

DESIGN AND IMPLEMENTATION OF MINIMAL MEMORY ABSTRACTION METHOD FOR REALTIME PATHFINDING HANDLING IN ANGRY DOG GAME

Listanto Tri Utomo¹, Farid Wahyudi²

¹ Teknik Informatika, Universitas Islam Raden Rahmat

² Sistem Informasi, Universitas Islam Raden Rahmat

listanto.tri@uniramalang.ac.id

ABSTRAK

Perkembangan permainan komputer yang semakin kompleks dengan menyajikan grafis yang realistis dan NPC (*Non Player Character*) yang canggih menuntut pengembang menitikberatkan pengembangan pada aspek grafis dan kecerdasan buatan. Pergerakan agen merupakan elemen penting dari aspek kecerdasan buatan dan strategi *pathfinding* merupakan bagian utama dalam pergerakan agen. Salah satu algoritma pembelajaran yang dapat diterapkan untuk menyelesaikan permasalahan *realtime pathfinding* adalah dengan menggunakan metode *minimal memory abstraction*. Pada penelitian ini nantinya akan dirancang sebuah game *pathfinding* bernama *Angry Dog* dan akan dilakukan pengujian apakah metode *minimal memory abstraction* dapat diimplementasikan pada game *Angry Dog*. Permasalahan *realtime pathfinding* pada game *Angry Dog* dapat diselesaikan dengan menggunakan metode *minimal memory abstraction* dan hasil perhitungan pada program sesuai dan mempunyai nilai yang sama dengan perhitungan manual seperti yang telah ditunjukkan pada tahap pengujian, serta ketika program melakukan perhitungan nilai *fitness* dan kemudian sistem menemukan kotak yang mempunyai nilai *fitness* terkecil yang sama, penentuan pengambilan keputusan ditentukan dari urutan kotak mana dulu yang dihitung.

Kata kunci: *realtime patfinding, minimal memory abstraction*

ABSTRACT

The development of increasingly complex computer games by presenting realistic graphics and sophisticated NPCs (*Non Player Characters*) requires developers to focus on developing aspects of graphics and artificial intelligence. Agent movement is an important element of artificial intelligence aspect and *pathfinding* strategy is a major part of agent movement. One of the learning algorithms that can be applied to solve *realtime pathfinding* problems is to use the *minimal memory abstraction* method. In this research, a *pathfinding* game called *Angry Dog* will be built and will be tested whether the *minimal memory abstraction* method can be implemented in the *Angry Dog* game. The *realtime pathfinding* problem in the *Angry Dog* game can be solved using the *minimal memory abstraction* method and the calculation results in the program match and have the same value as the manual calculation as shown in the testing phase, as well as when the program calculates the *fitness* value and then the system finds the correct box. have the same smallest *fitness* value, the decision making is determined from the order in which the boxes are calculated first.

Keywords: *realtime patfinding, minimal memory abstraction*

PENDAHULUAN

Latar Belakang

Perkembangan permainan komputer yang semakin kompleks dengan menyajikan grafis yang realistis dan NPC (*Non Player Character*) yang canggih menuntut pengembang untuk

melakukan pengembangan pada aspek grafis dan kecerdasan buatan. Pergerakan agen merupakan elemen penting dari aspek kecerdasan buatan dan strategi *pathfinding* merupakan bagian utama dalam pergerakan agen. *Game pathfinding* yang dibuat dengan AI

(Artificial Intelligence) dapat diterapkan dengan menggunakan algoritma A* (A Star). Namun algoritma A* membutuhkan komputasi yang lama dan hanya mendukung lingkungan statis. Penggunaan *physics engine* dalam permainan komputer merupakan salah satu penyebab adanya halangan dinamis (*dynamic obstacle*) dalam peta permainan sehingga menyebabkan algoritma *pathfinding* tradisional seperti A* tidak mampu bekerja dalam lingkungan dinamis (*dynamic environment*). Penyelesaian masalah *pathfinding* dalam lingkungan dinamis (*realtime pathfinding*) dapat dilakukan dengan menggunakan *realtime heuristic* dan algoritma pembelajaran (*learning algortm*). Objektif dalam *pathfinding* dapat dibagi menjadi dua yaitu, mencari path dari posisi awal ke tujuan akhir dan menghindari halangan-halangan saat proses pencarian rute tersebut (*obstacles*). Salah satu algoritma pembelajaran yang dapat diterapkan untuk menyelesaikan permasalahan *realtime pathfinding* adalah dengan menggunakan metode *minimal memory abstraction*.

Minimal memory abstraction melakukan abstraksi peta permainan dengan membagi peta permainan ke dalam beberapa klaster yang saling berhubungan (*linked local clusters*) dan kemudian membangun *high-level plan* dan secara bertahap membangun *low-level plan*.

