Penyelesaian Permainan Minesweeper Menggunakan Strategi Greedy by Probability

Jericho Russel Sebastian - 13521107
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13521107@std.stei.itb.ac.id

***Abstract*—Minesweeper merupakan salah satu permainan video terpopuler di dunia komputer pribadi pada awal abad 21. Puluhan, bahkan ratusan juta pemain yang tersebar di seluruh dunia memainkan permainan ini di komputer Windows pribadi mereka. Tergolong sebagai permainan yang mudah dipelajari namun sukar dikuasai, peraturan permainan Minesweeper yang relatif sederhana menimbulkan permainan yang kompleks dalam beberapa kasus dan menitikberatkan strategi yang mendalam untuk menyelesaikannya. Maka, tak heran bahwa Minesweeper menjadi salah satu bahan perhatian di kalangan sains komputer, di mana berbagai pendekatan algoritma diciptakan dengan tujuan menyelesaikan permainan secara programatis dengan seefektif mungkin. Makalah ini membahas salah satu pendekatan algoritma yang mungkin diambil dalam membangun sebuah program yang dapat menyelesaikan sebuah permainan *Minesweeper* dengan efektif, yang didasari atas konsep peluang dan kombinatorika serta diarahkan oleh prinsip *greedy*.**

Keywords—Minesweeper, Greedy, Peluang, Kombinatorika

I. Pendahuluan

Minesweeper merupakan permainan video yang dikembangkan oleh Microsoft, yang mulanya ditujukan untuk sistem operasi buatannya, Windows [1]. Dengan puluhan, bahkan mungkin ratusan juta pemain di seluruh dunia yang memainkan permainan ini, Minesweeper terbilang sebagai salah satu permainan video terpopuler dalam sejarah permainan video untuk komputer pribadi. Selain populer dalam kalangan pemain kompetitif memperebutkan waktu tersingkat menyelesaikan sebuah permainan, Minesweeper juga mengundang perhatian dari kalangan ilmuwan komputer sejak Richard Kaye membuktikan bahwa persoalan konsistensi Minesweeper merupakan persoalan NP-*complete* [3] dan menunjukkan bahwa *infinite* Minesweeper, varian dari Minesweeper dengan papan tak terbatas, bersifat Turing-*complete* [4].

Berbagai pendekatan untuk menyelesaikan permainan Minesweeper secara programatik telah dikembangkan, seperti pendekatan *single point* dan *constraint satisfaction* [5]. Pendekatan yang umumnya digunakan melibatkan pengenalan pola angka yang memastikan sekelompok petak berisi atau tidak berisi ranjau. Namun demikian, terdapat beberapa situasi permainan di mana tidak ada pola yang dapat memastikan isi dari petak-petak tertutup, sehingga tebakan diperlukan untuk melanjutkan penyelesaian. Idealnya, tebakan yang dilakukan tidak murni acak, namun meminimasi kemungkinan ditemukan ranjau pada petak yang ditebak.

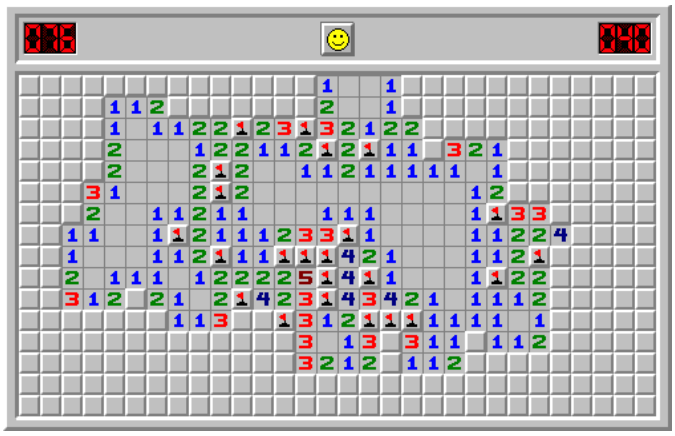
Dalam makalah ini, penulis melakukan eksplorasi terhadap salah satu pendekatan yang dapat digunakan untuk membangun algoritma penyelesaian permainan Minesweeper, menggunakan strategi *greedy* terhadap peluang sebuah petak berisi ranjau.

II. Dasar Teori

A. Permainan Minesweeper

Permainan Minesweeper merupakan permainan video yang tergolong sebagai permainan *puzzle* [1]. Permainan ini terdiri dari sebuah papan kisi (*grid*) yang mewakili sebuah ladang ranjau. Sejumlah petak dalam papan tersebut menyembunyikan sebuah ranjau.

