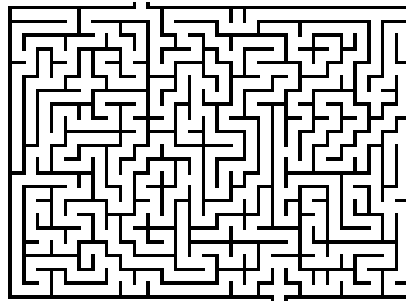


BAB II KAJIAN TEORI DAN KAKAS PENDUKUNG

Bab ini berisi penjelasan mengenai teori dasar yang berkaitan dengan pelaksanaan tugas akhir, antara lain labirin, agen, sistem multiagen, serta algoritma yang akan digunakan dalam pelaksanaan tugas akhir ini.

2.1 Labirin

Labirin adalah sebuah jaringan dari jalur-jalur yang saling berhubungan untuk dilalui dari awal hingga akhir yang dimaksudkan untuk sebuah tantangan. Istilah labirin berasal dari bahasa Yunani, yaitu *Labrys* yang berarti kapak bermata dua. Contoh gambar labirin dapat dilihat pada Gambar II-1.



Gambar II-1: Labirin

Labirin memiliki banyak jenis klasifikasi sesuai dengan karakteristik yang dimilikinya. Klasifikasi yang dimiliki oleh sebuah labirin yaitu antara lain dimensi, hiperdimensi, topologi, struktur unit pembentuknya (*Tessellation*), rute, tekstur, dan fokus. Sebuah labirin memiliki gabungan dari seluruh klasifikasi yang ada, tetapi mungkin memiliki tipe yang berbeda-beda. Klasifikasi labirin yang lebih lengkap berdasarkan tipe-tipe yang dimiliki oleh sebuah labirin dapat dilihat pada Lampiran A [PUL07].

Dari berbagai klasifikasi labirin yang disebutkan pada Lampiran A, maka dibutuhkan algoritma untuk membentuk labirin-labirin tersebut. Jenis-jenis algoritma yang biasa digunakan untuk membangun labirin adalah sebagai berikut:

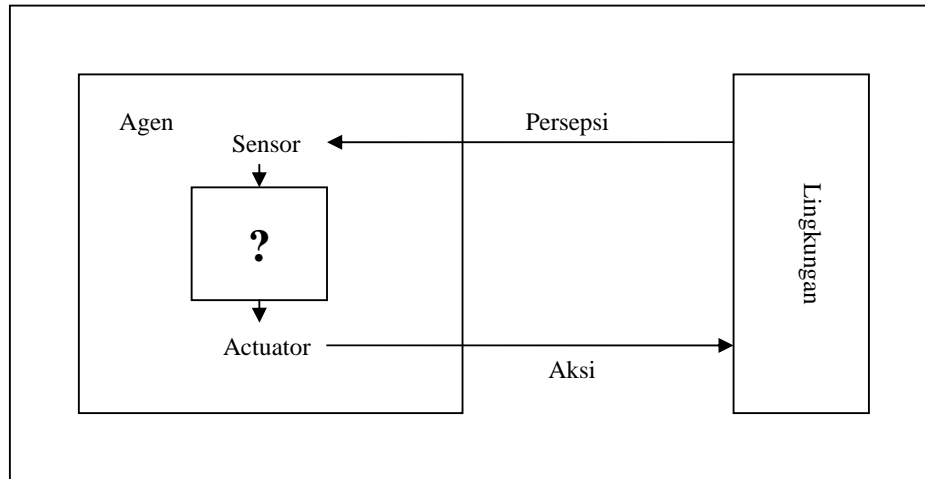
1. *Perfect*. Algoritma ini membangun sebuah labirin yang bersifat *perfect*, dimana pada labirin tersebut dipastikan bahwa tidak ada loop atau area yang tidak dapat dicapai. Kunci dari algoritma ini adalah memastikan bahwa dinding yang ditambahkan salah satu ujungnya menempel pada dinding yang sudah ada, dan ujung lainnya bebas.
2. *Braid*. Algoritma ini digunakan untuk membuat labirin yang tidak memiliki jalur buntu. Hal ini diperoleh dengan membuat loop dan memastikan bahwa labirin tidak menghasilkan jalur yang buntu.
3. *Unicursal*. Algoritma ini akan membuat labirin yang tidak memiliki persimpangan, sehingga tidak ada pemilihan jalur yang dilakukan selama penelusuran.
4. 3D. Algoritma ini pada dasarnya sama dengan yang digunakan pada labirin dua dimensi, tetapi dengan dua tambahan jalur, ke atas dan ke bawah.
5. *Weave*. Algoritma ini pada dasarnya sama dengan *perfect*, tetapi tidak selalu terhalang oleh jalur lainnya karena dapat melintasi bagian bawah jalur tersebut, sehingga tetap memiliki kualitas dari *perfect*.
6. *Crack*. Algoritma ini berjalan dengan memilih sebuah posisi yang berhimpitan dengan dinding yang sudah ada, kemudian secara acak memilih posisi lainnya kemudian menarik garis untuk menjadi dinding labirin yang baru, tetapi harus menghindari bertemu dengan dinding labirin lainnya yang sudah ada, agar tidak membuat area yang terisolasi.

