BAB IV

GAMBARAN UMUM OBYEK PENELITIAN

4.1 *PRE-PRODUCTION*

Tahap *pre-production* adalah tahap dalam pembuatan *game* yang berfokus pada ide dan konsep pengembangan *game*. Tujuan dari proses ini adalah untuk memperoleh gambaran jelas tentang *game* yang akan dibuat serta membantu dalam menentukan target serta *deadline* dalam proses pembuatan *game*.

4.1.1 *Story*

Game "Trooper" merupakan sebuah *game* yang memiliki alur cerita sebagai berikut, di suatu markas *army* terdapat sekelompok tentara yang sedang menjalankan tugas mereka. Di markas tersebut dipimpin oleh seorang komandan yang sangat gagah dan juga pemberani. Mereka bertugas untuk menjaga keamanan markas tersebut dari serangan robot perusak.

Namun suatu ketika hal yang tidak diinginkan pun terjadi, ada sebuah robot perusak yang dikirim oleh musuh untuk menyerang markas agar musuh dapat menguasai daerah tersebut. Padahal dimarkas tersebut banyak kotak berisi senjata dan pasokan bahan makanan yang juga diincar musuh.

Kemudian markas tersebut diserang secara diam-diam oleh robot perusak sehingga hanya tinggal komandan yang tersisa dari penyerangan tersebut. Dengan keberaniannya sang komandan tidak menyerah begitu saja. Disinilah misi penyelamatan markas oleh sang komandan yang

berjuang sendirian untuk mempertahankan markasnya, sekaligus menjadi misi dari permainan *game* ini.

4.1.2 Aturan Permainan

- Pemain memiliki senjata untuk mempertahankan diri dengan senjata yang dapat digunakan untuk menyerang bos di final stage.
- Pada level 1-3 dan level bos jika pemain tertangkap oleh robot maka pemain tidak dapat melanjutkan permainan.
- Pada level 1-3 pemain akan diberikan waktu untuk mengambil point, apabila point belum terambil semua dan waktu telah habis maka pemain tidak dapat melanjutkan permainan.
- Pada level bos pemain akan diberikan waktu untuk menyerang si bos dengan menggunakan senjata yang telah ada dipundaknya, apabila waktu habis atau pemain terkena maka akan *game over*.

4.1.3 Cara Kerja Algoritma A* *Pathfinding*

Didalam algoritma A* menggunakan dua senarai yaitu OPEN dan CLOSED. Pada algoritma A* memiliki tiga kondisi dimana *NPC* berada di keadaan OPEN, berada dikeadaan CLOSED, dan tidak berada dikeduanya. Jika *NPC* dalam keadaan OPEN maka akan dilakukan pengecekan apakah perlu pengubahan parent atau tidak tergantung pada nilai g melalui parent lama maupun parent baru. Jika diparent baru nilai g yang memiliki nilai lebih kecil, maka akan dilakukan pengubahan parent. Apabila dilakukan perubahan parent, maka akan dilakukan pembaruan nilai g dan f pada *NPC* tersebut. Dengan pembaruan ini, maka *NPC* tersebut akan memiliki

kesempatan yang lebih besar untuk terpilih sebagi simpul terbaik (*best node).*

Jika *NPC* sudah pernah berada di CLOSED, maka akan dilakukan pengecekan apakah perlu adanya pengubahan parent atau tidak. Apabila dilakukan pengubahan, maka akan dilakukan pembaharuan nilai g dan h pada *NPC* tersebut sampai keujung node dan juga yang sudah pernah ada di keadaaan OPEN. Dengan pembaharuan ini maka semua simpul yang terlewati *NPC* memiliki kesempatan lebih besar untuk terpilih sebagai simpul terbaik.

Jika *NPC* tidak berada dikeadaan OPEN maupun CLOSED, maka *NPC* tersebut akan dimasukan didalam keadaan OPEN. Kemudian hitung biaya *NPC* tersebut dengan rumus $f = g + h$. Algoritma A* ini akan memberikan solusi terbaik yang optimal didalam arena.

Fungsi evaluasi yang digunakan dalam algortima A* ini adalah:

$$f = g + h \text{ .....(2)}$$

Dimana :

f (n) : estimasi total biaya yang melalui n sampai node tujuan

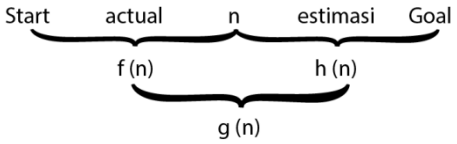
g (n) : biaya yang diperlukan untuk mencapai goal melalui node n

h (n): estimasi biaya dari node ke goal state

Sehingga pada algortima A* terdapat 2 faktor yang diperhatikan, yaitu:

1. Path dari start sampai ke current position n melalui fungsi g (n), ini merupakan nilai sebenarnya.
2. Path dari current position n melalui fungsi h (n), ini merupakan suatu biaya perkiraan atau estimasi biaya.

Dari faktor tersebut maka total dari kedua biaya inilah yang akan menghasilkan biaya estimasi path dari start ke goal melalui current position n.



Gambar 4.1 Visuialisasi fungsi A*

Dari perhitungan menggunakan algoritma A* akan didapat solusi yang akan selalu diterima apabila nilai h tidak melebihi biaya atau cost minimal untuk mencapai tujuan yang sebenarnya. Nilai heuristik h (n) juga harus kontant atau tetap agar nilai yang didapat dari perhitungan dengan algoritma A* dapat optimal.