Minesweeper dibuat oleh Robert Donner dan Curt Johnson, dua orang pegawai Microsoft. Mulanya dikenal sebagai *Mine* oleh kalangan rekan dan teman dari Donner, Minesweeper dirilis oleh Microsoft pada tanggal 8 Oktober 1990 sebagai salah satu dari enam permainan video yang termasuk ke dalam Microsoft Entertainment Pack for Windows, yang ditujukan kepada para pengguna Windows 3.0. Popularitas permainan ini melonjak pesat, yang kemudian mendorong Microsoft untuk mengintegrasikan Minesweeper sebagai bagian dari Windows 3.1 yang dirilis pada tahun 1992. Minesweeper juga dijadikan sebagai bagian dari setiap versi Windows yang dirilis selanjutnya, hingga Windows 7 yang dirilis pada tahun 2009.



Gambar 1.1 Sesi permainan Minesweeper pada papan dengan tingkat kesulitan tertinggi (Sumber: https://minesweeper.online/)

Tujuan akhir dari permainan ini adalah membuka semua petak dalam papan tanpa membuka petak ranjau. Untuk membantu pemain mencapai tujuan akhir ini, setiap petak yang dibuka menampilkan sebuah angka, yang bernilai antara 1 hingga 8, yang menunjukkan banyak ranjau yang berada di delapan petak di sekitarnya. Minesweeper juga menyediakan tiga tingkat kesulitan permainan, yaitu mudah (9×9 petak dengan 10 ranjau), menengah (16×16 petak dengan 40 ranjau), dan sulit (16×30 petak dengan 99 ranjau).

Terdapat beberapa mekanik yang dapat digunakan untuk membantu pemain mencapai tujuan akhir tersebut [2]. Pemain dapat membubuhi bendera (*flag*) pada petak-petak tertutup untuk menandai petak-petak yang dicurigai merupakan petak ranjau. Lebih lanjut lagi, jika semua petak ranjau di sekitar sebuah petak angka sudah dibubuhi bendera, petak angka tersebut dapat diklik kembali untuk membuka semua petak tertutup di sekitarnya yang tidak ditandai dengan bendera. Mekanik ini disebut sebagai *chording*, dan kerap digunakan untuk mengurangi banyak klik yang diperlukan untuk menyelesaikan permainan.

Meskipun peraturannya tergolong sederhana, Minesweeper merupakan permainan yang membutuhkan strategi mendalam agar penyelesaian dapat dicapai secepat mungkin [1]. Karenanya, Minesweeper merupakan salah satu permainan yang kerap dimainkan dalam konteks kompetitif oleh pemain-pemain dari berbagai penjuru dunia. Salah satu strategi yang banyak digunakan oleh pemain kompetitif adalah pengenalan pola-pola petak angka yang memastikan keberadaan petak kosong atau petak ranjau. Pola-pola ini umumnya dihafalkan dan dapat mempersingkat waktu penyelesaian dengan signifikan. Berbagai teknik, seperti *1.5 click*, juga digunakan untuk mengurangi banyak klik yang dilakukan.

Persoalan konsistensi Minesweeper, atau sederhananya dikenal sebagai persoalan Minesweeper, didefinisikan sebagai berikut: diberikan sebuah papan kisi yang terisi angka atau ranjau, dan sebagian petak tertutup, tentukan apakah terdapat suatu susunan ranjau pada petak-petak tertutup sedemikian hingga papan kisi memenuhi peraturan permainan Minesweeper. Hal ini bermakna bahwa setiap susunan ranjau solusi menghasilkan susunan petak angka yang terdapat pada papan kisi. Persoalan ini telah dibuktikan merupakan persoalan NP-*complete* oleh Kaye pada tahun 2000 [3].

B. Algoritma Greedy

Strategi *greedy* merupakan salah satu strategi algoritma yang digunakan untuk menyelesaikan persoalan optimasi, yaitu persoalan maksimasi dan minimasi [6]. Algoritma *greedy* tergolong sederhana dan relatif mudah diimplementasi.

Pada algoritma yang menggunakan strategi *greedy*, solusi dibentuk secara langkah per langkah. Pada setiap langkah, beberapa pilihan dievaluasi. Pilihan terbaik pada langkah tersebut diambil, dan algoritma maju ke langkah selanjutnya. Selama evaluasi dan pencarian solusi, algoritma tidak dapat mundur ke langkah sebelumnya. Maka, dasar dari strategi *greedy* adalah pemilihan optimum lokal dengan harapan bahwa langkah sisanya mengarah ke optimum global. Hal ini tidak dijamin karena algoritma *greedy* tidak mengevaluasi semua kemungkinan solusi yang ada secara menyeluruh, sehingga solusi yang dihasilkan oleh algoritma *greedy* umumnya disebut sebagai solusi suboptimum atau pseudo-optimum.

