# IMPLEMENTASI ALGORITMA *JUMP POINT SEARCH* PADA NPC MUSUH UNTUK MENGEJAR PEMAIN DALAM GAME LABIRIN

Ilham Ramadhan

Program Studi Teknik Informatika
Fakultas Teknik dan Ilmu Komputer Universitas Komputer Indonesia
Jl. Dipati Ukur 114 Bandung
E-mail : ilhamyudo27@gmail.com

## ABSTRAK

Salah satu algoritma *pathfinding* yang dapat dipakai dalam mencari dan mengenali jalur dari satu titik ke titik lain adalah algoritma *Jump Point Search* (JPS). Algoritma JPS merupakan pengembangan dari algoritma A* yang dikembangkan oleh Daniel Harabor dan Alban Grastien pada tahun 2011.Penelitian ini bertujuan untuk mengukur performansi algoritma *Jump Point Search* dalam mencari jalur dengan halangan yang bersifat dinamis. Penelitian pada algoritma *Jump Point Search* akan diterapkan pada sebuah *game* labirin.

Karakteristik *game* yang akan dibangun didasarkan pada sebuah *game* yaitu *game bomberman*, dimana *game* tersebut memiliki beberapa fitur yang dapat digunakan pada penelitian ini. Salah satu fitur tersebut adalah penggunaan sebuah halangan untuk menghalangi musuh dalam mengejar pemain.

Hasil penelitian menunjukkan bahwa, pada uji coba algoritma dengan program simulasi didapatkan algoritma ini dapat mengenali jalur yang berubah secara dinamis, hal ini ditunjukkan dari tingkat persentase keberhasilan menemukan jalur di atas 90%. Dengan demikian algoritma ini layak digunakan pada kasus *game* labirin.

**Kata kunci:** Kecerdasan buatan, *game*, labirin, algoritma *pathfinding*

## 1. PENDAHULUAN

*Game* bertemakan labirin adalah sebuah permainan yang mengharuskan pemain untuk mencari jalan sendiri dengan rintangan berupa jalur yang rumit, berliku-liku, serta memiliki banyak jalan buntu. Pada beberapa permainan labirin juga ditambahkan NPC atau *Non-playable Character* sebagai musuh yang akan mengejar pemain, contohnya ada pada permainan *bomberman*. NPC musuh pada permainan *bomberman* tersebut diharuskan mengenali keadaan sekitarnya, seperti adanya tembok, halangan yang ada atau menemukan pemain. Dalam kasus tersebut, maka NPC musuh menerapkan suatu algoritma dalam menemukan pemain, yaitu algoritma pathfinding. Pada permainan *bomberman*, algoritma *pathfinding* yang diterapkan untuk musuh dalam pengejaran pemain adalah algoritma HPA*(*star*) [1]. Namun dalam pengimplementasiannya, algoritma ini tidak berjalan dengan sempurna, dikarenakan jika jalan yang ditemukan tertutup oleh penghalang yang dinamis dan posisi *start* dan *goal* terdapat pada *cluster* yang berbeda maka musuh akan berhenti bergerak hingga jalan tersebut terbuka.

Salah satu algoritma *pathfinding* yang dapat dipakai dalam mencari dan mengenali jalur ini adalah algoritma *Jump Point Search* (JPS). Algoritma JPS merupakan pengembangan dari algoritma A* yang dikembangkan oleh Daniel Harabor dan Alban Grastien pada tahun 2011. Algoritma ini dikatakan adalah algoritma yang kompetitif dan lebih cepat dari HPA*, sederhana tetapi efektif, mempertahankan optimalitas tetapi tidak memerlukan memori tambahan, dan sangat cepat tetapi tidak memerlukan *preprocessing* [2]. Penerapan algoritma JPS dalam game labirin diterapkan untuk mengenali keadaan sekitar dari NPC sehingga dapat mencari jalur terbaik dalam mengejar pemain. Algoritma JPS dilakukan untuk mempercepat proses pencarian jalur dan mengurangi penggunaan memori untuk *open* dan *closed list* dalam menyimpan *nodes* pada algoritma A*. Dalam JPS penyimpanan *nodes* akan dipangkas menjadi hanya beberapa yang disebut sebagai *jump point*.

