BAB III

METODE PENELITIAN

Secara garis besar, penelitian akan dibagi dalam lima tahap. Langkah-langkah yang akan dilakukan:

1. Membuat program berdasarkan permainan "Hunt The Wumpus",
   a. membuat *user interface*,
   b. membuat *class*,
2. men-*generate* peta,
   a. inisialisasi peta,
   b. menempatkan obyek pada peta,
   c. menggambar peta,
3. merepresentasi peta ke dalam *graph*,
   a. inisialisasi *vertex*,
   b. menentukan isi *vertex* berdasarkan peta,
4. mengimplementasikan algoritma,
   a. merancang struktur data yang akan digunakan,
   b. implementasi algoritma Dijkstra dan Floyd-Warshall,
5. membandingkan kedua algoritma,
   a. menghitung dan mencatat performa yang dihasilkan berdasarkan *completeness*, *optimality*, *time complexity*, *space complexity*,dan
   b. membandingkan performa dari kedua algoritma.

3.1 Struktur Data

Permainan "Hunt The Wumpus" memiliki dua *class* yaitu *Main* dan *Vertex*. *ClassVertex* merepresentasikan setiap *vertex* dengan atribut sebagai berikut.

1. Bool Status, di mana *vertex* yang berisi lubang memiliki status *false* dan *vertex* lainnya bernilai *true*,
2. int jumTuj, yaitu jumlah *edge* yang dimiliki setiap *vertex*, dengan jumlah minimum 0 dan maksimum 4,
3. int x yang merupakan letak baris dari *vertex*, dimulai dari baris ke-0,
4. int y yang merupakan letak kolom dari *vertex*, dimulai dari kolom ke-0,
5. int nmr, yaitu nomor yang mewakili tiap *vertex*, di mana *vertex* dengan baris 0 dan kolom 0 diwakili dengan nomor 0, *vertex* dengan baris 0 dan kolom 1 diwakili dengan nomor 1, dan seterusnya,
6. arrayList tujuan, yang berisi nomor *vertex* tujuan yang dimiliki sebuah *vertex*,
7. arrayList cost, yang berisi *cost* dari *edge* yang dimiliki *vertex* di mana variabel *cost* dan tujuan untuk *edge* yang sama dari suatu *vertex* akan memiliki index yang sama,
8. array of int distance sebesar *dimension* x *dimension*, yang akan digunakan dalam algoritma Dijkstra dan berisi *cost* yang dibutuhkan untuk bergerak dari satu *vertex* ke *vertex* lainnya,

9. array of string prev sebesar *dimension* x *dimension*, yang akan digunakan dalam algoritma Dijkstra dan berisi nomor dari *previous vertex* dari suatu *vertex*, dan
10. array of boolean visited sebesar *dimension*x*dimension*, yang akan digunakan dalam algoritma Dijkstra dan menentukan apakah suatu *vertex* sudah dikunjungi atau belum.

Class vertex memiliki *constructor* yang menginisialisasi tiap variabel termasuk variabel yang akan digunakan untuk Dijkstra.

Class Main merupakan *class* utama yang terdiri dari *function-function* yang digunakan untuk men-*generate* peta, memetakan peta ke dalam *vertex*, dan mengimplementasikan *search algorithm. Class Main* memiliki atribut sebagai berikut.

1. Int dimension, yaitu besar dimensi dari permainan di mana terdapat dua pilihan yaitu 5 x 5 atau 30 x 30,
2. int cost, yaitu total *cost* yang dibutuhkan untuk berpindah dari satu *vertex* ke *vertex* lainnya,
3. int boxSize, yaitu besar kotak pada peta,
4. int source, yaitu nomor dari *vertex* awal,
5. int min, yaitu nomor dari *vertex* dengan *distance* terkecil yang akan digunakan pada Dijkstra,
6. vertex v, yaitu *object* dari *class Vertex*, yang akan merepresentasikan tiap *vertex* dalam permainan,
7. array of Vertex vertex sebesar dimensi x dimensi yang akan menampung tiap *vertex*,

8. array of int map yang merupakan *array* dua dimensi sebesar *dimension* yang mewakili baris dan kolom dari peta dan berisi angka yang mewakili obyek pada tiap kotak,
9. array of int matrix yang merupakan *array* dua dimensi dengan besar *dimension* x *dimension* yang akan digunakan dalam algoritma Floyd-Warshall,
10. array of int next yang merupakan *array* dua dimensi dengan besar *dimension* x *dimension* yang akan digunakan dalam algoritma Floyd-Warshall,
11. boolean _Dijkstra yang bernilai *true* bila algoritma Dijkstra dijalankan dan bernilai *false* jika algoritma Floyd-Warshall yang dijalankan, dan
12. graphics g, bitmap mybmp, icon wumpusIcon, icon wumpus2Icon, icon wumpus3Icon, icon goldIcon, icon pitIcon, icon playerIcon, dan brush myBrush yang merupakan atribut dari *class Drawing*, yaitu *class* turunan dari C# yang digunakan untuk menggambar.