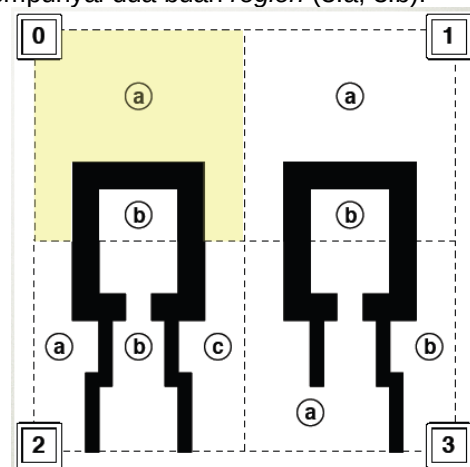
Landasan Teori

Minimal Memory Abstraction

Algoritma A* tidak efisien jika diterapkan dalam peta permainan beresolusi tinggi karena membutuhkan memori yang banyak dan waktu komputasi yang lama. *Automatic state abstractions* dapat mengubah peta permainan beresolusi tinggi menjadi peta permainan yang beresolusi rendah dan abstrak yang dapat digunakan untuk mempercepat proses *pathfinding*. Konsep abstraksi peta permainan dapat dianalogikan dengan cara abstraksi manusia dalam menentukan rute perjalanan dari kota satu ke kota lainnya. Dalam menentukan rute perjalanan, rencana yang dibuat pertama kali adalah rencana abstrak terlebih dahulu yaitu dengan menentukan kota mana yang akan dilewati. Baru kemudian akan direncanakan lebih detail jalan yang akan dilewati antara kota satu dengan kota lainnya sampai menuju tempat tujuan. Ada beberapa tahapan yang dilakukan dalam *minimal memory abstraction* yaitu (1) komputasi sektor (2) komputasi tepi (3) dan *pathfinding* menggunakan abstraksi.

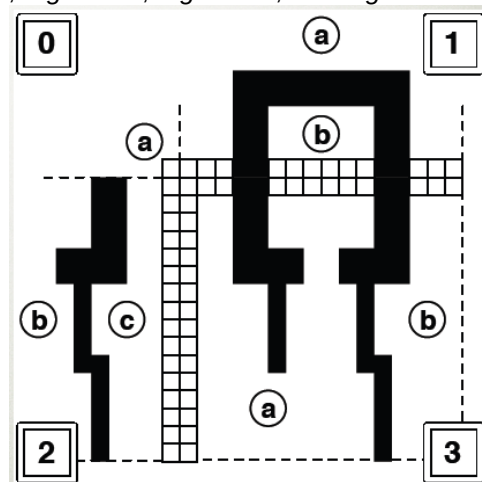
Komputasi sektor dilakukan dengan membagi peta permainan yang mempunyai representasi

ruang pencarian berupa grid menjadi beberapa sektor dalam ukuran yang tetap dan mempunyai indeks. Kemudian dilanjutkan dengan membagi setiap sektor tersebut menjadi beberapa region yang saling terhubung dan mempunyai titik tengah. Gambar 1 menjelaskan proses komputasi sektor. Dalam Gambar 1, wilayah yang berwarna hitam merupakan halangan atau wilayah yang tidak dapat dilewati. Peta sederhana yang ditunjukkan dalam Gambar 2.2 mempunyai empat sektor (0, 1, 2, 3), sektor 0 mempunyai dua buah *region* (0.a, 0.b), sektor 1 mempunyai dua buah *region* (1.a, 1.b), sektor 2 mempunyai tiga buah *region* (2.a, 2.b, 2.c), dan sektor 3 mempunyai dua buah *region* (3.a, 3.b).



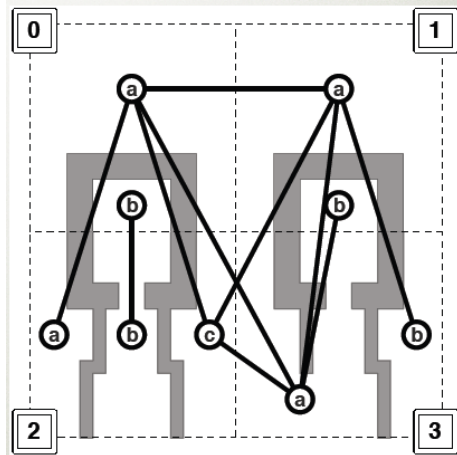
Gambar 1. Komputasi Sektor

Komputasi tepi dilakukan dengan menghubungkan *region-region* yang saling berdekatan (atau berhadapan) yang ada dalam sektor yang berbeda. Dalam Gambar 2, *region* 3.a mempunyai empat buah tepi ke arah *region* 2.c, *region* 0.a, *region* 1.a, dan *region* 1.b.



Gambar 2. Komputasi Tepi

Abstract graph setelah dilakukan komputasi tepi ditunjukkan dalam Gambar 2.4. Dalam *abstract graph* hanya terdiri dari 9 node dan 10 tepi. Jumlah *node* dalam *abstract graph* tersebut jauh lebih kecil daripada jumlah *node* dalam peta permainan yang asli yang mempunyai sekitar 1024 *node* dan ribuan tepi.



Gambar 3. Final Abstract Graph

Untuk mempercepat proses *pathfinding* dapat dilakukan tahap *pre-processing* dengan melakukan abstraksi peta permainan menggunakan *minimal memory abstraction*. *Minimal memory abstraction* merupakan teknik abstraksi yang membutuhkan sedikit memori penyimpanan hasil abstraksi. Proses abstraksi ini dapat dilakukan secara *offline*.

Setelah hasil abstraksi diperoleh maka tahap selanjutnya adalah melakukan proses *pathfinding* menggunakan abstraksi untuk memperoleh *abstract path* dan *real path*. *Real path* yang dihasilkan akan digunakan sebagai penunjuk jalan *neural agent* dalam menelusuri peta permainan sampai ke tujuan. Berikut merupakan *pseudo code pathfinding* menggunakan abstraksi :

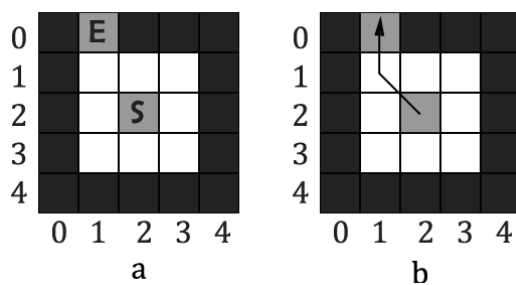
1. Cari *region* terdekat dengan posisi awal dengan Breadth First Search
2. Cari *region* terdekat dengan posisi tujuan dengan Breadth First Search
3. Cari *abstract path* menggunakan A* dari posisi awal sampai ke posisi tujuan (sektor dan *region* mana yang akan dilewati).

