Implementasi Algoritma Heuristik untuk Menyelesaikan Minesweeper

Hafizh Afkar Makmur/13514062
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
hafizhmakmur@gmail.com

***Abstract*—Banyak cara untuk menyelesaikan permainan Minesweeper di antaranya adalah metode Brute Force, Heuristik, Probabilitas, dan menghitung jumlah ranjau. Pada kesempatan ini penulis mengimplemntasikan metode Heuristik**

Keywords—Minesweper, heuristic, game solver

I. PERKENALAN GAME MINESWEEPER

Permainan Minesweeper adalah sebuah permainan untuk satu pemain. Permainan ini terdari sebuah "ladang ranjau" yang merupakan banyak tombol dalam sebuah bidang yang beberapa di antaranya mengandung ranjau. Pemain kalah ketika pemain mengklik tombol ini. Pada tombol yang tidak mengandung ranjau, ada sebuah angka yang menunjukkan berapa ranjau yang berada di sekitarnya. Pemain menang ketika pemain sudah membuka semua tombol yang tidak mengandug ranjau.

Gambar 1 : Contoh Permainan Minesweeper

Pada gambar terlihat beberapa unsur utama dalam sebuah permainan Minesweeper. Pertama, bidang ranjau yang terlihat berwarna hijau. Kedua, penghitung waktu di bagian bawah untuk mencatat waktu penyelesaian permainana. Ketiga, angka pada bidang ranjau yang menunjukkan jumlah ranjau yang berada di sekitar angka tersebut. Keempat, bendera yang ditujukan untuk membantu pemain menandai tombol yang didiuga mengandau ranjau. Kelima, jumlah ranjau tersisa di bagian bawah yang menunjukkan jumlah ranjau dikurangi jumlah bendera yang sudah diberikan.

II. TEKNIK PENYELESAIAN MINESWEEPER

A. Brute Force

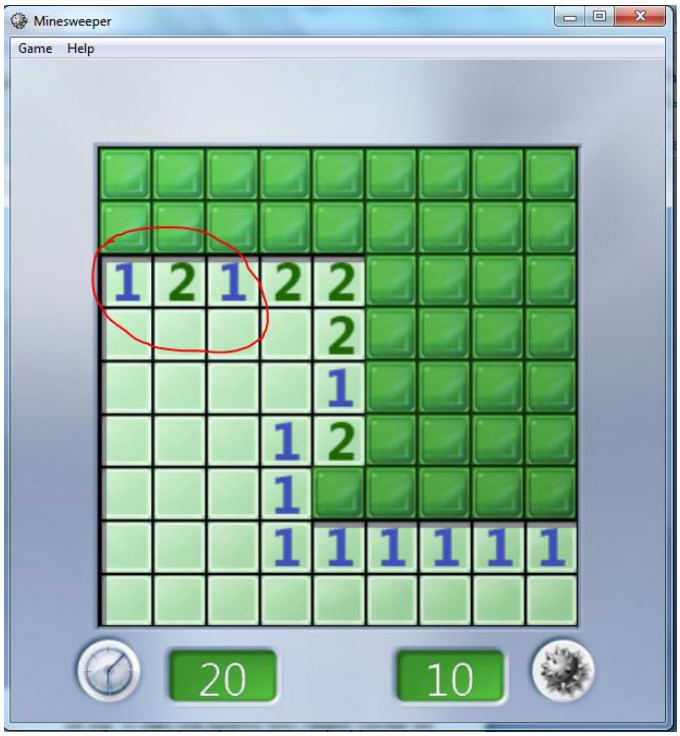
Teknik paling naïf dalam mencoba menyelesaikan permainan ini adalah dengan membuka semua kotak secara asal sampai permainan usai. Teknik ini merupakan teknik terburuk untuk menyelesaikan permainan namun mungkin menjadi satu-satunya cara saat tidak ada lagi petunjuk lebih lanjut untuk menyelesaikan permainan.

B. Teknik Heuristik

Teknik ini memanfaatkan info dari angka yang sudah terbuka untuk menentukan dengan pasti letak ranjau-ranjau dengan pasti. Salah satu contohnya adalah dengan melihat angka 1 yang hanya memiliki 1 kotak yang belum terbuka di sekitarnya. Kita bisa menebak dengan keyakinan 100% bahwa kotak yang tersisa itu berisi ranjau. Begitu juga dengan angka 2, 3, 4, dan seterusnya. Kasus-kasus tersebut adalah kasus yang paling mudah diselesaikan dengan cara ini.

