Penerapan Algoritma A* untuk Penyelesaian Permainan Sokoban

Ayu Rifanny Margareth - 13517038
Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13517038@std.stei.itb.ac.id

***Abstrak*—Algoritma A* (baca: *ei-star*) merupakan sebuah algoritma pencarian rute terpendek yang bertipe *informed search*. Algoritma ini merupakan salah satu algoritma yang efisien dalam mencari rute terpendek. Algoritma ini dapat diimplementasikan untuk memecahkan berbagai permasalahan. Salah satunya adalah untuk membentuk penyelesaian terhadap permainan Sokoban. Makalah ini akan membahas tentang proses penyelesaian permainan Sokoban dengan menggunakan algoritma A*, dimulai dari penentuan fungsi heuristik, menghindari terjadinya *deadlock* serta contoh penyelesaian dengan algoritma A*.**

Keywords—Sokoban; deadlock; A*; heuristik;

I. PENDAHULUAN

Permainan dengan kategori puzzle sudah menjadi sebuah kategori permainan yang bersifat mengasah otak. Permainan dengan kategori ini menuntut pemainnya untuk mencari cara yang seoptimal mungkin untuk bisa menyelesaikan permainan. Permainan puzzle juga kerap diikuti oleh beberapa tingkat kesulitan. Permainan biasanya awali dengan tingkat kesulitan yang rendah sehingga sebagian besar pemainnya dapat menyelesaikan permainan dalam waktu yang singkat. Perlahan-lahan, tingkatan permainan akan semakin sulit dan pemain akan membutuhkan waktu yang lebih lama untuk memikirkan penyelesaian permainan. Tentu saja apabila seorang pemain dapat menyelesaikan permainan tersebut, akan ada lagi tingkatan permainan yang lebih tinggi untuk diselesaikan. Hal ini yang membuat permainan puzzle menjadi menarik. Pemain akan terus merasa tertantang dengan tingkatan permainan yang semakin sulit dan pemain juga akan berusaha menyelesaikan permainan tersebut untuk membuktikan bahwa ia pemain terbaik ataupun hanya untuk sekedar menghabiskan waktu saja.

Namun, akan ada suatu masa dimana sebuah permainan sudah mencapai tingkatan yang sangat sulit sehingga manusia akan membutuhkan waktu yang relatif lama untuk bisa menyelesaika permainan tersebut. Selain itu juga, manusia bisa merasa jenuh ataupun menyerah saat menyelesaikan permainan dengan tingkatan yang sangat sulit. Apabila pemain berhenti sejenak untuk beristirahat dari permainan, mungkin pemain bisa merasa lupa dengan proses yang sudah dilaluinya, sehingga ketika ia kembali untuk melanjutkan permainan, ia harus mengulang untuk memproses jalannya permainan itu.

Keterbatasan manusia yang seperti itu yang membuat manusia harus bergantung kepada mesin. Mesin tidak akan merasa jenuh jika diberikan sebuah tugas/proses yang memakan waktu lama. Selain itu, mesin mampu menjalankan algoritma dalam waktu yang singkat juga untuk bisa menemukan hasil. Manusia hanya harus memikirkan algoritma yang tepat agar mesin dapat menjalankannya dengan optimal.

Terdapat sebuah permainan kategori puzzle yang sangat dikenal terutama pada tahun 2000-an. Permainan tersebut adalah Sokoban, di mana seorang pemain harus memindahkan kotak yang ada ke tempat yang seharusnya. Seiring bertambahnya tingkat kesulitan permainan, akan muncul lebih banyak kotak yang harus dipindahkan. Permainan akan menjadi semakin sulit, sehingga untuk dapat menyelesaikan permainan dalam waktu singkat dapat menggunakan algoritma. Salah satu algoritma yang cocok untuk penyelesaian permainan ini adalah algoritma A* yang dapat mencari rute terpendek dari kumpulan rute yang ada. Algoritma A* akan membantu pemain dalam mencari jawaban jalur yang bisa menempatkan kotak di tempat yang sudah ditentukan oleh permainan.

