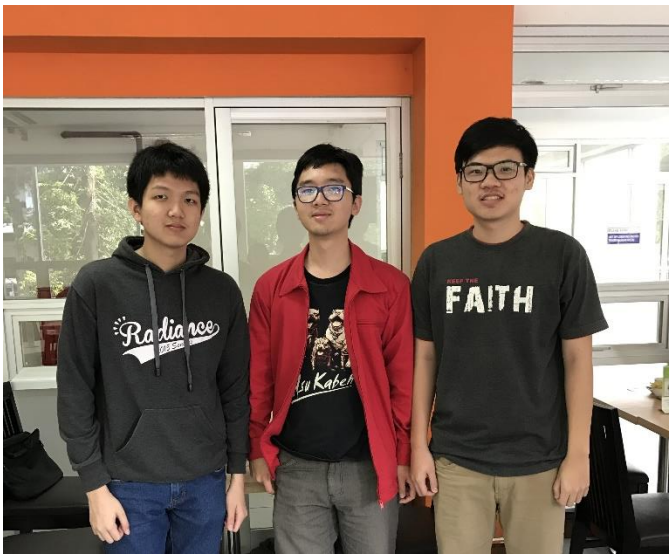
Tugas Besar I – IF2211 Strategi Algortima

APLIKASI ALGORITMA GREEDY PADA PERMAINAN BOMBERMAN



Disusun oleh:

Erick Wijaya / 13515057 / K-03

Adrian Mulyana / 13515075 / K-03

William / 13515144 / K-03

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2017

DAFTAR ISI

BAB I
DESKRIPSI PERSOALAN

Bomberman adalah *video game* berbasis labirin yang memerlukan strategi untuk menyelesaikannya. Aplikasi ini pertama kali dikembangkan oleh Hudson Soft dan dipublikasikan pertama kali pada tahun 1983. Tujuan umum dari permainan ini adalah menyelesaikan semua level dengan membuat strategi penempatan bom, dalam rangka membunuh musuh dan menghancurkan halangan. Salah satu jenis Bomberman membuat sebuah tim terdiri atas beberapa pemain. Yang menjadi pemenang dalam permainan ini adalah pemain terakhir yang masih hidup, atau tim yang salah satu pemain nya masih hidup saat pemain yang lain sudah mati.

Dalam tugas ini, kami mengimplementasikan strategi Greedy untuk memenangkan permainan. Aturan permainan bomberman yang digunakan antara lain:

1. Papan permainan Bomberman terdiri atas beberapa objek, dan objek yang mungkin ada pada papan adalah pemain, bom, tembok (bisa dihancurkan dengan bom), dan pembatas (tidak hancur oleh bom).
2. Setiap kelompok hanya memiliki satu pemain.
3. Pada satu saat, setiap pemain hanya membawa 1 bom yang aktif di layar, namun jumlah bom yang dimiliki bisa ditambah, dengan fitur *power up*. Waktu jeda antara peletakan bom hingga bom meledak silakan dipelajari pada *game engine* tersebut, karena jeda waktu ini bisa berubah bergantung keadaan papan permainan pada saat itu (apakah ada bom lain yang sedang meledak), diatur sendiri oleh bot (pemain) yang meletakkan bom, atau bergantung pada jumlah *power up*.
4. Radius bom secara *default* adalah 1 petak, namun radius tersebut bisa ditambah dengan fitur *power up*.
5. Fitur *power up* adalah fitur yang disediakan *game engine*, silakan dipelajari bagaimana mekanisme untuk mendapatkannya.

6. Pemain yang menghancurkan tembok dengan bom, menghancurkan pemain lawan dengan bom, akan mendapatkan nilai, dengan aturan penghitungan nilai dapat dibaca pada dokumentasi *game engine* tersebut.
7. Pemain dengan nilai tertinggi akan menjadi pemenang permainan Bomberman.
8. Permainan berhenti ketika hanya tinggal 1 pemain pada papan permainan, atau berdasarkan jumlah iterasi tertentu yang nanti akan ditentukan oleh asisten.

Strategi *Greedy* yang digunakan dikaitkan dengan proses penghitungan nilai yang disediakan *Game Engine*. Fungsi objektif penerpan *Greedy* adalah memenangkan permainan dengan mendapatkan nilai tertinggi.

BAB II
DASAR TEORI

2.1 Algoritma Greedy

Algoritma *greedy* merupakan metode yang populer untuk memecahkan persoalan optimasi. Algoritma ini termasuk sederhana dan lempang *(straight forward).* Prinsip dari algoritma *greedy* adalah "*take what you can get now*!"(Ambil apa yang dapat anda peroleh sekarang!). Algoritma ini membentuk solusi per langkah *(step by step).* Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkahnya harus dibuat keputusan terbaik penentuan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah berikutnya. Pendekatan yang digunakan di dalam algoritma *greedy* adalah membuat pilihan yang "tampaknya" memberikan perolehan terbaik, yaitu dengan membuat pilihan optimum lokal pada setiap langkah, dengan harapan bahwa sisanya mengarah ke solusi optimum global.

Dari pendekatan di atas, algoritma *greedy* dapat didefinisikan sebagai algoritma yang memecahkan masalah langkah per langkah, dan pada setiap langkah:

1. Mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan ("*take what you can get now*").
2. Berharap dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

2.2 Pemanfaatan Game Engine Bomberman

2.2.1 Penambahkan Pemain

Pada *Game Engine Bomberman*, jumlah pemain dibatasi maksimal 12 pemain. Pemain dibedakan menjadi 2 jenis, yaitu pemain *console* dan *bot*. Pemain *console* adalah pemain yang masukan langkahnya adalah manual

dari manusia, sedangkan pemain *bot* adalah pemain yang langkahnya ditentukan berdasarkan algoritma yang diimplementasikan pada *bot*. Penambahan pemain dilakukan saat eksekusi Bomberman.exe. Langkah yang dilakukan adalah sebagai berikut:

1. Pada terminal, masuklah ke direktori dimana Bomberman.exe berada.
2. Tambahkan argumen berikut saat melakukan *run*

```
-c <jumlah pemain console>
```

Sedangkan, untuk menambahkan bot, langkah yang dilakukan sebagai berikut:

1. Pada terminal, masuklah ke direktori dimana Bomberman.exe berada.
2. Tambahkan argumen berikut saat melakukan *run*

```
-b "<path menuju folder bot-1>" "<path menuju folder
    bot-2>" ... "<path menuju folder bot ke-n>"
```

2.2.2 Cara Menambahkan Strategi Greedy

Penyertaan strategi *greedy* pada bot dilakukan pada file SampleBot.cpp (atau jenis file lain sesuai bahasa pemrograman yang digunakan). Algoritma *Greedy* dituliskan pada prosedur writeMoveFile. Pada prosedur ini, akan dilakukan penentuan langkah apa yang akan dilakukan oleh bot. Langkah yang dilakukan bot akan berdasarkan suatu nilai yang dikeluarkan. Terdapat 6 jenis nilai yang merepresentasikan langkah yang dapat dilakukan oleh *bot*, yaitu:

1. Diam = 0,
2. Bergerak ke atas = 1,
3. Bergerak ke kiri = 2,
4. Bergerak ke kanan = 3,
5. Bergerak ke bawah = 4,
6. Meletakkan bom = 5,
7. Merubah waktu peledakan bom menjadi satu = 6.