Kasus lain yang bisa diselesaikan dengan cara heuristic ini adalah dengan melihat beberapa pola-pola khusus pada suatu barisan angka seperti pola 1-2-1 yang berbaris lurus seperti terlihat pada gambar 2. Pola 1-2-1 dengan tiga kotak yang belum terbuka di depannya hanya memliki satu penyelesiaan yaitu bahwa ada satu ranjau persis di depan angka 1 dan sebuah ranjau lagi persis di depan angka 1 yang lainnya. Penyelesaian seperti itu mungkin tidak terlalu terlihat jelas pada pandang pertama namun orang yang sudah sering

memainkan permainan ini bisa melihat langsung solusi seperti itu. Namun, tetap saja ada kasus dimana angka-angka itu tidak dapat memberikan informasi dengan pasti mengenai letak ranjau lain (atau letak yang tidak mungkin ada ranjau lain) .



Gambar 2. Contoh kasus Heuristik

C. Brute Force (Ranjau)

Cara lebih lanjut yang bisa terlihat lebih cocok untuk implementasi pada komputer adalah algoritma brute-force namun untuk menghitung ranjau. Cara ini berkisar pada mengenumerasi semua kemungkinan peletakan ranjau yang valid dan kemudian melihat tempat yang merupakan ranjau pada 100% kasus. Cara ini mrirp dengan cara heuristic namun dengan pendekatan yang berbeda karena jika heuristik berfokus pada angka, makan cara ini berfokus pada ranjau. Juga seperti heurisitik, belum tentu pada setiap enumerasi ditemukan letak yang 100% pasti (atau 0%) memiliki bom.

D. Probabilitas

Cara ini merupakan cara paling terakhir yang bisa digunakan untuk menebak letak bom ketika informasi yang ada belum cukup lagi, cara ini sebenarnya hanyalah optimasi dari cara pertama yaitu membuka sebuah kotak yang belum dibuka sebelumnya namun tanpa kepastian bahwa kotak itu tidak mengandung bom. Cara ini mengoptimasi cara itu dengan menebak kotak dengan kemungkinan tertinggi untuk tidak memiliki bom sehingga bisa dibuka (atau kemungkinan tertinggi memiliki bom sehingga tidak dibuka) kemudian mengeksekusi tebakan itu.. Cara ini tidak menjamin kemenangan permainan namun memungkinkan persentase kemenangan yang lebih tinggi pada waktu yang lama. Probabilitas kemungkinan bisa dibangkitkan melalui cara heuristic yaitu dengan melihat angka-angka dan menghitung probabilitas. Probabilitas juga bisa dihitung dengan cara brute-force ranjau yaitu dengan melihat persentase setiap kotak berisi ranjau pada semua enumerasi. Cara ini bisa tetap menyisakan kebuntuan ketia ditemukan beberapa kotak dengan kemungkinan sama persis seperti 50%-50% atau 25%-25%-25%-25%. Jika hal itu terjadi, maka langkah tersisa hanyalah brute-force seperti pada langkah pertama dengan membuka sembarang kotak kemudian mencoba menebak lagi dengan cara lain ketika ternyata kotak yang dibuka bukanlah bom. Namun, jika permainan hampir berakhir, ada satu cara lagi yang tidak terlalu efektif pada kebanyakan waktu namun efektif di akhir permainan yaitu:

E. Menghitung jumlah ranjau tersisa

Langkah ini harusnya sudah tertutupi pada metode ketiga yang sudah mengenumerasi dengan menggunakan hanya jumlah ranjau yang tersisa. Ada beberapa kasus dimana tanpa mengetahui jumlah ranjau tersisa, maka kemungkinan penyelesaian sebuah permainan hanyalah 50%. Namun, dua kemungkinan itu membutuhkan jumlah ranjau yang berbeda. Dengan melihat jumlah ranjau yang tersisa, dapat diputuskan dengan ketepatan 100% letak ranjau yang tersisa. Namun, kasus ini tidak terlalu sering terjadi sehingga tidak terlalu bisa diandalkan untuk terus-menerus digunakan untuk akhir permainan. Kasus umum akhir permainan yang terjadi biasanya bisa diselesaikan dengan cara-cara sebelumya atau tidak bisa siselesaikan sama sekali karena 50% kemungkinan yang sudah disebutkan dan semua kemungkinan memliki jumlah ranjau yang sama.