Fungsi heuristik dari algortima A* juga sangat mempengaruhi hasil dari perhitungannya, karena salah satu fungsi dari nilai heuristik dapat digunakan untuk melakukan perncarian untuk tipe map grid yang merupakan fungsi heuristik manhattan. Fungsi ini hanya akan menjumlahkan selisih nilai x dan nilai z dari dua buah titik. Berikut ini adalah perhitungan dari fungsi manhattan :

$$\boxed{h\,(n) = \text{abs}(n.x - tujuan.x) + \text{abs}(n.z - tujuan.z)} \quad \text{.....(3)}$$

Dimana :

n.x : Koordinat x titik start

n.z : Koordinat z titik start

tujuan.x : Koordinat x titik tujuan

tujuan.z : Koordinat z titik tujuan

Sebuah nilai heuristik dikatakan dapat diterima apabila nilai h'(n) ≤ (n), dimana besarnya biaya perkiraan yang mendekati tujuan tidak boleh lebih besar dibandingkan dengan biaya perkiraan pada node saat ini. Berikut adalah psudeocode algoritma A*:

1. Simpul awal atau current state. Simpan current state pada list dengan nama OPEN dan list CLOSED = { }.
2. Loop :
   a. Cari node (n) dengan nilai f (n) yang memiliki nilai minimum dalam list OPEN.
   b. Keluarkan current state dari dalam OPEN lalu simpan kedalam CLOSED list.
   c. Untuk setiap *NPC* dari node current state lakukan pengecekan seperti berikut :
      - Jika tidak dapat dilalui atau sudah berada didalam CLOSED, maka tidak akan dihitung lagi.
      - Jika belum pernah tersimpan didalam list OPEN, maka simpan didalam OPEN list. Kemudian buat parent baru dari node ini dan simpan nilai f, g, dan h dari node ini.
      - Jika sudah berada didalam list OPEN, lalu cek bila node saat ini memiliki nilai f(n) lebih kecil dari node sebelumnya. Jika node g'(n) < g(n) maka ganti parent dari node g'(n) yang ada didalam list OPEN menjadi current node. Kemudian lakukan perhitungan terbaru dari nilai f(n) yang sekarang.

d. Loop dihentikan apabila :

- Node tujuan telah terdapat didalam OPEN.
- Belum menemukan node tujuan, sementara list dalam OPEN sudah tidak ada lagi yang artinya tidak ditemukan solusi

3. Lakukan backtarck urutan dari node tujuan menuju node awal,maka solusi telah ditemukan.

4.2 *PRODUCTION*

Pada tahap ini merupakan proses dimana mengubah ide dari konsep yang telah dibuat menjadi sebuah *gameplay* pada layar. Sebelum mengimplementasikannya perlu adanya *Material Collection* untuk pengumpulan segala kebutuhan produksi, seperti karakter yang akan digunakan dalam *game*, sketsa konsep, *sound effect*, dan AI yang akan diterapkan pada karakter *game*.

AI pada *game* ini akan diterapkan pada karakter musuh atau *NPC*. Algoritma yang digunakan adalah A* untuk diimplementasikan pada musuh atau *NPC* yang nantinya akan berfungsi atau mengambil peran untuk menangkap atau mencari pemain. Dalam konsep Algoritma A* sangat erat kaitannya dengan *pathfinding* yang berfungsi salah satu dasar algoritma dalam menggerakan karakter dalam sebuah *game.* Musuh atau *NPC* nantinya akan dapat menangkap atau mencari pemain dengan menggunakan titik optimal dari jarak yang ditempuh.

Semua tool yang dibutuhkan pada pembuatan *game* ini harus kompatibel agar proses ini dapat diimport kedalam software unity supaya dapat diolah menjadi sebuah *game*. Hasil dari perhitungan Algoritma A* juga akan dimasukan di dalam unity dengan menggunakan bahasa C# menjadi *script game*.

4.3 *POST PRODUCTION*

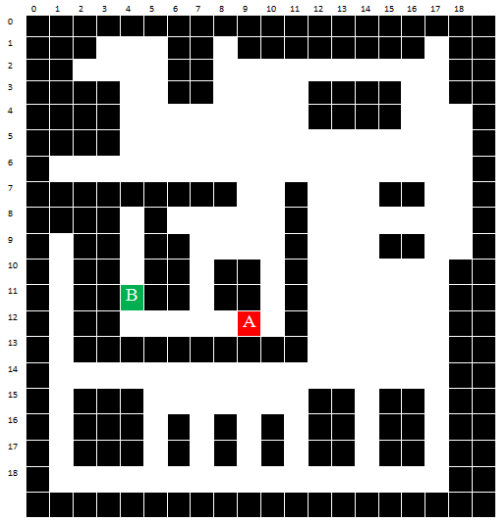
Setelah proses pembuatan *game* selesai, maka akan dilakukan beberapa pengujian atau testing. Untuk menguji fungsionalitas dari *game* yang telah dibuat yaitu menggunakan *Balckbox*, pada tahap ini akan dilakukan pengujian kompitabel terhadap *game* dengan beberapa perangkat smartphone. Adapun pengujian pada device terdapat beberapa jenis tes yaitu :

- Animasi : Pengujian gerak animasi player, musuh atau NPC, Main Camera yang ikut bergerak mengikuti gerak player dan perpindahan dari tiap scene berjalan sesuai dengan rancangan yang telah dibuat. Tidak ada bug atau error dalam mensetting pergerakan seluruh komponen yang terdapat dalam tiap scene.
- Audio : audio digunakan sebagai soundtrack saat di main menu maupun saat permainan sedang dimainkan dan sound effect pada tiap pergerakan melompat, berlari, menembak dan dipastikan semua sound berbunyi dengan baik saat permainan sedang berlangsung.
- Button : jenis pengujian ini maksudnya adalah apakah fungsi button dapat sesusai dengan fungsi semestinya.
- Grafis : pengujian grafis ditentukan oleh masih adakah sisi kosong dari layar device. Pada mestinya saat dimainkan di device apapun tampilan gambar teta fullscreen dengan tidak ada bagian terpotong maupun ada area kosong.

- Screenplay : semua screen yang tampil secara berurutan didalm aplikasi sehingga berjalan sesuai dengan alur storyboard atau tidak.
- *Game*play : jenis tes yang melakukan pengecekan pada alur permainan dari *game* ini apakah masih ada bug atau tidak.
- Smoothness : tes perpindahan dari tiap scene apakah terlihat kasar atau halus dan masih terjadi lag atau tidak.
- Memory : seberapa banyak *game* ini memakan kapsaitas memori pada setiap device.

