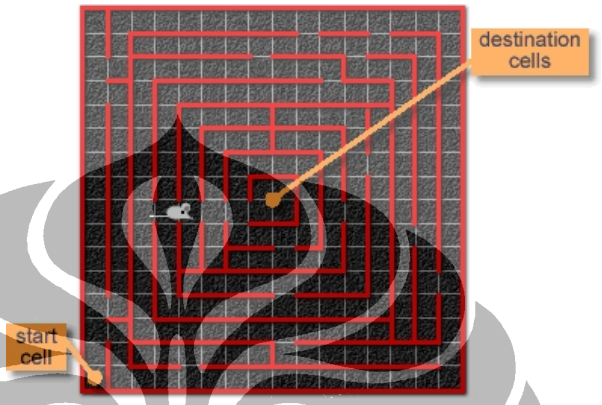
BAB III
PERANCANGAN
KECERDASAN-BUATAN ROBOT PENCARI JALUR

Kecerdasan-buatan yang dirancang untuk robot pencari jalur ini ditujukan pada lingkungan labirin (maze) dua dimensi seperti ditunjukkan oleh Gambar 3.1.



Gambar 3.1. Lingkungan Robot Pencari Jalur

Berdasarkan Gambar 3.1, lingkungan labirin tersusun atas matriks *n x n cell*, di mana *n* merupakan jumlah *cell* penyusunnya. Penelitian ini mencoba merancang dan mengimplementasikan kecerdasan-buatan untuk robot pencari jalur pada lingkungan labirin yang lebih kecil dengan ukuran matriks 5 x 5 (25 *cell* penyusun) seperti ditunjukkan pada Gambar 3.2

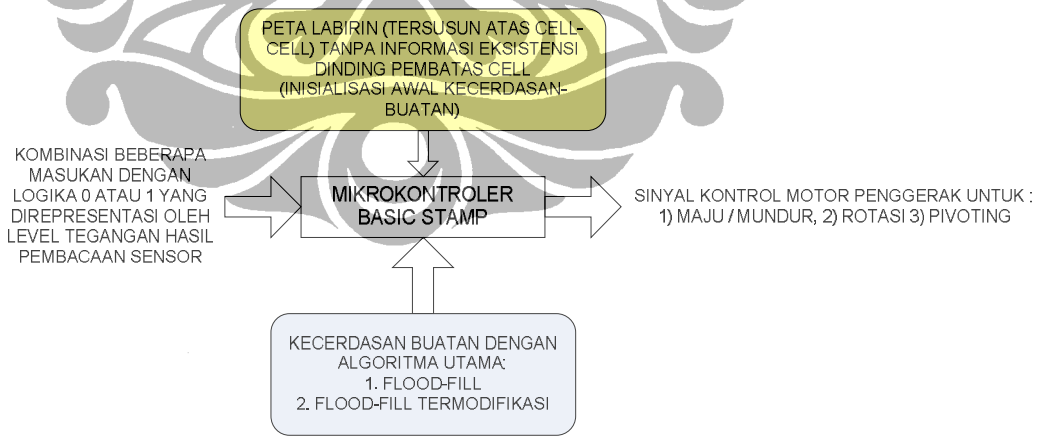
Gambar 3.2. Lingkungan Labirin 25 *Cell*

Perancangan meliputi aspek perangkat keras dan perangkat lunak di mana blok diagram secara keseluruhan dapat dilihat pada Gambar 3.3. Berdasarkan gambar tersebut, perancangan penelitian ini berorientasi pada tiga aspek besar, yaitu masukan, proses, dan keluaran.

Diagram blok secara keseluruhan menerima informasi eksistensi dinding pembatas *cell* di mana robot pencari jalur berada. Pendeteksian dilakukan oleh sensor infra merah di dalam blok masukan, menghasilkan keluaran berupa level tegangan yang merepresentasikan kondisi masukan. Level tegangan ini kemudian diproses oleh blok proses, di mana kecerdasan-buatan ini ditanamkan. Kecerdasan-buatan memiliki peta labirin, yang tersusun atas sejumlah *cell* tanpa informasi dinding pembatas *cell*. Kecerdasan-buatan dalam penelitian ini dibangun dengan dua algoritma utama (Gambar 3.3c), yaitu algoritma *Flood-Fill* algoritma dan *Modified Flood-Fill*. Hasil dari pemrosesan di blok proses bagi blok keluaran adalah berupa sinyal kontrol motor penggerak untuk manuver robot pencari jalur di dalam labirin. Sinyal kontrol ini akan memberikan kecepatan dan arah bagi perputaran roda penggerak robot pencari jalur.

Gambar 3.3a. Diagram Blok Sistem

Gambar 3.3b. Diagram Blok Masukan



Gambar 3.3c. Diagram Blok Proses

Gambar 3.3d. Diagram Blok Keluaran

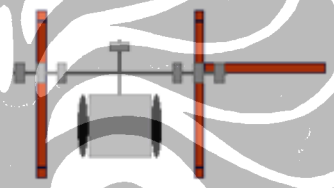
3.1 PERANGKAT KERAS

Kecerdasan-buatan akan digunakan pada rancangan perangkat keras, yang meliputi tiga aspek, yaitu perancangan subsistem masukan, proses, dan keluaran.

3.1.1 Perancangan Subsistem Masukan

Robot pencari jalur menggunakan sensor *near infrared*, yaitu sensor *proximity* untuk mendeteksi keberadaan dinding labirin. Dinding pada labirin dua dimensi ini direpresentasi menggunakan garis hitam.

Sensor *proximity* biasanya dipasang pada "sayap" robot dan menghadap ke bawah dari atas dinding labirin dua dimensi. Jika terdapat dinding di bawah, sensor akan mengembalikan nilai logika benar *(true)*. Jika tidak terdapat dinding di bawah, sensor akan mengembalikan nilai logika salah *(false)*.

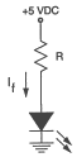


Gambar 3.4. Robot dengan Sensor ***Proximity***

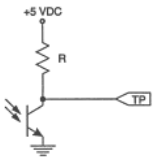
Paling tidak terdapat satu sensor di depan dan tiga sensor masing-masing di sisi kiri dan kanan robot. Sensor depan digunakan untuk mendeteksi keberadaan dinding di depan robot. Sensor sisi digunakan untuk mendeteksi keberadaan dinding samping serta untuk mengkoreksi posisi pergerakan robot. Berdasarkan gambar di atas sebagai contoh, jika sayap kiri mendapatkan pembacaan nilai 010 mengindikasikan robot berada dalam posisi di tengah-tengah *cell*. Jika terbaca 100, mengindikasikan robot terlalu jauh ke sebelah kanan dan perlu dikoreksi ke sebelah kiri. Jika terbaca 001, mengindikasikan robot terlalu jauh ke sebalah kiri dan perlu dikoreksi ke sebelah kanan.