Oleh karena itu dalam penelitian ini algoritma *Jump Point Search* akan digunakan untuk diimplementasikan dalam NPC sebagai suatu AI dalam mengenali keadaan sekitar sehingga dapat menentukan jalur terpendek yang dapat dilalui dalam mengejar musuh. Algoritma ini dipilih dikarenakan melihat karakteristik permasalahan pada NPC musuh game labirin sama dengan permasalahan pada pathfinding.

## 2. MODEL ANALISA DAN DESAIN IMPLEMENTASI

Untuk melihat bagaimana cara kerja algoritma *Jump Point Search* dalam *game* labirin maka berikut ini adalah analisis dari *game* labirin, *artificial intelligence* dan algoritma *Jump Point Search* itu sendiri.

### 2.1. *Game* Labirin

*Game* merupakan suatu program yang dirancang sedemikian rupa dimana pemainnya berinteraksi dengan konflik buatan yang memiliki aturan dan memiliki hasil keluaran [3]. *Game* memiliki beragam jenis tipe, salah satu tipe game ini adalah *game* labirin. *Game* labirin biasanya merupakan permainan yang mengharuskan pemain dalam mencari jalan keluar ada beberapa *game labirin* yang mengharuskan pemain menyelesaikan suatu tujuan dalam labirin tersebut. Labirin memiliki ciri dimana jalan yang dilalui memiliki jalur yang rumit, berliku-liku dan memiliki banyak jalan buntu.Contoh *game* labirin yang masih dikenal sampai saat ini adalah *game bomberman*.




**Gambar 1 *Game Bomberman***

### 2.2. *Artificial Intelligence*

*Artificial Intelligence* sering diterjemahkan sebagai kecerdasan buatan atau,kecerdasan artifisial. Definisi dari AI itu sendiri berbeda-beda tergantung dari sudut pandang dari para ahli masing-masing. Menurut Sarbjeet Singh dan Sukhvinder Singh [4], AI adalah bidang studi yang berdasarkan bahwa pemikiran yang cerdas dapat dianggap sebagai sebuah bentuk komputasi. Mereka mengatakan bahwa ada dua isu penting yang harus diperhatikan yaitu:

1. *Knowledge representation*

Sudah lama diketahui bahwa bahasa dan model yang digunakan untuk merepresentasikan realitas memiliki dampak yang mendalam bagi pemahaman seseorang mengenai realitas itu sendiri. Ketika manusia berpikir tentang suatu sistem tertentu, mereka membentuk sebuah mental model dari sistem tersebut dan membuat mereka mengetahui kenyataan dari sistem tersebut. Kenyataan ini menyebabkan mereka dapat membuat sebuah prediksi atau kesimpulan dari suatu sistem. Tetapi saat suatu model tidak cukup cocok dengan masalah yang asli, maka penemuan kenyataan dan kemampuan memprediksi menjadi sangat sulit.

Dalam *Artificial Intelligence*, upaya yang signifikan telah dilakukan dalam pengembangan bahasa yang digunakan untuk merepresentasikan pengetahuan dengan tepat. Dengan AI, para insinyur dan ilmuwan komputer mampu membuat sebuah mesin yang dapat mengganti manusia dalam menjalankan pekerjaan yang bahaya. Contohnya, robot polisi yang dapat menangani bom aktif. Dengan AI juga, mereka dapat memprediksikan kemungkinan-kemungkinan yang dapat terjadi pada suatu sistem, seperti perkiraan sampainya pesawat luar angkasa ke bulan.

2. *Knowledge manipulation*

Banyak masalah yang dialami oleh manusia tidak sepenuhnya dapat dipahami. Pemahaman yang setengah dicerminkan pada fakta bahwa solusi algoritmik yang kaku tidak dapat diterapkan. Sebaliknya, konsep pada pencarian digunakan untuk memecahkan masalah tersebut. Ketika pencarian digunakan untuk menjelajah setiap ruang solusi, dapat dikatakan itu sangat melelahkan. Karena itu, jika seseorang ingin mencari suatu solusi dalam suatu masalah, maka harus selektif menjelajahi ruang pencarian masalah**.**

Salah satu contohnya adalah pencarian heuristik. Dengan pencarian heuristik, hasil pencarian dari pengetahuan masalah-khusus didapatkan dari pencarian universal. Keberhasilan heuristik telah menyebabkan fokus penerapan teknik AI umum digunakan untuk domain masalah tertentu. Ini menyebabkan pengembangan sistem pakar yang mampu memberikan penalaran yang lebih baik dalam domain sempit yang dapat didefinisikan dalam bidang seperti kedokteran, matematika, kimia, robotika, dan penerbangan.