Algoritma *greedy* terdiri dari beberapa elemen, yaitu:

- Himpunan kandidat C: berisi kandidat pilihan yang akan dipilih pada setiap langkah;
- Himpunan solusi S: berisi kandidat yang sudah dipilih;
- Fungsi solusi: menentukan apakah S sudah memberikan solusi;
- Fungsi seleksi (*selection*): memilih kandidat dari C berdasarkan strategi *greedy* tertentu, umumnya bersifat heuristik;
- Fungsi kelayakan (*feasibility*): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam S atau tidak; dan
- Fungsi objektif: fungsi yang ingin dioptimumkan.

Beberapa persoalan dapat memiliki algoritma penyelesaian *greedy* yang terjamin menghasilkan solusi optimal. Optimalitas algoritma ini harus dibuktikan secara matematis.

C. Permutasi dan Kombinasi

Permutasi dari sebuah himpunan merupakan sebuah susunan dari sebagian atau seluruh elemen himpunan tersebut dengan suatu urutan tertentu. Banyak permutasi r elemen dari sebuah himpunan n elemen, ditulis sebagai P_r^n, dihitung sebagai:

$$P_r^n = \frac{n!}{(n-r)!} \tag{2.1}$$

Sedangkan, kombinasi dari sebuah himpunan merupakan sebuah susunan dari sebagian atau seluruh elemen himpunan tersebut tanpa memperhatikan urutan. Karena untuk setiap sub-himpunan r elemen dari suatu himpunan n elemen, terdapat $P_r^r = r!$ permutasi yang dapat dibuat yang berisi elemen-elemen yang sama, semua permutasi tersebut dapat dianggap sebagai satu kombinasi yang sama. Sehingga, banyak kombinasi r elemen dari himpunan n elemen, ditulis sebagai C_r^n, dihitung sebagai:

$$C_r^n = \frac{1}{r!} P_r^n \tag{2.2}$$

$$C_r^n = \frac{n!}{r!(n-r)!} \tag{2.3}$$

D. Peluang Kejadian Diskrit

Referensi [7] oleh William Feller menjelaskan definisi peluang kejadian diskrit sebagai berikut: diberikan sebuah ruang sampel diskrit $\mathfrak{S}$ berisi berbagai titik sampel E_1, E_2, …, setiap titik sampel E_i dapat dipasangkan dengan sebuah bilangan, yang disebut sebagai peluang dari E_i, dan ditulis sebagai $\mathrm{P}(E_i)$, sedemikian hingga $\mathrm{P}(E_i)$ non-negatif dan:

$$\sum_i \mathrm{P}(E_i) = P(\mathfrak{S}) = 1 \tag{2.4}$$

Peluang dari sebuah kejadian A, di mana $A \subseteq \mathfrak{S}$, merupakan jumlah dari peluang masing-masing titik sampel di dalamnya, sehingga:

$$0 \leq \mathrm{P}(A) \leq 1 \tag{2.5}$$

Peluang $\mathrm{P}(A)$ bernilai 0 bermakna bahwa kejadian A tidak mungkin terjadi, karena tidak ada titik sampel yang mungkin muncul. Sebaliknya, peluang $\mathrm{P}(A)$ bernilai 1 bermakna bahwa kejadian A pasti terjadi, karena A berisi semua titik sampel dalam $\mathfrak{S}$ yang mungkin terjadi.

Jika diberikan dua kejadian $A_1 \subseteq \mathfrak{S}$ dan $A_2 \subseteq \mathfrak{S}$, peluang kedua kejadian terjadi bersamaan adalah jumlah dari peluang masing-masing titik sampel yang berada pada irisan A_1 dan A_2. Sedangkan, peluang setidaknya salah satu dari kedua kejadian tersebut terjadi adalah jumlah dari peluang masing-masing titik sampel yang berada pada gabungan A_1 dan A_2:

$$\mathrm{P}(A_1 \cup A_2) = \mathrm{P}(A_1) + \mathrm{P}(A_2) - \mathrm{P}(A_1 \cap A_2) \tag{2.6}$$

Jika semua titik sampel dalam $\mathfrak{S}$ memiliki peluang seragam, maka peluang setiap kejadian A dalam $\mathfrak{S}$ adalah:

$$\mathrm{P}(A) = \frac{|A|}{|\mathfrak{S}|} \tag{2.7}$$

III. Penyelesaian Permainan Minesweeper Menggunakan Strategi Greedy by Probability

A. *Penyelesaian Permainan Minesweeper*

Proses penyelesaian permainan Minesweeper merupakan sebuah proses perulangan dari dua tahap: menentukan petak-petak aman dan petak-petak ranjau pada papan, dan memilih petak selanjutnya yang akan dibuka berdasarkan petak-petak yang telah diperiksa. Kedua langkah ini menyusun dasar dari sebagian besar variasi algoritma penyelesaian Minesweeper.