4.4 IMPLEMENTASI ALGORITMA A* *PATHFINDING*

Tujuan dari analisa dari algoritma A* *pathfinding* ini adalah untuk mengetahui bagaimana cara kerja dari algoritma ini untuk menyelesaikan permasalahan yang dihadapi dan untuk mengetahui apakah algoritma yang diterapkan dapat bekerja dengan baik atau tidak, sehingga dapat diketahui apakah benar-benar menghasilkan seperti yang diharapkan.



Gambar 4.2 Arena Graph

Dari gambar 4.2 menunjukan papan grid yang akan diasumsikan sebagai arena dari *game*. Grid yang telah dibuat menggunakan skala 1 : 10 dengan grid arena yang sebenarnya. Warna merah pada papan grid merupakan starting point atau titik awal dari *NPC* sebelum bergerak. Warna hijau pada papan grid merupakan end point atau titik yang akan dikejar oleh *NPC*. Warna hitam pada grid merupakan tembok penghalang yang tidak dapat dilewati.

Pada algoritma A* *pathfinding* mempertimbangkan biaya yang diperlukan untuk mencapai titik tujuan dimana biaya yang paling rendahlah yang akan diambil sebagai bestnode untuk melangkah. Dimisalkan perkiraan biaya yang diperlukan untuk melangkah secara horizontal dan vertikal adalah 10, sedangkan nilai heuristik dihitung menggunakan fungsi manhattan yang nanti akan digunakan dalam perhitungan A* pada papan grid yang memiliki start awal (9,12) dan goal (4,11) yang dapat disajikan dalam bentuk tabel 4.1 sebagai berikut :

Tabel 4.1 Nilai Heuristik

Start_X	Goal_X	Start _Z	Goal_Z	h(n)=abs(startX-goalX)+abs(startZ-goalZ)	Initial Node
3	1	4	11	11	
4	1	4	11	10	
5	1	4	11	11	
8	1	4	11	14	
17	1	4	11	23	
2	2	4	11	11	
3	2	4	11	10	
4	2	4	11	9	
5	2	4	11	10	
8	2	4	11	13	
9	2	4	11	14	
10	2	4	11	15	
11	2	4	11	16	
12	2	4	11	17	
13	2	4	11	18	
14	2	4	11	19	
15	2	4	11	20	
16	2	4	11	21	

Start_X	Goal_X	Start _Z	Goal_Z	h(n)=abs(startX-goalX)+abs(startZ-goalZ)	Initial Node
17	2	4	11	22	
4	3	4	11	8	
5	3	4	11	9	
8	3	4	11	12	
9	3	4	11	13	
10	3	4	11	14	
11	3	4	11	15	
16	3	4	11	20	
17	3	4	11	21	
4	4	4	11	7	
5	4	4	11	8	
6	4	4	11	9	
7	4	4	11	10	
8	4	4	11	11	
9	4	4	11	12	
10	4	4	11	13	
11	4	4	11	14	
16	4	4	11	19	
17	4	4	11	20	
18	4	4	11	21	
4	5	4	11	6	
5	5	4	11	7	
6	5	4	11	8	
7	5	4	11	9	
8	5	4	11	10	
9	5	4	11	11	
10	5	4	11	12	
11	5	4	11	13	
12	5	4	11	14	
13	5	4	11	15	
14	5	4	11	16	
15	5	4	11	17	
16	5	4	11	18	
17	5	4	11	19	
18	5	4	11	20	
1	6	4	11	8	
2	6	4	11	7	
3	6	4	11	6	
4	6	4	11	5	
5	6	4	11	6	
6	6	4	11	7	
7	6	4	11	8	
8	6	4	11	9	
9	6	4	11	10	
10	6	4	11	11	
11	6	4	11	12	
12	6	4	11	13	

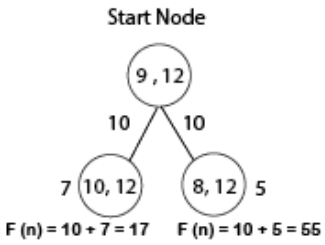
Start_X	Goal_X	Start _Z	Goal_Z	h(n)=abs(startX-goalX)+abs(startZ-goalZ)	Initial Node
13	6	4	11	14	
14	6	4	11	15	
15	6	4	11	16	
16	6	4	11	17	
17	6	4	11	18	
18	6	4	11	19	
9	7	4	11	9	
10	7	4	11	10	
12	7	4	11	12	
13	7	4	11	13	
14	7	4	11	14	
17	7	4	11	17	
18	7	4	11	18	
6	8	4	11	5	
7	8	4	11	6	
8	8	4	11	7	
9	8	4	11	8	
10	8	4	11	9	
12	8	4	11	11	
13	8	4	11	12	
14	8	4	11	13	
15	8	4	11	14	
16	8	4	11	15	
17	8	4	11	16	
18	8	4	11	17	
1	9	4	11	5	
4	9	4	11	2	
6	9	4	11	4	
7	9	4	11	5	
8	9	4	11	6	
9	9	4	11	7	
10	9	4	11	8	
12	9	4	11	10	
13	9	4	11	11	
14	9	4	11	12	
17	9	4	11	15	
18	9	4	11	16	
1	10	4	11	4	
4	10	4	11	1	
6	10	4	11	3	
7	10	4	11	4	
10	10	4	11	7	
12	10	4	11	9	
13	10	4	11	10	
14	10	4	11	11	
15	10	4	11	12	
16	10	4	11	13	