Pada kesempatan ini, penulis akan mengimplementasikan metode heuriistik untuk menyelsaikan permainan ini dengan memperhitungkan cara brute-force untuk kasus-kasus tertentu. Penulis tidak akan menggunakan program permainan minesweeper yang sudah ada di beberapa system operasi seperti Windows atau online namun penulis akan membuat program permainan sendiri untuk memudahkan pembuatan program penyelesai Minesweeper. Program permainan maupun program penyelesai menggunakan bahasa Java.

III. PERSIAPAN PROGRAM PERMAINAN MINESWEEPER

Seperti sudah disinggung sebelumnya, penulis membuat program sendiri untuk membantu implementasi program penyelesai permainan Minesweeper/. Program ini pada dasarnya adalah program yang menyimpan setiap ranjau yang ada dalam sebuah bidang dan juga setiap angka yang mennjukkan jumlah ranjau yang berada di sekitar angka tersebut. Program permainan ini meniru program permainan Minesweeper yang dibuat pada Windows 7.

A. Desain Kelas

Rancangan kelas untuk permainan ini adalah sebagai berikut :

```
1.  import java.util.List;
2.  import java.util.ArrayList;
3.
4.  class Board {
5.      private int[][] board = null;
6.      private boolean[][] landMine = null;
7.      private Cell start = null;
8.      private int height;
9.      private int width;
10.     private int jumlahMine;
11.
12.     public Board(int h,int w, Cell s, int n) {

13.
14.
15.     private boolean isMined(int r, int c)  {
16.     }
17.
18.     private boolean isMined(Cell cell)  {
19.     }
20.
21.     public Cell getStartCell() {
22.     }
23.
24.     public int getHeight() {
25.     }
26.
27.     public int getWidth() {
28.     }
29.
30.     public int getMine() {
31.     }
32.
33.
34.     public void testMine(int r, int c)  throws
     mineExplodedException {
35.     }
36.
37.     public void testMine(Cell cell)  throws mi
    neExplodedException {
38.
39.     }
40.
41.     public String toString() {
42.
43.     }
44.
45.     public int getAdjacent(Cell cell) {
46.
47.     }
48.
49.     public List<Cell> getAdjacentCell(Cell cel
    l) {
50.
51.     }
52. }
```

Bagian yang patut diperhatikan dari desain kelas tersebut adalah penggunaan beberapa kelas lain yaitu Cell dan mineExplodedException. Kelas Cell digunakan sebagai penyimpan koordinat dari sebuah kotak dalam board dan kelas mineExplodedException digunakan untuk menghentikan permainan secara instan. Properti mineExplodeException memungkinkan program untuk menghentikan program penyelesai secara paksa jika salah satu ranjau terinjak. mineExplodedException tidak mempunyai properti special dan hanya berfungsi untuk mengirimkan pesan „Game Over „ sebagai Exception. Sebaliknya, kelas Cell memiliki beberapa metode khusus untuk membantu kelas yang memakainya di antaranya metode standar seperti getter dan setter dan metode equals untuk membantu operasi yang membutuhkan perbandingan seperti akan dibicarakan di program penyelesai.

B. Implementasi Kelas Board

Terlebih dulu dijelaskan sebuah metode pada board yang akan sangat membantu baik program permainan maupun program penyelesai adalah metode getAdjacentCell. Metode ini menerima input sebuah Cell dan memberi output daftar Cell yang berada di sekitarnya. Daftar Cell ini secara umum akan berisi 8 Cell sekitar kecuali jika Cell yang diinput berada di pinggir atau di sudut sehingga jumlah Cell di sekitar akan menjadi kurang dari 8. Metode ini akan sangat berguna baik untuk menghitung jumlah ranjau di sekitar sebuah Cell tertentu atau untuk memperkirakan apakah sebuah area sudah aman atau belum untuk dibuka seluruhnya.