2.2 Agen

Dalam [WEI99], agen didefinisikan sebagai sistem komputer yang terletak pada sebuah lingkungan dan dapat melakukan aksi secara otonom dalam lingkungannya tersebut untuk memenuhi tujuannya. Sedangkan [RUS03] mendefinisikan agen sebagai sesuatu yang mengamati lingkungannya melalui sensor dan melakukan aksi terhadap lingkungannya tersebut melalui aktuator.

Gambaran mengenai definisi tersebut dapat dilihat pada Gambar II-2. Seorang agen manusia memiliki mata, telinga, dan organ lainnya sebagai sensor serta tangan, kaki, mulut dan bagian tubuh lainnya sebagai aktuator. Sebuah agen robot mungkin memiliki kamera dan sensor inframerah sebagai sensor dan berbagai motor sebagai

aktuatornya. Sedangkan agen perangkat lunak menerima masukan dari *keyboard*, isi dokumen, dan *network packet* sebagai sensor masukan dan beraksi terhadap lingkungannya dengan menampilkan sesuatu pada layar, menulis ke dokumen, dan mengirimkan *network packet*.



Gambar II-2: Interaksi agent dengan lingkungannya melalui sensor dan aktuator

[WEI99] menyebutkan tiga syarat yang diperlukan oleh sebuah agen untuk dapat disebut *intelligent*. Syarat pertama yaitu reaktif, yaitu agen dapat mengamati lingkungannya dan memberikan respon pada selang waktu tertentu terhadap perubahan yang terjadi. Syarat kedua yaitu proaktif, yaitu agen dapat memperlihatkan *goal-directed behaviour* dengan mengambil inisiatif. Syarat ketiga yaitu memiliki kemampuan sosial, yang dimaksud dengan hal ini yaitu sebuah agen dapat berinteraksi dengan agen lainnya, atau dengan manusia.

2.3 Sistem Multiagen

Walaupun penggunaan agen dapat membantu menyelesaikan masalah tersebut, penggunaan satu agen saja mungkin tidak cukup. Penggunaan satu agen yang terpusat mungkin secara umum akan lebih efisien daripada penggunaan agen yang terdistribusi. Walaupun begitu, pada kenyataannya sebagian besar masalah yang ada tidak cukup diselesaikan hanya menggunakan agen tunggal saja. Alasan dari hal ini yaitu karena permasalahannya sendiri mungkin sudah dalam bentuk terdistribusi, atau adanya data atau informasi rahasia yang tidak boleh diketahui pihak lain, atau domain

permasalahannya sendiri sangat luas. Penggunaan agen yang lebih dari satu disebut dengan Sistem Multiagen (*Multiagent System/MAS*).

Selanjutnya, lingkup permasalahan yang biasa diselesaikan menggunakan sistem multiagen yaitu pengambilan keputusan secara independen tanpa campur tangan manusia. Beberapa contoh dari hal ini yaitu pengontrol udara pada suatu bangunan, pelelangan melalui internet, permainan catur, dan pencarian rute [WEI99]. Seluruh sistem di atas menggunakan sistem multiagen dalam proses pengambilan keputusan yang dilakukan.

Karakteristik dari MAS yaitu (1) setiap agen yang ada hanya memiliki informasi dan kemampuan penyelesaian masalah yang bersifat parsial, sehingga memiliki pandangan yang terbatas; (2) tidak memiliki sistem kontrol yang bersifat global dari sistem tersebut; (3) data terdesentralisasi dalam sistem; (4) komputasi bersifat asinkron [SYC98].