Start_X	Goal_X	Start _Z	Goal_Z	h(n)=abs(startX-goalX)+abs(startZ-goalZ)	Initial Node
17	10	4	11	14	
1	11	4	11	3	
4	11	4	11	0	**Goal**
6	11	4	11	2	
7	11	4	11	3	
10	11	4	11	6	
12	11	4	11	8	
13	11	4	11	9	
14	11	4	11	10	
15	11	4	11	11	
16	11	4	11	12	
17	11	4	11	13	
1	12	4	11	4	
4	12	4	11	1	
5	12	4	11	2	
6	12	4	11	3	
7	12	4	11	4	
8	12	4	11	5	
9	12	4	11	6	**Start**
10	12	4	11	7	
12	12	4	11	9	
13	12	4	11	10	
14	12	4	11	11	
15	12	4	11	12	
16	12	4	11	13	
17	12	4	11	14	
1	13	4	11	5	
12	13	4	11	10	
13	13	4	11	11	
14	13	4	11	12	
15	13	4	11	13	
16	13	4	11	14	
17	13	4	11	15	
1	14	4	11	6	
2	14	4	11	5	
3	14	4	11	4	
4	14	4	11	3	
5	14	4	11	4	
6	14	4	11	5	
7	14	4	11	6	
8	14	4	11	7	
9	14	4	11	8	
10	14	4	11	9	
11	14	4	11	10	
12	14	4	11	11	
13	14	4	11	12	
14	14	4	11	13	

Start_X	Goal_X	Start _Z	Goal_Z	h(n)=abs(startX-goalX)+abs(startZ-goalZ)	Initial Node
15	14	4	11	14	
16	14	4	11	15	
17	14	4	11	16	
1	15	4	11	7	
5	15	4	11	5	
6	15	4	11	6	
7	15	4	11	7	
8	15	4	11	8	
9	15	4	11	9	
10	15	4	11	10	
11	15	4	11	11	
14	15	4	11	14	
17	15	4	11	17	
1	16	4	11	8	
5	16	4	11	6	
7	16	4	11	8	
9	16	4	11	10	
11	16	4	11	12	
14	16	4	11	15	
17	16	4	11	18	
1	17	4	11	9	
5	17	4	11	7	
7	17	4	11	9	
9	17	4	11	11	
11	17	4	11	13	
14	17	4	11	16	
17	17	4	11	19	
1	18	4	11	10	
2	18	4	11	9	
3	18	4	11	8	
4	18	4	11	7	
5	18	4	11	8	
6	18	4	11	9	
7	18	4	11	10	
8	18	4	11	11	
9	18	4	11	12	
10	18	4	11	13	
11	18	4	11	14	
12	18	4	11	15	
13	18	4	11	16	
14	18	4	11	17	
15	18	4	11	18	
16	18	4	11	19	
17	18	4	11	20	

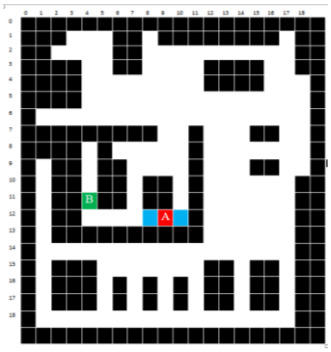
Kemudian dengan tabel 4.1 pada biaya perkiraan dan juga nilai cost untuk bergerak atau berpindah grid dengan dimisalkan perkiraan biaya sebesar 10, maka akan dilakukan penghitungan sesuai fungsi *f(n) = g(n) + h(n)* pada node yang terhubung langsung dengan node awal saat ini. Berikut langkah pencarian pada algoritma A* *Pathfinding* :

- **Langkah 1**



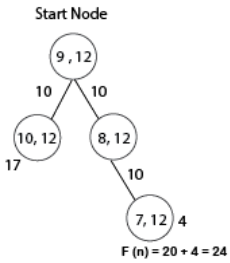
Gambar 4.3 Perhitungan Langkah 1

Dari gambar 4.3 didapat nilai pada simpul pertama yaitu f(n) = 10 + 7 =17 dan pada simpul kedua didapat nilai f(n) = 10 + 5 =15, maka akan diperlebar pada simpul kedua terlebih dahulu. Simpan node [9,12] didalam Closed, sedangkan node [10,12] dan [8,12] didalam Open. Maka pergerakan didalam papan grid akan ditandai dengan kotak warna biru sebagai Open List dan kotak warna kuning merupakan Closed List seperti gambar 4.4 :



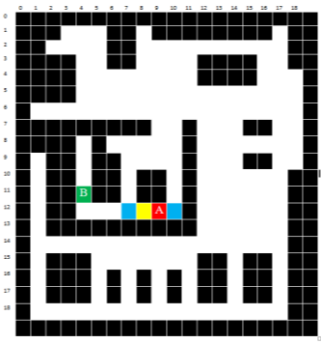
Gambar 4. 4 Langkah 1 pergerakan papan grid

- **Langkah 2**



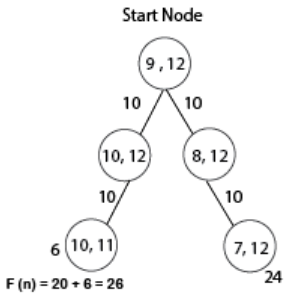
Gambar 4.5 Perhitungan Langkah 2

Untuk memilih simpul selanjutnya seperti gambar 4.5 pilih simpul dengan nilai f yang paling rendah, kemudian hitung dengan rumus A* untuk menentukan harga selanjutnya. Simpan node [9,12] dan [8,12] kedalam Closed list serta node [10,12] dan [7,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.6 :



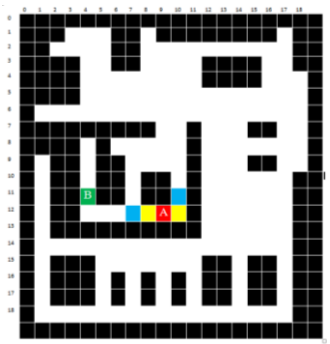
Gambar 4. 6 Langkah 2 pergerakan papan grid