Selain itu juga perlu dilakukan pengaturan identitas *bot* pada *file* **bot.json**. Berikut adalah isi serta pengaturan yang dilakukan pada file tsb:

{

"Author": "<nama pemrogram bot>",

“Email": "<alamat surel pemrogram bot>",

"NickName": "<nama bot>",

"BotType": "<bahasa pemrograman bot>",

"ProjectLocation" : "<path menuju SampleBot.cpp>”

"RunFile" : "<path menuju SampleBot.exe>"

}

2.2.3 Cara Menjalankan Game Engine

Setelah algoritma disertakan pada file Bot, untuk menjalankan *Game Engine*, dilakukan langkah-langkah berikut:

1. Pada terminal, masuk pada direktori tempat penyimpanan Bomberman.exe
2. Terdapat beberapa argumen yang dapat disertai saat melakukan *run*, antara lain:

`-b, --bot`	(default: Empty String) Digunakan untuk menambahkan bot pada game. Tuliskan path menuju folder yang menyimpan bot. Path ini dituliskan setelah –b. Dapat dimasukkan lebih dari 1 bot, dengan memisahkan path nya dengan spasi
`-c, -console`	(default:0) Digunakan untuk mengatur jumlah pemain bukan bot.
`-r, --rules`	Digunakan untuk menampilkan aturan permainan

`--clog`	Menyimpan *log* dari console pada file log.txt
`--pretty`	Menampilkan peta permainan
`-l, --log`	Untuk menjalankan *replay log* files tertentu
`-s, --seed`	Seed untuk penciptaan peta
`--nolimit`	Menonaktifkan batas waktu eksekusi bot
`--debug`	Untuk menghentikan permainan apabila terjadi *error*
`--help`	Digunakan untuk menampilkan petunjuk

3. Misalnya, untuk menjalankan game bomberman dengan 1 pemain dan 1 bot digunakan perintah berikut:

```
Bomberman.exe --pretty -b "<path ke folder bot>" -c 1
```

BAB III
PEMANFAATAN STRATEGI GREEDY

3.1 Dasar Penyusunan Strategi Greedy

Dalam penyusunan strategi greedy yang digunakan pada bot, penulis dasarkan untuk pada bagaimana cara pemain untuk mempertahankan diri dan mendapatkan poin sebanyak-banyaknya. Berikut adalah sistem poin yang terdapat pada permainan:

- Pemain mendapatkan 10 poin jika menghancurkan tembok (Jika ada 2 bom yang menghancurkan tembok yang sama, kedua pemain mendapatkan 10 poin)
- Pemain yang berhasil membunuh pemain lain akan mendapat poin dengan perhitungan berikut:

$$\text{poin yang didapat} = \frac{100 + \text{Max point per peta untuk tembok}}{\text{jumlah pemain pada peta}}$$

- Pemain mendapatkan poin bila bergerak ke tempat yang belum pernah didahuli sebelumnya
- Pemain mendapatkan poin bila mendapatkan Power Up
- Jika banyak bom yang meledak akibat rantai peledakan bom, semua pemain yang bom nya membentuk rantai tersebut akan mendapatkan poin untuk semua entitas yang hancur
- Jika pemain mati pada suatu ronde, pemain tersebut kehilangan semua poin yang didapatkan pada ronde tersebut. Pemain juga akan kehilangan poin sebanyak poin yang didapat bila pemain membunuh pemain lain.

Hal lain yang juga dipertimbangkan dan tidak kalah penting dalam strategi *greedy* ini adalah Power Up. Terdapat 3 jenis Power Up, antara lain:

- Bomb Bag Power Up

  Berfungsi untuk menambahkan jumlah bom yang dapat diletakkan oleh pemain ketika waktu pada bom lain sudah berkurang. Pada peta, disimbolkan dengan karakter '&'.
- Bomb Radius Power Up

  Berfungsi untuk mengalikan radius bom yang dimiliki pemain dengan dua. Pada peta, disimbolkan dengan karakter '!'.
- Special Power Up

  Power Up ini adalah Power Up yang memiliki fungsi mencakup fungsi Bomb Bag Power Up dan Bomb Radius Power Up. Selain itu juga akan memberikan 50 poin. Pada peta, disimbolkan dengan karakter '$'.

3.2 Strategi Greedy yang Digunakan

Berdasarkan hal-hal yang kami pertimbangkan di atas, berikut adalah strategi *Greedy* yang kami gunakan dalam pembuatan *bot*:

- Jika *bot* berada di zona yang tidak aman (ada pada radius peledakan bom), cari jalan terpendek untuk melarikan diri ke daerah yang aman.
- Jika *bot* berada di zona yang aman, lakukan salah satu dari hal berikut (diurutkan berdasarkan prioritas):
  - Jika terdapat *Power Up* di sekitar *bot* kami, yaitu berjarak 6 langkah atau kurang, cari jalan terpendek dan ambil *Power Up* tersebut.
  - Jika terdapat pemain lain di sekitar bom milik *bot*, ubah waktu peledakan bom menjadi satu.
  - Jika ada pemain lain yang berhadapan dengan *bot*, dimana radius bom yang *bot* miliki lebih dari sama dengan jarak antara *bot* kami dengan pemain lain tersebut, letakkan bom.
  - Jika ada pemain lain di sekitar *bot* (jarak 3 langkah atau kurang), dan lebih lemah (dilihat dari radius bom), cari jarak terpendek untuk menghampiri pemain lain tersebut.

- Jika masih ada tembok (yang bisa dihancurkan) pada peta,
  - Jika berada di sebelah tembok dan masih memiliki bom yang aktif di tempat lain, ledakan bom tersebut terlebih dulu.
  - Jika berada di sebelah tembok dan tidak ada bom aktif, letakkan bom.
  - Jika tidak berada di sebelah tembok, cari jalan terpendek menuju tembok.
- Jika tembok sudah habis, cari jalan terpendek menuju pemain terlemah.