Kelebihan penggunaan MAS dibandingkan dengan agen tunggal yaitu (1) lebih toleran terhadap kesalahan, hal ini terjadi karena banyaknya jumlah agen yang digunakan, sehingga jika beberapa agen tersebut mengalami kegagalan, masih ada agen lainnya; (2) penggunaan sumber daya yang lebih efisien, hal ini dimungkinkan jika menggunakan paralelisme [YE07].

Tidak semua algoritma atau aplikasi yang ada dapat memanfaatkan sistem multiagen ini. Sebuah algoritma atau aplikasi dapat dikatakan memanfaatkan sistem multiagen apabila pada suatu waktu tertentu memiliki beberapa agen yang aktif beroperasi.

2.4 Algoritma Pencarian Rute Terdekat pada Labirin

2.4.1 A^*

A^* adalah salah satu bentuk *best-first search* yang paling dikenal. Pemilihan node yang akan diekspansi dilakukan dengan melakukan evaluasi $g(n)$, yaitu biaya yang digunakan untuk mencapai sebuah node, dan $h(n)$, yaitu biaya yang diperlukan untuk mencapai tujuan dari node tersebut.

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

Strategi ini sangat baik untuk digunakan, karena jika nilai daripada $h(n)$ memenuhi syarat tertentu, maka strategi ini akan lengkap dan optimal. A^* disebut optimal jika $h(n)$ bersifat *admissible heuristic*, yaitu tidak pernah memakai nilai yang terlalu melampaui perkiraan untuk mencapai tujuan [RUS03]. Untuk dapat memenuhi syarat *admissible heuristic* ini, maka diperlukan pengetahuan heuristik dari pakar.

Nilai $h(n)$ bisa didapatkan melalui beberapa cara, antara lain dengan menarik garis lurus antara node saat ini dengan node akhir dan menghitung jaraknya atau dengan menghitung jumlah busur minimum yang perlu dilalui untuk mencapai node akhir dari node saat ini. Deskripsi algoritma A^* dapat dilihat pada Algoritma II-1.

```

OPEN = priority queue containing START
CLOSED = empty set
while lowest rank in OPEN is not the GOAL:
    current = remove lowest rank item from OPEN
    add current to CLOSED
    for neighbors of current:
        cost = g(current) + movementcost(current, neighbor)
        if neighbor in OPEN and cost less than g(neighbor):
            remove neighbor from OPEN, because new path is better
        if neighbor in CLOSED and cost less than g(neighbor): **
            remove neighbor from CLOSED
        if neighbor not in OPEN and neighbor not in CLOSED:
            set g(neighbor) to cost
            add neighbor to OPEN
            set priority queue rank to g(neighbor) + h(neighbor)
            set neighbor's parent to current

reconstruct reverse path from goal to start
by following parent pointers

```

Algoritma II-1: Algoritma A^*

2.4.2 Wall Follower

Algoritma ini bekerja dengan menelusuri dinding, dan saat menemui persimpangan, akan selalu berbelok ke kiri (atau kanan). Hal ini persis dengan bagaimana manusia menyelesaikan labirin dengan menempelkan tangannya ke dinding dan berjalan terus tanpa melepaskan tangannya. Metode ini tidak selalu menemukan solusi terbaik, dan bahkan tidak dapat bekerja saat tujuan akhir terletak di tengah area dan terdapat sirkuit tertutup mengelilinginya. Algoritma ini tidak menggunakan memori tambahan [PUL07].

2.4.3 Recursive Backtracker

Algoritma ini bekerja dengan menelusuri labirin dengan bebas. Jika menemui jalur buntu, maka akan mengembalikan nilai gagal, jika sampai posisi tujuan akan mengembalikan nilai berhasil, selain itu secara rekursif bergerak ke empat arah. Jalur-jalur yang sudah dilalui akan ditandai agar tidak kembali ditelusuri dari arah lainnya. Dalam istilah sains komputer, metode ini pada dasarnya adalah *depth first search*. Metode ini akan selalu menemukan solusi, tetapi tidak selalu solusi yang terbaik [PUL07].