Pembeda utama dari algoritma-algoritma penyelesaian yang ada adalah metode penentuan petak aman dan ranjau. Pendekatan yang berbeda menggunakan metode yang berbeda, namun idealnya memberikan hasil yang optimal dan sama untuk papan yang sama.

Makalah ini mengeksplorasi metode penentuan petak aman dan ranjau berdasarkan peluang petak tersebut berisi ranjau.

B. *Perhitungan Peluang Petak Beranjau*

Diberikan sebuah papan Minesweeper berukuran $r \times c$ (berisi $n = rc$ petak) dan terdapat x ranjau yang tersembunyi di dalamnya, sedemikian hingga $x < n$.

Asumsikan bahwa belum ada petak yang dibuka. Banyak susunan ranjau yang dapat dibuat pada papan tersebut adalah C_x^n. Dari setiap susunan tersebut, untuk setiap petak tertutup, terdapat C_{x-1}^{n-1} susunan di mana petak tersebut berisi ranjau. Maka, peluang sebuah petak tertutup i berisi ranjau dalam kondisi ini adalah p_i sedemikian hingga:

$$p_i = C_{x-1}^{n-1} / C_x^n$$

$$p_i = \frac{(n-1)!}{(x-1)!(n-x)!} \times \frac{x!(n-x)!}{n!}$$

$$p_i = \frac{x}{n} \tag{3.1}$$

Jika terdapat petak-petak angka yang terbuka, maka terdapat kondisi yang harus dipenuhi oleh susunan ranjau, yaitu bahwa susunan tersebut harus menghasilkan susunan petak angka yang sama dengan susunan petak angka yang ditampilkan pada papan. Dengan demikian, banyak susunan ranjau yang mungkin dibuat pada n' petak tertutup akan kurang dari C_x^n:

$$p_i < \frac{x}{n} \tag{3.2}$$

Jika sebuah petak angka bernilai v bertetangga dengan tepat v buah petak tertutup, maka semua susunan ranjau yang sah pada papan pasti mengandung ranjau pada semua petak tertutup tersebut. Dengan demikian, nilai p_i untuk petak-petak tersebut adalah 1.

Selanjutnya, diketahui bahwa papan berisi petak-petak yang ditandai bendera, dengan asumsi bahwa setiap petak yang ditandai bendera pasti berisi ranjau. Jika suatu petak angka bertetangga dengan petak-petak bendera yang banyaknya sama dengan angka yang ditampilkan oleh petak tersebut, maka semua susunan ranjau yang sah harus berisi ranjau di petak-petak bendera tersebut, dan tidak berisi ranjau di petak-petak yang tidak ditandai bendera. Dengan kata lain, untuk setiap petak tertutup i yang bertetangga dengan petak angka tersebut, peluang i berisi ranjau adalah:

$$p_i = 1, \text{ jika } i \text{ ditandai bendera} \tag{3.3}$$

$$p_i = 0, \text{ jika } i \text{ tidak ditandai bendera} \tag{3.4}$$

Menggunakan aturan-aturan di atas, berdasarkan sebuah papan permainan Minesweeper, dapat dibuat sebuah matriks yang seukuran dengan papan tersebut, di mana setiap elemen dalam matriks bernilai p_i. Matriks ini kemudian dapat digunakan untuk menentukan petak yang paling aman untuk dibuka.

C. *Pembangunan Algoritma Penyelesaian Menggunakan Strategi* Greedy by Probability

Algoritma *greedy* yang digunakan dalam penyelesaian yang akan dibahas dalam makalah ini terdiri dari elemen-elemen berikut:

- Himpunan kandidat C: matriks berisi nilai p_i untuk setiap petak papan; matriks ini akan dibangun pada awal setiap langkah;
- Himpunan solusi S: himpunan berisi petak-petak yang dibuka pada setiap langkah;
- Fungsi solusi: bernilai benar jika permainan sudah berakhir, baik dengan kondisi menang ataupun kalah;
- Fungsi seleksi: pilih petak pada papan yang nilai p_i-nya terkecil; pilih salah satu secara acak jika ada lebih dari satu petak yang memenuhi;

- Fungsi kelayakan: bernilai benar jika petak yang dipilih merupakan petak tertutup; dan
- Fungsi objektif: menyelesaikan permainan dengan banyak langkah seminimal mungkin, tanpa menggunakan *chording* dan tidak menghitung pemasangan bendera.