Algoritma A*

Algoritma A* (Hart, Nilsson, & Raphael, 1968) digunakan untuk menemukan sebuah *path* terpendek diantara dua titik dalam sebuah peta permainan. Dalam algoritma A*, peta permainan terdiri dari *node-node*. Setiap *node-node* tersebut memiliki tiga atribut yaitu *f*, *g*,

dan *h* yang seringkali disebut berurutan sebagai nilai *fitness*, *goal* dan *heuristik*. Nilai *goal* atau *g* adalah biaya dari *node* awal sampai ke *node* yang ditelusuri sekarang (*current node*). Nilai *heuristik* atau *h* adalah biaya estimasi dari *current node* sampai ke *node* tujuan. Nilai *fitness* atau *f* adalah penjumlahan dari nilai *g* dan *h*. Semakin kecil nilai *f* semakin bagus *path* yang dihasilkan. *Node-node* dalam A* disimpan dalam *open list* dan *closed list*. *Open list* berisi *node-node* yang belum dijelajahi sedangkan *closed list* berisi *node-node* yang telah dijelajahi. Sebuah *node* dikatakan telah dijelajahi jika algoritma A* telah memeriksa semua *node* yang terhubung, menghitung nilai *f*, *g*, dan *h*, serta menyimpannya dalam *open list* untuk dijelajahi kemudian. *Open list* dan *closed list* dibutuhkan untuk menyimpan jejak penelusuran *node-node* tersebut. Berikut merupakan *pseudo-code* algoritma A* (Matthews, 2002) :

1. Misal P sebagai titik awal.
2. Hitung nilai *f*, *g*, dan *h* dari P
3. Tambahkan P ke dalam *open list*.
4. Misal B merupakan *node* terbaik (*node* yang mempunyai nilai *f* terkecil) yang ada dalam *open list*.
 - a. Jika B merupakan *node* tujuan maka penelusuran dihentikan dan sebuah *path* ditemukan.
 - b. Jika isi *open list* kosong maka penelusuran dihentikan dan *path* tidak berhasil ditemukan.
5. Misal C adalah *node* valid yang terhubung ke *node* B.
 1. Hitung nilai *f*, *g*, dan *h* dari *node* C.
 2. Periksa apakah C ada dalam *open list* atau *closed list*.
 - i. Jika berada dalam *open list*, maka periksa apakah sebuah *path* baru lebih efisien (atau memiliki nilai *f* yang lebih kecil).
 1. Jika ya, maka perbarui *path*
 - ii. Jika berada dalam *closed list*, maka tambahkan *node* C ke dalam *open list*.
 3. Ulangi langkah ke-5 untuk semua *node* anak dari B.
6. Ulangi langkah 4



Gambar 4. a) Peta permainan sederhana
b) Solusi path

Contoh sederhana berikut akan memperjelas *pseudo-code* algoritma A* diatas. Sebagai contoh, misal dalam peta permainan yang ditunjukkan dalam Gambar 2.1 akan ditentukan solusi path dari node S(2,2) ke node E(1,0). Penelusuran diawali dari node S dan kemudian dilakukan penghitungan nilai f, g, dan h. Nilai g sama dengan nol dikarenakan node yang ditelusuri hanya node S. Nilai h dapat dihitung menggunakan manhattan distance. Misal (dx,dy) merupakan node tujuan dan (sx, sy) merupakan node awal maka nilai h dapat dihitung menggunakan *manhattan distance* dengan persamaan berikut,

$$h = |dx - sx| + |dy - sy|$$

Jadi nilai h dari node yang ditelusuri sekarang S(2,2) ke node E(1,0) sama dengan 3 dan nilai f sama dengan 3. Setelah nilai f, g, dan h dari node S selesai dihitung maka node yang ditelusuri sekarang (node S) akan dimasukkan ke dalam open list. Kemudian semua node anak yang berdekatan dengan node yang ditelusuri sekarang (node S) akan dimasukkan ke dalam open list. Nilai g untuk masing-masing node anak tersebut sama dengan 1. Sedangkan nilai h masing-masing node anak berbeda. Nilai h paling kecil adalah nilai node pada posisi (1,1) dan nilai node tersebut akan ditelusuri selanjutnya.

Node (1,1) mempunyai empat buah node anak : (1,0), (1,2), (2,1), dan (2,2). Node (2,2) ada dalam closed list, sedangkan node (1,0), (1,2), (2,1) ada dalam open list. Kemudian setelah dihitung nilai f, g, h dari masing-masing node anak didapatkan node (1,0) mempunyai nilai terbaik dan merupakan node tujuan.

METODE PENELITIAN

Analisis Kebutuhan Non Fungsional

Analisis kebutuhan non-fungsional merupakan analisis yang dibutuhkan untuk menentukan spesifikasi kebutuhan sistem. Spesifikasi ini juga meliputi elemen atau komponen-komponen apa saja yang dibutuhkan untuk sistem yang akan dibangun sampai dengan sistem tersebut diimplementasikan.

Pada analisis kebutuhan non-fungsional ini dijelaskan analisis mengenai analisis kebutuhan perangkat keras (*hardware*), perangkat lunak (*software*), dan pengguna (*user*) sebagai bahan analisis kekurangan dan kebutuhan yang harus dipenuhi dalam perancangan sistem yang akan diterapkan.