Di awal permainan board terlebih dahulu haruslah ditentukan besar board permainan, jumlah ranjau yang disebar, juga dimana kotak yang paling awal yang dibuka. Menurut pengamatan penulis, program permainan Minesweeper yang idbuat oleh Windows selalu menjamin bahwa kotak pertama yang dibuka tidak berisi ranjau dan juga semua kotak disekitarnya. Karena itu, penulis membuat program konstruktor untuk board seperti ini.

```
1.  public Board(int h,int w, Cell s, int n) {
2.          height = h;
3.          width = w;
4.          board = new int[h+1][w+1];
5.          landMine = new boolean[h+1][w+1];
6.          for (int i=1;i<=height;++i) {
7.              for (int j=1;j<=width;++j) {
8.                  board[i][j] = 0;
9.                  landMine[i][j] = false;
10.             }
11.         }
12.
13.         start = s;
14.
15.         jumlahMine = n;
16.         List<Cell> spesial = getAdjacentCell(s
    tart);
17.         for (int i=0;i<jumlahMine;++i) {
18.             int baris;
19.             int kolom;
20.             do {
21.                 baris = (int) (Math.random() *
     height + 1);
22.                 kolom = (int) (Math.random() *
     width + 1);
23.             } while (isMined(baris,kolom) || (
    (baris == start.getRow()) && (kolom == start.g
    etColumn())) || spesial.contains(new Cell(bari
    s,kolom)));
```

```
            landMine[baris][kolom] = true;
        }


        for (int i=1;i<=height;++i) {
            for (int j=1;j<=width;++j) {
                Cell cell = new Cell(i,j);
                List<Cell> daftar = getAdjacen
    tCell(cell);

                for (Cell k : daftar) {
                    if (isMined(k)) {
                        board[cell.getRow()][c
    ell.getColumn()]++;
                    }
                }
            }
        }
    }
```

Seperti terlihat pada program, saat board diciptakan, ranjau akan disebar ke seluruhh permukaan array landmine dengan membuat perkecualian pada Cell start dan semua Cell disekitarnya. Setelah itu, dihitung jumlah ranjau di sekitar Cell untuk semua Cell, baik untuk Cell yang tidak memiliki ranjau ataupun yang tidak memiliki ranjau. Hal ini dilakukan agar penyelesai tidak mencurangi permainan dengan bertanya angka untuk sebuah daerah dengan menggunakan metode getAdjacent dan menerima angka yang aneh ketika kebetulan sebuah daerah mengandung ranjau.

Pada program permainan ini satu-satunya cara bagi program luar untuk mengetahui bahwa suatu kotak mengandung ranjau atau tidak adalah dengan menggunakan metode testMine yang bisa saja memberikan exception mineExplodedException. Karena itu, untuk sebuah program penyelesai yang memakai program permainan ini untuk mengetahui sebuah kotak mengandung ranjau atau tidak, program itu harus mengambil resiko menerima exception dan mengakhiri permainan.

IV. DETAIL IMPLEMENTASI PROGRAM PENYELESAI

Program penyelesai ini didesain untuk menyelesaikan program permainan yang sudah dijelaskan sebelumnya. Secara teknis, program ini juga bisa digunakan untuk menyelesaikan program permainan Minesweeper lain namun dengan beberapa perubahan pada implementasi metode sehingga mendukung pergerakan mouse atau hal lain yang dibutuhkan untuk memainkan permainan tersebut.

A. Desain Kelas

Desain kelas untuk program penyelesai ini adalah sebagai berikut :

```
import java.util.List;
import java.util.ArrayList;
import java.util.concurrent.CopyOnWriteArrayLi
st;
import java.util.Queue;
import java.util.HashSet;

class Solver {
    private int mineGuessed;
    private Board board = null;
    private int[][] tipe = null;
 // 0 = unknown, 1 = mine, 2 = angka
    private ArrayList[][] exception = null;
    private int[][] curse = null;
    private List<Cell> target = null;
    private boolean changed;

    public Solver(Board b) throws mineExploded
Exception {
    }

    private int getTipe(Cell cell) {    }

    private int getAdjacent(Cell cell) {    }


    private int getCurse(Cell cell) {
    }

    private List<Cell> getException(Cell cell)
 {  }

    private void openCell(Cell cell) {
    }

    private List<Cell> getIntersect(List<Cell>
 l1, List<Cell> l2) {   }

    public void clickCell(Cell cell) throws mi
neExplodedException {
    }

    public void randomClick() throws mineExplo
dedException {    }

    public void flagCell(Cell cell) {
    }

    private void obviousGuess() throws mineExp
lodedException {  }

    private void advancedGuess(Cell cell) thro
ws mineExplodedException {
    }

    public void solve() throws mineExplodedExc
eption {  }

    public String toString() {
    }
 }
```

.