Teknik pemecahan masalah yang ada didalam AI dibagi menjadi empat, yaitu [5]:

*a. Searching*

*Searching* merupakan teknik menyelesaikan masalah dengan cara merepresentasikan masalah ke dalam state dan ruang masalah serta menggunakan strategi pencarian untuk menemukan solusi.

*b. Reasoning*

*Reasoning* merupakan teknik menyelesaikan masalah dengan cara merepresentasikan masalah ke dalam basis pengetahuan (knowledge base) menggunakan logic atau bahasa formal.

*c. Planning*

*Planning* merupakan teknik menyelesaikan masalah dengan cara memecah masalah kedalam

sub-sub masalah dan menyelesaikan sub-sub masalah satu demi satu.

*d. Learning*

*Learning* merupakan teknik menyelesaikan masalah dengan mengotomatisasi mesin untuk menemukan aturan yang diharapkan bisa berlaku umum untuk data-data yang belum pernah diketahui.

Pemilihan teknik yang digunakan dalam penyelesaian masalah tergantung dari karakteristik permasalahan yang akan diselesaikan. Contohnya implementasi kecerdasan buatan pada game yang sering diterapkan yaitu pada pathfinding. Penggunaan pathfinding yang paling sering digunakan pada game adalah untuk mempengaruhi pergerakan dan pengambilan keputusan yang diterapkan pada *Non-Playable Character* atau disingkat NPC. Sedangkan teknik untuk pemecahan masalah pathfinding dalam game menggunakan teknik pencarian atau searching.

### 2.3. Algoritma *Jump Point Search*

*Pathfinding* dalam lingkungan *uniform-cost grid* adalah masalah yang sering kali ditemui pada bidang robotika dan *video games*. Terdapat beberapa algoritma yang dikhususkan untuk *pathfinding*, contohnya ada pada algoritma HPA*. Tetapi algoritma ini tidak dapat bergerak menuju tujuan jika jalur yang sudah didapat ditutup oleh halangan yang dinamis serta *start* dan *goal* berada pada *cluster* yang berbeda [1].

Algoritma *Jump Point Search* atau dapat disingkat algoritma JPS merupakan salah satu algoritma *pathfinding* yang ada. Algoritma ini dipublikasikan pada tahun 2011 oleh D. Harabor dan A. Grastien [2]. Algoritma ini merupakan salah satu pengembangan yang dilakukan terhadap algoritma A* yang bertujuan untuk mengurangi penggunaan memori dan membuat pencarian jalur menjadi lebih cepat.

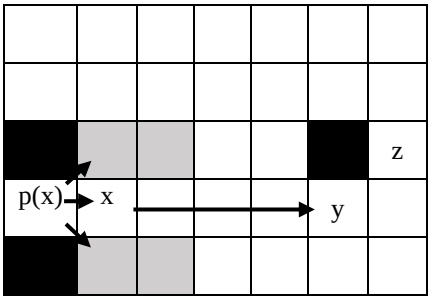
Notasi dan terminologi yang dipakai dalam algoritma JPS adalah sebagai berikut

1. p(x) adalah *start* atau *node* awal
2. z merupakan *goal* atau *node* tujuan
3. *neighbours*(x) adalah *node* sekitar *node* x
4. y dapat dicapai dari x dengan melewati beberapa *node* pada arah yang ditentukan, y dapat dinyatakan sebagai berikut:

$$y = x + k\vec{d} \quad (1)$$

x = *node* yang aktif
k = the number of units of x step towards y
$\vec{d}$ = arah yang dituju
y = *node* calon suksesor x

Pada JPS terdapat sebuah strategi baru yang dapat mempercepat pencarian jalur dengan cara secara selektif melakukan ekspansi ke *nodes* tertentu pada *grid map* yang disebut sebagai *jump point*.