1. Analisis kebutuhan perangkat keras.

Agar aplikasi dapat berjalan dengan baik, maka dibutuhkan perangkat keras sesuai dengan kebutuhan aplikasi. Spesifikasi minimum perangkat keras yang dibutuhkan agar dapat menjalankan game *realtime pathfinding angry dog* sebagai berikut :

Processor : Intel(R) Core(TM) Duo CPU
T5750 @ 2.00GHz

Memory : 1.00 GB

Resolusi : 1280x800

2. Analisis kebutuhan perangkat keras yang dibutuhkan agar dapat menjalankan game *realtime pathfinding angry dog* sebagai berikut :

OS : Windows XP

Editor IDE : NetBeans versi 6.5

UML Modeler : Rational Rose Enterprise Edition

Image Editor : Corel Draw X3, Adobe Photoshop CS3

3. Analisis pengguna (*user*)

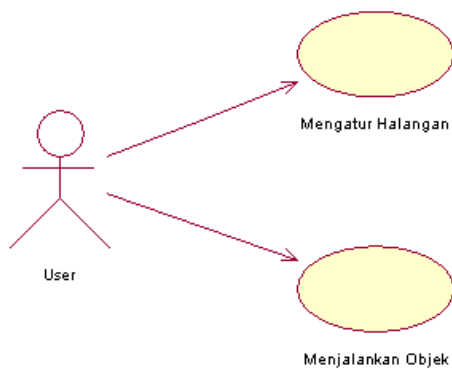
Pengguna yaitu *public* yang dapat mengerti dan memahami komputer sehingga dapat menggunakan aplikasi yang akan dibangun dan mengerti cara memainkan permainan.

Analisis Kebutuhan Fungsional

Pemodelan yang digunakan pada aplikasi yang akan dibuat adalah analisis perancangan dan pengembangan lunak berbasis objek, dimana alat yang digunakan untuk pengembangan perangkat lunak, yaitu menggunakan pemodelan UML (*Unified Modelling Language*).

Use Case Diagram

Sebuah use case merepresentasikan sebuah interaksi antara *user* dengan sistem, serta merepresentasikan proses *realtime pathfinding* dengan metode yang dipakai dalam permainan *Angry Dog* ini yaitu metode minimal *memory abstraction*. Pada use case, user digambarkan secara keseluruhan antara interaksi yang dilakukan user dalam menjalankan permainan.

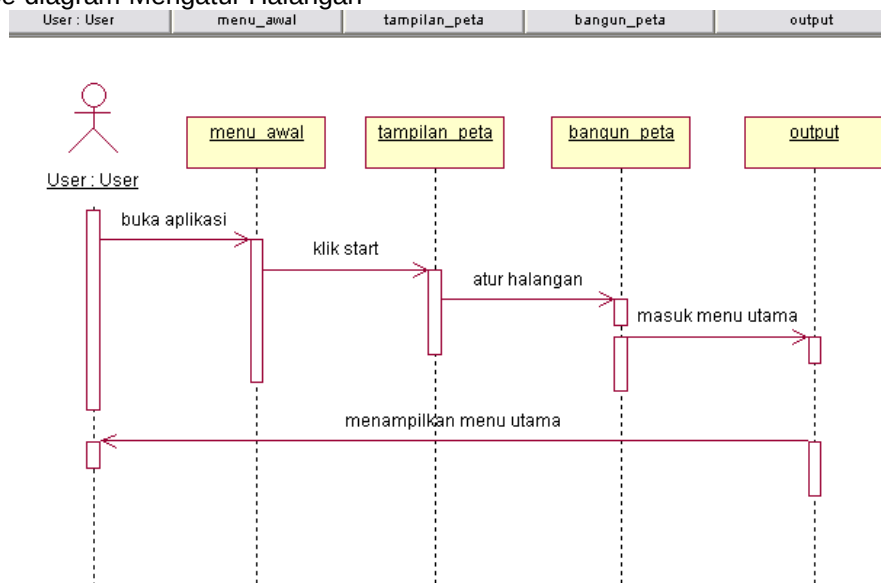


Gambar 5. Use Case Diagram

Sequence Diagram

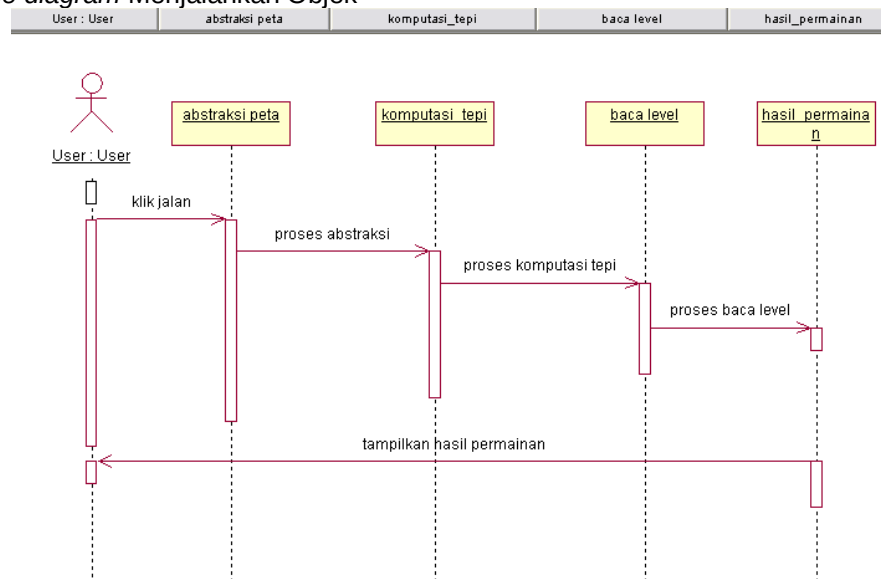
Aksi dari sebuah objek yang mengirimkan informasi disebut sebagai event. Untuk lebih memahami tentang event-event yang terjadi pada suatu sistem, bisa memakai sequence diagram. Di dalam UML, *sequence diagram* digunakan untuk menentukan fungsi yang kelak dapat diterapkan pada suatu kelas/objek. Dengan *sequence diagram* kita bisa melakukan pemetaan aliran event antar kelas/objek. *Sequence diagram* terdiri atas dimensi vertikal (waktu) dan dimensi horisontal (objek-objek terkait).