Pembangunan matriks C merupakan tahap paling kompleks dalam algoritma ini. Matriks C dibangun dengan mengiterasi setiap kemungkinan susunan ranjau yang dapat dibuat pada papan, dan untuk setiap petak tertutup i, mencatat banyak susunan ranjau di mana i berisi ranjau.

Berikut merupakan *pseudocode* yang menggambarkan implementasi naif dari algoritma pembangunan matriks C:

```
function hitungPeluang(papan: Papan) -> real[][]
KAMUS LOKAL
p: real[][]
c: integer[][]
count: integer
i, j: integer
cRanjau: Petak[]
petak: Petak

ALGORITMA
{ Inisialisasi matriks }
i traversal [0..papan.nBaris] do
    j traversal [0..papan.nKolom] do
        c[i][j] <- 0

{ Iterasi setiap kombinasi ranjau yang mungkin }
count <- 0
cRanjau traversal KOMBINASI(papan.petakTertutup,
papan.nRanjauTersisa) do
    { Jangan hitung jika susunan tidak sah }
    if not papan.valid(cRanjau) then continue

    { Inkremen nilai counter setiap petak }
    petak traversal cRanjau do
        c[petak.bar][petak.kol]++
    count++

{ Hitung nilai p_i masing-masing petak }
i traversal [0..papan.nBaris] do
    j traversal [0..papan.nKolom] do
        p[i][j] <- c[i][j] / count

-> p
```

Pada algoritma di atas, setiap kemungkinan susunan ranjau M berisi x' ranjau di atas papan dibangkitkan dan diperiksa keabsahannya terhadap papan permainan. Jika susunan sah, maka setiap petak mencatat susunan tersebut sebagai salah satu susunan di mana petak tersebut berisi ranjau. Nilai p_i kemudian dihitung sebagai rasio antara banyak susunan sah di mana petak i berisi ranjau terhadap banyak susunan sah secara keseluruhan.

Algoritma ini menghasilkan nilai p_i sesungguhnya dari setiap petak pada papan permainan. Namun, kekurangan terbesar dari algoritma ini adalah proses pembangkitan susunan ranjau yang akan memakan waktu sangat lambat untuk papan yang jumlah petak tertutupnya banyak; pemeriksaan terhadap papan permainan Minesweeper dengan tingkat kesulitan termudah, yang berisi 81 petak dan 9 ranjau, akan membangkitkan $C_9^{81} = 260887834350$ buah susunan ranjau pada langkah pertama penyelesaian.

Salah satu cara untuk mempercepat proses pembangkitan adalah dengan membangkitkan cukup susunan ranjau pada petak-petak tertutup yang bertetangga dengan petak terbuka saja, atau disebut sebagai petak-petak *frontier*. Hal ini karena petak-petak tertutup yang tidak bertetangga dengan petak terbuka tidak memiliki *constraint* apapun yang membatasi nilai p_i-nya, terlepas dari banyak ranjau pada papan yang belum ditemukan. Dengan demikian, pembangkitan susunan ranjau cukup dilakukan pada petak-petak *frontier*. Sedangkan, susunan ranjau pada petak-petak tertutup lainnya diabaikan, dan nilai p_i-nya dapat disimpulkan dari banyak ranjau yang tersisa.

Misal terdapat i_f petak tertutup *frontier*. Karena hanya sebagian ranjau saja yang termasuk ke dalam susunan ranjau *frontier*, susunan ranjau harus dibangkitkan untuk beberapa kemungkinan banyak ranjau *frontier*. Banyak ranjau *frontier* x_f bernilai positif dan dibatasi oleh dua nilai, sebagai berikut:

- Banyak ranjau *frontier* tidak lebih dari banyak petak tertutup *frontier* dan banyak ranjau yang tersisa. Hal ini karena petak-petak tertutup *frontier* tidak mungkin menampung lebih banyak ranjau dari petak yang tersedia, dan banyak ranjau *frontier* tidak mungkin lebih dari banyak ranjau secara keseluruhan.
- Banyak ranjau *frontier* tidak kurang dari selisih banyak ranjau yang belum ditemukan dengan banyak petak tertutup non-*frontier*. Hal ini karena jumlah dari banyak ranjau yang tersembunyi pada petak tertutup *frontier* dan banyak ranjau yang tersembunyi pada petak tertutup non-*frontier* pasti sama dengan banyak ranjau yang tersisa.