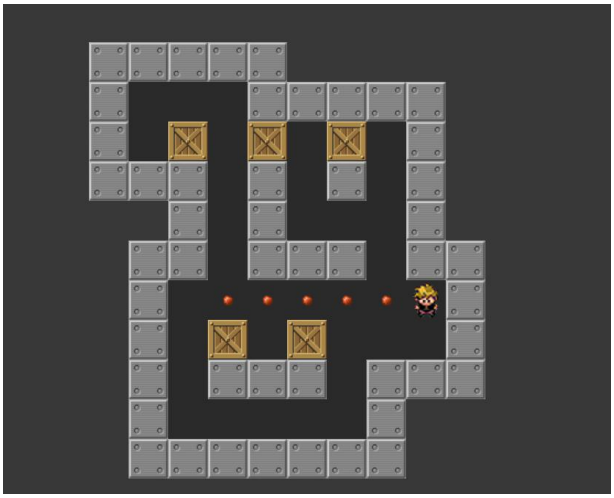
II. DASAR TEORI

A. *Permainan Sokoban*

Sokoban merupakan sebuah permainan *puzzle* klasik. Pertama kali di buat oleh Hiroyuki Imabayashi pada tahun 1980-an. Pada mulanya, permainan ini dibuat untuk kompetisi pemrograman. Hiroyuki berhasil menjadi juara karena permainan Sokoban yang ia buat. Selanjutnya pada tahun 1982, terbentuklah versi komersial untuk permainan Sokoban ini. Versi ini diluncurkan pada tahun 1984. Versi komersial untuk permainan ini pertama kali dibuat oleh perusahaan Thinking Rabbit dipimpin oleh Hiroyuki Imabayashi sendiri. Sebagai versi pertama dari permainan ini, Thinking Rabbit meluncurkan Sokoban yang bisa kompatibel dengan perangkat komputer IBM. Sokoban versi pertama ini di kembangkan oleh Spectrum Holobyte. Sekarang ini permainan Sokoban telah banyak dikembangkan dengan tampilan permainan yang lebih baik. Meskipun demikian, konsep dari permainan ini tetap sama seperti pada versi pertamanya.

Permainan Sokoban merupakan permainan yang menempatkan pemain sebagai ‘penjaga gudang’. Pemain harus mendorong kotak yang ada pada sebuah ruangan agar kotak-kotak tersebut bisa berada pada tempat yang telah ditentukan. Pemain dapat mendorong kotak ke empat arah, Utara, Selatan, Timur dan Barat. Dalam menggeser kotak, pemain hanya diperbolehkan untuk menggeser satu kotak dalam satu waktu.

Sehingga, pemain tidak bisa mendorong beberapa kotak secara bersamaan. Selain itu, pemain hanya bisa mendorong kotak jika pemain berada di satu posisi sebelum kotak.



Gambar 2.1 Contoh permainan Sokoban

Sumber : https://sokoban.info/?16

Permainan Sokoban ini terdiri atas beberapa tingkat kesulitan. Tingkat kesulitan ditentukan oleh jumlah kotak yang harus dipindahkan, struktur ruangan serta titik tujuan kotak diletakkan. Pemain bisa dikatakan menyelesaikan sebuah level apabila pemain dapat menempatkan seluruh kotak ke titik tujuannya.

Namun, tidak semua level dapat diselesaikan. Terdapat keadaan dimana pemain dinyatakan kalah akibat tidak bisa melanjutkan permainan. Kondisi ini terjadi ketika terdapat kotak yang sudah sampai di sudut ruangan. Karena berada di sudut dan dihalangi oleh 2 tembok, kotak pasti tidak dapat digerakkan lagi. Keadaan seperti ini biasa disebut dengan *deadlock.* Jika pemain sampai ke kondisi *deadlock*, maka pemain dinyatakan kalah.

Permainan ini cenderung lebih mengutamakan kecerdasan otak pemain dalam menentukan jalur yang dipilih daripada ketangkasan aksi tangan pemain. Selain itu, apabila pemain dapat menyelesaikan permainan ini dengan perpindahan yang lebih sedikit, maka skor pemain akan memiliki bobot yang lebih besar juga.