Jenis sensor ini mempunyai keuntungan lain. Jika sayap kanan dibaca dari gambar di atas, akan memberikan nilai 011, mengindikasikan terdapat dinding yang tegak lurus terhadap dinding sebelah kanan. Pemetaan dinding pada *cell* yang bersebelahan memberikan waktu eksplorasi yang lebih singkat.

Implementasi sensor *proximity* bisa dilihat pada Gambar 3.5.



Gambar 3.5a. IR LED Sensor *Proximity*



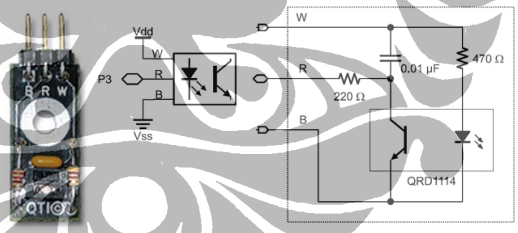
Gambar 3.5b. Phototransistor Sensor *Proximity*

Gambar 3.5c. Sensor *Proximity* (Satu Kemasan)

Resistor pada bagian *emitter* (Gambar 3.5a) dipilih untuk menghasilkan arus sehingga IR LED bisa memancarkan sinar infra merah. Jika tidak dihubungkan secara langsung ke tegangan konstan +5 V, cara lain dapat digunakan, yaitu dengan menghubungkannya melalui transistor yang dapat diaktifkan oleh mikrokontroler. Dengan cara tersebut, LED infra merah dapat dinyalakan oleh mikrokontroler hanya pada saat sensor perlu untuk dibaca. Phototransistor pada bagian sensor (Gambar 3.5b) bertindak sebagai saklar pada saat saturasi. LED infra merah dan sensor dapat berupa komponen yang terpisah, atau berada dalam satu kemasan plastik (Gambar 3.5c).

Penelitian ini menggunakan sensor *proximity* berbasis komponen sensor opto pantul *(reflective object sensor)* Fairchild Semiconductor QRD1114 [14].

Sensor *proximity* berbasis QRD1114 ditunjukkan oleh Gambar 3.6.



Gambar 3.6. Sensor *Proximity* Berbasis QRD1114 [8]

Kotak hitam kecil di atas label QTI merupakan sensor opto pantul. Sensor ini mempunyai sebuah diode infra merah di balik layar bening dan sebuah transistor infra merah di balik jendela hitam. Ketika infra merah dipancarkan oleh diode, memantul pada sebuah permukaan dan kembali ke jendela hitam, infra merah tersebut mengenai basis transistor, mengakibatkan transistor menghantarkan arus. Semakin banyak infra merah mengenai basis transistor, semakin banyak arus yang dapat dihantarkan [13].

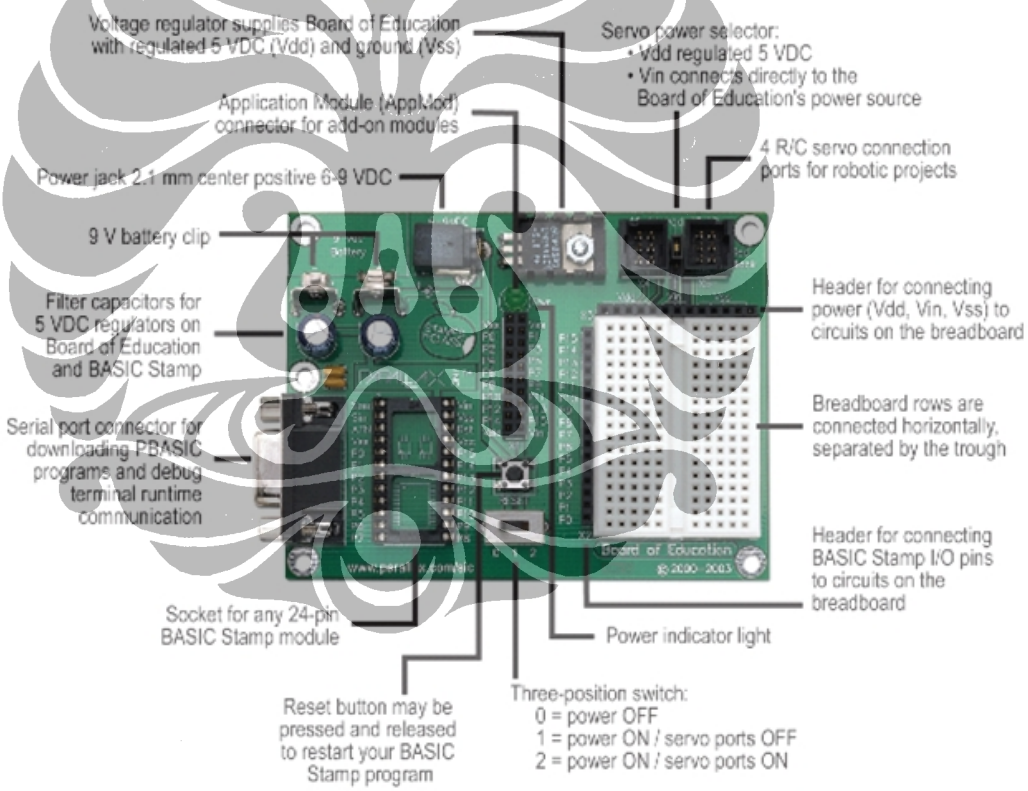
3.1.2 Perancangan Subsistem Proses

Pemrosesan dilakukan oleh mikrokontroler, yang melakukan pembacaan masukan (sensor), mengontrol keluaran (pergerakan motor), dan mencari jalur di lingkungan labirin.

Beberapa hal yang harus diperhatikan dalam pemilihan mikrokontroler, yaitu jumlah memori dan port I/O yang dimiliki.

Jumlah memori tertentu diperlukan dalam pemrosesan algoritma kecerdasan-buatan sedangkan jumlah port I/O diperlukan oleh masukan (sensor, saklar, tombol) dan keluaran (motor, LCD, LED, speaker).