B. Detail Implementasi

Metode yang paling utama dalam kelas penyelesai ini tentu saja adalah metode konstruktor dan **solve()**. Konstruktor berfungsi untuk mengassign sebuah board ke pada class Solver dan kemudian metode solve menyelesaikannya. Semua metode lain berfungsi untuk membantu metode-metode ini.

Implementasi kelas konstruktor adalah seperti ini :

```
1.  public Solver(Board b) throws mineExplodedExcept
    ion {
2.          board = b;
3.          tipe = new int[board.getHeight()+1][boar
    d.getWidth()+1];
4.          curse = new int[board.getHeight()+1][boa
    rd.getWidth()+1];
5.          target = new CopyOnWriteArrayList<Cell>
    ();
6.          exception = new ArrayList[board.getHeigh
    t()+1][board.getWidth()+1];
7.          for (int i=1;i<=board.getHeight();++i) {

8.              for (int j=1;j<=board.getWidth();++j
    ) {
9.                  tipe[i][j] = 0;
10.                 curse[i][j] = 0;
11.                 exception[i][j] = new ArrayList(
    );
12.             }
13.         }
14.
15.         clickCell(board.getStartCell());
16.     }
```

Ada beberapa atribut yang akan sangat berguna untuk penyelesaian metode heuristik berikutnya yaitu curse (salah satu kata kunci dalam implementasi penulis) dan exception (juga salah satu kata kunci). Atribut lain seperti tipe dan mineGuessed berguna untuk mencatat progress penyelesaian dan atribut target berguna untuk mencatat semua angka yang belum seluruh kotak di sekitarnya terselesaikan.

Metode solve yang penulis gunakan diimplementasikan seperti ini :

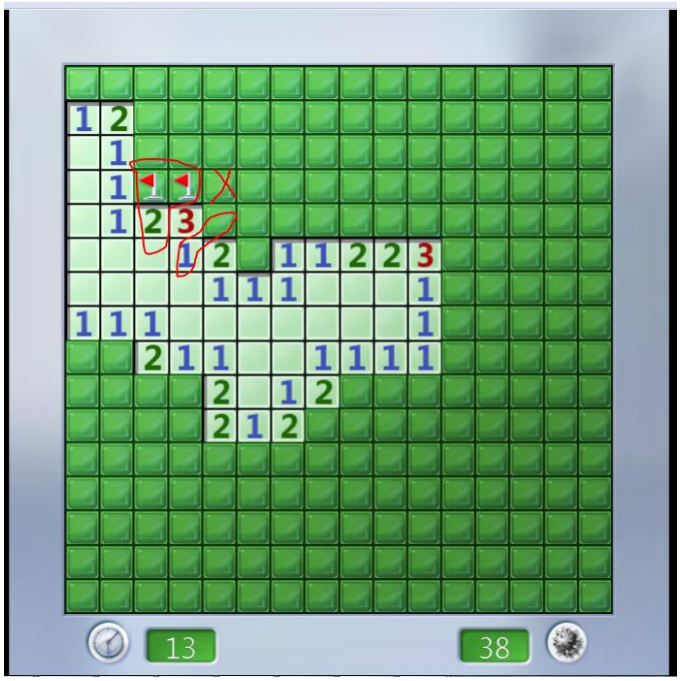
```
1.  public void solve() throws mineExplodedException
     {
2.          while (mineGuessed < board.getMine()) {

3.              changed = false;
4.              obviousGuess();
5.              if (mineGuessed < board.getMine()) {

6.                  randomClick();
7.              }
8.          }
9.      }
```

Inti dari implentasi metode solve ini adalah selama belum seluruh ranjau terdata, maka seluruh metode di dalamnya akan diulang terus.