Untuk setiap nilai x_f yang memenuhi kedua kondisi di atas, sebuah susunan ranjau M_f dibangkitkan. Susunan ini ditambahkan dengan susunan ranjau yang sudah ditemukan, dan susunan hasilnya diperiksa keabsahannya. Jika susunan tersebut sah, maka banyak susunan ranjau secara keseluruhan yang mengandung susunan ranjau *frontier* tersebut dihitung sebagai berikut: jika M_nf merupakan himpunan semua susunan ranjau non-*frontier* dan i_nf banyak petak tertutup non-*frontier*, maka banyak susunan ranjau dalam M_nf sama dengan:

$$|M_\mathrm{nf}| = C_{x'-i_\mathrm{f}}^{i_\mathrm{nf}} \tag{3.5}$$

Hal ini bermakna bahwa untuk setiap susunan M_f yang mungkin dibuat, terdapat sebanyak $|M_\mathrm{nf}|$ susunan x' ranjau yang juga dapat dibuat, sedemikian hingga $|M_\mathrm{nf} \cup M_\mathrm{f}| = x'$. Perhatikan bahwa $M_\mathrm{nf} \cup M_\mathrm{f}$ sama dengan M, namun pembangkitan pola hanya dibutuhkan untuk M_f, karena keabsahan M hanya ditentukan oleh ranjau-ranjau yang berada dalam M_f. Dengan demikian, pembangkitan susunan ranjau menjadi lebih efisien.

Untuk setiap susunan M_f yang dibangkitkan, banyak kemunculan setiap petak tertutup non-*frontier* dalam semua kemungkinan M dapat dihitung sebagai:

$$k_{\text{nf}} = C^{i_{\text{nf}}-1}_{x'-i_{\text{f}}-1} \quad (3.5)$$

Berikut merupakan *pseudocode* yang menggambarkan implementasi teroptimasi dari algoritma yang sama:

```
function hitungPeluang(papan: Papan) -> real[][]
KAMUS LOKAL
p: real[][]
c: integer[][]
count, cSah, xfMin, xfMax: integer
x, xf, Mnf, knf, If, Inf: integer
i, j, r: integer
F, NF, M: Petak[]
petak: Petak

ALGORITMA
{ Inisialisasi matriks }
i traversal [0..papan.nBaris - 1] do
    j traversal [0..papan.nKolom - 1] do
        c[i][j] <- 0

{ Inisialisasi variabel }
F <- papan.petakTertutupFrontier
NF <- papan.petakTertutupNonfrontier
If <- F.Length
Inf <-NF.Length
x <- papan.nRanjauTersisa

{ Hitung batas nilai x_f }
xfMin <- max(0, x − Inf)
xfMax <- min(If, x)

{ Iterasi setiap nilai x_f yang mungkin }
count <- 0
xf traversal [xfMin..xfMax] do
    { Hitung nilai |M_nf| dan k_nf }
    Mnf <- C(Inf, x - If)
    Knf <- C(Inf − 1, x − If - 1)

    { Iterasi setiap kombinasi ranjau frontier
yang mungkin pada petak-petak frontier }
    cSah <- 0
    cRanjau traversal KOMBINASI(F, x) do
        { Tambahkan ranjau yang sudah ditemukan
ke dalam susunan }
        M <- cRanjau ∪ papan.petakFlagged

        { Jangan hitung jika susunan tidak sah }
        if not papan.valid(M) then continue

        { Inkremen counter petak frontier }
        petak traversal cRanjau do
            r <- c[petak.bar][petak.kol]
            c[petak.bar][petak.kol] <- r + Mnf
        cSah++
        count <- count + Mnf

{ Inkremen counter petak nonfrontier }
    petak traversal NF do
        r <- c[petak.bar][petak.kol]
        c[petak.bar][petak.kol] <- r + Knf*cSah

{ Hitung nilai p_i masing-masing petak }
i traversal [0..papan.nBaris] do
    j traversal [0..papan.nKolom] do
        p[i][j] <- c[i][j] / count

-> p
```