2.4.4 Collision Solver

Algoritma ini disebut juga dengan *amoeba solver*, dimana strategi ini akan menemukan seluruh solusi terpendek. Strategi ini dapat diibaratkan dengan membanjiri labirin dengan air, sehingga seluruh jalur dari posisi awal dipenuhi pada saat yang sama, saat menemui percabangan, maka aliran air akan terbagi dan akan melanjutkan perjalanannya hingga menemui jalur yang buntu atau bertemu dengan aliran air lainnya. Seluruh aliran air yang terbentuk mengetahui jalur yang sudah dilaluinya dan dilalui aliran air lainnya, serta mengingat jarak yang telah ditempuhnya. Saat salah satu aliran sudah mencapai posisi akhir yang diinginkan, maka selain aliran yang menemui jalur buntu atau bertemu aliran lainnya akan melanjutkan kerjanya sampai juga mencapai posisi akhir. Setelah seluruh jalur sudah terbanjiri, maka dipilih solusi terpendek dari beberapa kemungkinan solusi yang ada [PUL07]. Algoritma ini menggunakan memori sebesar labirin yang digunakan, selain menggunakan memori sebesar ukuran dari labirin itu sendiri.

2.5 Portable Graymap (PGM)

Format *portable graymap* adalah sebuah format citra dalam bentuk ASCII yang didesain untuk dapat digunakan pada berbagai sistem operasi. Secara struktur, format ini tidak efisien dalam segi ukuran kapasitas, tetapi dapat ditangani oleh berbagai aplikasi manipulasi citra [SAU07].

Sebuah citra *pgm* merepresentasikan sebuah citra gambar dalam bentuk *grayscale*. Terdapat beberapa format semi-PGM yang digunakan, dimana hampir seluruhnya sama dengan spesifikasi aslinya kecuali arti dari setiap pixel yang ada. Pada

umumnya, sebuah citra PGM dapat dipandang sebagai sebuah kumpulan bilangan integer. Sebagai akibatnya, sebuah aplikasi yang mengira sedang memproses sebuah citra *grayscale* dapat dengan mudah ditipu untuk memproses hal lainnya.

Salah satu variasi resmi dari PGM adalah *transparency mask*. *Transparency mask* dalam Netpbm direpresentasikan dalam citra PGM, kecuali sebagai pengganti nilai intensitas pixel, digunakan nilai *opaqueness*. Sebuah file PGM dapat mengandung satu atau lebih dari citra PGM. Tidak ada data, *delimiter*, atau *padding* sebelum, setelah atau antara citra yang ada.

Setiap citra PGM mengandung hal-hal sebagai berikut:

1. Kode khusus untuk mengidentifikasi tipe file. Kode khusus untuk citra PGM adalah "P5".
2. *Whitespace* (blanks, TABs, CRs, LFs).
3. Lebar citra, dituliskan dalam karakter desimal ASCII.
4. *Whitespace*.
5. Tinggi citra, dalam desimal ASCII.
6. *Whitespace*.
7. Nilai *gray value* maksimum (Maxval), dalam desimal ASCII. Jangkauan Maxval adalah dari 0 hingga 65536.
8. Sebuah karakter *whitespace* atau *newline*.
9. Sekumpulan baris sejumlah tinggi citra, dari atas ke bawah. Setiap baris mengandung *gray values* sejumlah lebar citra, dari kiri ke kanan. *Gray value* yang diizinkan adalah dari 0 hingga Maxval, dimana 0 adalah hitam dan Maxval adalah putih. Setiap nilai direpresentasikan dengan nilai biner dalam 1 atau 2 byte. Jika Maxval kurang dari 256, maka yang digunakan adalah 1 byte. Selain dari itu digunakan 2 byte. Pixel dalam citra berbentuk persegi dan kontigu.
10. String yang dimulai dengan karakter "#" adalah komentar.

Ada satu variasi dari format PGM yang jarang digunakan, yaitu format *plain* PGM. Format normal sebagaimana yang disebutkan di atas disebut dengan format *raw* PGM. Perbedaan dalam format *plain* adalah sebagai berikut:

1. Dalam satu file, ada tepat satu citra.
2. Kode yang digunakan adalah P2.
3. Pixel direpresentasikan dalam desimal ASCII.
4. Setiap pixel harus memiliki minimal sebuah spasi sebelum dan sesudah pixel tersebut, tapi tidak ada batas maksimum.
5. Sebuah baris tidak boleh melebihi 70 karakter.