**Gambar 2 *Jumping* secara lurus pada JPS**

Pada gambar 2 dapat dilihat bahwa *node* x melakukan ekspansi dengan *parent* p(x). Arah yang dituju dari ekpansi *node* x adalah lurus ke kanan atau horizontal ke kanan. Tidak seperti algoritma A* yang memasukkan semua *nodes* yang ada dari x menuju y, maka pada JPS hal ini tidak dilakukan. JPS akan langsung melakukan pencarian ke arah kanan sampai menemukan *node* seperti y. Jika ditemukan *node* seperti y, maka tandai *node* tersebut sebagai suksesor dari x dan memberikan nilai g(y) pada *node* tersebut.

$$g(y) = g(x) + dist(x,y) \quad (2)$$

g(x) = biaya dari *node* x
dist(x, y) = jarak dari *node* x menuju y
g(y) = biaya dari *node* awal menuju y

Perhitungan nilai heuristik pada setiap *node* di JPS juga diperlukan untuk menghitung jarak *node* dengan *node* tujuan. Perhitungan nilai heuristik untuk simpul ke-*n* dalam *jump point* dapat menggunakan perhitungan *Manhattan Distance*, *Eucliden Distance* atau *Chebyshev Distance* yang rumusnya adalah sebagai berikut:

a. *Manhattan Distance*

$$h(y) = (abs|currentX - targetX| + abs|currentY - targetY|) \quad (3)$$

*h*(*y*) = perkiraan biaya minimum dari simpul y ke simpul tujuan
currentX = posisi X simpul awal
currentY = posisi Y simpul awal
targetX = posisi X simpul tujuan
targetY = posisi Y simpul tujuan

b. *Euclidean Distance*

$$h(y)=\sqrt{(currentX - targetX)^2 + (currentY - targetY)^2} \quad (4)$$

*h*(*y*) = perkiraan biaya minimum dari simpul y ke simpul tujuan

currentX = posisi X simpul awal
currentY = posisi Y simpul awal
targetX = posisi X simpul tujuan
targetY = posisi Y simpul tujuan

c. *Chebyshev Distance*

$$h(y)=\max(|targetX - currentX|,|targetY-currentY|) \quad (5)$$

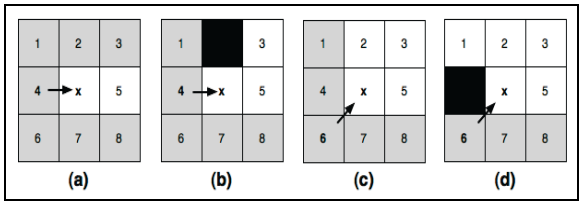
*h*(*y*) = perkiraan biaya minimum dari simpul y ke simpul tujuan
currentX = posisi X simpul awal
currentY = posisi Y simpul awal
targetX = posisi X simpul tujuan
targetY = posisi Y simpul tujuan

Setelah ditemukan nilai biaya suatu *node* yang ditempuh dari awal menuju y dalam bentuk g(y) dan nilai heuristik dari y menuju tujuan, maka diperlukan persamaan yang dapat menentukan nilai sebenarnya dari *node* yang berada di *jump point*, oleh karena itu persamaan dalam A* yaitu f(x) = g(x) + h(x) tetap dipakai dalam JPS. Persamaannya dalam JPS adalah sebagai berikut:

$$f(y) = g(y) + h(y) \quad (6)$$

f(y) = nilai sebenarnya dari *node* y
g(y) = biaya dari *node* awal menuju y
*h*(*y*) = perkiraan biaya minimum dari simpul y ke simpul tujuan

Jika pencarian menuju arah yang ditentukan menemukan penghalang, maka diasumsikan pencarian menuju arah tersebut tidak menghasilkan apa – apa. Saat melakukan ekspansi dari x, dapat dilihat bahwa terdapat evaluasi terhadap *node* lain yang berwarna abu – abu. Pada proses *jumping* ini juga dibutuhkan suatu aturan yang dapat mengenali apakah sebuah *nodes* perlu dilihat atau dapat diabaikan. Aturan ini disebu *pruning*.




