OPTIMISASI ALGORITMA A* UNTUK PENCARIAN RUTE MENGGUNAKAN MEDIA ROBLOX

Restu Andra Ahmad Saeroji[1], Suastika Yulia Riska[2]

[1,2] Teknik Informatika, Fakultas Teknologi dan Desain, Institut Teknologi dan Bisnis ASIA Malang, Indonesia
[1]rsstst.ann@gmail.com, [2] riska.suastika@asia.ac.id

Abstrak

Pengembangan Non-Player Character (NPC) yang realistis dalam platform *metaverse* seperti Roblox membutuhkan sistem yang efisien. Namun, permasalahan utama yang dihadapi pengembang adalah tingginya biaya komputasi dan kurangnya data mengenai kinerja algoritma A* pada bahasa pemrograman Luau. Penelitian ini bertujuan untuk mengevaluasi kinerja algoritma A* pada Roblox dengan bahasa pemrograman Luau melalui analisis komparatif dengan memvariasikan fungsi heuristik (Manhattan, Euclidean, Chebyshev, dan Octile) dan metode *sorting* (Quick Sort dan Min-Heap Priority Queue). Penelitian dilakukan dengan pendekatan eksperimental kuantitatif di dalam Roblox Studio. Pengujian dilaksanakan pada tiga skenario labirin statis dengan ukuran grid 64x64, 128x128, dan 256x256 studs. Evaluasi didasarkan pada dua metrik utama, yaitu total waktu eksekusi dan total panjang rute yang dihasilkan. Hasil penelitian menunjukkan bahwa penggunaan Min-Heap Priority Queue membuat waktu eksekusi mengalami penurunan rata – rata 46,5% dibandingkan implementasi *default* dengan efektivitas tertinggi sebesar 73,3% pada skenario ukuran 256 studs x 256 studs. Waktu eksekusi dan hasil rute untuk setiap fungsi heuristik memiliki perbedaan yang tidak signifikan kecuali Euclidean Distance. Fungsi heuristik Euclidean Distance mencatatkan waktu eksekusi tercepat di antara fungsi lain sebesar 2,03ms di 64x64, 5,8ms di 128x128, dan 42,35ms di 256x256. Selain itu, fungsi Euclidean Distance menghasilkan rute yang kurang optimal dengan jarak terjauh sebesar 134 studs di 64x64, 427 studs di 128x128, dan 1062 studs di 256x256 dibandingkan fungsi heuristik lainnya. Penelitian ini membuktikan bahwa dalam pengembangan Roblox, pemilihan konfigurasi dan optimisasi algoritma yang tepat sangat krusial bagi pengembang untuk menyeimbangkan antara kecepatan proses dan akurasi rute sesuai kebutuhan.

Kata kunci : algoritma a*, fungsi heuristik, pathfinding, priority queue, roblox

1. Pendahuluan

Metaverse saat ini berkembang dengan sangat pesat (Park & Kim, 2022). Salah satu platform Metaverse, yaitu Roblox bahkan memiliki pengguna aktif bulanan hingga 380 juta pengguna terhitung pada tahun 2025 (Singh, 2025). Selain menjadi platform metaverse, Roblox juga memberikan kesempatan kepada pengguna untuk membuat dunianya sendiri (Rospigliosi, 2022). Melalui Roblox Studio, pengguna dapat mengatur dunianya sendiri dengan fleksibel dan dapat membagikannya langsung ke publik.

Pencarian rute merupakan komponen penting dalam pengembangan sebuah game modern untuk menggerakkan *Non-Player Character* (NPC) yang pintar dan realistis (Morina & Rafuna, 2023; Pardede dkk., 2022). Dari banyaknya algoritma yang digunakan dalam pencarian rute, algoritma A* menjadi standar industri karena waktu eksekusi dan akurasi rute terpendek yang seimbang (Liu, 2023). Namun, implementasi pada platform Roblox, sebuah platform berbasis *user-generated content* memiliki tantangannya tersendiri (Kang dkk., 2024). Roblox menggunakan Luau yang merupakan bahasa turunan dari Lua (Brown dkk., 2021). Seiring meningkatnya kompleksitas, efisiensi algoritma menjadi sangat penting dalam menjaga stabilitas dan performa pengguna.