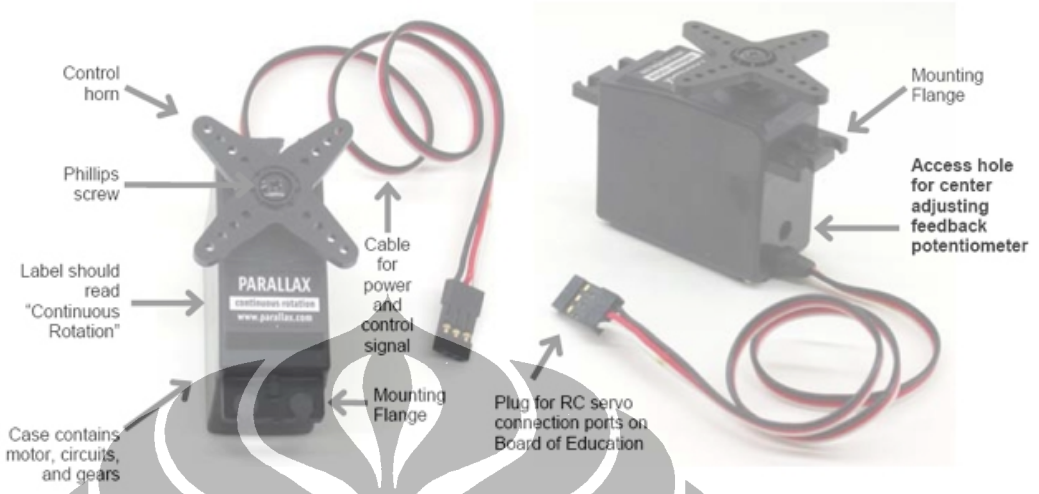
Penelitian ini direncanakan menggunakan *Board of Education* dari Parallax sebagai *carrier board* dari prosesor BASIC Stamp.



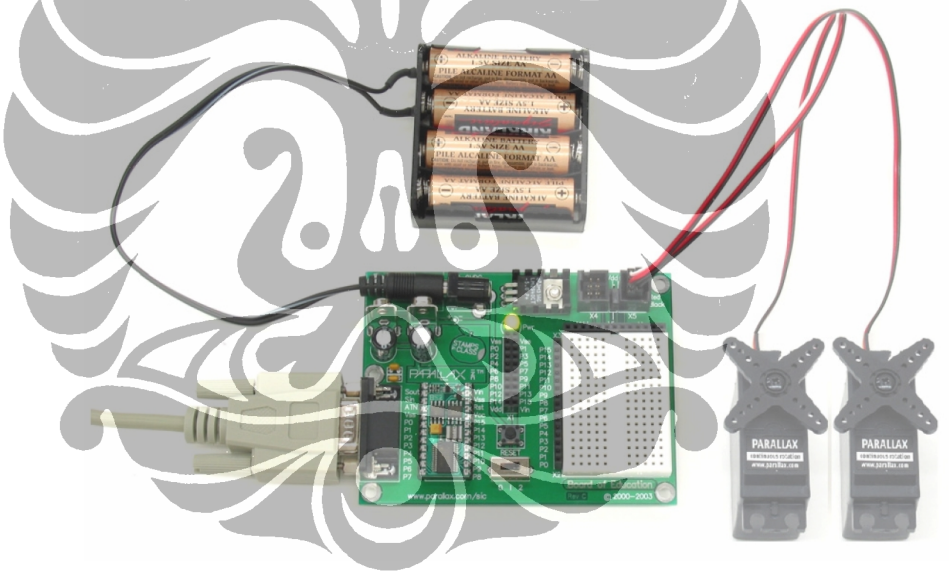
Gambar 3.7. *Board of Education* (BOE) Rev. C *Carrier Board* untuk Modul Mikrokontroler BASIC Stamp [11]

3.1.3 Perancangan Subsistem Keluaran

Motor penggerak robot dalam penelitian ini menggunakan jenis *continuous rotation servo*.

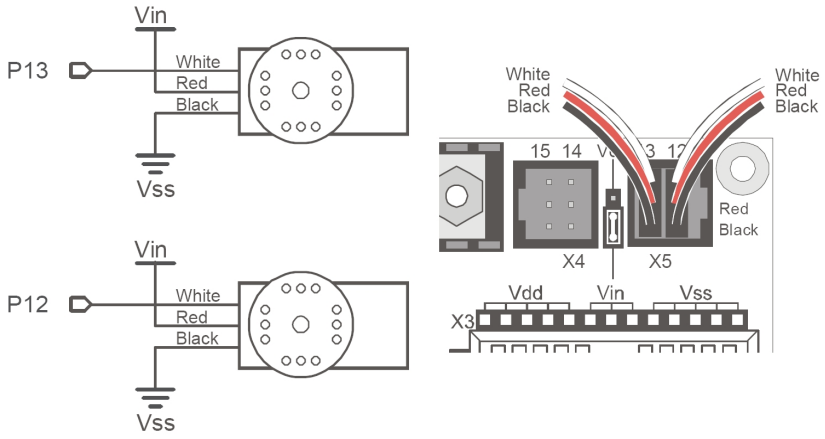


Gambar 3.8. *Continuous Rotation Servo* [8]



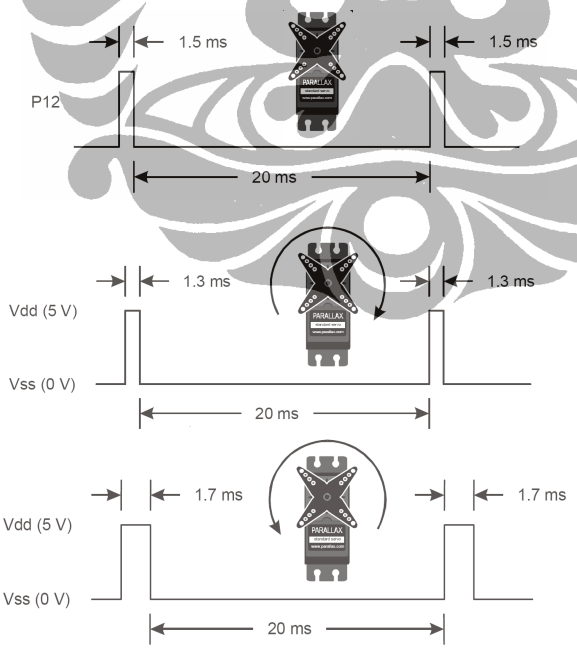
Gambar 3.9. Koneksi *Board of Education* dan *Servo* [8]

Servo mempunyai tiga koneksi, yaitu +5 V, *ground*, dan *signal*. *Signal* merupakan pulsa dengan panjang beberapa mili detik. Dengan mengontrol lebar pulsa ke *servo*, mikrokontroler akan dapat menentukan apakah perputaran bergerak maju *(forward)* atau mundur *(backward)*, juga dapat menentukan seberapa cepat perputaran tersebut.