Gambar II-3 adalah contoh dari sebuah file PGM dalam format *plain*.

```
P2
# feep.pgm
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 0 0 0 7 7 7 0 0 0 0 11 11 11 0 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Gambar II-3: Format Penulisan Citra PGM

2.6 Mazesmith

Mazesmith adalah aplikasi berbasis javascript yang dapat digunakan untuk membangun labirin secara acak dengan cepat yang dibuat oleh Robert Klein. Aplikasi ini memiliki parameter yang cukup lengkap untuk membuat sebuah labirin yang unik. Aplikasi ini tidak memiliki batas maksimal ukuran labirin yang dapat dibuatnya, melainkan hanya tergantung kepada besar dan kecepatan dari memori yang dimiliki oleh komputer itu sendiri.

Beberapa parameter penting dari Mazesmith dituliskan sebagai berikut:

1. *Braid*, semakin tinggi nilainya, akan semakin tinggi kemungkinan untuk menghilangkan jalur buntu, mengakibatkan terciptanya sebuah loop. Parameter ini tidak berjalan dengan baik pada algoritma Prim
2. *Direction bias*, yaitu kecenderungan arah dari jalur yang dibuat. Semakin positif nilainya, maka jalur pada labirin akan semakin horizontal, sedangkan semakin negatif nilainya, maka jalur pada labirin akan semakin vertikal.
3. *Versions*, yaitu algoritma pembentukan labirin. Algoritma pembentukan labirin yang digunakan pada aplikasi ini adalah:
 - a. Grow: memiliki jalur buntu yang lebih panjang dibandingkan dengan DFS, menghasilkan labirin yang lebih sulit.
 - b. DFS: memiliki jalur solusi yang lebih panjang, jalur buntu yang pendek dan beberapa yang sangat panjang, dan lebih mudah diselesaikan dibandingkan dengan Grow.
 - c. Prim: memiliki jalur solusi yang tidak rumit, memiliki banyak jalur buntu yang pendek, mudah diselesaikan.
 - d. Binary: memiliki jalur solusi yang paling tidak rumit, banyak jalur buntu, baik panjang maupun pendek, paling mudah diselesaikan.
 - e. Eller: memiliki jalur solusi yang rumit dengan jalur buntu yang panjang, mampu membuat labirin terpanjang di dunia.

2.7 Recursive Porous Agent Simulation Kit (Repast)

Repast adalah salah satu dari berbagai kaskas pemodelan agen yang ada. Repast meminjam banyak konsep dari Swarm. Repast berbeda dengan Swarm karena Repast memiliki banyak implementasi murni dalam beberapa bahasa dan fitur adaptif yang terkandung di dalamnya, seperti algoritma genetik dan regresi.

Repast adalah kaskas *free open source* yang pada awalnya dikembangkan oleh Sallach, Collier, Howe, dan North. Repast dikembangkan di University of Chicago. Selanjutnya, Repast diteruskan oleh Argonne National Laboratory. Saat ini Repast dikelola oleh organisasi sukarela nonprofit Repast Organization for Architecture and

Development (ROAD). ROAD dipimpin oleh badan direksi yang beranggotakan individu dari pemerintahan, akademik, dan industri.

Repast mendukung pengembangan model yang fleksibel dari living social agents, tetapi tidak hanya terbatas pada hal itu saja. Tujuan dari Repast adalah sebagai berikut:

"Our goal with Repast is to move beyond the representation of agents as discrete, self-contained entities in favor of a view of social actors as permeable, interleaved, and mutually defining; with cascading and recombinant motives. We intend to support the modeling of belief systems, agents, organizations, and institutions as recursive social constructions."

Maksud dari pernyataan di atas yaitu bahwa Repast ditujukan untuk melampaui batasan representasi agen tidak hanya sebagai objek yang diskret, tetapi menjadi pelaku sosial, sehingga dapat memodelkan organisasi dan institusi sebagai konstruksi sosial rekursif.

Pada intinya, kaskas Repast 3 dapat dipandang sebagai sebuah spesifikasi dari fungsi pemodelan agen. Terdapat tiga implementasi dari spesifikasi konseptual ini. Tentu saja ketiga implementasi ini memiliki fungsi inti yang membentuk sistem Repast. Ketiga implementasi tersebut adalah Repast untuk Java (Repast J), Repast untuk framework Microsoft.Net (Repast.Net), dan Repast untuk Phyton (RepastPy) [REP08].