Permasalahan utama yang dihadapi pengembang Roblox adalah tingginya biaya komputasi dalam pencarian rute pada peta yang luas. Pemilihan komponen algoritma seperti fungsi heuristik atau struktur data sering menjadi pelaku utama yang memperlambat proses algoritma (Riska & Noercholis, 2024). Meskipun algoritma A* menjamin ditemukannya rute terpendek, performa waktu eksekusi belum tentu yang terbaik tergantung pada optimisasi yang digunakan. Kurangnya data yang spesifik mengenai hal perilaku algoritma ini pada lingkup Roblox membuat pengembang bergantung hanya pada implementasi generik yang tidak teroptimisasi.

Berbagai penelitian terdahulu umumnya menggunakan Python, C++, C#. Penelitian dalam konteks video game biasanya memanfaatkan game engine seperti Unity 3D. Pendekatan tersebut membuktikan bahwa perubahan pada fungsi heuristik dan struktur data memiliki dampak performa yang signifikan terhadap algoritma A*. Modifikasi pada fungsi heuristik secara konsisten meningkatkan performa dan efisiensi algoritma pathfinding (Lu &

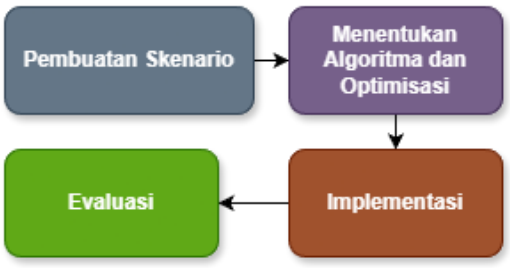
Sun, 2024; Luo, 2024). Penggunaan fungsi heuristik yang berbeda juga mempengaruhi jumlah titik yang dievaluasi dan berdampak langsung pada kecepatan pencarian (Yan dkk., 2020). Penelitian yang dilakukan pada C++ membuktikan bahwa struktur data yang telah dioptimisasi dapat mengurangi waktu dan biaya eksekusi saat pemrosesan data titik hasil algoritma (Hong dkk., 2021). Di sisi lain, terdapat penelitian lainnya yang membuktikan bahwa dampak optimisasi tidak selalu signifikan dalam segala kondisi. Sebuah studi yang menggunakan Python menemukan bahwa pada peta ruang terbuka dengan skala kecil, perbedaan performa antara fungsi heuristik dapat diabaikan (Rachmawati & Gustin, 2020).

Implementasi pathfinding pada Roblox saat ini menghadapi tantangan akibat terbatasnya data empiris mengenai kinerja algoritma dalam lingkungan Roblox Luau. Kondisi ini memaksa pengembang untuk terus menggunakan metode yang kurang optimal. Selain itu, banyaknya penelitian terdahulu yang berfokus pada bahasa pemrograman umum seperti C++ dan Python menciptakan kesenjangan yang signifikan, mengingat arsitektur teknis dan karakteristik performa Roblox Luau berbeda secara fundamental.

Penelitian ini berupaya mengatasi kesenjangan tersebut dengan melakukan evaluasi algoritma dalam ekosistem Roblox Luau guna memberikan wawasan mengenai metode optimisasi yang efektif. Studi ini menerapkan analisis komparatif performa algoritma A* dengan menguji variasi fungsi heuristik dan metode *sorting* melalui pendekatan eksperimental kuantitatif. Hasil penelitian ini diharapkan dapat menjadi rujukan pengembang dalam menentukan konfigurasi algoritma yang sesuai dengan kebutuhan pengembang Roblox.

2. Metode

Penelitian dilakukan langsung menggunakan Roblox Studio versi terbaru. Algoritma saat melakukan *testing* juga dijalankan di dalam Roblox Studio dalam mode *Run*, mode yang menjalankan *script* dari sudut pandang *client* dengan perspektif yang bisa terbang untuk memudahkan *debugging* dan melihat hasil. Alur penelitian dapat dilihat pada blok diagram pada Gambar 1.



Gambar 1. Blok Diagram Alur Penelitian

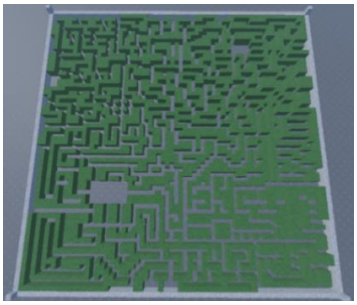
2.1 Pembuatan Skenario

Skenario dibuat berupa labirin 2 dimensi dengan berbagai rintangan. Rintangan terdiri dari ruang terbuka, lorong panjang, belokan berputar, dan berbagai persimpangan. Skenario ini sendiri berupa skenario statis yang tidak ada perubahan secara langsung saat algoritma berjalan. Skenario dibuat di berbagai ukuran, dengan ukurannya dihitung menggunakan satuan studs, satuan panjang yang digunakan Roblox Studio dalam pengembangan dunia virtualnya.