- **Langkah 3**



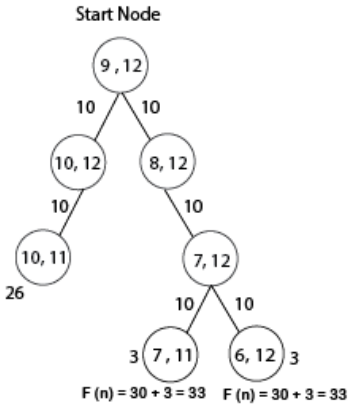
Gambar 4.7 Perhitungan Langkah 3

Ulangi langkah seperti sebelumnya dengan mencari nilai f yang paling rendah pada gambar 4.7 kemudian hitung menggunakan rumus A*. Simpan node [9,12] [8,12] dan [10,12] kedalam Closed list serta node [10,11] dan [7,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.8:



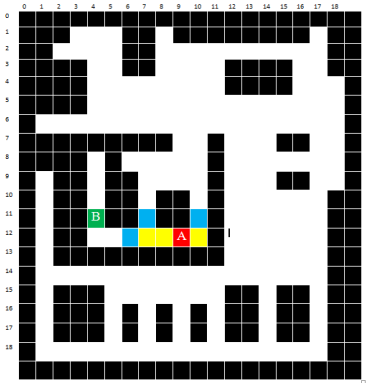
Gambar 4. 8 Langkah 3 pergerakan papan grid

- **Langkah 4**



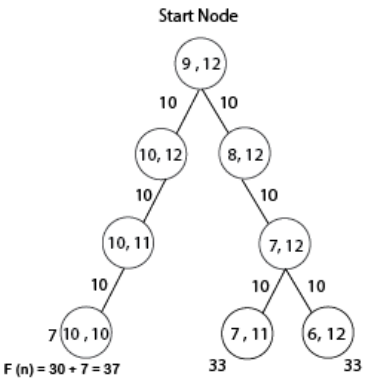
Gambar 4.9 Perhitungan Langkah 4

Dari gambar 4.9 simpan node [9,12] [8,12] [10,12] dan [7,12] kedalam Closed list serta node [10,11] [7,11] dan [6,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.10 :



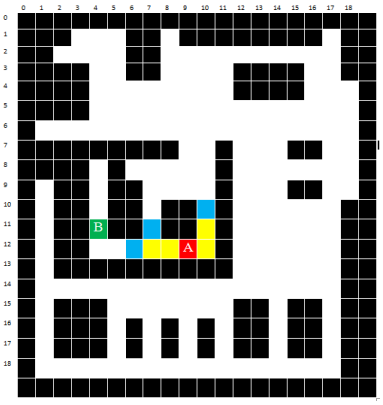
Gambar 4. 10 Langkah 4 pergerakan papan grid

- **Langkah 5**



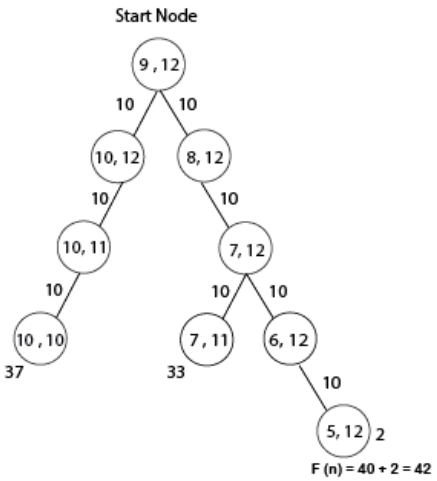
Gambar 4.11 Perhitungan Langkah 5

Pada gambar 4.11 simpan node [9,12] [8,12] [10,12] [7,12] dan [10,11] kedalam Closed list serta node [10,10] [7,11] dan [6,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.12 :



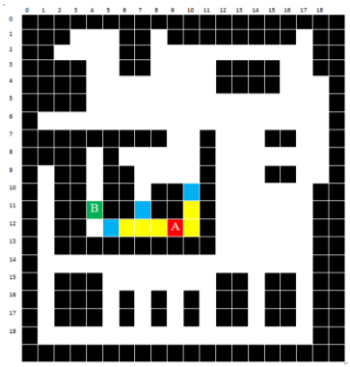
Gambar 4. 12 Langkah 5 pergerakan papan grid

- **Langkah 6**



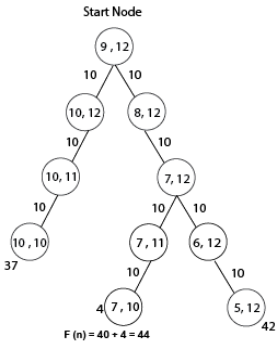
Gambar 4.13 Perhitungan Langkah 6

Dari gambar 4.13 simpan node [9,12] [8,12] [10,12] [7,12] [10,11] dan [6,12] kedalam Closed list serta node [10,10] [7,11] dan [5,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.14 :



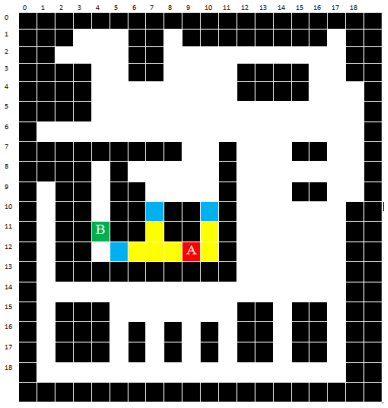
Gambar 4. 14 Langkah 6 pergerakan papan grid

- **Langkah 7**



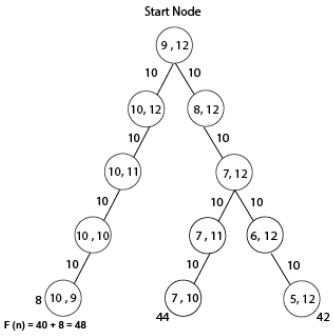
Gambar 4.15 Perhitungan Langkah 7

Dari gambar 4.15 simpan node [9,12] [8,12] [10,12] [7,12] [10,11] [7,11] dan [6,12] kedalam Closed list serta node [10,10] [7,10] dan [5,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.16 :