1) Sequence diagram Mengatur Halangan



Gambar 6. Sequence Diagram Mengatur Halangan

2) Sequence diagram Menjalankan Objek

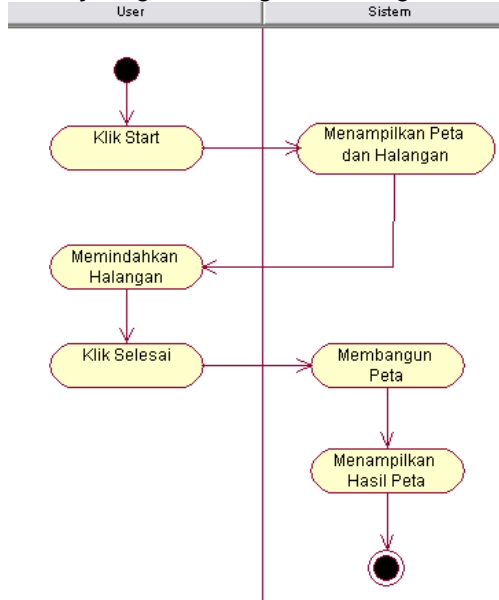


Gambar 7. Sequence Diagram Menjalankan Objek

Activity Diagram

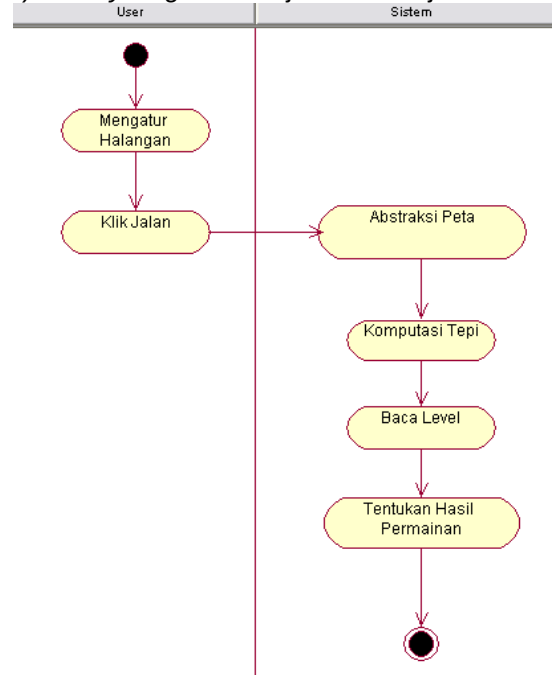
Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang akan dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi, dan bagaimana proses berakhir dalam game realtime *pathfinding Angry Dog* menggunakan *minimal memory abstraction* ini.

