. Ini semua dilakukan terus menerus hingga program berhasil mendapatkan solusi.

Algoritma ini harusnya lebih efisien daripada *brute force* dan DFS yang normal. Namun, setelah diteliti lebih lanjut terdapat sebuah fenomena yang membuat algoritma ini tidak lebih baik daripada sebuah DFS biasa. Hal ini disebabkan karena *nature* atau sifat dari permainan ini. Kita hanya bisa mengecek apakah solusi kita valid jika kita telah meletakkan semua bola pada papan permainan. Ini berarti kita tidak bisa melakukan pengecekan lebih awal sebelum mencapai simpul daun dari pohon pencarian. Padahal teknik itulah yang menyebabkan *backtracking* cukup efisien. Akibatnya, secara teknis kita hanyalah melakukan DFS murni tanpa trik *heuristic* apapun.

Fakta ini tidak terlihat pada pengujian awal program di mana hasilnya durasi pencarian *brute force* cukup jauh daripada pencarian *backtracking*.

```
Solving KBlackBox: N=6, atoms=2, shots=16
Branch-and-bound solution: [(3, 3), (4, 1)]
Time: 0.0016s

Verifying shots:
  N1: expected absorb, got absorb ✓
  N2: expected exit to ('W', 2), got exit to ('W', 2) ✓
  N3: expected absorb, got absorb ✓
  N4: expected exit to ('E', 2), got exit to ('E', 2) ✓
  N5: expected exit to ('S', 5), got exit to ('S', 5) ✓
  N6: expected exit to ('S', 6), got exit to ('S', 6) ✓
  E1: expected exit to ('W', 1), got exit to ('W', 1) ✓
  E3: expected absorb, got absorb ✓
  E4: expected exit to ('S', 4), got exit to ('S', 4) ✓
  E5: expected exit to ('S', 2), got exit to ('S', 2) ✓
  E6: expected exit to ('W', 6), got exit to ('W', 6) ✓
  S1: expected absorb, got absorb ✓
  S3: expected absorb, got absorb ✓
  W3: expected reflect, got reflect ✓
  W4: expected absorb, got absorb ✓
  W5: expected reflect, got reflect ✓

Solution verified: True
Trying brute force...
Brute force solution: [(3, 3), (4, 1)]
Time: 0.0089s
```

***Gambar 4**. Hasil pengujian dengan papan 6x6 dan 2 buah bola atom*
(Sumber: dokumentasi pribadi)

Pada hasil pengujian tersebut, durasi pencarian *brute force* nilainya lebih dari lima kali lipat durasi pencarian dengan *backtracking*. Namun, pada kasus dengan ukuran papan 8x8 dan jumlah bola atom adalah 4, perbedaan waktu pencarian antara *backtracking* dan *brute force* hampir sama. Berikut adalah hasil dari beberapa percobaan.

***Tabel 1**. Hasil percobaan solver dengan dua algoritma berbeda*

No.	*Backtracking*	*Brute Force*
1	1.2637s	1.3778s
2	1.3011s	1.3520s
3	1.3858s	1.4267s
4	1.3119s	1.3563s
5	1.2683s	1.3642s

Dari Tabel 1 terlihat bahwa durasi waktu pencarian tidak jauh berbeda, ini karena algoritma *backtracking* yang dilakukan pada akhirnya tetap mencoba hampir semua kemungkinan yang ada secara DFS tanpa adanya *pruning* atau penghentian cabang pencarian yang tidak menuju ke solusi.

Fakta tersebut dapat terjadi karena tidak dapat dicek apakah suatu *state* peletakan bola atom secara parsial akan menuju ke solusi. Maksudnya adalah untuk mengetahui apakah peletakan bola-bola atom menuju kepada solusi, kita perlu meletakkan semua bola terlebih dahulu alih-alih hanya sebagian bola atom.

Memang beberapa kasus khusus seperti meletakkan bola di depan tempat sinar masuk yang berekspektasi untuk keluar dari papan atau direfleksikan kembali dapat langsung kita kategorikan sebagai jalur pencarian yang salah. Namun, kasus-kasus lainnya yang merupakan bagian besar dari algoritma ini tetap tidak dapat dicek sebelum kita meletakkan semua bolanya. Sebagai contoh, misalkan sinar A berekspektasi untuk masuk dari sisi kiri urutan kedua dan keluar dari sisi kanan dari urutan kelima. Untuk kasus ini ada banyak sekali kemungkinan peletakan bola yang memungkinkan sinar dapat dibelokkan sehinga keluar dari tempat tersebut. Sedangkan kita tidak tahu bagaimana bola lain harus diletakkan dan bagaimana sinar lain akan berinteraksi dengan bola tadi. Maka dari itu untuk sebagian besar kasus kita tidak bisa langsung menebak bahwa solusi akan salah.