Maka, algoritma penyelesaian papan secara keseluruhan menggunakan strategi *greedy* adalah sebagai berikut:

```
procedure solve(papan: Papan)
KAMUS LOKAL
C: real[][]
minP: real
next: <int bar, int kol>[]
selected: <int bar, int kol>
i, j: integer

ALGORITMA
{ Ulangi hingga permainan selesai }
while not papan.selesai do
    { Bangun matriks C }
    C <- hitungPeluang(papan)

    { Iterasi setiap petak, cari p terkecil }
    minP <- 1
    i traversal [0..papan.nBaris - 1] do
        j traversal [0..papan.nKolom - 1] do
            { Tandai petak yang pasti berisi
ranjau dengan bendera }
            if C[i][j] = 1 then
                papan.bendera(i, j)

            { Catat petak dengan p terkecil }
            if C[i][j] < minP then
                minP <- C[i][j]
                next <- {}

            { Masukkan petak ke kumpulan }
            if C[i][j] = minP then
                next <- next ∪ {<i, j>}

    { Pilih salah satu petak dalam kumpulan,
jadikan sebagai pilihan langkah ini }
    selected <- RANDOMCHOICE(next)
    papan.klik(selected.bar, selected.col)

{ Permainan selesai: tunjukkan hasil }
if papan.status = MENANG then
    print("Menang!")
else
    print("Kalah.")
```

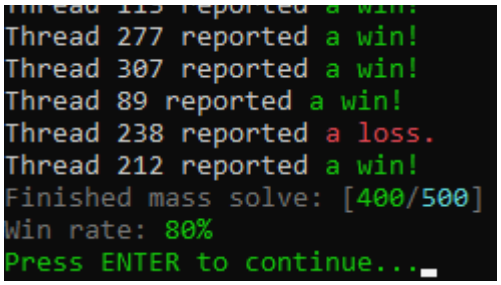
D. Implementasi dan Pengujian

Penulis menuangkan implementasi algoritma di atas ke dalam sebuah program sederhana yang ditulis dalam bahasa C#. Program menerima masukan ukuran papan permainan dan banyak ranjau yang tersembunyi untuk membangun papan permainan yang akan diselesaikan. Program juga menerima masukan banyak *run* uji yang akan dilakukan.

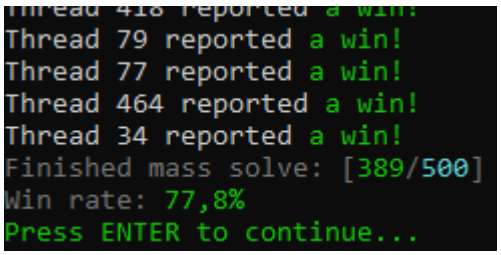
Pengujian dilakukan menggunakan konfigurasi standar permainan Minesweeper pada tingkat kesulitan termudah, yaitu papan berukuran 9×9 petak dan berisi 10 ranjau. Pengujian dilakukan dalam dua *batch* yang berisi 500 *run*, dan jumlah *run* yang berakhir dengan kemenangan dicatat. Di akhir pengujian, program menampilkan rasio menang terhadap banyak *run* yang

dilakukan untuk menunjukkan efektivitas algoritma penyelesaian yang dibuat.

Berikut adalah hasil pengujian yang dijalankan:



Gambar 3.1 Hasil Pengujian Algoritma, *Batch* 1

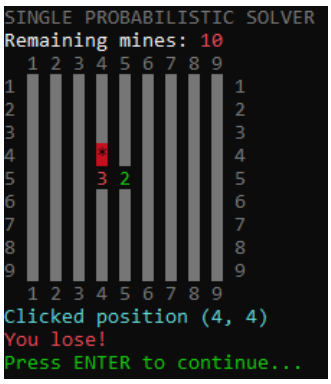


Gambar 3.2 Hasil Pengujian Algoritma, *Batch* 2

Pengujian *batch* pertama menghasilkan rasio menang sebesar $400:500$, sedangkan pengujian *batch* kedua menghasilkan rasio menang sebesar $389:500$. Secara keseluruhan, pengujian dilakukan sebanyak 1000 *run* dan menghasilkan rasio menang sebesar $789:1000$ atau 78,9 persen.

Hasil pengujian di atas menunjukkan bahwa algoritma dapat memenangkan kira-kira 4 dari setiap 5 permainan Minesweeper pada tingkat kesulitan termudah. Meski angka ini terlihat relatif tinggi, rasio menang umum yang diterima pada tingkat kesulitan termudah untuk pemain berpengalaman adalah di atas 90 persen.

Salah satu kemungkinan penyebab dari rendahnya rasio menang relatif terhadap rasio teoritis adalah penebakan petak pada awal permainan. Terdapat kasus di mana petak-petak pertama yang dibuka oleh algoritma merupakan petak angka, bukan petak nol yang dapat memberikan lebih banyak informasi untuk membuat penilaian yang lebih baik.