1) Activity diagram Mengatur Halangan

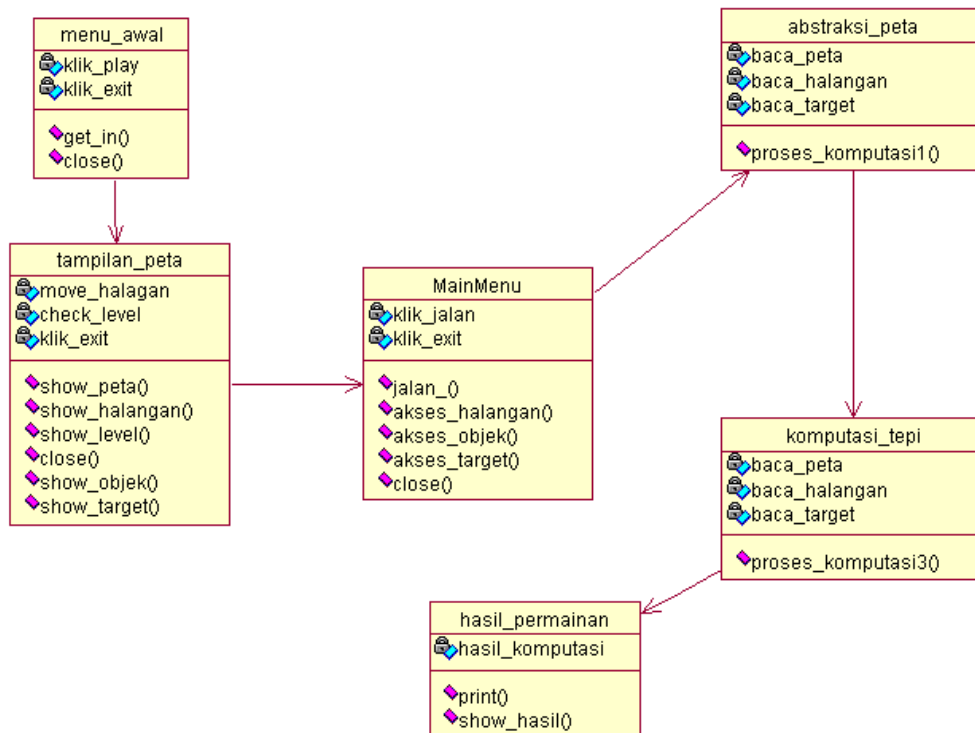


Gambar 8. Activity diagram Mengatur Halangan

2) Activity diagram Menjalankan Objek



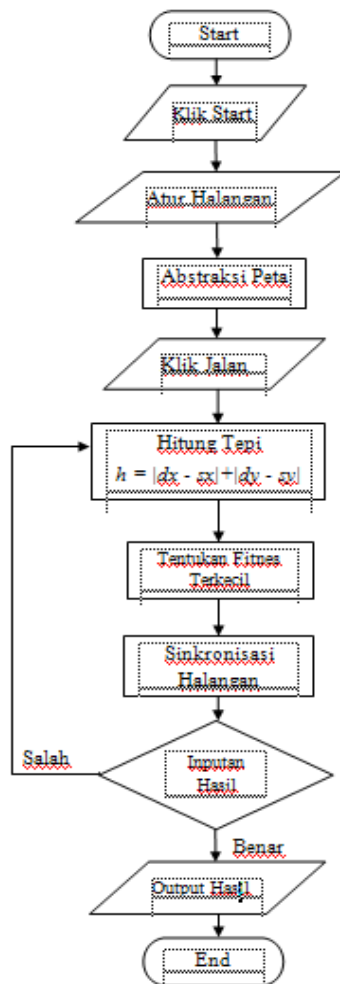
Gambar 9. Activity diagram Menjalankan Objek



Gambar 10. Class Diagram

Class Diagram

Selanjutnya, kita bisa membuat class diagram dengan atribut-atribut yang bisa didapatkan dari skenario sebelumnya. *Class* didefinisikan sebagai kumpulan atau himpunan objek yang memiliki kesamaan dalam atribut/properti, perilaku (*operasi*), serta cara berhubungan dengan orang lain. *Class* juga berfungsi untuk membungkus (*encapsulating*) objek-objek. Class diagram adalah sebuah diagram yang menggambarkan struktur dan deskripsi *class*, package dan objek beserta hubungan satu sama lain seperti hubungan dinamis, pewarisan, asosiasi, dan agregasi. Dibawah ini adalah class diagram dari sistem game *realtime pathfinding Angry Dog* menggunakan *minimal memory abstraction*.



Gambar 11. Flowchart Alur Sistem

Flow Chart

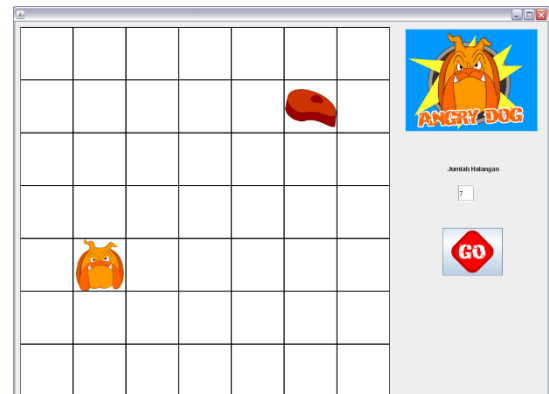
Flowchart adalah tipe diagram yang merepresentasikan algoritma atau proses, menampilkan tahapan-tahapan dalam bentuk kotak, oval dan bentuk lain yang dihubungkan

dengan tanda panah. Diagram tersebut menunjukkan tahap demi tahap penyelesaian sebuah permasalahan. Operasi proses ditunjukkan oleh bentuk kotak dan tanda panah merepresentasikan aliran dari sebuah kontrol. *Flowchart* digunakan dalam proses analisis, desain, dokumentasi atau mengatur proses dalam sebuah program pada bermacam-macam *field*. Berikut ini adalah diagram *flowchart* dari game *realtime pathfinding Angry Dog* menggunakan *minimal memory abstraction* :

HASIL DAN PEMBAHASAN

Implementasi Antar Muka

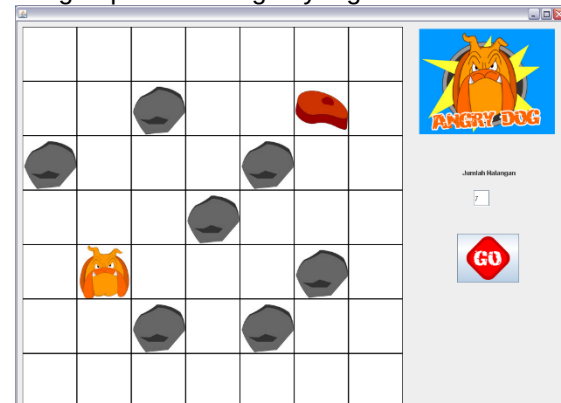
Implementasi antarmuka permainan yang dibangun dapat dilihat pada gambar dibawah ini.