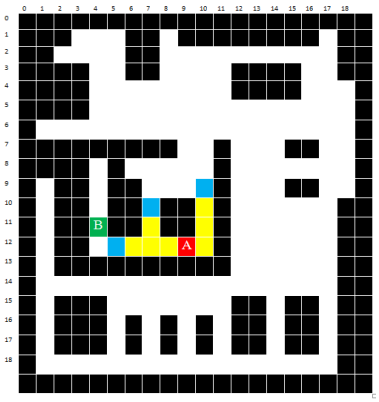
Gambar 4. 16 Langkah 7 pergerakan papan grid

- **Langkah 8**



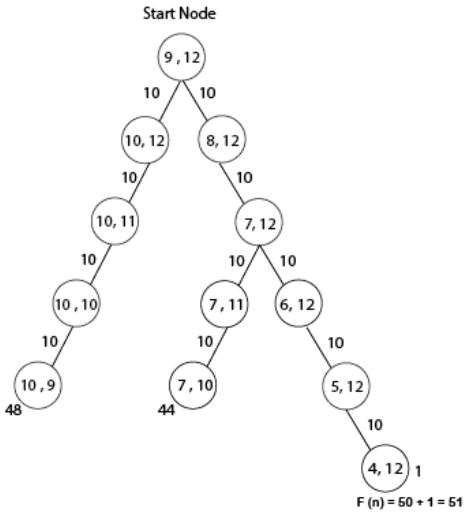
Gambar 4.17 Perhitungan Langkah 8

Simpan node [9,12] [8,12] [10,12] [7,12] [10,11] [6,12] [7,11] dan [10,10] kedalam Closed list serta node [10,9] [7,10] dan [5,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.18 :



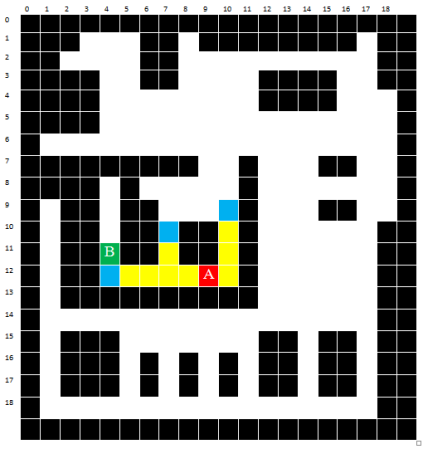
Gambar 4. 18 Langkah 8 pergerakan papan grid

- **Langkah 9**



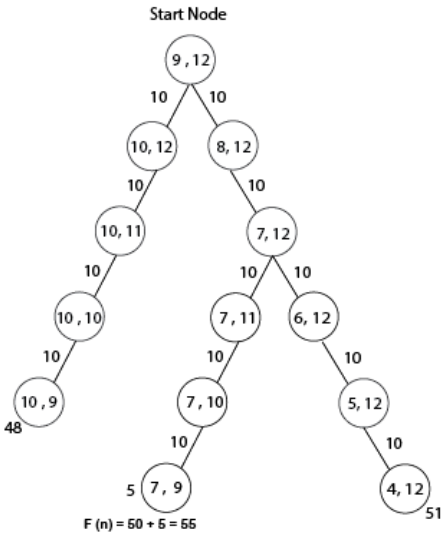
Gambar 4.19 Perhitungan Langkah 9

Simpan node [9,12] [8,12] [10,12] [7,12] [10,11] [6,12] [7,11] [10,10] dan [5,12] kedalam Closed list serta node [10,9] [7,10] dan [4,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.20 :



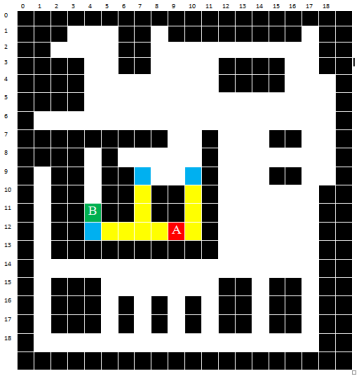
Gambar 4. 20 Langkah 9 pergerakan papan grid

- **Langkah 10**



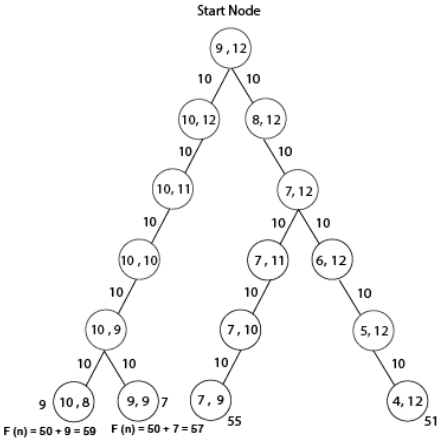
Gambar 4.21 Perhitungan Langkah 10

Simpan node [9,12] [8,12] [10,12] [7,12] [10,11] [6,12] [7,11] [10,10] [5,12] dan [7,10] kedalam Closed list serta node [10,9] [7,9] dan [4,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.22 :



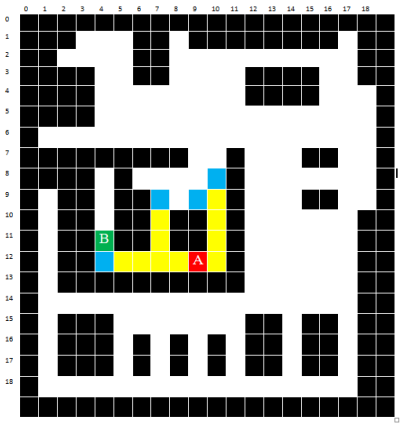
Gambar 4. 22 Langkah 10 pergerakan papan grid