Gambar 3.10. *Pinout Servo* (+5 V, ground, dan signal) [8]

Gambar 3.11 memperlihatkan kode program pengontrol *servo*. Pada intinya, untuk membuat *servo* tidak berputar, *pin* keluaran ke *servo* dari mikrokontroler BASIC Stamp diberi pulsa dengan lebar 1,5 mili detik (gambar bagian atas). Untuk membuat *servo* berputar searah jarum jam dengan kecepatan maksimum, *pin* keluaran diberi pulsa dengan lebar 1,3 mili detik. Untuk membuat *servo* berputar berlawanan arah jarum jam dengan kecepatan maksimum, *pin* keluaran diberi pulsa dengan lebar 1,7 mili detik.




```
DO
  PULSOUT 12, 750
  PAUSE 20
LOOP
```

```
DO
  PULSOUT 12, 650
  PAUSE 20
LOOP
```

```
DO
  PULSOUT 12, 850
  PAUSE 20
LOOP
```

Gambar 3.11. Kode Program Pengontrol *Servo* [8]

3.2 PERANGKAT LUNAK

Kecerdasan-buatan yang dirancang mengacu kepada dua algoritma utama, yaitu algoritma *Flood-Fill* dan algoritma *Modified Flood-Fill*.

3.2.1 Perancangan dengan Algoritma *Flood-Fill*

Algoritma ini melibatkan pemberian nilai pada masing-masing *cell* penyusun labirin di mana nilai ini merepresentasikan jarak dari sebarang *cell* ke *cell* tujuan. Selanjutnya *cell* tujuan diberikan nilai 0. Jika robot pencari jalur berada di sebuah *cell* dengan nilai 1, robot tersebut 1 *cell* jauhnya dari tujuan. Jika robot berada pada *cell* dengan nilai 3, robot tersebut 3 *cell* jauhnya dari tujuan.

Diasumsi robot tidak melakukan pergerakan secara diagonal, maka nilai untuk labirin dengan ukuran matriks 5 x 5 tanpa dinding diperlihatkan oleh Gambar 3.12.

Gambar 3.12. Pergerakan Awal Robot dalam Algoritma *Flood-Fill*

Untuk labirin dengan ukuran 16 baris kali 16 kolom sama dengan 256 nilai *cell*, diperlukan memori sebanyak 256 *byte* untuk menyimpan nilai jarak.

Ketika pergerakan dimulai, robot harus memeriksa semua *cell* bersebelahan yang tidak dipisahkan oleh dinding dan memilih satu dengan nilai jarak terendah. Pada contoh di atas, robot akan mengabaikan sisi kiri (Barat) sebab ada dinding, dan selanjutnya akan mengecek nilai jarak pada *cell* di bagian atas (Utara), *cell* di sisi kanan (Timur), dan *cell* di bagian bawah (Selatan) karena *cell-cell* tersebut tidak dipisahkan oleh dinding. *Cell* ke arah Utara bernilai 2, *cell* ke arah Timur bernilai 2, dan *cell* ke arah Selatan bernilai 4. Rutin pada kecerdasan-buatan akan

men-sorting nilai-nilai tersebut untuk menentukan *cell* mana yang mempunyai nilai terendah. *Cell* ke arah Utara dan Timur mempunyai nilai jarak 2. Itu berarti robot dapat pergi ke arah Utara atau Timur dan melewati jumlah *cell* yang sama dalam perjalanan ke *cell* tujuan. Karena berputar akan menghabiskan waktu lebih lama, robot akan memilih lurus ke arah *cell* Utara. Jadi proses pengambilan keputusan akan seperti Gambar 3.13.

Menentukan *cell* bersebelahan dengan nilai jarak terendah

Apakah *cell* Utara dibatasi dinding ?
Ya **à** Abaikan *cell* Utara
Tidak **à** *Push cell* Utara ke *stack* untuk diperiksa

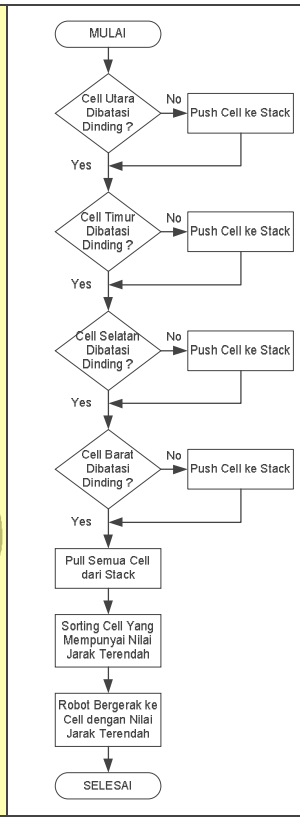
Apakah *cell* di Timur dibatasi dinding ?
Ya **à** Abaikan *cell* Timur
Tidak **à** *Push cell* Timur ke *stack* untuk diperiksa

Apakah *cell* Selatan dibatasi dinding ?
Ya **à** Abaikan *cell* Selatan
Tidak **à** *Push cell* Selatan ke *stack* untuk diperiksa

Apakah *cell* Barat dibatasi dinding ?
Ya **à** Abaikan *cell* Barat
Tidak **à** *Push cell* Barat ke *stack* untuk diperiksa

Pull semua *cell* dari *stack* (*stack* kosong sekarang)
Sorting mencari nilai jarak terendah *cell-cell* tersebut

Robot bergerak ke *cell* bersebelahan dengan nilai jarak terendah



Gambar 3.13. Algoritma dan Diagram Alir Menentukan *Cell* dengan Nilai Jarak Terendah

Dinding labirin mempengaruhi nilai jarak suatu *cell* sehingga informasi dinding *cell* tersebut perlu disimpan. Pada labirin yang tersusun atas 256 *cell* (matriks 16 x 16) perlu dialokasikan lagi memori sebanyak 256 *byte* untuk keperluan penyimpanan informasi tersebut. Terdapat 8 *bit* pada 1 *byte* untuk sebuah *cell*. 4 *bit* pertama dapat merepresentasi informasi keberadaan dinding *cell*, menyisakan 4 *bit* untuk keperluan lain.

Bit No.	7	6	5	4	3	2	1	0
Dinding					B	S	T	U

Gambar 3.14. *Byte* Merepresentasi Status Dinding *Cell*

di mana : B = dinding Barat, S = dinding Selatan, T = dinding Timur, dan U = dinding Utara.