3.3 Struktur Data

Dalam penyusunan solusi algoritma greedy untuk *bot*, dibutuhkan struktur data yang berfungsi untuk mendukung jalannya algoritma greedy dan kinerja bot. Struktur data ini didefinisikan sebagai kelas (*class*) pada program. Kelas utama yang didefinisikan adalah *class Player*, *class Map*, dan *class Bomb*. Selain kelas, struktur data yang digunakan adalah *queue* yang berperan penting dalam algoritma greedy, yang dapat dilihat pada bagian spesifikasi program.

1. *Class Map*

   Kelas Map adalah kelas yang mendefinisikan objek peta, pada kasus ini adalah peta permainan bomberman. Kelas ini memiliki atribut seperti matriks (memiliki baris dan kolom) dan memiliki metode yang standar, yaitu konstruktor, operator, predikat, dan mencetak peta. Kelas ini menampung setiap petak pada peta permainan dalam tipe data *char*. Karakter yang ditampung memiliki representasi tertentu pada permainan bomberman. Berikut adalah tabel representasi karakter beserta isi header dari kelas Map.

Karakter	**Makna**
‘#’	Tembok yang tidak dapat dihancurkan.
‘+’	Tembok yang dapat dihancurkan.
Alfabet	Pemain
Angka	Bomb beserta jumlah langkah bom akan meledak.
‘!’, ‘&’, ‘$’	Power Up.
‘*’	Ledakan bom.
‘ ‘	Jalur.

```
class Map{
   public:
       /* Constructor */
       Map();
       Map(int, int);
       Map(const Map&);
       /* Destructor */
       ~Map();
       /* Getter */
       int GetRow();
       int GetCol();
       /* Output */
       friend ostream& operator<<(ostream&, const Map&);
       /* Operator */
       Map& operator=(const Map&);
       void search(char, int&, int&);
       /* Operator[][] */
       class Proxy {
          friend class Map;
          public:
              char& operator[](int c)
              {
                  return parent.data[r][c];
              }
          private:
              Proxy(Map &parent_, int row_) :
                  parent(parent_),
                  r(row_)
              {}
              Map& parent;
              int r;
       };

      Proxy operator[](int x)
      {
          return Proxy(*this, x);
      }

   private:
       int row;
       int col;
       char ** data;
};
```

2. *Class Player*

   Kelas Player mendefinisikan objek pemain yang bermain setiap permainan berlangsung. Kelas Player memiliki atribut yang dimiliki oleh pemain pada permainan bomberman, mulai dari identitas pemain hingga kekuatan pemain berdasarkan power up yang dimilikinya. Metode kelas Player yang perlu disorot adalah predikat yang membandingkan dua buah Player, yaitu Player yang manakah yang lebih lemah. Metode ini cukup penting untuk mengaplikasikan algortima greedy dalam menentukan apakah *bot* bersedia mengejar dan membunuh pemain lain (karena pemain tersebut lebih lemah

dari *bot*) atau menghiraukan pemain lain tersebut. Berikut adalah header dari kelas Player.

```
class Player{
   public:
      /* Konstruktor */
      Player();
      Player(char,int,int,int,int,int,int);
      Player(const Player&);
      Player& operator=(const Player&);
      /* Getter */
      int GetKey();
      int GetBombs();
      int GetMaxBag();
      int GetRadius();
      /* Predikat */
      bool IsWeaker(const Player&);
      /* Output */
      friend ostream& operator<<(ostream&, const Player&);
   private:
      char key;
      int point;
      int x;
      int y;
      int bombs;
      int max_bag;
      int blast_radius;
};
```

3. Class Bomb

   Kelas Bomb merupakan *blueprint* yang menggambarkan objek Bomb sesuai pada permainan bomberman. Kelas ini memiliki atribut berupa identitas bom, jangkauan ledakan bom, dan waktu yang dibutuhkan bom untuk meledak.

```
class Bomb{
   public:
      /* Konstruktor */
      Bomb();
      Bomb(char,int,int);
      Bomb(const Bomb&);
      /* Operator */
      Bomb& operator=(const Bomb&);
      /* Getter */
      char GetID();
      int GetRadius();
      int GetFuse();
      /* Output */
      friend ostream& operator<<(ostream&, const Bomb&);
   private:
      char id;
      int radius;
      int fuse;
};
```

3.4 Spesifikasi Program

Berikut adalah spesifikasi dari bagian-bagian program yang membentuk algoritma greedy. Spesifikasi berupa program utama beserta beberapa bagian program yang menjadi kunci dari tercapainya algoritma greedy.

1. Program Utama

   Program utama menerima dua buah argumen, yaitu argc dan argv. Argv menyimpan data identitas pemain dan akses path untuk membuka file. Identitas pemain sangat penting karena dengan identitas ini kita dapat mengetahui siapakah yang merupakan *bot* pengguna dari semua player yang ada.

```
int main(int argc, char** argv)
{
    char playerKey = argv[1][0];
    string filePath = argv[2];

    cout << "Args: " << argc << std::endl;
    cout << "Player Key: " << argv[1] << std::endl;
    cout << "File Path: " << argv[2] << std::endl;

    readStateFile(filePath);
    writeMoveFile(playerKey, filePath);
    return 0;
}
```