Penulis membagi metode heuristik ini menjadi 3 level. Satu, menandai semua ranjau yang berada di kotak yang tersisa yang jumlahnya sama persis dengan angka tersebut. Contohnya adalah sebagai berikut :



Gambar 3. Contoh Implementasi Heuristic 1

Pertama-tama target akan mencatat semua angka yang bukan 0, kemudian program akan mencari dalam target semua angka yang memiliki jumlah kotak terbuka sama dengan angka itu. Kemudian, semua kotak terbuka itu akan diberi bendera. Bendera itu akan memberi “curse” pada semua angka disekitarnya yaitu mengurangi semua angka di sekitarnya sebesar 1. Kemudian langkah-langkah tersebut di ulang lagi sampai tidak ada perubahan pada board.

Semua langkah itu terimplementasikan dalam obivousGuess seperti terlihat dalam solve yang diimplementasikan seperti ini :

```
1.  private void obviousGuess() throws mineExplode
    dException {
2.          do {
3.              changed = false;
4.              for (Cell cell : target) {
5.                  if ((getTipe(cell) == 2) && ((
    getAdjacent(cell) + getCurse(cell)) == 0)) {

6.                      target.remove(cell);
7.
8.                      List<Cell> daftar = board.
    getAdjacentCell(cell);
9.                      for (Cell i : daftar) {
10.                         if ((getTipe(i) == 0)
    && (!getException(cell).contains(i))) {
11.                             clickCell(i);
12.                         }
13.                     }
14.                 } else {
15.                     List<Cell> daftar = board.
    getAdjacentCell(cell);
16.
17.                     int closedCell = 0;
18.                     for (Cell i : daftar) {
```

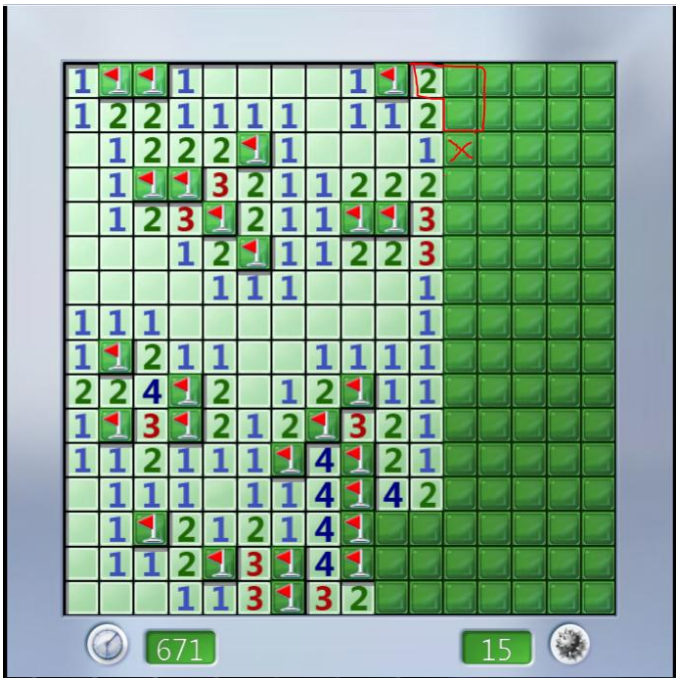
```
                        if ((getTipe(i) == 0) && (!getException(cell).contains(i))) {
                            closedCell++;
                        }
                    }

                    if (closedCell == (getAdjacent(cell) + getCurse(cell))) {
                        for (Cell i : daftar) {
                            if ((getTipe(i) == 0) && (!getException(cell).contains(i))) {
                                flagCell(i);

                                List<Cell> daftar2 = board.getAdjacentCell(i);
                                for (Cell j : daftar2) {
                                    if ((getAdjacent(j) + getCurse(j) == 0) && target.contains(j)) {
                                        target.remove(j);
                                        List<Cell> daftar3 = board.getAdjacentCell(j);
                                        for (Cell k : daftar3) {
                                            if ((getTipe(k) == 0) && (!getException(j).contains(k))) {
                                                clickCell(k);
                                            }

                                        }
                                    }
                                }
                            }
                        }
                    }
                    advancedGuess(cell);
                }
            } while (changed);
        }
```

Metode heuristic level 1 sudah cukup untuk menyelesaikan beberapa board standar "Beginner" namun belum cukup untuk menyelesaikan board dengan level lebih tinggi. Untuk itu, diiimplementasikan heuristic level 3 yang lebih memfokuskan pada hubungan antar angka. Ilustrasi untuk ini bisa terlihat sebagai berikut :



Gambar 4. Contoh implementasi Heuristic 2

Gambar di atas menunjukkan contoh board yang sudah menjalankan algoritma obviousGuess sampai tidak ada lagi yang bisa diteruskan. Dengan algoritma heuristik level 2, kita masih bisa melanjutkan pencarian. Pada kasus ini angka 2 paling atas sudah terkena "curse" sehingga bernilai 1, begitu juga dengan angka 2 di bawahnya. Namun, karena masih ada dua kotak terbuka di sekitar mereka, tidak ada kesimpulan yang bisa didapatkan.