*B. Algoritma A**

Salah satu algoritma yang dapat digunakan untuk mencari jalur terpendek adalah algoritma A*. Algoritma A* pertama kali ditemukan oleh Peter Hart, Nils Nilsson dan Bertram Raphael pada tahun 1968. Algoritma ini merupakan pengembangan dari algoritma BFS. Kekurangan dari algoritma BFS adalah algoritma tersebut merupakan algoritma *uninformed search* sehingga pada saat proses pencarian jalur program tidak dapat memperkirakan *cost* yang akan ditempuh. Dengan algoritma A*, penelusuran jalur dapat dilakukan dengan menelusuri jalur dengan mempertimbangkan biaya yang sudah dikeluarkan untuk menempuh suatu jalur serta optimasi biaya selanjutnya dari jalur yang akan ditempuh.

Optimasi biaya dari rute yang akan di tempuh diperoleh dari fungsi heuristik. Fungsi heuristik merupakan sebuah proses yang melakukan estimasi biaya pada suatu node (posisi) tertentu. Nilai dari estimasi biaya tersebut diperoleh dengan mengambil jarak terpendek dari sebuah titik node ke titik tujuannya. Algoritma A* akan menggunakan informasi dari heuristik ini untuk menentukan rute mana yang nantinyaa akan dipilih. Sehingga fungsi heuristik ini perlu dibentuk untuk menghindari adanya pencarian pada jalur yang memiliki biaya yang lebih besar. Nilai estimasi biaya yang ditentukan oleh fungsi heuristik tidak boleh lebih besar dari biaya yang sesungguhnya agar algoritma dapat mendapatkan hasil yang optimal.

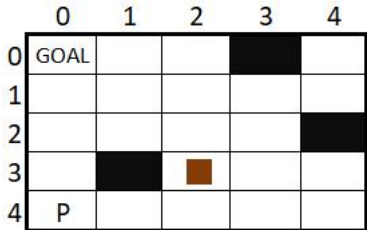
Dalam mencari rute terpendek, algortima A* menggunakan fungsi evaluasi yang dinyatakan dengan rumus berikut ini:

$$f(n) = g(n) + h(n)$$

Di mana f(n) adalah fungsi evaluasi yaitu fungsi yang menetukan jarak yang sudah ditempuh dan mengestimasi langkah terbaik selanjutnya. Sedangkan g(n) merupakan jumlah langkah yang telah ditempuh sebelumnya dari posisi awal hingga berada di posisi sekarang. Selanjutnya h(n) merupakan fungsi heuristik yang akan mengestimasi jumlah biaya selanjutnya yang akan dikeluarkan dari posisi saat ini ke posisi akhir nanti. Fungsi ini akan diimplementasikan pada setiap algoritma A* dan fungsi ini juga yang membuat optimasi dari algoritma ini menjadi lebih baik dibandingkan algoritma yang lain.

III. Penyelesaian Permainan Sokoban dengan Menerapkan Algoritma A*

Penyelesaian permainan Sokoban dapat diperoleh dengan pendekatan algortima A*. Pada dasarnya, pemain dan box sama-sama memiliki posisi awal. Selain itu juga terdapat posisi akhir yang merupakan tempat tujuan di mana kotak akan diletakkan. Sebagai contoh, terdapat peta Sokoban seperti di bawah ini:



Gambar 3.1 Contoh Panel Permainan

Pada gambar 3.1, terdapat huruf P yang melambangkan posisi awal pemain. Terdapat juga sebuah kotak kecil berwarna coklat sebagai kotak yang harus dipindahkan menuju *goal.*

Pada panel permainan tersebut juga terdapat beberapa sel yang di hitamkan. Sel tersebut merupakan gambaran dari tembok-tembok yang ada di permainan ini. Pemain tidak akan bisa melewati sel hitam tersebut, demikian juga kotak tidak dapat didorong melalui sel hitam tersebut.

Sebelum melakukan pemrosesan dengan algoritma A*, program harus terlebih dahulu menentukan fungsi heuristik. Fungsi heuristik yang dibuat tidak boleh *overestimate*, yang berarti biaya yang di estimasi lebih besar daripada biaya penelusuran yang sebenarnya. Di samping itu juga, fungsi heuristik harus konsisten agar penelusuran rute bisa dilakukan dengan optimal. Terdapat beberapa pilihan fungsi heuristik yang bisa digunakan untuk menyelesaikan permainan Sokoban ini, diantaranya:

1. Heuristik Nol

   Menggunakan heuristik dengan nilai nol untuk setiap jalur. Apabila menggunakan heuristik ini, maka implementasi algoritma A* akan menjadi mirip dengan algoritma BFS. Sehingga penggunaan heuristik ini kurang disarankan.