Informasi dinding *cell* dapat dimanfaatkan oleh dua *cell* yang bersebelahan sehingga jika terdapat *updating* informasi dinding untuk satu *cell*, dapat juga dilakukan *updating* informasi dinding untuk *cell* yang bersebelahan. Instruksi untuk *updating* peta dinding dapat seperti Gambar 3.15

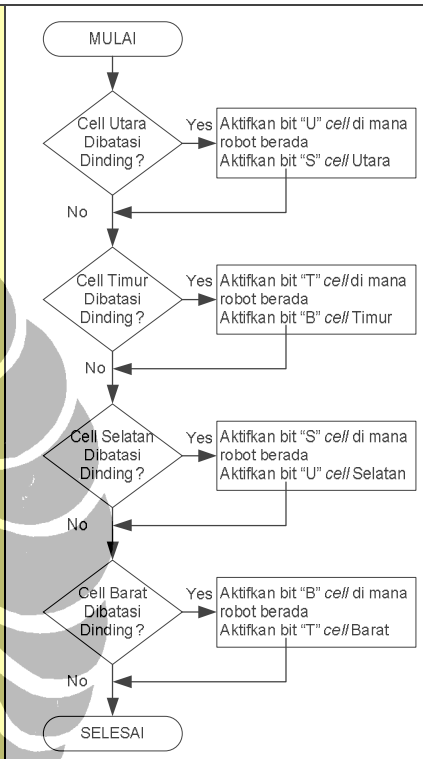
Updating peta dinding

Apakah *cell* Utara dibatasi dinding ?
Ya à Aktifkan *bit* "U" *cell* di mana robot berada
Aktifkan *bit* "S" *cell* Utara
Tidak à tidak ada *update*

Apakah *cell* Timur dibatasi dinding ?
Ya à Aktifkan *bit* "T" *cell* di mana robot berada
Aktifkan *bit* "B" *cell* Timur
Tidak à tidak ada *update*

Apakah *cell* Selatan dibatasi dinding ?
Ya à Aktifkan *bit* "S" *cell* di mana robot berada
Aktifkan *bit* "U" *cell* Selatan
Tidak à tidak ada *update*

Apakah *cell* Barat dibatasi dinding ?
Ya à Aktifkan *bit* "B" *cell* di mana robot berada
Aktifkan *bit* "T" *cell* Barat
Tidak à tidak ada *update*



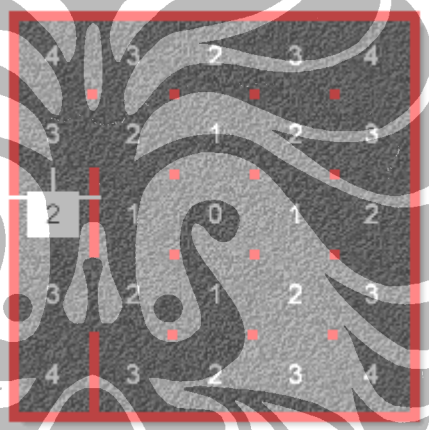
Gambar 3.15. Algoritma dan Diagram Alir *Updating* Peta Dinding *Cell*

Berdasarkan mekanisme pada Gambar 3.15, kecerdasan-buatan yang dibangun mempunyai cara untuk menyimpan informasi dinding *cell*. Jika dinding baru ditemukan, nilai jarak *cell* akan terpengaruh sehingga diperlukan suatu cara untuk melakukan *updating* nilai jarak tersebut.

Berdasarkan Gambar 3.16, seandainya robot menemukan dinding, robot tidak dapat menuju ke Barat (ke kiri) dan ke Timur (ke kanan), hanya dapat bergerak ke Utara (ke atas) atau ke Selatan (ke bawah).

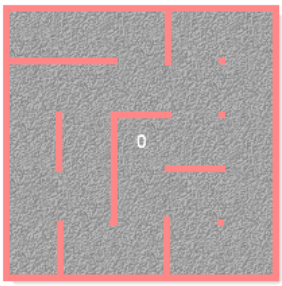
Jika bergerak ke Utara atau ke Selatan berarti menaikkan nilai jarak, sehingga perlu dilakukan *updating* pada nilai *cell* sebagai akibat ditemukannya dinding baru ini. Untuk melakukan ini, kecerdasan-buatan melakukan *"flood"* pada labirin dengan nilai baru.

"Flood" dalam hal ini berarti memberikan nilai jarak lebih besar pada *cell* bersebelahan yang tidak dibatasi dinding dari arah *cell* di mana robot berada *(open neighbour cell)*. Nilai yang diberikan pada *open neighbour cell* lebih besar satu daripada nilai *cell* di mana robot berada.

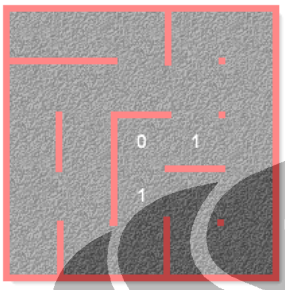


Gambar 3.16. Penemuan Dinding Labirin dalam Algoritma *Flood-Fill*

Sebagai contoh *"flooding"* pada labirin, asumsikan robot telah melakukan pergerakan dan menemukan beberapa dinding. Simulasi *flooding* diperlihatkan pada Gambar 3.17.

1. 

Rutin mulai menginisialisasi array yang menyimpan nilai jarak dan memberikan nilai 0 pada *cell* tujuan.

2. 

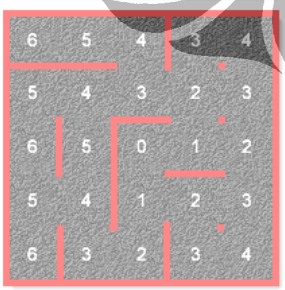
Rutin kemudian mengambil *cell* bersebelahan yang tidak dibatasi dinding dan memberi nilai jarak yang lebih tinggi, yaitu 1.

3. 

Rutin kembali menemukan *cell* bersebelahan yang tidak dibatasi dinding dan memberi nilai jarak yang lebih tinggi, yaitu 2.

4. 

Beberapa iterasi dilakukan

5. 

Hal ini diulang sebanyak yang diperlukan sampai *cell-cell* yang lain mempunyai nilai jarak

Gambar 3.17. Simulasi Algoritma *Flood-Fill*

Terlihat bagaimana nilai-nilai ini mengarahkan robot dari *cell* awal menuju *cell* tujuan melalui jalur terpendek.

Intruksi untuk “flooding” pada labirin dengan nilai jarak diperlihatkan oleh Gambar 3.18.

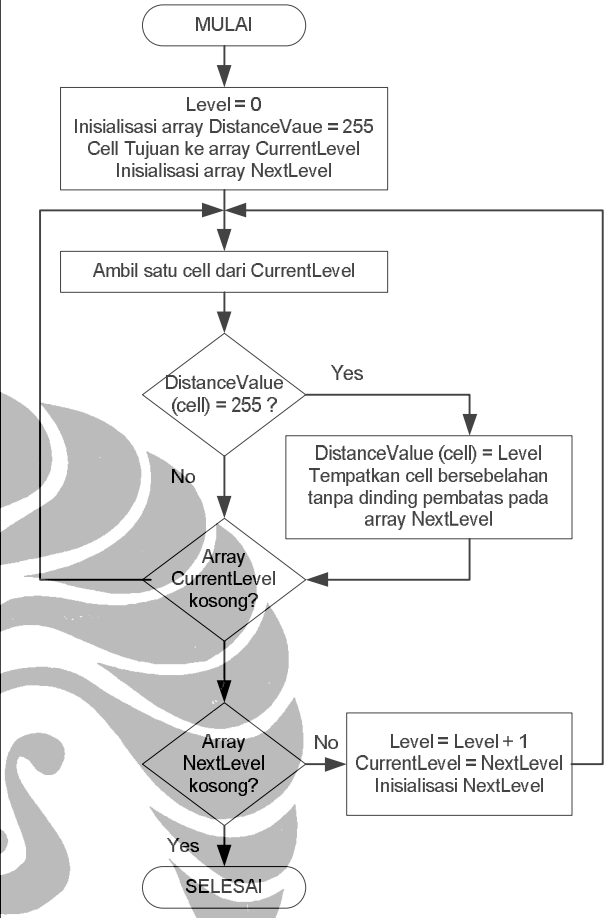
Flood labirin dengan nilai jarak yang baru
Variabel Level = 0