Heuristic level 2 menerapkan semacam flag seperti pada level 1 namun bedanyta ini berlaku pada beberapa kotak sekaligus. Pada kasus ini, kita "flag" dua kotak yang berhubungan dengan angka 2 paling atas dengan 1 "flag" dan kemudian cari semua kotak baik angka maupun belum terbuka yang berhubungan dengan semua kotak yang di flag. Kotak itu kemudian akan terkena "curse" juga dan berkurang angkanya sebesar 1. Pada contoh di atas, karena dua kotak itu diflag bersamaan, maka angka 2 yang kedua dari atas akan terkena curse itu dan karena curse sebelumnya, menjadi 0 dan akan membuka semua kotak di sekitarnya. Namun, itu menjadi masalah jika dia membukan kotak yang memberinya curse dan permberi curse itu sebenarnya tidak benar-benar diberi flag sehingga tidak ada jaminan bahwa kotak itu tidak akan dibuka.

Untuk itulah diberikan sebuah daftar baru bernama exception. Untuk setiap kotak yang dicurse oleh flag block, maka kotak itu akan menerima exception berupa daftar semua kotak dalam block yang memberi dia curse. Semua kotak yang ada dalam exception tidak boleh dibuka dengan pengaruh oleh kotak yang memiliki exception itu. Jika suatu saat salah satu kotak dalam block itu di flag dengan flag sebenarnya, dia juga tidak boleh lagi memberi curse pada kotak yang dia merupakan anggota dari exception itu. Dengan begitu, pencarian lebih lanjut bisa berjalan dengan lancar.

Heuristic level 2 diimplementasikan dalam advancedGuess yang dijalankan dalam obviousGuess dan diimplementasikan sebagai berikut :

```
1.  private void advancedGuess(Cell cell) throws min
    eExplodedException {
2.          if (getAdjacent(cell) + getCurse(cell) >
     0) {
3.              List<Cell> daftar = board.getAdjacen
    tCell(cell);
4.
5.              List<Cell> daftar2 = null;
6.              List<Cell> daftar3 = new ArrayList<C
    ell>();
7.              boolean first = true;
8.              for (Cell cell2 : daftar) {
9.                  if (getTipe(cell2) == 0) {
10.                     if (first) {
11.                         daftar2 = board.getAdjac
    entCell(cell2);
12.                         first = false;
13.                     } else {
14.                         daftar2 = getIntersect(d
    aftar2,board.getAdjacentCell(cell2));
15.                     }
16.                     daftar3.add(cell2);
17.
18.                 }
19.
20.             }
21.
22.             if ((daftar2 != null) && !daftar2.is
    Empty()) {
23.                 daftar2.remove(cell);
24.                 for (Cell cell2 : daftar2) {
25.                     int total1 = getAdjacent(cel
    l) + getCurse(cell);
26.                     int total2 = getAdjacent(cel
    l2) + getCurse(cell2);
27.                     if (getIntersect(getExceptio
    n(cell2),daftar3).isEmpty() && (total1<=total2))
     {
28.                         curse[cell2.getRow()][ce
    ll2.getColumn()] -= total1;
29.                         changed = true;
30.                         for (Cell i : daftar3) {

31.                             exception[cell2.getR
    ow()][cell2.getColumn()].add(i);
32.                         }
33.                     }
34.                 }
35.             }
36.         }
37.     }
```

Bahkan masih ada kasus yang belum bisa terpecahkan dengan heuristic level 2 seperti pada kasus di bawah ini :

0 0 0 0 0 1 * 1 1 ^^^^^^^^^^^^^^^^^^^^^

1 1 1 0 0 1 1 1 1 ^^^^^^^^^^^^^^^^^^^^^

2 * 1 1 2 2 1 0 1 2 ^^^^^^^^^^^^^^^^^^^^

* 3 3 3 * * 1 0 1 ^^^^^^^^^^^^^^^^^^^^

^^^^^ 3 2 **1 3** ^^^^^^^^^^^^^^^^^^^^

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Contoh kasus Heuristic 3.