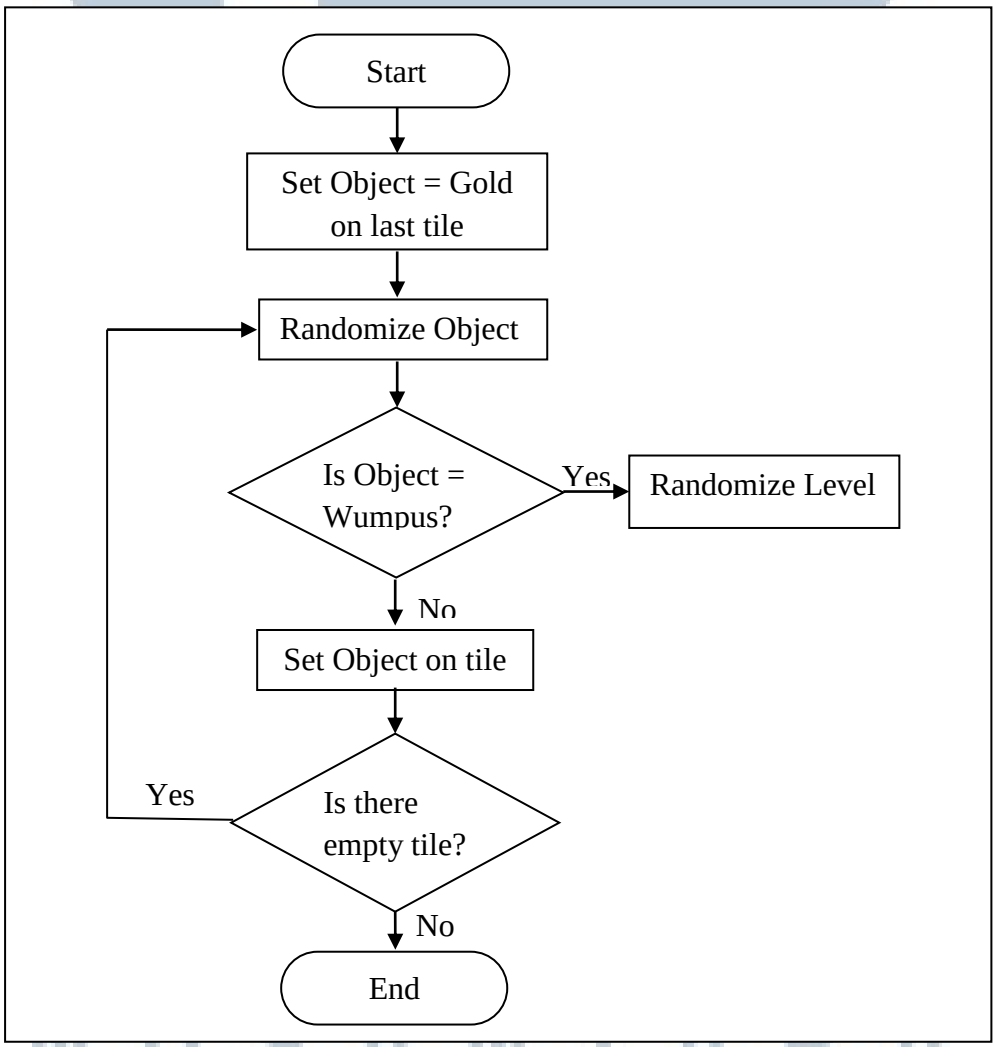
3.2 Men-*generate* Peta

Generate peta dilakukan dalam tiga tahapan yaitu inisialisasi peta, penempatan obyek pada peta, dan menggambar peta. Inisialisasi peta dilakukan dengan inisialisasi *array* dan mengosongkan peta.

Akan terdapat tujuh macam obyek pada permainan ”Hunt The Wumpus”. Obyek tersebut adalah lokasi awal, kotak kosong, lubang, emas, dan tiga jenis wumpus.