Gambar 3.3 Kasus Petak-petak Pertama Bukan Petak Nol

Strategi yang umumnya digunakan oleh pemain berpengalaman ketika memulai permainan adalah membuka petak-petak sudut. Hal ini karena pada awal permainan, setiap petak i dengan k tetangga memiliki peluang bernilai nol sebagai berikut [5]:

$$\mathrm{P}(i=0) \approx \left(1-\frac{x}{n}\right)^{k+1} \tag{3.6}$$

Nilai peluang ini minimum ketika k minimum. Nilai k terkecil yang mungkin adalah 3, yaitu ketika i adalah petak sudut. Hal ini menunjukkan bahwa petak sudut merupakan petak yang paling mungkin bernilai nol di awal permainan, dan algoritma seharusnya memulai permainan berdasarkan informasi ini.

IV. Kesimpulan

Meskipun diperlukan pengembangan lebih lanjut, algoritma yang dibahas dalam makalah ini memiliki performa yang cukup baik. Ini menunjukkan bahwa pendekatan penyelesaian permainan Minesweeper secara *greedy* merupakan pendekatan yang layak untuk sebagian besar kasus permainan.

Efisiensi algoritma masih terbilang buruk dalam keadaan di mana terdapat banyak petak *frontier* di atas papan permainan, yang mengakibatkan banyaknya susunan ranjau yang dibangkitkan. Masalah ini dapat dimitigasi dengan optimasi seperti mengelompokkan petak-petak tertutup ke dalam 'pulau-pulau' yang dapat diperiksa secara terpisah dan konkuren.

Pendekatan *greedy by probability* untuk menyelesaikan permainan Minesweeper melibatkan banyak bidang studi, seperti teori peluang dan kombinatorika. Meskipun pendekatan ini bukan pendekatan paling efektif dan efisien sejauh ini, pendekatan ini termasuk salah satu pendekatan edukasional yang mendorong pemahaman teoritis dan praktis dalam memecahkan masalah.

V. Ucapan Terima Kasih

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa; oleh karena kasih karunia dan penyertaan-Nya penulis dapat menyelesaikan makalah berjudul "Penyelesaian Permainan Minesweeper Menggunakan Strategi *Greedy by Probability*" ini dengan baik. Penulis mengucapkan terima kasih kepada semua pihak yang terlibat dalam proses penyusunan makalah ini, yaitu:

1. Para dosen pengajar mata kuliah IF2211 Strategi Algoritma, terutama Bapak Dr. Rinaldi Munir, M.T. selaku dosen pengajar mata kuliah IF2211 kelas K01, atas bimbingan dan ilmu yang dibagikan kepada penulis melalui kegiatan perkuliahan,
2. Kedua orang tua penulis, atas dorongan dan dukungan kepada penulis senantiasa,
3. Para sahabat dan teman penulis, atas dukungan dan kontribusi dalam penyusunan dan penyempurnaan makalah ini,
4. Para penulis jurnal, artikel, dan buku, yang karyanya penulis jadikan sebagai sumber informasi di sepanjang proses penulisan.

Akhir kata, penulis mengucapkan terima kasih kepada pembaca. Penulis memohon maaf bila ada kesalahan, baik dalam penulisan maupun muatan. Besar harapan penulis bahwa makalah ini dapat menjadi manfaat bagi pembaca.

DAFTAR PUSTAKA

[1] Edwards, Benj. 2021. *30 Years of 'Minesweeper' (Sudoku with Explosions).* https://www.howtogeek.com/693898/30-years-of-minesweeper-sudoku-with-explosions/ diakses pada 21 Mei 2023, pukul 23:30 WIB.

[2] *How to Play Minesweeper*. https://minesweepergame.com/strategy/how-to-play-minesweeper.php diakses pada 21 Mei 2023, pukul 23:30 WIB.

[3] Kaye, Richard. 2000. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2), 9-15.

[4] Kaye, Richard. 2000. *Infinite Versions of Minesweeper are Turing-complete*. University of Birmingham. https://web.mat.bham.ac.uk/R.W.Kaye/minesw/infmsw.pdf

[5] Becerra, David J. 2015. *Algorithmic Approaches to Playing Minesweeper*. Skripsi Sarjana, Harvard College. http://nrs.harvard.edu/urn-3:HUL.InstRepos:14398552

[6] Munir, Rinaldi. 2021. *Algoritma Greedy (bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf diakses pada 22 Mei 2023, pukul 10:45 WIB.

[7] Feller, William. 1968. *An Introduction to Probability Theory and Its Applications (Vol. 1), 3rd Edition*. Wiley.

LAMPIRAN

Kode sumber program yang dibuat untuk mengetes algoritma penyelesaian dapat dilihat dan diunduh pada *repository* yang diacu oleh tautan berikut:

https://github.com/JerichoFletcher/Minesolver

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 Mei 2023

Jericho Russel Sebastian – 13521107