**Gambar 3 Aturan pruning pada algoritma JPS**

*Pruning* memiliki artian yaitu pemangkasan yang dalam hal ini berarti *node-node* yang tidak diperlukan akan diabaikan. *Node – node* tersebut dapat diabaikan dengan cara melihat arah yang dipakai dalam ekspansi. Dalam *pruning* terdapat dua istilah yang digunakan, yaitu *natural neighbours* dan *forced neighbours*. *Natural neighbours* merupakan *node* yang dapat tidak diabaikan, sedangkan *forced neighbours* merupakan *node* yang dipaksa untuk tidak diabaikan.

Ada beberapa keadaan yang harus dicermati, tergantung pada bagaimana ekspansi dari p(x) menuju x dilakukan, apakah dilakukan secara lurus baik itu horizontal dan vertikal atau dilakukan secara diagonal.

***Pruning* lurus** : Pangkas semua *node* n di dalam *neighbours*(x), selain *node* yang dapat dituju dengan menggunakan arah ekspansi yang digunakan secara horizontal atau vertikal. Dapat dilihat pada gambar 3 (a) bahwa *node* yang berwarna abu-abu merupakan *nodes* yang diabaikan dikarenakan tidak dapat dicapai menggunakan arah ekspansi yang digunakan yaitu horizontal ke kanan. Pada gambar 3 (a) hanya *node* dengan nomor 5 saja yang diambil, dikarenakan dapat dicapai dengan arah ekspansi yang dapat digunakan.

***Pruning* diagonal** : Pangkas semua *node* n di dalam *neighbours*(x), selain *node* yang dapat dituju dengan menggunakan arah ekspansi yang digunakan secara diagonal. Dapat dilihat pada gambar 3 (c) bahwa *node* yang berwarna abu-abu merupakan *nodes* yang diabaikan dikarenakan tidak dapat dicapai menggunakan arah ekspansi yang digunakan yaitu diagonal ke atas kanan, sedangkan *node* dengan nomor 2 dan 5 tidak berwarna abu-abu dikarenakan *nodes* tersebut dianggap tidak dapat dicapai dari p(x). Oleh karena itu *nodes* yang tidak diabaikan pada gambar 3(c) hanya *nodes* dengan nomor 2, 3 dan 5 saja.

*Node – node* hasil dari *pruning* yang telah dilakukan, baik secara lurus maupun diagonal dipanggil sebagai *natural neighbours* sedangkan *node* yang berwarna abu-abu pada gambar 3 dipanggil sebagai *non-natural neighbours*. Ketika *neighbours*(x) memiliki *node* yang merupakan halangan, maka hasil *pruning* akan menghasilkan *forced neighbours*. *Forced neighbours* dapat dikenali jika n bukan merupakan *natural neighbours* dari x yang dapat dilalui dan x memiliki *neighbours nodes* yang tidak dapat dilalui yang jaraknya dekat dengan n.

Pada gambar 3(b) dapat dilihat bahwa *node* dengan nomor 3 merupakan *forced neighbour* pada *pruning* lurus dikarenakan terdapat *nodes* yang tidak dapat dilalui berwarna hitam yang dekat dengan *node* nomor 3, sedangkan pada gambar 3(d) n = 1 merupakan *forced neighbour* pada *pruning* diagonal.

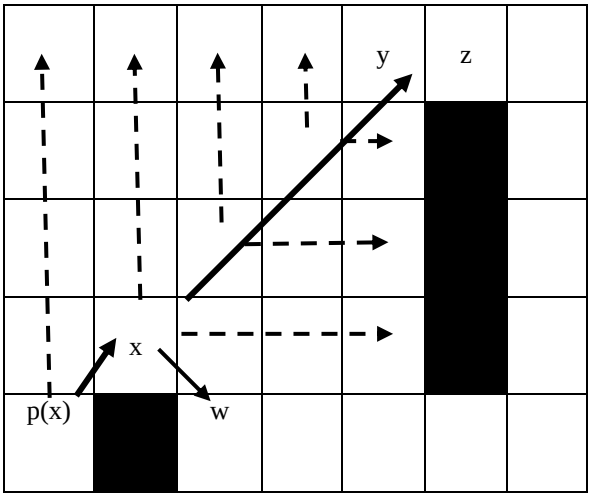
Algoritma JPS memiliki kesamaan terhadap algoritma A*, dimana *nodes* yang akan digunakan dalam penentuan jalur disimpan di dalam memori. Tetapi perbedaannya adalah algoritma JPS tidak menyimpan semua *nodes* yang menjadi jalur menuju tujuan. Pada penjelasan algoritma A* sebelumnya telah dijelaskan tentang bagaimana cara penyimpanan *nodes* di algoritma A* dalam bentuk *open* dan *closed list*. Pada JPS penyimpanan *nodes* pada *closed list* dihilangkan. Algoritma ini