Gambar 11. Tampilan Menu Angry Dog

Pengujian

Berikut adalah contoh kondisi permainan dengan posisi halangan yang sudah diatur:



Gambar 12. Tampilan Kondisi Kalangan

Gambar di atas menunjukkan setelah user selesai mengatur halangan, user dapat menekan tombol go untuk menjalankan objek, dan kemudian sistem akan merespon. Pergerakan objek ditunjukkan oleh gambar jejak warna merah. Berikut adalah perhitungan nilai *fitness* sesuai dengan gambar di atas:

Rumus menghitung *fitness* terkecil :

$$h = |dx - sx| + |dy - sy|$$

Keterangan:

h : nilai *fitness*

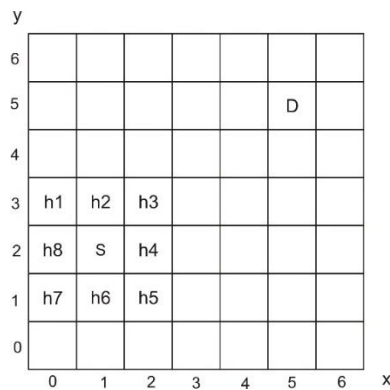
dx : destination x atau nilai x target

sx : start x atau nilai x objek

dy : destination y atau nilai y target

sy : start y atau nilai y target

Untuk memudahkan membaca nilai x dan y pada peta, gambar di bawah adalah peta manual dalam penentuan nilai array atau nilai x dan y pada peta Permainan Game *Angry Dog*.



Gambar 13. Peta manual 1

Langkah pertama yang dilakukan adalah menghitung $h1$ sampai dengan $h8$ dengan rumus yang sudah tersedia di atas, seperti berikut:

$$h1 = |dx - sx| + |dy - sy| = |5-0| + |5-3| = 5+2 = 7$$

$$h2 = |dx - sx| + |dy - sy| = |5-1| + |5-3| = 4+2 = 6$$

$$h3 = |dx - sx| + |dy - sy| = |5-2| + |5-3| = 3+2 = 5$$

$$h4 = |dx - sx| + |dy - sy| = |5-2| + |5-2| = 3+3 = 6$$

$$h5 = |dx - sx| + |dy - sy| = |5-2| + |5-1| = 3+4 = 7$$

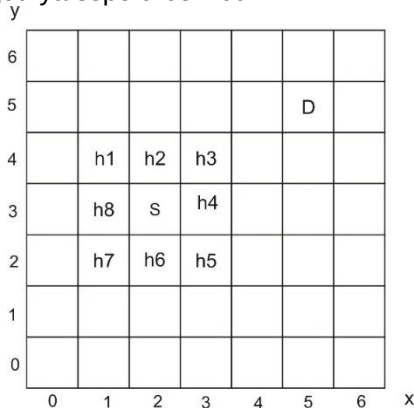
$$h6 = |dx - sx| + |dy - sy| = |5-1| + |5-1| = 4+4 = 8$$

$h7$ tidak dihitung karena terdapat halangan

$$h8 = |dx - sx| + |dy - sy| = |5-0| + |5-2| = 5+3 = 8$$

Hasil telah didapat yaitu nilai *fitness* terkecil pada kotak $h3$ atau kotak dengan array $[2][3]$.

Setelah objek berpindah ke kotak $[2][3]$, maka akan dilakukan perhitungan nilai *fitness* selanjutnya seperti berikut:



Gambar 14. Peta manual 2

$$h1 = |dx - sx| + |dy - sy| = |5-1| + |5-4| = 4+1 = 5$$

$$h2 = |dx - sx| + |dy - sy| = |5-2| + |5-4| = 3+1 = 4$$

$$h3 = |dx - sx| + |dy - sy| = |5-3| + |5-4| = 2+1 = 3$$

$h4$ tidak dihitung karena terdapat halangan

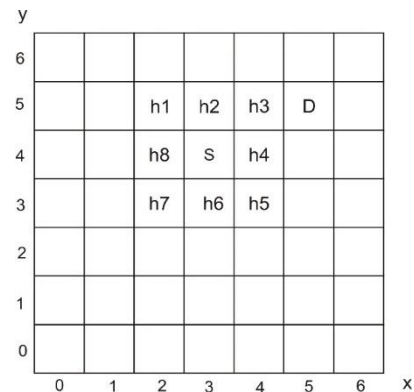
$$h5 = |dx - sx| + |dy - sy| = |5-3| + |5-2| = 2+3 = 5$$

$$h6 = |dx - sx| + |dy - sy| = |5-2| + |5-2| = 3+3 = 6$$

$$h7 = |dx - sx| + |dy - sy| = |5-1| + |5-2| = 4+3 = 7$$

$$h8 = |dx - sx| + |dy - sy| = |5-1| + |5-3| = 4+2 = 6$$

Hasil telah didapat yaitu nilai *fitness* terkecil pada kotak $h3$ atau kotak dengan array $[3][4]$. Setelah objek berpindah ke kotak $[3][4]$, maka akan dilakukan perhitungan nilai *fitness* selanjutnya seperti berikut:



Gambar 15. Peta manual 3

$h1$ tidak dihitung karena terdapat halangan

$$h2 = |dx - sx| + |dy - sy| = |5-3| + |5-5| = 2+0 = 2$$

$$h3 = |dx - sx| + |dy - sy| = |5-4| + |5-5| = 1+0 = 1$$

$$h4 = |dx - sx| + |dy - sy| = |5-4| + |5-4| = 1+1 = 2$$

$$h5 = |dx - sx| + |dy - sy| = |5-4| + |5-3| = 1+2 = 3$$

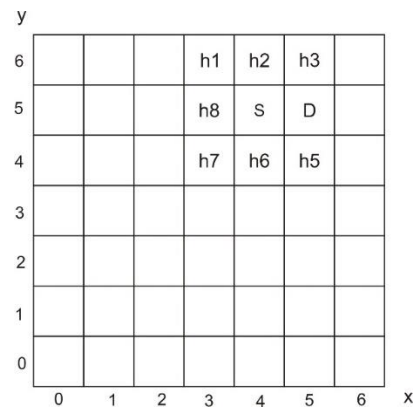
$$h6 = |dx - sx| + |dy - sy| = |5-3| + |5-3| = 2+2 = 4$$

$$h7 = |dx - sx| + |dy - sy| = |5-2| + |5-3| = 3+2 = 5$$

$$h8 = |dx - sx| + |dy - sy| = |5-2| + |5-4| = 3+1 = 4$$

Hasil telah didapat yaitu nilai *fitness* terkecil pada kotak $h3$ atau kotak dengan array $[4][5]$.