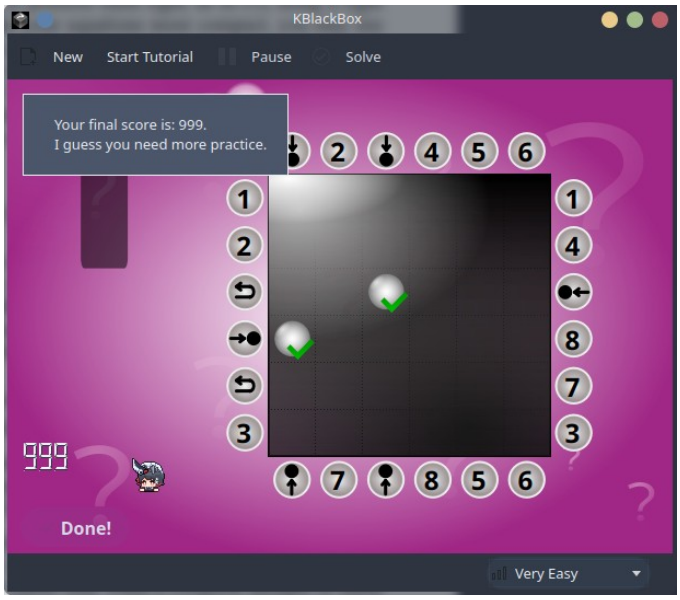
Hal tersebut di ataslah yang menyebabkan mengapa pencarian dengan *backtracking* untuk permainan teka-teki ini hampir tidak ada bedanya dengan *brute force* karena pada akhirnya kita harus mengecek semua kemungkinan yang ada.

Untuk pencarian dengan ukuran papan 8x8 dan jumlah atom 4 buah, kita masih bisa mencari solusinya dalam hitungan beberapa detik, salah satunya ditunjukkan oleh Tabel 1. Namun, untuk kasus yang lebih kompleks seperti papan ukuran 12x12 dan jumlah atom 8 buah, algoritma ini tidak bisa memberikan solusinya dalam waktu yang baik. Ini karena untuk papan 8x8 dan atom 4 buah akan dibangkitkan sebanyak C(8^2; 4) = C(64; 4) atau 635,376 kombinasi yang harus dicek semuanya. Dengan pengecekan 1000 kombinasi per mili detik, kita akan dapat menyelesaikannya dalam waktu 0,635 detik. Sedangkan untuk papan 12x12 dan atom 8 buah akan diperlukan C(144; 8) atau sekitar 3.76207936 x 10^12 kombinasi. Bahkan dengan pengecekan 1000 kombinasi per mili detik pun akan memakan waktu satu setengah bulan untuk bisa menyelesaikannya. Hal ini jelas tidak ideal karena kita sedang membuat *solver* untuk permainan KBlackBox, sebuah gim vidio. Kita tidak ingin memerlukan waktu selama itu hanya untuk menyelesaikan satu teka-teki yang bahkan bukan jenis teka-teki tersulit dalam tingkat kesulitan KBlackBox.

Maka dari itu, diperlukan suatu algoritma yang dapat menyelesaikan permainan teka-teki ini dengan algoritma yang lebih efisien. Implementasinya mungkin tetap dapat menggunakan *brute force* atau *backtracking* tetapi formulasi masalah dan lojik pengecekan serta lojik pemilihan posisi bola memerlukan metode yang jauh lebih baik agar kita bisa mengeliminasi banyak kemungkinan yang perlu ditelusuri. Diharapkan metode lojik tersebut juga dapat memberikan fungsi *bounding* yang dapat menghentikan cabang pencarian bermasalah secara efektif dan efisien sedini mungkin.