hanya akan menyimpan beberapa *nodes* yang dimasukkan ke dalam *open list*.

*Open list* dalam JPS merupakan *nodes* suksesor atau *node* y dari *node* x yang disebut sebagai *jump point* yang disimpan ke dalam memori. Suatu *node* y dapat dimasukkan ke dalam *open list* dan dipanggil sebagai *jump point* jika memenuhi syarat-syarat sebagai berikut:

1. *Node* y merupakan *goal node.*
2. *Node* y memiliki setidaknya satu *forced neighbours.*
3. Pada pencarian diagonal, pencarian secara horizontal atau vertikalnya memiliki *node* yang memiliki kondisi 1 atau kondisi 2.

Untuk lebih jelasnya pada kondisi 3 dapat dilihat di gambar 4, dimana x melakukan ekspansi secara diagonal, pada ekspansi tersebut ditemukan *node* y yang dimasukkan ke dalam *open list* dikarenakan pada ekspansi secara horizontal ke kanan pada *node* y terdapat *goal*.




**Gambar 4. *Jumping* secara diagonal pada JPS**

Pada gambar 4 dapat dilihat bahwa x melakukan ekspansi secara diagonal. Pada awal ekspansi dilakukan secara vertikal atau horizontal terhadap *node* x dimana w merupakan *forced neighbors* dikarenakan tidak dapat dicapai oleh ekspansi p(x). Pada setiap ekspansi vertikal maupun horizontal setiap *node* yang dikunjungi akan diterapkan aturan *pruning*. Jika ekspansi secara vertikal atau horizontal tersebut tidak ditemukan *node* yang dapat dimasukkan ke dalam *open list* atau menemui penghalang, maka lanjutkan ekspansi secara diagonal. Dapat dilihat di gambar bahwa ekspansi berhenti di *node* y dikarenakan ekspansi secara horizontal ditemukan *node* z atau *node* yang dituju.

## 3. PENGUJIAN

Berikut adalah pengujian pada algoritma *Jump Point Search* yang dilakukan menggunakan perhitungan heuristik *Manhattan Distance*, *Euclidean Distance* dan *Chebyshev Distance*. Pengujian dilakukan sebanyak 30 kali dengan bentuk labirin, posisi musuh dan pemain yang berbeda.

Pengujian ini diterapkan pada 3 kasus dimana kasus tersebut dapat dilihat pada tabel 1.

Tabel 1 Tabel Kasus

| Kasus | Keterangan |
|---|---|
| 1 | Labirin berukuran 14 grid horizontal dan 11 grid vertical |
| 2 | Labirin berukuran 18 grid horizontal dan 15 grid vertical |
| 3 | Labirin berukuran 24 grid horizontal dan 21 grid vertical |

### 3.1 Pengujian Kasus 1

Setelah dilakukan pengujian dengan menggunakan aturan kasus 1 dengan algoritma *Jump Point Search* dengan pengujian sebanyak 30 kali, maka didapatkan rekap hasil pengujian sebagai berikut pada tabel 2.