2. Algoritma Greedy

   Algoritma greedy didasarkan dari strategi greedy yang dibahas pada bab ini pada poin kedua. Algoritma ini menjadi kunci yang menentukan segala gerakan dan kinerja dari *bot*. Algoritma ini memiliki *core* yaitu mencari jarak minimum dari suatu posisi menuju suatu benda. Oleh karena itu, algoritma ini didukung dengan fungsi/prosedur yang berguna mencari nilai jarak minimum. Berikut adalah algoritma greedy yang diterapkan pada prosedur menulis langkah pada file.

```
void writeMoveFile(char playerKey, string filePath)
{
    cout << "Writing move file " << filePath + "/" + "move.txt" <<
    std::endl;
    ofstream outfile(filePath + "/" + "move.txt");
```

```
if (outfile.is_open()){
    /* Object Instances */
    Map GameMap;
    map<char,Player> PlayerList;
    map<pair<int,int>, Bomb> BombList;

    /* Read Object from File */
    string FileName = filePath + "/" + "map.txt";
    Parse(GameMap, PlayerList, BombList, FileName);

    /* Initial Boolean */
    initBool(danger);

    /* Set Danger Area */
    for(map<pair<int,int>,Bomb>::iterator it = BombList.begin(); it
    != BombList.end(); it++){
        setDanger(danger, GameMap, it->first.first, it->first.second,
        it->second.GetRadius());
    }

    /* Get Player Position */
    int a, b;
    GameMap.search(playerKey, a, b);

    /* Greedy */
    int x, out;
    pair<int,int> CurrPos (a,b);
    if (danger[a][b]){
        x = FindEscape(GameMap, CurrPos); // Find Nearest Escape
        out = MoveToEscape(GameMap, CurrPos, x);
    }else{
        int p, q, r, s;
        char c;

        p = FindAny(GameMap, CurrPos, '$');
        q = FindAny(GameMap, CurrPos, '!');

        s = Min2(p,q);
        if ((s >= 0) && (s <= 6)){ // Found PowerUp(s) Nearby
            if (s == p)
                out = MoveToAny(GameMap, CurrPos, s, '$');
            else
                out = MoveToAny(GameMap, CurrPos, s, '!');
        }

        else if (IsParallelBomb(GameMap, BombList, playerKey)){
            out = 6; // Trigger Player's Bomb if enemy enter radius
        }

        else if (IsParallelPlayer(GameMap, CurrPos,
        PlayerList[playerKey].GetRadius())){
            out = 5; // Put Bomb Parallel to Enemy, for Self-Defence
        }

        else if (( (c = IsPlayerNearby(GameMap, CurrPos, 3)) != -1)
        && !PlayerList[playerKey].IsWeaker(PlayerList[c])){
            out = MoveToAny(GameMap, CurrPos, 3, c);  // Chase Weaker
            Enemy Nearby
        }

        else if ((x = FindWall(GameMap, CurrPos)) != -1){ // Find
        Nearest Wall, If Exist
```

```
        if (x == 0){
            if (PlayerList[playerKey].GetBombs() > 0
                out = 6; // Detonate Existing Bombs, Save Guaranteed
            else
                out = 5; // Place Bomb
           }
           else{
                out = MoveToWall(GameMap, CurrPos); // Go To Nearest
                Wall
           }

        }

        else { // Find Weakest Enemy
            map<char,Player>::iterator it = PlayerList.begin();
                if (it->first == playerKey){
                    it++;
                }
        char weakest = it->first;

        for( ; it != PlayerList.end(); it++){
        if (it->second.IsWeaker(PlayerList[weakest]) && (it->first !=
        playerKey))
            weakest = it->first;
        }

        out = MoveToAny(GameMap, CurrPos, 10, weakest); // Chase
        Weakest Enemy
        if (out == -1) // Unable to Chase because of Bombs
            out = 6;
    }
}

/* Clean Up */
initBool(danger);

/* Output To File */
outfile << out << std::endl;
outfile.close();
}
}
```

3. Pencarian jarak minimum

   Pencarian jarak minimum merupakan salah satu bagian yang selalu muncul pada bagian strategi greedy yang dijelaskan sebelumnya. Penentuan jarak minimum dilakukan dengan algoritma BFS (*Breadth-First Search*), yaitu dengan menjelajahi elemen peta hingga ditemukan elemen yang ingin dicari. Algoritma ini memanfaatkan struktur data *queue* untuk menjelajahi elemen peta di sekeliling posisi awal, baru kemudian menyebar secara radial. Pada kasus ini hanya akan ditunjukkan algoritma BFS untuk menentukan jarak menuju tembok terdekat. Pada program utama digunakan berbagai variasi BFS, yaitu BFS untuk mencari tembok, mencari area yang aman dari ledakan bom, mencari power up, dan mencari player lain. Namun, prinsip dari semua

program pencarian tersebut sama saja sehingga disini hanya akan dibahas salah satu kasus, yaitu mencari tembok terdekat. Berikut adalah algoritma pemanfaatan BFS untuk menentukan nilai minimum.

```
int FindWall(Map &m, pair<int,int> pos){
    bool visited[MAX][MAX];
    initBool(visited);

    int i = pos.first;
    int j = pos.second;
    queue<pair<int,int>> q;
    queue<int> qn;
    int n = 0;
    bool found = false;

    q.push(pos);
    qn.push(n);
    visited[i][j] = true;

    while(!q.empty() && !found){
        pair<int,int> currentIdx = q.front();
        n = qn.front();

        i = currentIdx.first;
        j = currentIdx.second;
        q.pop();
        qn.pop();

        if ((i+1 < m.GetRow()) && !visited[i+1][j] && !danger[i+1][j]
        && (m[i+1][j] != '#') && !isalpha(m[i+1][j]))
        {
            if (m[i+1][j] == '+')
                found = true;
            else{
                q.push(pair<int,int>(i+1, j));
                qn.push(n+1);
                visited[i+1][j] = true;
            }
        }

        if ((i-1 >= 0) && !visited[i-1][j] && !danger[i-1][j] && (m[i-
        1][j] != '#') && !isalpha(m[i-1][j]))
        {
            if (m[i-1][j] == '+')
                found = true;
            else{
                q.push(pair<int,int>(i-1, j));
                qn.push(n+1);
                visited[i+1][j] = true;
            }
        }

        if ((j+1 < m.GetCol()) && !visited[i][j+1] && !danger[i][j+1]
        && (m[i][j+1] != '#') && !isalpha(m[i][j+1]))
        {
            if (m[i][j+1] == '+')
                found = true;
            else{
                q.push(pair<int,int>(i, j+1));
                qn.push(n+1);
                visited[i][j+1] = true;
            }
        }
```

```
        if ((j-1 >= 0) && !visited[i][j-1] && !danger[i][j-1] &&
        (m[i][j-1] != '#') && !isalpha(m[i][j-1]))
        {
            if (m[i][j-1] == '+')
                found = true;
            else{
                q.push(pair<int,int>(i, j-1));
                qn.push(n+1);
                visited[i][j-1] = true;
            }
        }
    }

    if (found)
        return n;
    else
        return -1;
}
```