Inisialisasi array DistanceValue
sehingga semuanya bernilai = 255

Tempatkan *cell* tujuan dalam array CurrentLevel

Inisialisasi array kedua NextLevel

Mulai :
Ulangi instruksi berikut sampai
CurrentLevel kosong :
{
Ambil satu *cell* dari CurrentLevel
If DistanceValue(*cell*) = 255 then
DistanceValue(*cell*) = Level, dan
tempatkan *cell* bersebelahan tanpa
dinding pembatas pada NextLevel
End if
}
Array CurrentLevel sekarang kosong
Apakah array NextLevel kosong ?
Tidak **à**
{
Level = Level + 1
CurrentLevel = NextLevel
Inisialisasi NextLevel
Kembali ke ‘Mulai :’
}
Ya **à** Flooding labirin selesai



Gambar 3.18. Algoritma dan Diagram Alir Flood Labirin dengan Nilai Jarak Baru

Algoritma *Flood-Fill* memberikan cara yang baik untuk menemukan jalur terpendek dari *cell* awal menuju *cell* tujuan. Untuk labirin tersusun atas 256 *cell*, diperlukan 512 *byte* RAM untuk mengimplementasikan rutin kecerdasan buatan: satu array 256 *byte* untuk menyimpan informasi nilai jarak dan satu array 256 *byte* untuk menyimpan informasi peta dinding.

Setiap kali robot memasuki sebuah *cell*, kecerdasan buatannya akan melakukan langkah-langkah berikut

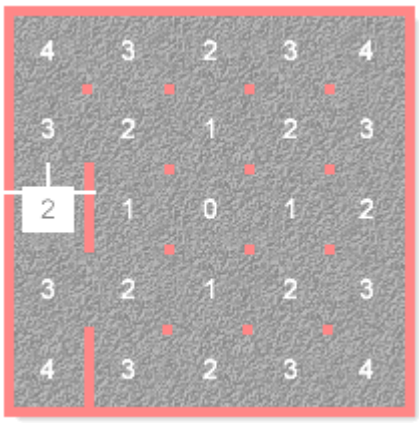
1. *Update* peta dinding
2. Flood labirin dengan nilai jarak yang baru
3. Menentukan *cell* bersebelahan yang mempunyai nilai jarak terendah
4. Bergerak ke *cell* bersebelahan yang mempunyai nilai jarak terendah

3.2.2 Perancangan dengan Algoritma *Modified Flood-Fill*

Algoritma ini sama dengan algoritma *Flood-Fill* regular di mana kecerdasan-buatan menggunakan nilai jarak untuk bergerak di dalam labirin. Nilai jarak, yang merepresentasi seberapa jauh robot dari *cell* tujuan, diikuti secara menurun (descending order) sampai robot mencapai tujuannya.

Gambar 3.19. Pergerakan Awal Robot dalam Algoritma *Modified Flood-Fill*

Ketika robot menemukan dinding baru, nilai jarak perlu di-*update*. Algoritma *Modified Flood-Fill* hanya mengubah nilai-nilai yang perlu diubah. Berbeda dengan algoritma food-fill regular, yang melakukan flooding seluruh labirin dengan suatu nilai. Sebagai contoh, berdasarkan Gambar 3.19, robot bergerak maju satu *cell* dan menemukan sebuah dinding.



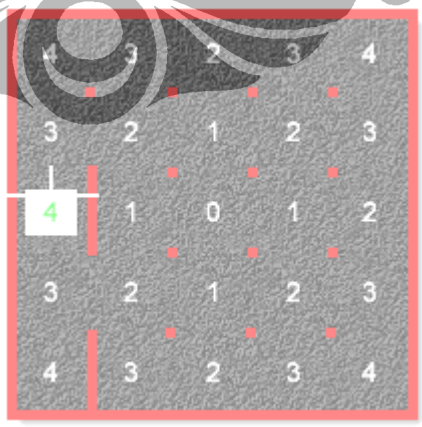
Gambar 3.20. Penemuan Dinding Labirin dalam Algoritma *Modified Flood-Fill*

Robot tidak dapat bergerak ke Barat dan ke Timur, hanya dapat bergerak ke Utara atau ke Selatan. Jika bergerak ke Utara atau ke Selatan berarti menaikkan nilai jarak, sehingga nilai *cell* perlu untuk di-*update*.

Ketika robot menghadapi kasus tersebut, kecerdasan-buatan akan melakukan hal berikut

Jika sebuah *cell* bukan merupakan *cell* tujuan, nilai jaraknya adalah satu plus nilai minimum *cell* bersebelahan tanpa dinding pembatas.

Berdasarkan contoh di atas, nilai minimum *cell* bersebelahan tanpa dinding pembatas adalah 3. Tambahkan 1 pada nilai ini menghasilkan nilai 3 + 1 = 4. *Update* nilai jarak *cell* dalam labirin diperlihatkan pada Gambar 3.21.



Gambar 3.21. *Updating* Nilai Jarak *Cell* dalam Algoritma *Modified Flood-Fill*

Ada waktunya ketika *updating* nilai jarak *cell* menyebabkan *cell* bersebelahan menyalahi aturan "1 + nilai minimum", sehingga hal tersebut harus dicek juga. Berdasarkan contoh di atas *cell* Utara dan Selatan mempunyai nilai minimum 2. Tambahkan 1 pada nilai ini menghasilkan 2 + 1 = 3 sehingga *cell* Utara dan Selatan tidak menyalahi aturan dan *updating* rutin selesai. Selanjutnya, jika nilai-nilai *cell* telah di-*update* semua, robot bisa bergerak mengikuti nilai jarak tersebut secara menurun *(descending order).* Prosedur algoritma *Modified Flood-Fill* dalam melakukan *updating* nilai jarak, diperlihatkan oleh Gambar 3.22 berikut.