- **Langkah 11**



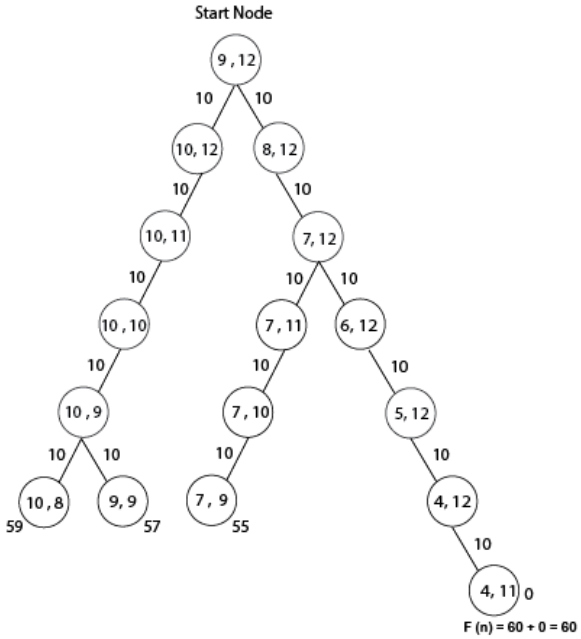
Gambar 4.23 Perhitungan Langkah 11

Simpan node [9,12] [8,12] [10,12] [7,12] [10,11] [6,12] [7,11] [10,10] [5,12] [7,10] dan [10,9] kedalam Closed list serta node [10,8] [9,9] [7,9] dan [4,12] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.24 :



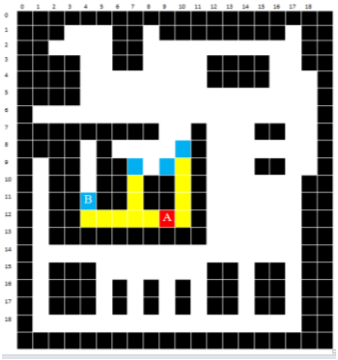
Gambar 4. 24 Langkah 11 pergerakan papan grid

- **Langkah 12**



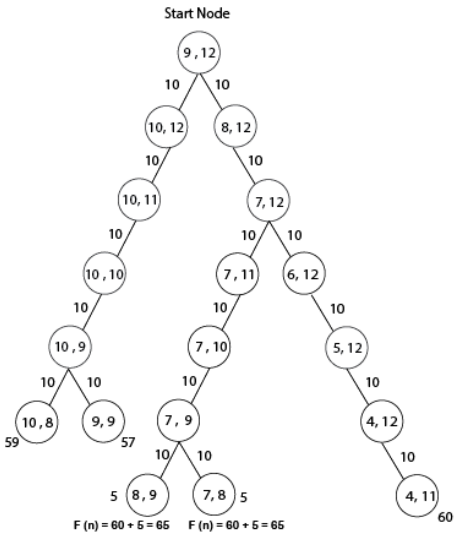
Gambar 4.25 Perhitungan Langkah 12

Simpan node [9,12] [8,12] [10,12] [7,12] [10,11] [6,12] [7,11] [10,10] [5,12] [7,10] [10,9] dan [4,11] kedalam Closed list serta node [10,8] [9,9] [7,9] dan [4,11] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.25 :



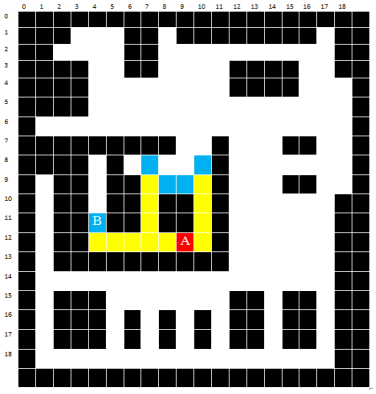
Gambar 4. 26 Langkah 12 pergerakan papan grid

- **Langkah 13**



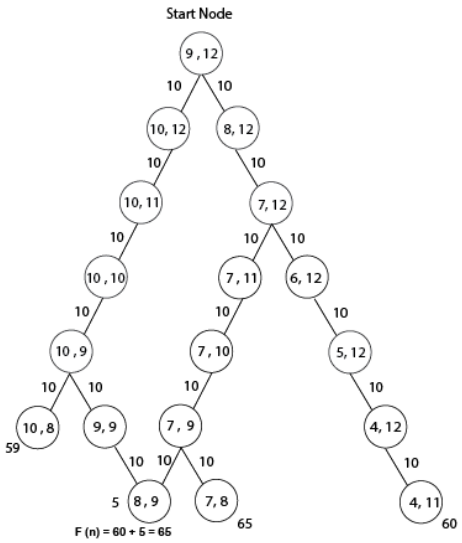
Gambar 4.27 Perhitungan Langkah 13

Simpan node [9,12] [8,12] [10,12] [7,12] [10,11] [6,12] [7,11] [10,10] [5,12] [7,10] [10,9] [4,11] dan [7,9] kedalam Closed list serta node [10,8] [9,9] [8,9] [7,8] dan [4,11] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.28 :



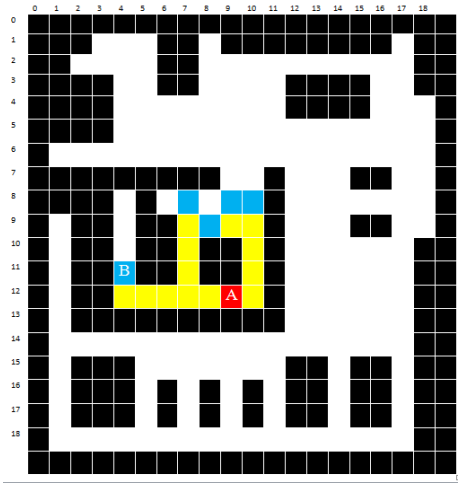
Gambar 4. 28 Langkah 13 pergerakan papan grid