4. Penentuan langkah *bot*

   Penentuan langkah *bot* juga merupakan bagian program yang sangat penting untuk tercapainya algoritma greedy. Bagian program ini menentukan arah yang terbaik bagi *bot* untuk bergerak, entah itu maju, mundur, kiri, maupun kanan, untuk mencapai tempat yang ingin dicapai dengan langkah yang minimal. Penentuan langkah ini juga memanfaatkan BFS, tetapi bedanya BFS dilakukan pada keempat sisi disekitar posisi awal. Tujuannya untuk mengetahui apakah BFS dari sisi manakah yang menghasilkan nilai yang lebih kecil. Nilai yang terkecil yang kemudian akan menjadi arah langkah bagi *bot*.

```
int MoveToWall(Map &m, pair<int,int> pos){
    vector<int> dir;
    vector<int> val;
    int i = pos.first;
    int j = pos.second;
    int x;

    /* Up */
    x = FindWall(m, pair<int,int>(i-1,j));
    if ((i-1 >= 0) && (x != -1) && !danger[i-1][j] && !isdigit(m[i-
    1][j]) && ((m[i-1][j] == ' ') || (m[i-1][j] == '$') || (m[i-1][j]
    == '&') || (m[i-1][j] == '!'))){
        val.push_back(x);
        dir.push_back(1);
    }

    /* Left */
    x = FindWall(m, pair<int,int>(i,j-1));
    if ((j-1 >= 0) && (x != -1) && !danger[i][j-1] && !isdigit(m[i][j-
    1]) && ((m[i][j-1] == ' ') || (m[i][j-1] == '$') || (m[i][j-1] ==
    '&') || (m[i][j-1] == '!'))){
        val.push_back(x);
        dir.push_back(2);
    }
```

```
    /* Down */
    x = FindWall(m, pair<int,int>(i+1,j));
    if ((i+1 < m.GetRow()) && (x != -1) && !danger[i+1][j] &&
    !isdigit(m[i+1][j]) && ((m[i+1][j] == ' ') || (m[i+1][j] == '$')
    || (m[i+1][j] == '&') || (m[i+1][j] == '!'))){
        val.push_back(x);
        dir.push_back(4);
    }

    /* Right */
    x = FindWall(m, pair<int,int>(i,j+1));
    if ((j+1 < m.GetCol()) && (x != -1) && !danger[i][j+1] &&
    !isdigit(m[i][j+1]) && ((m[i][j+1] == ' ') || (m[i][j+1] == '$')
    || (m[i][j+1] == '&') || (m[i][j+1] == '!'))){
        val.push_back(x);
        dir.push_back(3);
    }

    if (val.empty()){
        return 0;
    }

    else{
        int imin = 0;
        for(int idx = 1; idx < (int)val.size(); idx++){
            if (val[idx] < val[imin])
                imin = idx;
        }
        return dir[imin];
    }
}
```

BAB IV
IMPLEMENTASI DAN PENGUJIAN

Setelah menyusun strategi Greedy dan mendesain struktur data untuk mendukung algoritma Greedy, berikutnya dilakukan pengujian dan analisis pada setiap hasil uji. Berikut adalah *pseudocode* program yang diikuti dengan pengujian dan analisis hasil pengujian.

4.1 Pseudocode Program

Seperti yang sudah dijelaskan sebelumnya pada bab 2, penambahan algoritma Greedy pada bot dilakukan pada file **SampleBot.cpp** di prosedur writeMoveFile. Pada bagian ini akan ditampilkan dua jenis pseudocode, yaitu yang menyerupai bahasa manusia, dan yang lebih spesifik mendefinisikan variabel dan prosesnya.

Pseudocode Greedy:

```
IF bot berada di zona yg tidak aman (in bomb radius)
    FIND jalan terpendek utk kabur
ELSE IF bot berada di zona aman:
    IF ada powerup di sekitar bot (jarak 6 langkah atau kurang)
           FIND jalan terpendek untuk kesana
    ELSE IF ada player lain yang berada di radius bomb bot
           DETONATE bomb kita
    ELSE IF ada player lain yang berhadapan dengan bot
           PUT bomb
    ELSE IF ada player lain di sekitar bot (jarak 3 langkah atau kurang)
    AND lebih lemah dari bot
           FIND jalan terpendek untuk menghampiri player tersebut
    ELSE IF masih ada tembok di peta
           IF di sebelah bot ada TEMBOK
                  IF masih ada bomb milik bot di arena
                         DETONATE bomb
                  ELSE
                         PUT bomb
           ELSE
                  FIND jalan terpendek untuk ke TEMBOK
    ELSE (bila tembok sudah habis)
           FIND jalan terpendek untuk menghampiri enemy terlemah
```