Tabel 2 Tabel Hasil Kasus 1

| Perhitungan | Keterangan | Rata-rata memori (kb) | Jumlah menemukan jalur | Akurasi (%) |
|---|---|---|---|---|
| *Manhattan* | Tanpa Penghalang | 71.8 | 30 | 100 |
| | Dengan Penghalang | 88.1 | 30 | 100 |
| *Euclidean* | Tanpa Penghalang | 80.4 | 30 | 100 |
| | Dengan Penghalang | 98.4 | 30 | 100 |
| *Chebyshev* | Tanpa Penghalang | 93.6 | 30 | 100 |
| | Dengan Penghalang | 105.6 | 30 | 100 |
| **Rata-rata akurasi (%)** | | | | 100 |

### 3.2 Pengujian Kasus 2

Setelah dilakukan pengujian dengan menggunakan aturan kasus 2 dengan algoritma *Jump Point Search* dengan pengujian sebanyak 30 kali, maka didapatkan rekap hasil pengujian sebagai berikut pada tabel 3.

Tabel 3 Tabel hasil kasus 2

| Perhitungan | Keterangan | Rata-rata memori (kb) | Jumlah menemukan jalur | Akurasi (%) |
|---|---|---|---|---|
| *Manhattan* | Tanpa Penghalang | 145.7 | 30 | 100 |
| | Dengan Penghalang | 180.8 | 27 | 90 |
| *Euclidean* | Tanpa Penghalang | 247.8 | 30 | 100 |
| | Dengan Penghalang | 260.2 | 28 | 93.3 |
| *Chebyshev* | Tanpa Penghalang | 259.1 | 30 | 100 |
| | Dengan Penghalang | 491.7 | 25 | 83.3 |
| **Rata-rata tingkat akurasi (%)** | | | | 94.4 |

### 3.3 Pengujian Kasus 3

Setelah dilakukan pengujian dengan menggunakan aturan kasus 3 dengan algoritma *Jump Point Search* dengan pengujian sebanyak 30 kali, maka didapatkan rekap hasil pengujian sebagai berikut pada tabel 4.

Tabel 4 Tabel hasil kasus 3

| Perhitungan | Keterangan | Rata-rata memori (kb) | Jumlah menemukan jalur | Akurasi (%) |
|---|---|---|---|---|
| *Manhattan Distance* | Tanpa Penghalang | 188.8 | 30 | 100 |
| | Dengan Penghalang | 402.2 | 29 | 96.7 |
| *Euclidean Distance* | Tanpa Penghalang | 274.3 | 30 | 100 |
| | Dengan Penghalang | 507.9 | 28 | 93.3 |
| *Chebyshev Distance* | Tanpa Penghalang | 584.3 | 30 | 100 |
| | Dengan Penghalang | 692.8 | 27 | 90 |
| **Rata-rata tingkat akurasi (%)** | | | | 96.7 |

## 4. KESIMPULAN DAN SARAN

### 4.1. Kesimpulan

Berdasarkan hasil dari pengujian dan implementasi algoritma JPS pada *game* labirin, maka dapat diambil kesimpulan bahwa algoritma JPS (*Jump Point Search*) dapat diimplementasikan pada *game* labirin dalam mengenali perubahan lingkungan yang dinamis, hal ini dapat dilihat dari tingkat keberhasilan yang rata-rata mencapai di atas 90%..

### 4.2. Saran

Pengunaan algoritma *Jump Point Search* disarankan menggunakan perhitungan heuristik *Manhattan Distance* dimana penggunaan memori paling sedikit di antara *Manhattan*, *Euclidean* dan *Chebyshev*. Saran-saran yang dapat dikemukakan agar menjadi bahan masukan dan pertimbangan adalah mencoba menerapkan dua musuh atau lebih dalam labirin.

## 5. DAFTAR PUSTAKA


[1] Dewi, K.A.E. 2014. “Implementasi HPA Star Pada Permainan bomberman (Skripsi)”. Yogyakarta: Duta Wacana Christian University.

[2] Harabor, D., and Grastien, Al. 2011.”Online Graph Pruning for Pathfinding on Grid Maps”, Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence

[3] Salen, K., and E. Zimmerman. 2003. “Rules of Play: Game Design Fundamentals”. Massachusetts London, England: The MIT Press Cambridge.

[4] Sarbjeet. S., and Sukhvinder. S. 2010. “Artificial Intelligence”. International Journal of Computer Application (0975 - 8887), Vol. 6 No. 6.

[5] Suyanto. 2007. “Artificial Intelligence”. Bandung: Informatika.