Penempatan obyek pada peta dilakukan secara acak, di mana setiap kotak memiliki peluang 25% kosong, 25% berisi lubang, dan 50% berisi monster. Kotak berisi monster dapat ditempati oleh monster *level* 1, *level* 2, atau *level* 3 dengan peluang 33% untuk tiap monster.

Berbeda dengan obyek lainnya, emas yang merupakan kotak tujuan tidak diacak melainkan berada pada kotak terakhir yaitu pojok kanan bawah. Sedangkan lokasi awal dapat ditentukan oleh pemain. Penempatan obyek pada peta dilakukan dengan *flowchart* sebagai berikut.



Gambar 3.1 *Flowchart* Penempatan Obyek

Setelah inisialisasi peta dan penempatan obyek, langkah terakhir yang dilakukan adalah menggambar peta. Proses ini akan dilakukan dengan *class Drawing* yang merupakan *class* turunan dari C#. Penggambaran peta akan dilakukan dalam beberapa tahap.

Tahap pertama adalah menggambar papan berwarna putih yang akan menjadi *background* dari peta. Tahap berikutnya adalah menggambar tiap kotak pada peta. Setelah tiap kotak tergambar, langkah terakhir adalah menggambar obyek dapat dilakukan. Papan permainan memiliki dua pilihan dimensi yaitu 5x5 dan 30x30. Hal ini menyebabkan penggambaran obyek untuk kedua pilihan dimensi dilakukan berbeda, mengingat keterbatasan *space* pada aplikasi.Pada dimensi 5x5, obyek direpresentasikan dengan gambar yang sesuai, namun pada dimensi 30x30, obyek direpresentasikan menggunakan lingkaran, dengan warna yang berbeda untuk tiap obyek. Lubang diwakilkan oleh lingkaran berwarna hitam, emas diwakilkan oleh lingkaran berwarna kuning, dan monster akan diwakilkan oleh lingkaran berwarna merah, di mana semakin besar *level* monster, maka warna merah akan semakin gelap.

3.3 Merepresentasikan Peta

Setelah peta di-*generate*, maka peta direpresentasikan dalam *graph* agar dapat mengimplementasi *search algorithm. Graph* direpresentasikan dengan *weight matrix* dan *priority queuenya* diimplementasikan dengan *unordered array,*Representasidilakukan dengan memeriksa isi tiap kotak dan memeriksa isi kotak sekitarnya. Pemeriksaan dilakukan per baris dan dilakukan untuk kotak

yang tidak berisi lubang. Setelah pemeriksaan selesai dilakukan, maka atribut dari *vertex* akan di-*update* sesuai dengan hasil pemeriksaan.

3.4 Pseudocode Algoritma Dijkstra dan Floyd-Warshall

Implementasi algoritma Dijkstra dan Floyd-Warshall dilakukan untuk mencari jarak dengan *cost* minimal. Implementasi Dijkstra dilakukan berdasarkan *pseudocode*yang dijabarkan oleh Anany Levitin (Levitin, 2007: 323) seperti gambar 2.15.*Pseudocode*tersebut digunakan untuk *single-source* Dijkstra, sehingga membutuhkan modifikasi agar Dijkstra dilakukan untuk setiap *vertex* dalam *graph.*Modifikasi cukup dilakukan dengan menjalankan algoritma Dijkstra pada semua *vertex.*

Implementasi Floyd-Warshall dilakukan berdasarkan *pseudocode*yang dijabarkan oleh Anany Levitin (Levitin, 2007: 290) seperti gambar 2.16. Berbeda dengan Dijkstra yang dapat digunakan untuk merekonstruksi jalur yang digunakan, Floyd-Warshall hanya menghitung *cost* dari setiap *vertex* yang berpasangan. Hal ini menyebabkan Floyd harus dimodifikasi sehingga dapat merekonstruksi jalur yang digunakan. Modifikasi pada Floyd tidak mempengaruhi performa algoritma sehingga tidak mempengaruhi hasil penelitian.Gambar 3.3 memperlihatkan*pseudocode* untuk Floyd-Warshall yang sudah dimodifikasi.

```
for (k=0;k<n;k++)
  for (i=0;i<n;i++)
    for (j=0;j<n;j++)
      if (d[i][k] + d[k][j] < d[i][j]) {
        d[i][j] = d[i][k]+d[k][j];
        p[i][j] = p[k][j];
      }
print_path (int i, int j) {
  if (i!=j)
    print_path(i,p[i][j]);
  print(j);
}
```