Berikut adalah pseudocode program yang lebih detail dalam menunjukkan variabel, proses, dan komentar.

```
void writeMoveFile(char playerKey, string filePath)
{Menentukan langkah yang dilakukan pada bot pada setiap langkahnya.
Langkah direpresentasikan dengan suatu variabel integer out, dengan
keterangan:
       0: tidak melakukan apa-apa
       1: bergerak ke atas
       2: bergerak ke kiri
       3: bergerak ke kanan
       4: bergerak ke bawah
       5: meletakkan bom
       6: mengubah waktu peledakan bom menjadi 1
}

Algoritma

output("Writing move file " + filePath + "/" + "move.txt")
open(filePath + "/" + "move.txt")

if (outfile.is_open())then
       {Instantiasi Objek}
       Map GameMap
       map<char,Player> PlayerList
       map<pair<int,int>, Bomb> BombList

       {Membaca objek dari file}
       string FileName ← filePath + "/" + "map.txt"
       Parse(GameMap, PlayerList, BombList, FileName)

       {Inisialisasi seluruh daerah peta menjadi aman}
       initBool(danger)

       {Mengatur daerah berbahaya pada peta}
       for(map<pair<int,int>,Bomb>::iterator it ← BombList.begin(); it !=
       BombList.end(); it++)
              setDanger(danger, GameMap, it->first.first, it->first.second,
              it> second.GetRadius())
       endfor

       {Mendapatkan posisi pemain}
       int a, b
       GameMap.search(playerKey, a, b)

       {Algoritma Greedy, penentuan langkah yang dipilih}
       int x, out
       pair<int,int> CurrPos (a,b)
       if (danger[a][b]) then {Jika berada pada daerah berbahaya}
              x ← FindEscape(GameMap, CurrPos) {Mencari daerah aman
terdekat}
              out ← MoveToEscape(GameMap, CurrPos, x) {Menuju daerah aman
terdekat}
       else {Berada di daerah aman}
              {Mencari Power Up}
              int p, q, r, s
              char c

              p ← FindAny(GameMap, CurrPos, '$') {Mencari jarak ke power up
$}
              q ← FindAny(GameMap, CurrPos, '!') {Mencari jarak ke power up
!}
              s ← Min2(p,q); {Menentukan power up mana yang lebih dekat}
```

```
            if ((s >= 0) and (s <= 6))then
            {Menemukan power up yang dekat, yaitu berjarak 6 langkah atau
      kurang}
                  if (s = p)then
                  {Jika power up terdekat adalah power up $}
                        out ← MoveToAny(GameMap, CurrPos, s, '$')
                        {bergerak menuju power up $}
                  else
                  {Power up terdekat adalah power up !}
                        out ← MoveToAny(GameMap, CurrPos, s, '!')
                        {bergerak menuju power up !}

                  endif
            else if (IsParallelBomb(GameMap, BombList, playerKey)) then
            {Jika bom yang dimiliki bot sejajar dengan pemain lain dan
            pemain tersebut berada pada radius bom bot}
                  out ← 6 {mengaktifkan bom}

            else if (IsParallelPlayer(GameMap, CurrPos,
PlayerList[playerKey].GetRadius()))then
            {Jika ada pemain lain yang sejajar dengan bot, dan radius bom
            bot kurang dari sama dengan jarak antara pemain lain dengan
            bot}
                  out ← 5
                  {Letakkan bom sejajar dengan musuh, sebagai pertahanan
            diri}

            else if (( (c = IsPlayerNearby(GameMap, CurrPos, 3)) ≠ -1)
and !PlayerList[playerKey].IsWeaker(PlayerList[c]))then
            { Terdapat pemain lain yang lebih lemah dibanding bot dengan
            jarak 3 langkah atau kurang}
                  out ← MoveToAny(GameMap, CurrPos, 3, c)
                  {bergerak ke arah pemain lain tersebut}

            else if ((x ← FindWall(GameMap, CurrPos)) ≠ -1)then
            {Jika terdapat tembok yang bisa dihancurkan}
                  if (x = 0) then
                  {Jika berada di samping tembok}
                        if (PlayerList[playerKey].GetBombs() >0) then
                        {masih ada bom yang belum meledak}
                              out ← 6 {ledakkan bom yang sudah ada}
                        else {Masih ada bom}
                              out ← 5 {Letakkan bom}
                        endif
                  else {Tidak berada di sebelah tembok}
                        out ← MoveToWall(GameMap, CurrPos)
                        {bergerak menuju tembok terdekat}
                  endif

            else {sudah tidak ada tembok}
            {mencari pemain lain terlemah}
                  map<char,Player>::iterator it ← PlayerList.begin()
                  if (it->first = playerKey) then
                        it++
                  endif
                  char weakest ← it->first

                  for( ; it ≠ PlayerList.end(); it++)
                        if (it->second.IsWeaker(PlayerList[weakest]) and
(it->first ≠ playerKey)) then
                              weakest ← it->first
                  endfor
                  out ← MoveToAny(GameMap, CurrPos, 10, weakest)
                  {Kejar musuh terlemah}
                  if (out = -1) then {tidak dapat mengejar, karena bom}
```

```
                            out ← 6 {ubah waktu peledakan bom menjadi 1}
                    endif
            endif
        endif

        {Inisialisasi ulang daerah bahaya}
        initBool(danger)

        {Menuliskan pilihan langkah ke outfile}
        outfile << out << std::endl
        outfile.close()
endif
```

4.2 Pengujian dam Analisis Hasil Pengujian

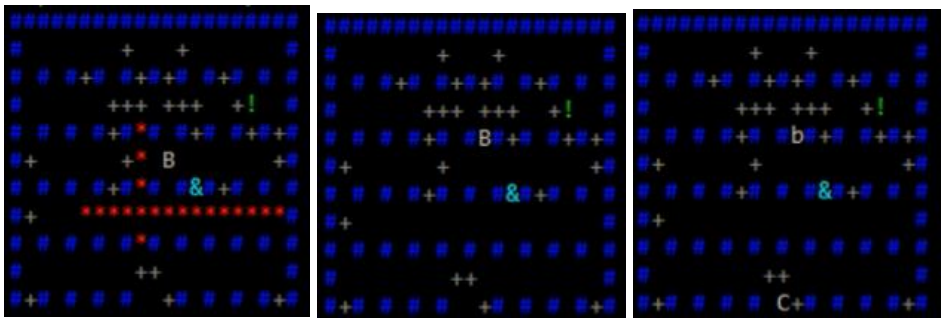
Setelah kode program selesai disusun, program diuji beberapa kali berdasarkan parameter greedy. *Bot* dikatakan lulus uji apabila berhasil melakukan strategi greedy. Strategi greedy berupa berjalan kearah tembok terdekat, berjalan ke daerah yang aman dari bom dengan jalan tersingkat, menghiraukan tembok ketika mengejar power up, dan mengejar musuh dengan jalan tersingkat. Berikut adalah beberapa pengujian beserta hasil dan analisisnya.

Catatan: Pada setiap gambar hasil pengujian, bot kelompok kami memiliki key huruf B pada peta. Gambar hasil pengujian diambil dari dokumentasi video yang kelompok kami buat dan sudah diunggah di Youtube.

1. Mengejar tembok terdekat

   Strategi greedy yang diterapkan pada *bot* adalah mengejar tembok terdekat untuk meledakan sehingga *bot* dapat mengumpulkan *points* dan *power up* sebanyak-banyaknya untuk memenangkan permainan. *Bot* dikatakan lulus uji apabila berhasil menghampiri tembok dengan jumlah langkah minimal dan meledakkan tembok tersebut.

   Hasil Pengujian:

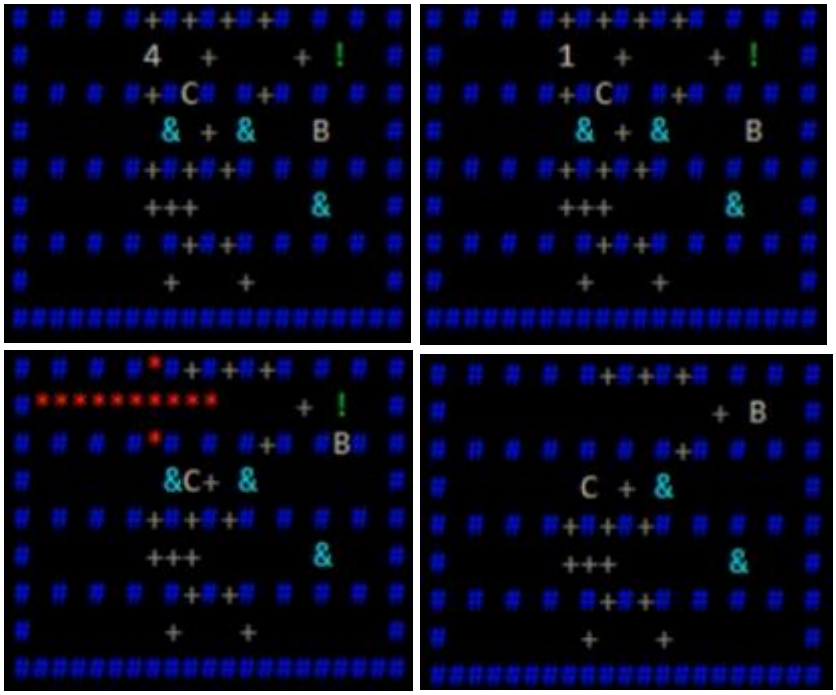


Hasil pengujian diatas menunjukkan tiga langkah yang dilakukan bot kelompok kami (huruf B) secara berurut dari kiri hingga kanan. Dari hasi pengujian diatas, jelas terlihat bahwa bot melangkah ke arah utara, kemudian meletakkan bom. Hal ini menunjukkan bahwa bot mampu mencari jalan terpendek menuju tembok, kemudian menaruh bom ketika sudah persis disebelah tembok. Dari hasil tersebut, disimpulkan bahwa *bot* **lulus uji mengejar tembok terdekat**.