2. Jarak Manhattan dari kotak dan lokasi tujuan kotak

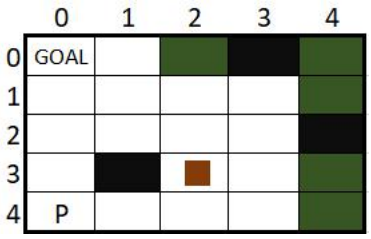
   Jarak Manhattan berarti mengukur jarak dari suatu titik ke titik lainnya pada bidang Cartesian dengan hanya menyusuri bidang horizontal dan vertikal saja. Pada kasus ini, fungsi heuristik akan menghitung jarak antar kotak ke lokasi tujuan kotak pada bidang horizontal dan vertikal.

3. Jarak Manhattan dari posisi pemain ke posisi kotak

   Konsep metode ini mirip dengan poin 2. Hanya saja pada metode ini yang di perhatikan adalah jarak antara pemain dengan kotak terdekat pada bidang horizontal dan vertikal saja.

4. Heuristik Kompleks

   Pada dasarnya, heuristik kompleks memanfaatkan jarak Manhattan juga dalam implementasinya. Namun selain jarak Manhattan, terdapat algoritma yang dapat digunakan untuk menentukan heuristik pada bagian-bagian sel yang berpotensi menyebabkan *deadlock*, sehingga nantinya algoritma A* akan memprioritaskan penelusuran ke jalur lain dibandingkan jalur yang berpotensi menyebabkan *deadlock*.



Gambar 3.2 Contoh posisi *deadlock*

Pada Gambar 3.2, terdapat sel yang diwarnai dengan hijau tua. Sel tersebut menggambarkan posisi *deadlock*. Apabila kotak berada di posisi-posisi tersebut, pemain dinyatakan kalah karena pemain tidak bisa lagi memindahkan kotak ke posisi mana pun. Heuristik kompleks akan memberikan bobot yang lebih besar untuk posisi-posisi tersebut setelah menghitung heuristik jarak Manhattan antara kotak dengan posisi tujuan.

Makalah ini akan menggunakan versi heuristik kompleks di di mana fungsi heuristik yang akan diimplementasikan memiliki algoritma tambahan untuk mencegah kotak berada di posisi *deadlock*. Potongan algoritma yang akan digunakan untuk menentukan fungsi heuristik akan ditulis dalam bahasa Java seperti dibawah ini:

```
 public static int[][] heuristicBox(int goalX,
int goalY, int[][] room) {
        //inisialisasi matriks heuristic
        int[][] heu = new
int[maxVertical][maxHorizontal];
        //fungsi yang mengeluarkan potetial
deadlock
        boolean[][] potDeadlock =
potentialDeadlock(r);

        for (int i = 0; i<maxVertical; i++) {
            for (int j = 0; j<maxHorizontal;
j++) {
                //sel hitam
                if (r[i][j] == 0) {
                    heu[i][j] = 99;
                }
                //sel hijau tua
                else if (potDeadlock[i][j]) {
                    heu[i][j] = Math.abs(goalX-
i)+Math.abs(goalY-j)+10;
                }
                //sel putih
                else {
                    heu[i][j] = Math.abs(goalX-
i)+Math.abs(goalY-j);
                }
            }
        }
        //return tabel
        return heu;
    }
```

Potongan kode untuk fungsi heuristik

Fungsi di atas mencari jarak Manhattan antara posisi sebuah sel yang mungkin di lalui oleh kotak dan posisi tujuan sel yang menjadi tempat penyimpanan kotak yang seharusnya. Dengan algoritma ini, seluruh sel pada lingkungan permainan akan memiliki nilai estimasi biaya yang diperlukan jika akan menelusuri ke tujuan permainan melalui sel tersebut. Pada fungsi heuristik ini terdapat sebuah fungsi lain yang dipanggil yaitu fungsi potDeadlock(int[][] room) : boolean[][] yang akan menentukan apakah sebuah sel berpotensi menjadi titik *deadlock* apabila dilalui oleh kotak. Algoritma untuk penentuan posisi *deadlock* akan dijelaskan pada potongan program di bawah ini:

:

```
 public static boolean[][]
potentialDeadlock(int[][] r) {
        boolean[][] potential = new
boolean[maxVertical][maxHorizontal];

        //inisialisasi semua dengan false
        for (int i = 0; i<maxVertical; i++) {
            for (int j = 0; j<maxHorizontal; j++) {
               potential[i][j] = false;
            }
        }

        //cari sel dengan potensial deadlock
        for (int i = 0; i<maxVertical; i++) {
            for (int j = 0; j<maxHorizontal; j++) {
                if ((i == 0 && j == 0) || (i == 0
&& j == maxHorizontal-1) || (i == maxVerical-1 && j
== maxHorizontal-1) || (i == maxVertical-1 && j ==
0)) {
                    potential[i][j] = true;
                }
                else if (r[i][j] == 0) {
                    if (j == 0 || j ==
maxHorizontal-1) {
                        //bagian atas dan bawahnya
                        potential[i+1][j] = true;
                        potential[i-1][j] = true;
                     }
                     if (i == 0 || i ==
maxVertical-1) {
                         //bagian atas dan
bawahnya
                         potential[i][j+1] =
true;
```

Potongan program untuk fungsi potentialDeadlock

Potongan kode diatas merupakan potongan kode untuk fungsi heuristik dari kotak menuju tujuan. Perlu dipertimbangkan kasus *deadlock* karena akan ada banyak kemungkinan posisi kotak yang bisa menyebabkan *deadlock*.

Untuk fungsu heuristik dari pemain, dapat menggunakan algoritma pencarian jarak Manhattan yang biasa. Dari sebuah posisi pada sel harus dihitung jarak terpendeknya menuju posisi kotak. Potongan program untuk fungsi heuristik akan di jabarkan dibawah ini:

```
 public static int[][] heuristicPlayer(Box B,
int[][] r) {

int[][] heu = new int[maxHorizontal][maxVertical];

     for (int i = 0; i<maxVertical; i++) {
            for (int j = 0; j<maxHorizontal; j++) {
        //jarak manhattan
                heu[i][j] = Math.abs(B.getX()-
i)+Math.abs(B.getY()-j);
                }
            }

        return heu;
    }
```

Kode program untuk heuristik pemain

Setelah menentukan fungsi heuristik, algortima A* dapat diibentuk dan akan selanjutnya dijalankan agar bisa menemukan solusi yang tepat untuk penyelesaian permainan Sokoban ini.

Algoritma A* yang digunakan adalah algoritma A* biasa yang menggunakan *priority queue* yang dapat menyimpan sel dengan yang diurutkan sesuai dengan fungsi evaluasi f(n). Fungsi evaluasi merupakan nilai biaya yang sudah dikeluarkan dari titik awal ke titik saat ini dan ditambahkan dengan estimasi biasa yang akan sikeluarkan dari titik asal ini hingga ke titik tujuan nantinya.

Untuk algoritma yang akan digunakan pada pencarian solusi dengan A* akan dijelaskan oleh potongan *pseudocode* di bawah ini:

```
Inisialisasi matriks ruangan dan titik tujuan
Inisialisasi posisi pemain
Inisialisasi posisi kotak-kotak yang akan dipindahkan
Inisialisasi priority queue

ADD sel AWAL ke priority queue

WHILE priority queue is not empty:
        REMOVE elemen (yang pertama) pada queue
        IF elemen is GOAL
                Return jalur
        ADD  elemen TO jalur
        FOREACH langkah yang mungkin dari elemen
                g(langkah) = g(elemen)+1
                hitung h(langkah)
                IF langkah ada pada queue dan lebih optimal contiune;
                IF langkah ada pada jalur dan lebih optimal continue;
                Else REMOVE langkah pada jalur maupun queue
                //untuk kerunutan jalur
                ADD elemen sebagai parent dari langkah
                ADD langkah ke queue untuk evaluasi langkah
        selanjutnya
        Endfor
```

*Pseudocode untuk Alogritma A**

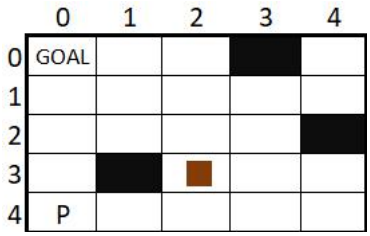
Bagian yang akan diproses terlebih dahulu adalah antrian yang memiliki nilai yang paling kecil. Saat melakukan proses, elemen paling kecil harus di hapus dari antrian. Selanjutnya akan di evaluasi, apabila sudah mencapai tujuan, maka program akan di hentikan dan mengeluarkan hasil jalur yang sudah diperoleh.