Algoritma Heurstic 2 tidak lagi bisa melanjutkan pencarian di board ini. Namun, sebenarnya masih ada satu kali lagi optimasi yang bisa dilakukan untuk melanjutkan pencarian.

Jika algortima Heuristic 2 hanya bisa memberi flag pada seluruh kotak yang tersisa dari sebuah angka, maka pada tahap ini flag pada kotak bisa diberikan sebagian saja. Kita ambil contoh pada kasus ini dengan angka 3 yang diberi warna merah. Dia sudah mendapatkan curse dan exception dari dua kotak yang ada di sebelah timur laut dan timurnya sehingga nilai efektifnya hanya tinggal 2. Sekarang angka 3 itu masih memliki 3 kotak bebas di bawahnya sehingga belum bisa diambil keputusan kotak mana yang tidak ada bom. Angka 1 di sebelahnya juga demikian.

Kita analisa tempat dimana kotak terbuka yang berdekatan dengan angka 3 berinterseksi dengan kotak yang bersebelahan dengan angka 1 dan kemudian menaruh semua ranjau pada angka 3 pada kotak yang tidak berinterseksi. Ternyata masih ada satu ranjau yang tersisa di angka 3 yang harus ditaruh di kotak yang berinterseksi. Dengan itu kita bisa menaruh flag block pada dua kotak yang berinterseksi itu dengan satu buah flag dan mengcurse semua angka di sekitarnya seperti biasa.

V. PENGUJIAN

Berikut adalah salah satu contoh output permainan standar “Intermediate”.

```
-------*----***-
-*------**------
-*----------*--
--------*-------
--*-----**-*-**-
*---------*-*---
----***-------*-
--*---*--**-*---
--------*-------
--**------------
-------*-------*
----*-----------
---------**---*-
-----------*----
-------*---*----
```

Contoh board dengan start cell di (2,5)

GAME OVER!!!

1110001*3-----#-

2*200012*---3---
2*2000023-------
12210002*-------
12*10002**3--*--
*212232223*3*432
1212***2133423*1
01*224*32**2*211
02331112*3221100
01**101221000011
0123211*1000001*
0001*11122210122
000111001**211*1
00000011224*2111
0000001*102*2000
0000001110111000

Contoh game over dengan # melambangkan tempat dimana bom terinjak

0000001121100111
0000001*2*1002*2
00000011333102*3
111112112**1023*
2*21*2*13*4101*2
3*3212223*200111
*3*1001*21101110
1211001110001*21
00111001111133*1
012*1001*23*4*31
01*320012*3*5*41
012*223221224**1
00112***1001*332
221012322111111*
**1111112*100011
2211*11*21100000

Contoh board yang sudah terselesaikan

Program ini mampu menyelesaikan standar "Beginner" untuk permainan Minesweeper Windows dengan akurasi mendekati 100%, "Intermediate" dengan akurasi mendekati 80%, dan"Advanced" dengan akurasi mendekati 10%.

VI. PENGEMBANGAN LEBIH LANJUT

Program penyelesai ini didesain untuk menyelesaikan program permainan yang sudah dijelaskan sebelumnya. Secara teknis, program ini juga bisa digunakan untuk menyelesaikan program permainan Minesweeper lain namun dengan beberapa perubahan pada implementasi metode sehingga mendukung pergerakan mouse atau hal-hal lain.

Pada implementasi ini penulis belum memasukkan metode heuristic level terakhir yang bisa meningkatkan akurasi dari penyelesai ini. Program ini juga masih bisa ditambahkan metode Probabilitas dan perhitungan ranjau tersisa untuk lebih melengkapkan program ini.

PERSEMBAHAN

Penulis ucapkan rasa syukur dan terima kasih kepada Tuhha Yang Maha Esa yang sudah memberikan kesempatan pada penulis untuk menulis karya ini.

References

[1] http://web.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.pdf

[2] https://luckytoilet.wordpress.com/2012/12/23/2125/

I. 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, 9 Mei 2016

Hafizh Afkar Makmur - 13514062