Gambar 3.2*PseudocodeModified* Floyd-Warshall

Untuk mengimplementasikan algoritma Floyd, pertama-tama harus dibuat matriks dua dimensi sebesar jumlah *vertex* pada *graph.* Tiap baris pada matriks menunjukkan *vertex* awal dan tiap kolom menunjukkan *vertex* tujuan. Bila terdapat *edge* antara *vertex* awal *i* dan *vertex* tujuan *j*, maka matriks (i,j) dan matriks (j,i) akan berisi *cost* dari *edge* tersebut. Apabila tidak terdapat *edge*, maka matriks akan diisi dengan *integer value* yang besar untuk menunjukkan bahwa *edge* tidak dapat dilewati. Selain matriks untuk menyimpan *vertex*, juga dibutuhkan matriks serupa yang digunakan untuk menyimpan jalur untuk proses rekonstruksi jalur.

Pada algoritma Dijkstra maupun Floyd, terdapat kemungkinan di mana terdapat lebih dari satu jalur minimal dengan *cost* yang sama. Walaupun demikian jalur yang dipilih kedua algoritma akan sama, sebab proses pemeriksaan*vertex* dilakukan dengan urutan yang sama untuk kedua algoritma.

3.5 Perbandingan Algoritma

Kedua algoritma akan dibandingkan berdasarkan empat kriteria yakni *completeness*, *optimality*, *space complexity,* dan *time complexity*. Perbandingan didapatkan melalui percobaan yang dilakukan beberapa kali untuk tiap algoritma dan dimensi yang berbeda.

Untuk *completeness,* percobaan dilakukan dengan melihat dan merekonstruksi jalur secara manual untuk menentukan apakah solusi dapat ditemukan apabila solusi tersebut ada. Dalam "Hunt The Wumpus", terdapat dua kondisi yang menyebabkan solusi tidak ada. Kondisi pertama adalah apabila tidak terdapat jalur antara kotak awal dan tujuan karena terhalang oleh lubang. Hal ini dapat diatasi dengan meng-*generate* ulang peta. Sedangkan kondisi yang kedua adalah apabila kotak awal yang ditentukan berisi lubang. Kondisi ini dapat diatasi dengan menghindari penempatan kotak awal pada kotak yang berisi lubang. Percobaan untuk membandingkan *completeness* akan dilakukan pada dimensi 5x5 dan 30x30 untuk tiap algoritma menggunakan *vertex* 0 sebagai *source*. Percobaan akan diulang sebanyak sepuluh kali.

Sedangkan untuk *optimality,* percobaan dilakukan dengan menghitung secara manual jalur minimal yang dapat ditempuh dan membandingkannya dengan hasil perhitungan algoritma Dijkstra dan Floyd. Percobaan akan dilakukan hanya pada dimensi 5 x 5 untuk tiap algoritma dengan menggunakan *vertex* 0 sebagai *source.*Percobaan hanya dilakukan pada dimensi 5x5 karena hasil perhitungan algoritma akan sama untuk setiap dimensi dan untuk memudahkan penghitungan jalur terpendek secara manual. Percobaan akan dimulai dari kasus yang hanya

memiliki satu solusi dan dilanjutkan dengan kasus yang memiliki dua solusi dan seterusnya hingga memiliki 10 solusi.

Untuk *space complexity*, perbandingan akan dilakukan dengan mencari batas maksimum dari dimensi peta di mana algoritma dapat berjalan. Percobaan akan dimulai pada dimensi 30x30 untuk kedua algoritma dan dimensi akan terus ditambah hingga program tidak dapat dijalankan. Setelah batas maksimum dimensi didapatkan, akan dilakukan penghitungan *space usage* secara manual. Penghitungan dilakukan dari dimensi 30x30 hingga dimensi maksimum untuk kedua algoritma.

Perbandingan *time complexity* akan dilakukan dengan menghitung berapa lama waktu yang dibutuhkan untuk menjalankan algoritma. Terdapat tombol Dijkstra dan Floyd pada permainan yang akan menjalankan *function* Dijkstra dan Floyd. Penghitungan waktu akan dimulai saat tombol ditekan dan berhenti saat proses selesai.Penghitungan waktu akan dilakukan menggunakan *class* timer yang tersedia pada C#. Percobaan dilakukan sebanyak sepuluh kali pada dimensi 20x20, 25x25, 30x30 untuk tiap algoritma. Selain itu, akan dilakukan percobaan sebanyak tiga kali untuk dimensi 40x40, 42x42, 44x44, dan seterusnya hingga mencapai batas maksimum dimensi.

Semua hasil percobaan akan dicatat dan digunakan untuk membandingkan performa dari algoritma Dijkstra dan Floyd secara keseluruhan.