Pada pemrosesan ini, ada dua fungsi heuristik yang dipakai. Fungsi heuristik ini lah yang akan mengarahkan pergerakan dari pemain untuk bisa sampai ke tujuan dengan membawa kotak. Fungsi heuristik pertama yang akan digunakan adalah fungsi heuristik dengan jarak Manhattan ke kotak. Sehingga pemain akan bergerak dari posisi awalnya hingga posisi sebelum kotak.

Namun, apabila pemain sudah berada berdekatan dengan kotak, fungsi yang akan digunakan adalah fungsi heuristik kompleks yang disudah dijelaskan pada potongan kode program sebelumnya. Dengan nilai heuristik ini, algoritma A* akan membawa pemain bersamaan dengan kotak menuju titik tujuan dari kotak tersebut. Langkah yang diikuti masih tetap sama seperti algoritma A*, namun hanya menggunakan fungsi heuristik yang berbeda saja.

IV. Penerapan dan Analisis

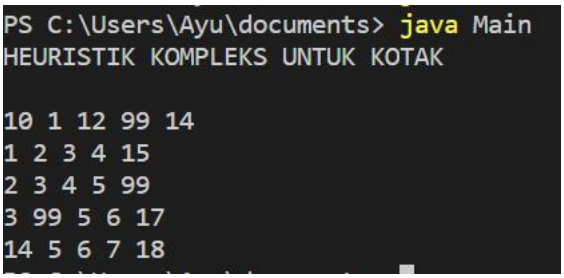
Misalkan terdapat lingkungan permainan yang sesuai contoh panel permanan pada bab 3.



Gambar 4.1 Lingkungan Permainan

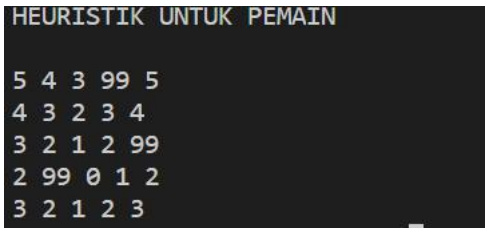
Terdapat pemain pada indeks (4,4) dan sebuah kotak pada indeks (3,2). Selain itu terdapat berbagai tembok di titik (0,3), (2,4) dan (1,3).

Langkah pertama yang harus dilakukan adalah menghitung fungsi heuristik, baik untuk pemain maupun untuk kotak. Adapun hasil dari fungsi heuristik untuk kotak adalah sebagai berikut:



Gambar 4.2 Nilai heuristik kotak

Langkah selanjutnya adalah dengan menentukan fungsi heuristik untuk pemain. Dengan algoritma yang sudah dijelaskan pada bab 3, maka tampilan fungsi heuristik akan menjadi :

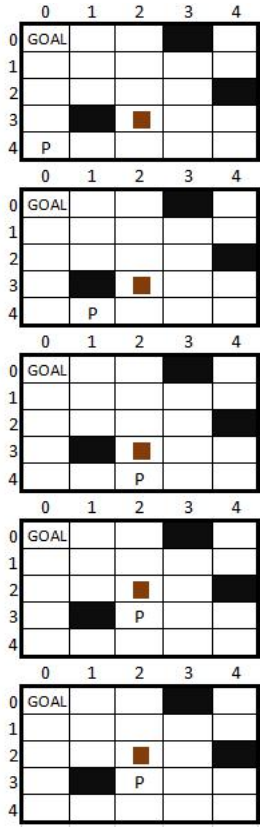


Gambar 4.3 Nilai heuristik Pemain

Pada bagian ini bisa dilihat bahwa di indeks matriks yang memiliki nilai heuristik 99 merupakan sel yang berwarna hitam. Sel tersebut tidak bisa dilalui sehingga harus memiliki nilai heuristik yang besar agar tidak dipilih. Terdapat berbagai cara lain untuk bisa menyatakan bahwa sel tersebut tidak boleh dilewati, namun untuk kasus ini metode yang dipilih adalah dengan mengisi nilai heuristik untuk sel tersebut dengan angka 99.