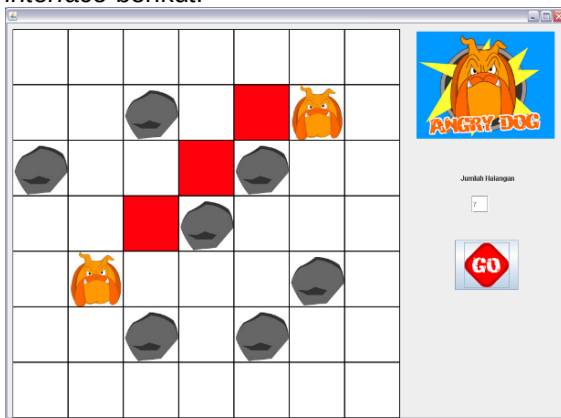
Setelah objek berpindah ke kotak $[4][5]$, maka akan dilakukan perhitungan nilai *fitness* selanjutnya seperti berikut:



Gambar 16. Peta manual 4

$h1 = |dx - sx| + |dy - sy| = |5-3| + |5-6| = 2+1 = 3$
 $h2 = |dx - sx| + |dy - sy| = |5-4| + |5-6| = 1+1 = 2$
 $h3 = |dx - sx| + |dy - sy| = |5-4| + |5-5| = 1+0 = 1$
 $h4 = D$
 $h5$ tidak dihitung
 $h6$ tidak dihitung
 $h7$ tidak dihitung
 $h8$ tidak dihitung

Hasil telah didapat yaitu kotak yang mempunyai status D atau target. Ketika sistem sudah mendapatkan posisi D atau target, maka sistem akan berhenti tanpa melanjutkan menghitung nilai h kotak yang lain. Dan hasil dari perhitungan di atas dapat dilihat pada gambar *interface* berikut.



Gambar 17. Interface Hasil Proses *Pathfinding*

KESIMPULAN DAN SARAN

Kesimpulan

Dari hasil uji coba yang telah dilakukan oleh penulis pada akhir perancangan aplikasi tersebut dapat dirumuskan beberapa kesimpulan:

1. Permasalahan *realtime pathfinding* pada game Angry Dog dapat diselesaikan dengan menggunakan metode minimal memory abstraction.
2. Hasil perhitungan pada program sesuai dan mempunyai nilai yang sama dengan perhitungan manual seperti yang telah ditunjukkan pada tahap pengujian.
3. Ketika program melakukan perhitungan nilai fitness dan kemudian sistem menemukan kotak yang mempunyai nilai fitness terkecil yang sama, penentuan pengambilan keputusan ditentukan dari urutan kotak mana dulu yang dihitung.

Saran

Adapun saran agar dapat meningkatkan aplikasi ini dimasa mendatang adalah:

1. Membuat tampilan permainan yang lebih menarik dan interaktif.

2. Mengaplikasikan metode minimal memory abstraction pada peta yang lebih besar atau dengan jumlah grid yang lebih banyak.
3. Mencari solusi lain dalam melakukan penentuan pengambilan keputusan jika menemukan dua kotak dengan nilai fitness yang sama.

REFERENSI

- Abdul Kadir.2008. Dasar Pemrograman WEB Dinamis Menggunakan PHP.Yogyakarta: Andi Offset.
- Jang JSR, Sun CT and Mizutani E. 1997. Neuro-Fuzzy And Soft Computing A Computational Approach to Learning and Machine Intelligence. New Jersey: Prentice-Hall.
- Kurnianti A.I., Saptono Fajar., & Hidayat Taufik. 2007. Sistem Pendukung Keputusan Penanganan Kesehatan Balita Menggunakan penalaran Fuzzy.
- Kusumadewi, S.2002. Analisis & Desain Sistem Fuzzy Menggunakan Tool Box MathLab. Graha Ilmu, Yogyakarta.
- Mamdani.Universitas Islam Indonesia. Susilo, F.2003.Pengantar Himpunan & Logika Kabur serta Aplikasinya. Yogyakarta : Graha Ilmu.
- Mohammad Glesung G.2010.Penentuan Jurusan di SMAN Surakarta dengan Fuzzy Inference Sistem FIS
- Sri Kusuma Dewi & Hari Purnomo.2004. Aplikasi Logika Fuzzy untuk Pendukung Keputusan.Gava Media.
- Synaptic. 2006. Fuzzy Math, Part 1, The Theory.http://www.scholarpedia.org/article/Fuzzy_Logic. juli 2012.
- Vrusias, B.L.2005. Fuzzy <http://www.2dix.com/ppt/fuzzy.php>. Juni 2012.
- Wahyudi.2005. Implementasi Fuzzy Logic Controller pada Sistem Pengeraman Kereta Api, Transmisi, Vol. 10, No.2, Desember 2005, 10-3.
- Yuyut Ika Indrawati.2010. Penentuan Nillai produktifitas Perusahaan dengan Metode Logika Fuzzy Mamdani.Universitas Muhammadiyah Malang.
- Zimmermann, H-J.1991. Fuzzy Set Theory and Its Application. Kluwer Academic Publisher, Dordrect.