Update nilai jarak (jika diperlukan)

Pastikan *stack* kosong

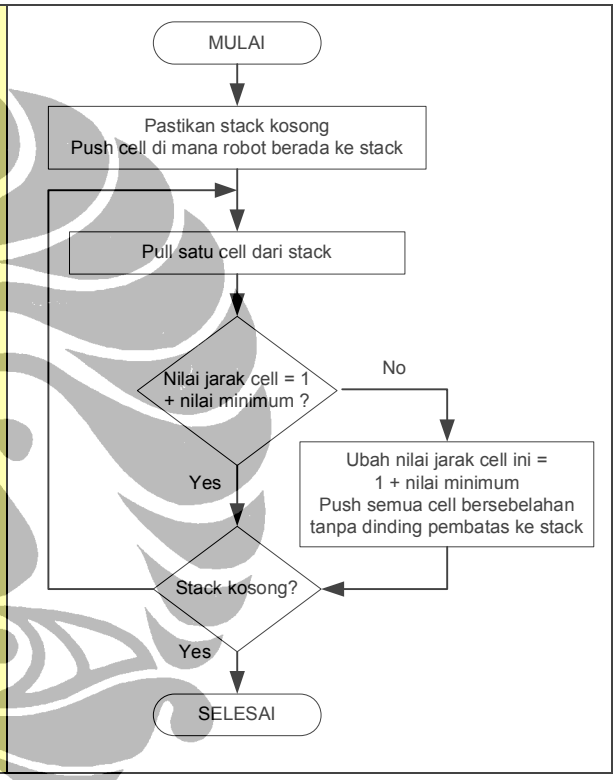
Push cell (di mana robot berada) ke *stack*

Ulangi set instruksi berikut sampai *stack* kosong :

{

Pull satu *cell* dari *stack*
Apakan nilai jarak *cell* ini = 1 + nilai minimum *cell* bersebelahan tanpa dinding pembatas ?
Tidak **à**
Ubah nilai jarak *cell* ini = 1 + nilai minimum *cell* bersebelahan tanpa dinding pembatas
dan
push semua *cell* bersebelahan tanpa dinding pembatas ke *stack* untuk diperiksa
Ya **à** Tidak ada *update*

}

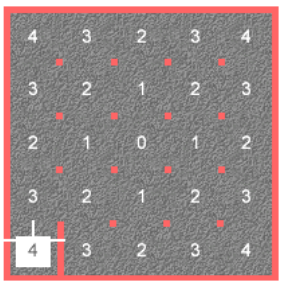


Gambar 3.22. Algoritma dan Diagram Alir *Update* Nilai Jarak (*Modified Flood-Fill*)

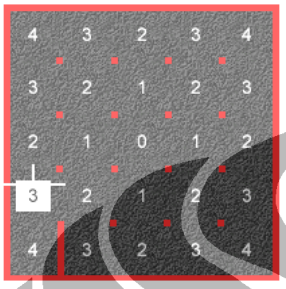
Dengan hanya melakukan *updating* pada nilai-nilai jarak yang perlu di-*update*, robot dapat melakukan pergerakan jauh lebih cepat. Setiap kali robot memasuki satu *cell*, kecerdasan-buatan melakukan langkah-langkah berikut

1. *Update* peta dinding
2. *Update* nilai jarak (hanya jika diperlukan)
3. Menentukan *cell* bersebelahan yang mempunyai nilai jarak terendah
4. Bergerak ke *cell* bersebelahan yang mempunyai nilai jarak terendah

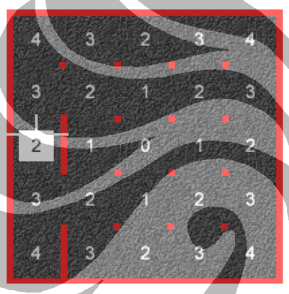
Simulasi Algoritma *Modified Flood-Fill*

1. 

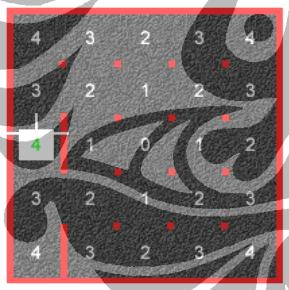
Robot berada di *cell* awal. Nomor mengindikasikan jarak sebarang *cell* ke *cell* tujuan di tengah labirin.

2. 

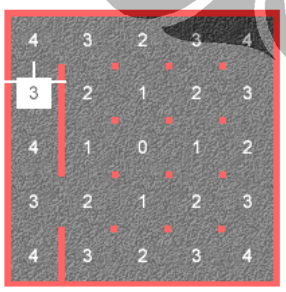
Dalam setiap pergerakan, kecerdasan-buatan mengecek *cell* bersebelahan dan pindah ke *cell* dengan nilai jarak terendah. Dalam kasus ini ada dua *cell* dengan nilai 2 namun berhubung berbelok memerlukan waktu lebih, kecerdasan-buatan menginstruksikan robot bergerak maju.

3. 

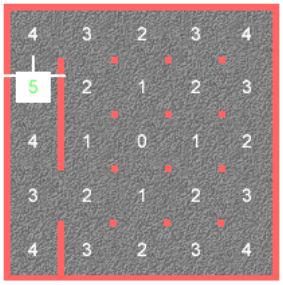
Di *cell* ini, robot menemukan dinding di sisi Timur sehingga robot tidak dapat bergerak ke sisi tersebut. Dua *cell* bersebelahan lain mempunyai nilai jarak lebih tinggi daripada *cell* di mana robot berada sehingga nilai jaraknya diubah menjadi 3 + 1 = 4.

4. 

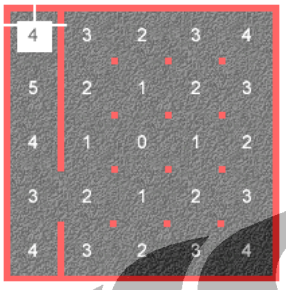
Selanjutnya, robot bisa melakukan pergerakan ke *cell* dengan nilai jarak terendah. Karena dua *cell* bersebelahan mempunyai nilai jarak 3, robot memilih melakukan pergerakan maju.

5. 

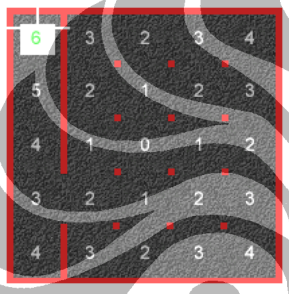
Menemukan dinding lain dan *update* dilakukan pada *cell* ini.

6. 

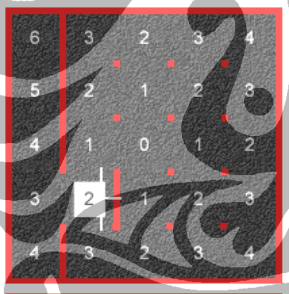
Update telah dilakukan.

7. 

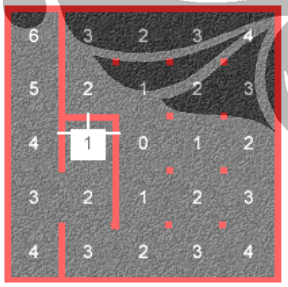
Jalan buntu. *Update* nilai jarak dilakukan dan robot melakukan putaran 180 derajat. Selanjutnya, pergerakan hanya masalah mengikuti nilai jarak *cell* secara menurun (descending order)

8. 