C. Solusi Valid

Meski dengan segala kekurangan algoritma ini yang disampaikan pada subbab B sebelumnya, algoritma ini tetap dapat mencari solusi untuk papan dengan ukuran 6x6 dengan 2 atom (Gambar 4), bahkan hingga papan 8x8 dengan 4 bola (Tabel 1). Berikut merupakan pembuktian solusi permainan 6x6 dengan 2 bola yang jawabannya diberikan oleh program *solver* kita yaitu (3,3) dan (4,1). Kita letakkan atom pertama di baris ketiga kolom ketiga lalu atom kedua pada baris keempat kolom pertama. Bisa dilihat bahwa jawaban diterima oleh permainan sebagai solusi valid (semua atom ditandai *checkmark*).



***Gambar 5**. Validasi solusi dari program solver dengan mengeceknya pada permainan KBlackBox*
(Sumber: dokumentasi pribadi)

D. Kompleksitas

Karena implementasi *brute force* dan *backtracking* pada makalah ini tidak jauh berbeda, bahkan hampir sama, kita asumsikan kompleksitasnya sama. Kompleksitas program dapat dihitung dari kombinasi yang perlu dibangkitkan dan perlu dicek yaitu $C(N^2, T)$ dengan N ukuran panjang sisi papan permainan dan T jumlah atom yang harus disusun. Kemudian untuk sejumlah S sinar yang masuk ke papan diperlukan $O(N^2)$ untuk mengecek apakah solusi valid. Maka dari itu kompleksitas algoritma adalah $O(C(N^2, T) \times S \times N^2)$.

IV. KESIMPULAN

Algoritma *brute force* dan *backtracking* dapat digunakan untuk menyelesaikan permainan teka-teki KBlackBox, khususnya untuk permainan yang ukuran papannya hingga 8x8 dan jumlah bola atom 4 buah. Untuk teka-teki yang lebih kompleks dari itu seperti ukuran papan 12x12 dan jumlah atom 8 buah maka diperlukan algoritma yang jauh lebih sangkil dan dapat melakukan *pruning* untuk memangkas jauh jumlah kemungkinan yang harus ditelusuri. Hal tersebut dikarenakan waktu yang diperlukan untuk mencari solusi dari permainan dengan papan 12x12 dan 8 buah atom dapat memerlukan waktu hingga lebih dari satu bulan untuk ditemukan solusinya.

Pada implementasi ini, algoritma *brute force* tidak jauh berbeda dengan algoritma *backtracking* dalam hal efisiensi dan waktu pencarian. Hal in disebabkan karena implementasi *backtracking* yang kurang sempurna sehingga sama saja dengan DFS murni. Program harus menelusuri hingga simpul daun dari pohon pencarian sebelum bisa mengecek apakah solusi valid. Ini menyebabkan program tidak bisa di-*pruning* karena tidak bisa dideteksi apakah simpul akan mengarahkan ke solusi atau justru menyesatkan.

UCAPAN TERIMA KASIH

Penulis mengucapkan banyak terima kasih kepada Tuhan Yang Maha Esa atas rahmat dan berkatnya sehingga penulis dapat menyelesaikan makalah ini dengan lancar dan tepat waktu. Penulis juga sangat berterima kasih kepada Dr. Nur Ulfa Maulidevi, S.T, M.Sc. selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma 2025 kelas 01, karena berkat ilmu dan wawasan yang beliau sampaikan dan bagikan dalam setiap kesempatan belajar mengajar di kelas sehingga penulis dapat memahami materi mata kuliah ini selama semester keempat ini. Penulis juga sangat berterimakasih juga kepada orang tua penulis juga teman-teman penulis yang memberikan semangat dan dukungan kepada penulis. Penulis berharap isu pada makalah ini dapat dibahas lebih lanjut sehingga ditemukan solusi dan analisis yang lebih baik.

REFERENSI

[1] Games Publications, "Games magazine #8", 1978. https://archive.org/details/games-8-1978-november/page/48/mode/2up. (Diakses pada 21 Juni 2025).

[2] https://apps.kde.org/id/kblackbox/. (Diakses pada 21 Juni 2025).

[3] R. Munir, "Bahan kuliah IF2211 strategi algoritma: algoritma brute force (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf. (Diakses pada 21 Juni 2025).

[4] R. Munir, "Algoritma runut-balik (backtracking) bahan kuliah IF2211 strategi algoritma". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/15-Algoritma-backtracking-(2025)-Bagian1.pdf. (Diakses pada 21 Juni 2025).

[5] https://www.geeksforgeeks.org/dsa/backtracking-algorithms/. (Diakses pada 21 Juni 2025).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Juni 2025

Fajar Kurniawan
13523027