2. Mendahulukan Power Up

   Untuk memenangkan permainan, strategi greedy *bot* kelompok kami adalah mendahulukan power up ketimbang tembok. Power up yang didahulukan adalah power up *blast radius* dan *super power up*, sedangkan power up *bomb bag* sengaja dihiraukan karena berdasarkan pengalaman kelompok kami bermain, power up *bomb bag* tidak secara signifikan membantu *bot* untuk mengejar *points* karena *fuse* dari bomb menjadi semakin lama. Power up tersebut sengaja dihiraukan dan biasanya akan diperoleh secara tidak sengaja oleh *bot* setidaknya sekali, dan menurut kelompok kami power up tersebut cukup dimiliki setidaknya satu saja, tidak perlu lebih.

   Hasil Pengujian:

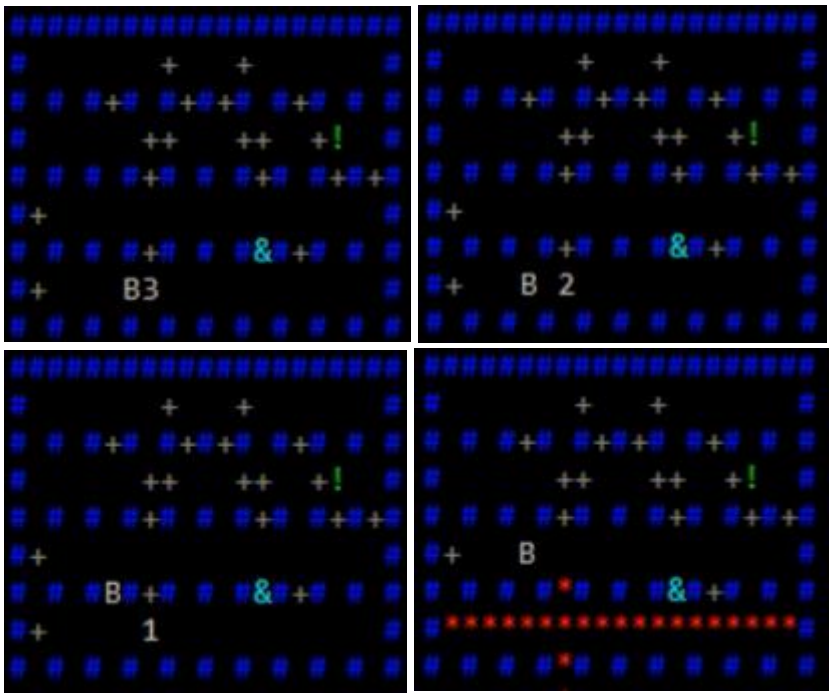


Urutan hasil pengujian diatas adalah dari kiri atas, kanan atas, kiri bawah, dan terakhir kanan bawah. Pada gambar kiri atas terlihat bahwa *bot* B sudah dekat dengan tembok, akan tetapi pada gambar selanjutnya *bot* berjalan kearah lain dan akhirnya mendapatka power up *blast radius*. Adapun power up *bomb bag* sengaja dihiraukan oleh *bot* karena kelompok kami sengaja tidak memfokuskan *bot* untuk mengejar power up *bomb bag*. Dari hasil pengujian diatas, *bot* dikatakan **lulus uji mendahulukan power up**.

3. Menghindari ledakan bom

   Pengujian ketiga dan yang paling penting adalah kemampuan *bot* untuk menghindari daerah ledakan bom. Ketika *bot* berada di daerah ledakan bom, *bot* mencari jalan tersingkat menuju petak yang aman dari ledakan bom. Berikut adalah hasil pengujiannya.

   Hasil Pengujian:

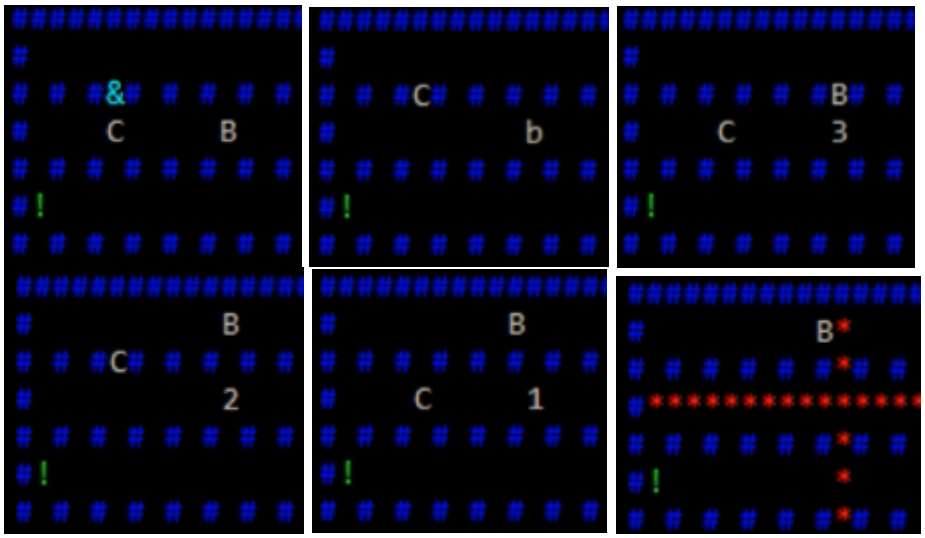


   Urutan berawal dari kiri atas, kanan atas, kiri bawah, kanan bawah. Dari hasil pengujian diatas, perhatikan transisi gerakan *bot* pada gambar urutan kedua menuju ketiga. Apabila dilihat seksama, ada tembok dengan jarak yang pendek di kiri, tetapi *bot* justru memilih untuk naik. Hal ini terjadi karena *bot* mengetahui daerah kiri tersebut tidak aman. Pada gambar terakhir bom

meledak dan *bot* berhasil menghindari ledakan tersebut. Artinya bot **lulus uji menghindari ledakan bom**.