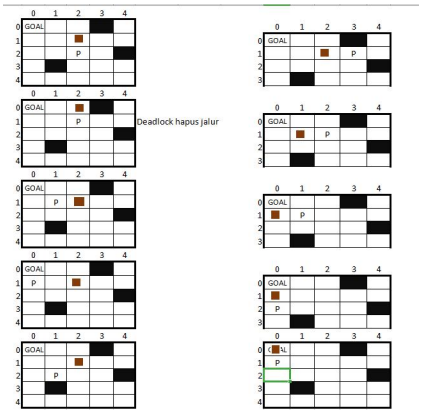
Selain itu juga pada heuristik kompleks untuk kotak, untuk posisi sel yang diwarnai dengan warna hijau memiliki nilai yang lumayan besar. Bisa dibandingkan dengan heuristik untuk pemain di posisi yang sama, nilainya tidak terlalu besar. Hal ini terjadi karena jika kotak yang menempati posisi tersebut, pasti akan menyebabkan pemain kalah, sehingga pilihan jalur dari rute itu diberukan prioritas yang rendah (dilambangkan dengan nilai heuristik yang tinggi). Sedangkan untuk pemain, apabila pemain harus melewati sel tersebut tidak akan terjadi masalah apapun karena pemain bisa bergerak dengan bebas di lingkungan permainan.

Selanjutnya akan dilakukan proses penelusuran jalur dengan algoritma A*. Sesuai dengan pseudocode pada bagian 3 di atas, pertama akan di cari jalur menuju kotak dengan menggunaakan heuristik dari pemain.



Gambar 4.4 Proses Pemain ke Kotak

Proses ini menggambarkan kerunutan lagoritma yang dijalankan hingga pemain sampai di titik sebelum kotak. Pemain akan mengikuti algoritma A* dan mendekati kotak. Saat sudah sampai di dekat kotak, maka fungsi heuristik yang akan dipakai adalah heuristik kompleks dari kotak, dan akan dijalankan hingga mencapai tujuan dari kotak tersebut. Berikut adalah proses untuk menjalankan algoritmanya:



Gambar 4.5 Proses Kotak ke Tujuan

Dapat dilihat pada proses ini bahwa ada proses yang berhenti di titik deadlock. Sehingg jalur tersebut dihapus dari list jalur, dan diganti dengan pembuatan jalur yang baru.

V. KESIMPULAN

Permainan Sokoban memiliki beberapa hal yang menjadi aspek permainan, diantaranya: Ruangan beserta tembok yang merupakan lingkungan permainan. Posisi tembok yang berdekatan dapat menyebabkan *deadlock.* Apabila hal itu terjadi, permainan akan berakhir. Posisi, posisi sangat menentukan dalam penyelesaian permainan ini. Apabila salah mengambil langkah, maka akan sulit untuk kembali lagi. Selain itu, terdapat beberapa pengaturan posisi yang memaksa pemain untuk kalah. Pengaturan posisi itu menyebabkan permainan benjadi tidak bisa di selesaikan. Untuk dapat menyelesaikan permainan Sokoban ini, dapat dibentuk sebuah algoritma yang memanfaatkan konsep algoritma A*. Proses pencarian jalur bisa ditemukan dengan waktu yang singkat. Dengan algoritma ini juga pemain bisa menentukan apakah model pengaturan posisi lingkungan permainan memiliki solusi atau tidak.

ACKNOWLEDGMENT

Saya sangat berterimakasih kepada Tuhan YME yang memberikan kekuatan untuk bisa menyelesaikan makalah Strategi Algoritma ini. Saya juga berterimakasih kepada orang tua yang selalu mengdukung saya, serta teman-teman saya yang juga sama-sama saling mendukung satu sama lain agar makalah ini bisa diselesaikan dengan baik.

REFERENCES

[1] http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html diakses tanggal 26 April 2019 pukul 01.01

[2] http://www.talistreit.com/AStar.html diakses tanggal 26 April 2019, pada pukul 07.00

[3] http://www.cs.huji.ac.il/~ai/projects/2012/SokobanWP/files/report.pdf. diakses tanggal 25 April 2019 pada pukul 18.05

[4] http://sokobano.de/wiki/index.php?title=Solver, diakses tanggal 26 April 2019 pukul 13.00

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 April 2019

Ayu Rifanny Margareth