- **Langkah 14**



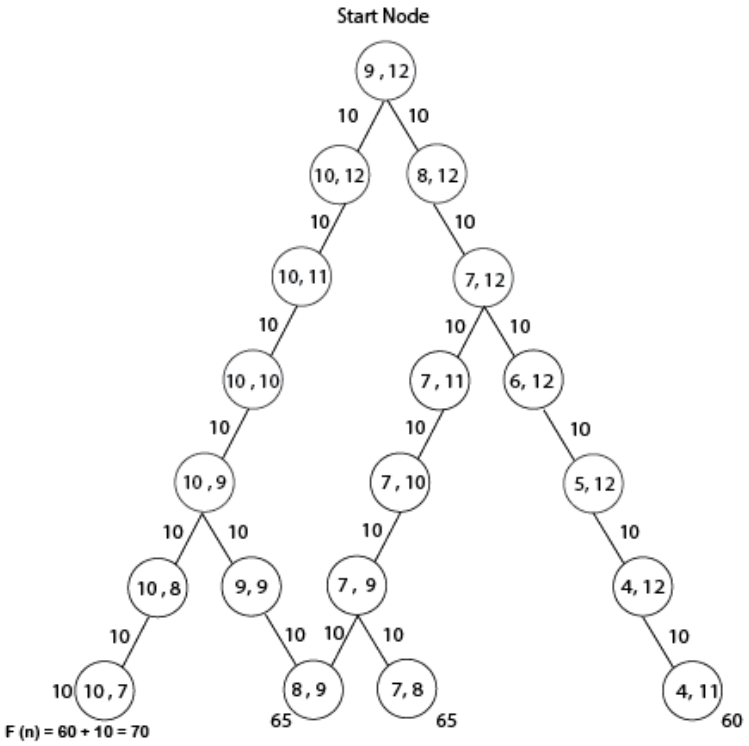
Gambar 4.29 Perhitungan Langkah 14

Simpan node [9,12] [8,12] [10,12] [7,12] [10,11] [6,12] [7,11] [10,10] [5,12] [7,10] [10,9] [4,11] [7,9] dan [9,9] kedalam Closed list serta node [10,8] [8,9] [7,8] dan [4,11] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.30 :



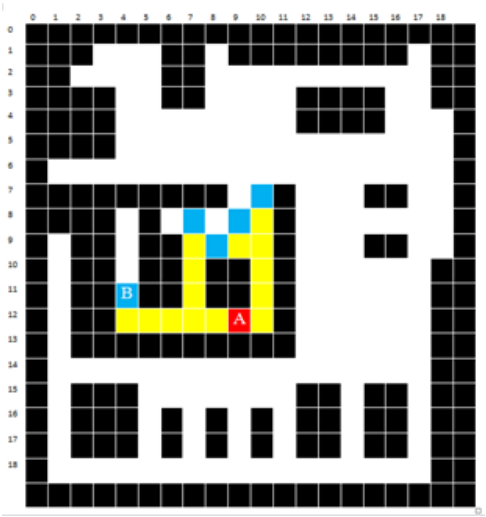
Gambar 4. 30 Langkah 14 pergerakan papan grid

- **Langkah 15**



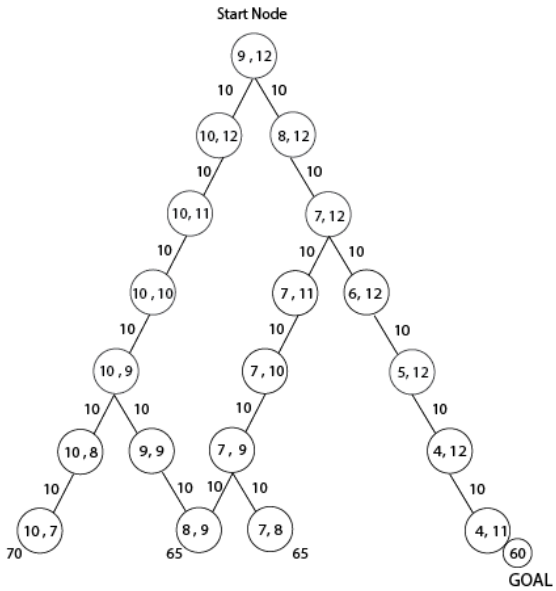
Gambar 4.31 Perhitungan Langkah 15

Simpan node [9,12] [8,12] [10,12] [7,12] [10,11] [6,12] [7,11] [10,10] [5,12] [7,10] [10,9] [4,11] [7,9] [9,9] dan [10,7] kedalam Closed list serta node [10,7] [8,9] [7,8] dan [4,11] ke dalam Open list. Maka pergerakan dalam papan grid seperti gambar 4.32 :



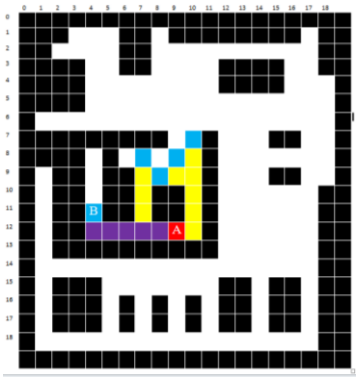
Gambar 4. 32 Langkah 15 pergerakan papan grid

Langkah selanjutnya mencari f(n) yang memiliki nilai paling kecil akan diperiksa, dan didapati [4,11] adalah tujuann sehingga pencarian dihentikan. Kemudian dilakukan backtracking sesuai parent yang telah disimpan didalam closed.



Gambar 4.33 Perhitungan Langkah 16

Dari gambar 4.33 didapatkan hasil pencarian Algoritma A* pathfiding didapat hasil jalur terpendeknya dimulai dari node [9,12] [8,12] [7,12] [6,12] [5,12] [4,11] dengan jumlah biaya 60 yang merupakan biaya paling kecil seperti gambar 4.34 merupakan hasil yang diperoleh pada papan grid ditandai dengan kotak warna ungu.



Gambar 4. 34 Backtraking papan grid