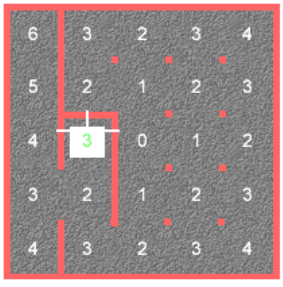
Update telah dilakukan

9. 

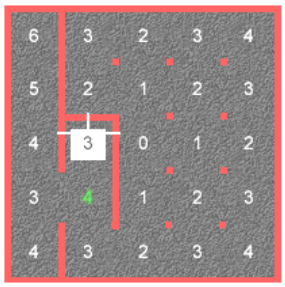
Menemukan dinding lain. Karena *cell* di sebelah Utara mempunyai nilai jarak terendah, robot melakukan putaran 90 derajat ke arah tersebut.

10. 

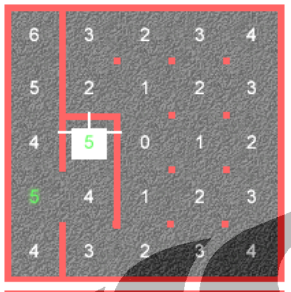
Jalan buntu lagi. *Update* nilai jarak *cell* ini.

11. 

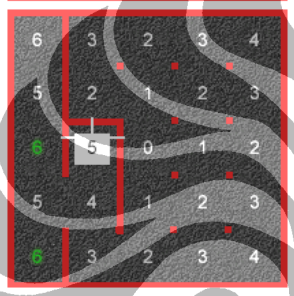
Tetapi ini menyebabkan nilai jarak *cell* di sebelah Selatan menjadi salah. *Update* akan dilakukan.

12. 

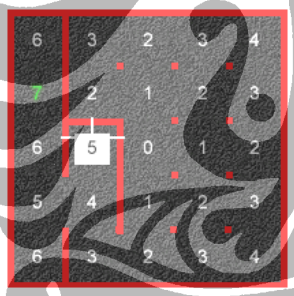
Beberapa *update* lagi untuk nilai jarak *cell*...

13. 

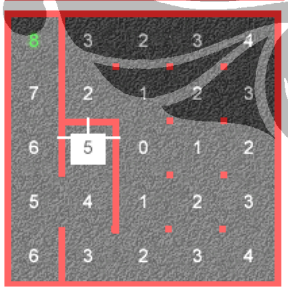
Beberapa *update* lagi untuk nilai jarak *cell*...

14. 

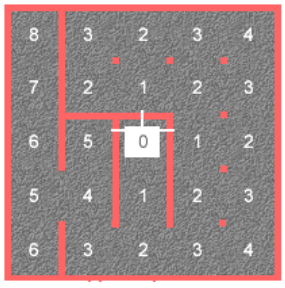
Beberapa *update* lagi untuk nilai jarak *cell*...

15. 

Beberapa *update* lagi untuk nilai jarak *cell*...

16. 

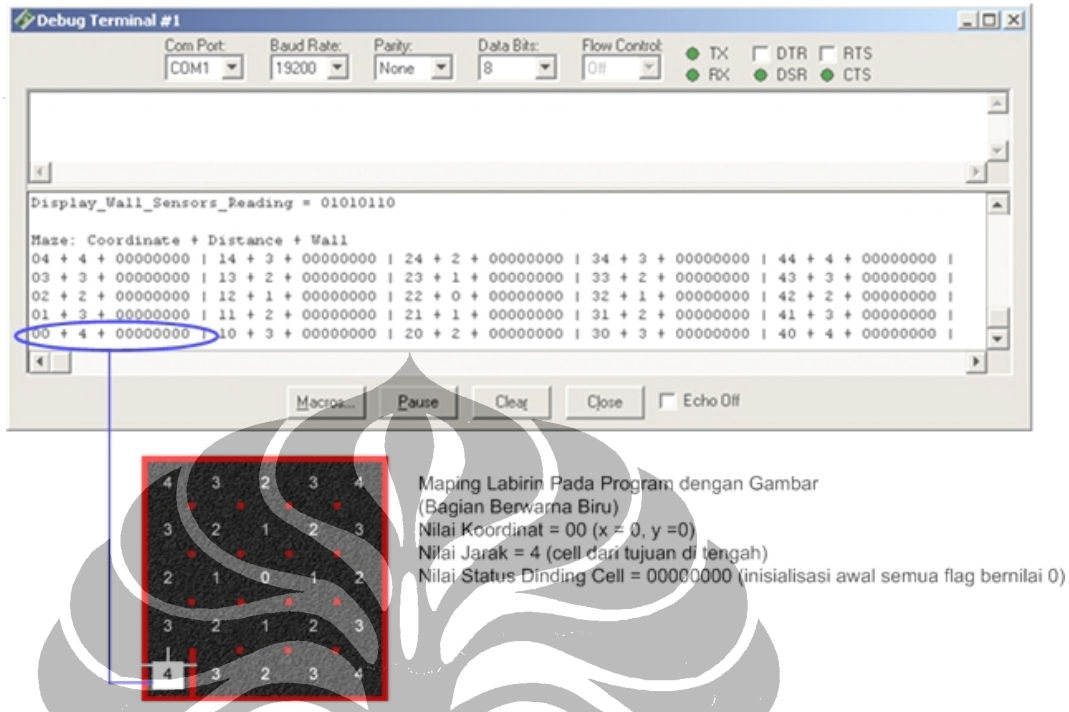
Update selesai. Robot melakukan putaran 180 derajat dan mengikuti nilai jarak secara menurun.

17. 

Terlihat bahwa bagian sebelah kanan labirin belum tereksplorasi. Hal tersebut tidak menjadi masalah karena algoritma *Modified Flood-Fill* telah memberikan jalur terpendek dari *cell* awal menuju *cell* tujuan.

Gambar 3.23. Simulasi Algoritma *Modified Flood-Fill*

Sebagian implementasi algoritma bisa dilihat pada Gambar 3.24.



Gambar 3.24. Inisialisasi Awal Peta Labirin Pada Program PBASIC

Inisialisasi awal peta labirin telah dilakukan program yang kemudian di-debug memberikan visualisasi labirin dengan *cell-cell* penyusun yang mempunyai nilai koordinat, nilai jarak, dan nilai *byte* merepresentasi keberadaan dinding *cell*.

Sebagai contoh, *cell* kiri bawah dari peta labirin diberikan nilai koordinat = (0,0), nilai jarak dari *cell* tujuan di tengah = 4, dan nilai *byte* masih bernilai *reset* = 00000000.