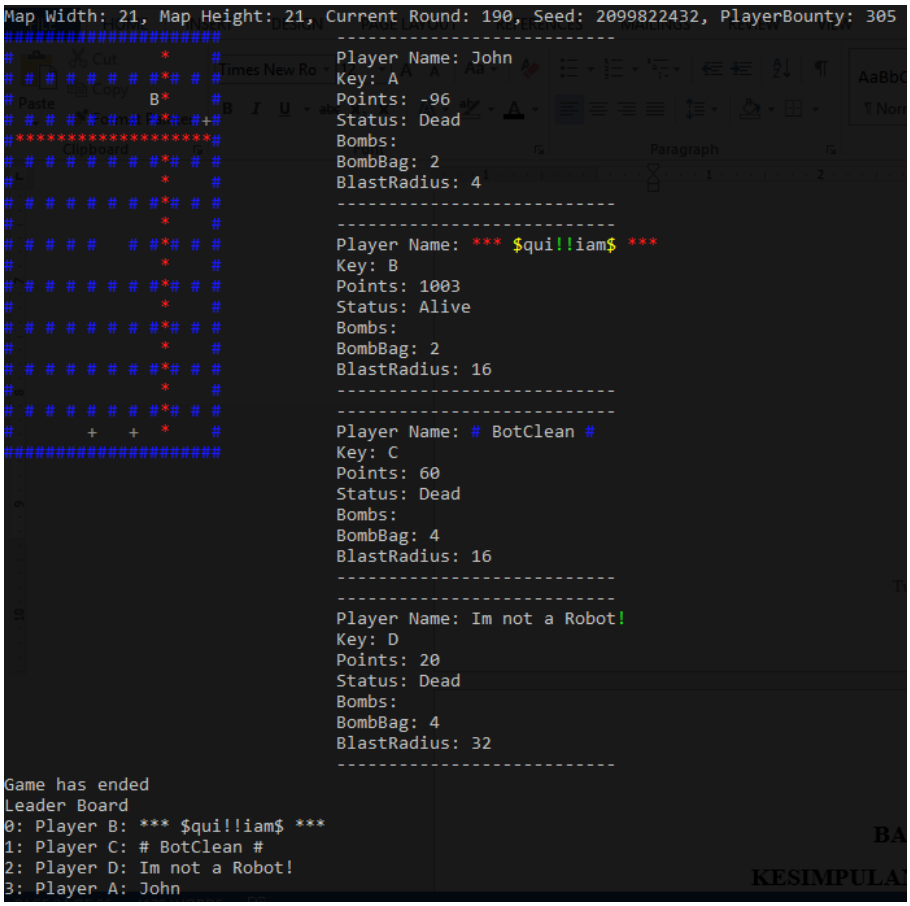
4. Mengejar musuh dan meletakkan bom

   Ketika ada musuh yang sudah dekat dengan *bot*, *bot* perlu melakukan aksi yang dapat melindungi dirinya, khususnya bila musuh tersebut mencoba membunuhnya. Untuk itu, setiap kali ada pemain lain yang mendekati *bot*, *bot* akan mengancam dengan meletakkan bom. Selain itu, apabila pemain lain yang mendekati *bot* cukup lemah relatif terhadap *bot*, justru *bot* akan mengejar pemain tersebut hingga pemain tersebut mati atau sudah cukup jauh dari *bot*. Pengujian dapat dilihat pada gambar dibawah.



   Urutan dimulai dari kiri atas hingga kanan bawah. Pada gambar pertama, terlihat bahwa *bot* B dan C saling berhadapan. Karena *bomb radius* yang dimiliki *bot* B cukup jauh untuk mengancam C, *bot* B meletakkan bom. Setelah itu *bot* B berusaha menghindari ledakan dari bomnya sendiri. Pada akhirnya *bot* C gugur dan B berhasil mengalahkan C. Dari pengujian ini, dapat disimpulkan bahwa *bot* **lulus uji mengejar musuh dan meletakkan bom**.

5. Hasil pertandingan



Diatas adalah hasil pertandingan antara *bot* kelompok kami (huruf B) dengan ketiga *reference bots*. Hasil diatas menunjukkan bahwa *bot* kelompok kami berhasil memenangkan pertandingan dengan skor yang tinggi. Skor sebesar 1003 diperoleh dengan membunuh ketiga *reference bots* dan menghancurkan tembok. Dari hasil pertandingan ini, dapat dikatakan bahwa *bot* kelompok kami sukses memenangkan pertandingan dengan pendekatan strategi greedy.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Algoritma Greedy dapat dimanfaatkan dalam penentuan langkah pada pembuatan bot permainan Bomberman. Algoritma greedy yang digunakan didasarkan pada cara untuk mempertahankan diri dalam permainan dan mendapatkan poin tertinggi. Algoritma greedy memanfaatkan BFS dan struktur data *queue* untuk mencari nilai minimum ketika mengejar tembok maupun melindungi diri. Secara umum, kelompok kami berhasil memanfaatkan strategi algoritma greedy dalam pembuatan bot permainan bomberman ini dengan baik. Nilai optimal berhasil kami dapatkan saat melakukan pengujian. Namun, bot kami masih memiliki kekurangan, yaitu belum memperhitungkan fuse dari bom ketika sedang mencoba meloloskan diri dari ledakan, sehingga terkadang bot akan berjalan pada arah yang menyebabkan bot terkena ledakan, khususnya bila arah tersebut adalah arah untuk keluar dari radius bomb dengan langkah terpendek.

5.2 Saran

Pembuatan bot ini sungguh bermanfaat untuk melatih kemampuan dalam merancang suatu strategi algoritma. Selain itu, proses pembuatan bot sangat mengasah pola pikir dan logika. Atas hal-hal tersebut, penulis menawarkan para pembaca, terutama yang memiliki ketertarikan dalam bidang game programming, ataupun kecerdasan buatan (artificial intelligence) untuk ikut mencoba membuat bot permainan Bombermannya sendiri.

DAFTAR PUSTAKA

https://github.com/EntelectChallenge/2016-Bomberman, akses terakhir 18 Februari 2017.

Munir, Rinaldi. 2009. *Diktat Kuliah Strategi Algoritmik IF2211 Strategi Algoritmik.* Departemen Teknik Informatika ITB.