Skenario dibuat menggunakan sistem grid dengan ukuran setiap grid sebesar 4 x 4 studs. Sistem grid dipilih untuk menyeragamkan ukuran sehingga perhitungan akan konsisten serta memastikan kompabilitas dengan variasi fungsi heuristik. Dibuat 3 skenario yang memiliki ukuran berbeda beda, yaitu 64 studs x 64 studs, 128 studs x 128 studs, dan 256 studs x 256 studs yang dapat dilihat pada Gambar 2, Gambar 3, dan Gambar 4. Variasi ukuran 64x64, 128x128, dan 256x256 dipilih untuk meningkatkan kompleksitas ruang pencarian secara eksponen dengan tujuan menguji skalabilitas algoritma.

Gambar 2. Skenario Labirin 64 studs x 64 studs

Gambar 3. Skenario Labirin 128 studs x 128 studs



Gambar 4. Skenario Labirin 256 studs x 256 studs

ISSN: 2614-6371 E-ISSN: 2407-070X

2.2 Menentukan Algoritma dan Optimisasi

Algoritma pathfinding yang digunakan dalam penelitian ini adalah algoritma A*. Optimisasi yang dilakukan berupa perubahan fungsi heuristik dan implementasi algoritma sorting yang lebih baik. Perubahan fungsi heuristik dipilih sebagai optimisasi karena setiap fungsi memiliki cara menentukan nilai terendah sebuah titik yang berbeda (Diana & Herdiana, 2021). Algoritma sorting dipilih sebagai optimisasi karena dapat meningkatkan performa pengolahan data titik yang ada (Rizvi dkk., 2024).

Fungsi heuristik yang digunakan pada penelitian ini antara lain Manhattan Distance, Euclidean Distance, Chebysev Distance, dan Octile Distance (Elshahed dkk., 2025). Algoritma sorting yang digunakan adalah Quick Sort dan Min-Heap Priority Queue (Al-Aydi & Abu-Naser, 2025). Variasi tersebut bertujuan untuk membandingkan efisiensi, akurasi, dan performa algoritma A* secara keseluruhan di berbagai optimisasi dan skenario.

2.3 Implementasi

Algoritma diimplementasikan pada Roblox menggunakan Roblox Studio dengan bahasa pemrograman Luau. Luau sendiri merupakan turunan dari Lua yang dibuat oleh Roblox Corporation. Perbedaan paling mencolok antara Lua dan Luau terdapat pada segi performa. Algoritma dari penelitian ini dapat dilihat pada pseudocode di Gambar 5.

```
1    openSet = {}
2    closedSet = {}
3
4    add startNode to openSet
5
6    while LOOP
7            Current = get Node in openSet
     with lowest F costs do:
8            remove Current from openSet
9            add Current to closedSet
10
11           if Current is targetNode then:
12                   return
13
14           for each Neighbour in Current
15                   if
     isTraversable(Neighbour) or Neighbour in
     closedSet
16                           continue
17                   if newPath to Neighbour
     is shorter or Neighbour is not in openSet
18                           set FCost of
     Neighbour
19                           set Parent of
     Neighbour to Current
20
21                           if Neighbour not
     in openSet
22                                   add
     Neighbour in openSet
```

Gambar 5. Pseudocode Algoritma Pathfinding A*

Dalam pseudocode, dapat dilihat bahwa algoritma yang digunakan pada penelitian ini terdiri dari 3 fungsi utama. Pertama, inisialisasi openSet untuk titik yang akan dievaluasi dan closedSet untuk titik yang sudah dievaluasi, dilanjutkan dengan penambahan titik awal ke dalam openSet. Kedua, dilakukan perulangan yang terus berjalan selama ada titik pada openSet yang dalam setiap iterasinya, algoritma memilih Current atau titik saat ini yang memiliki F costs terendah yang kemudian dipindahkan dari openSet ke closedSet. Terakhir, melakukan pemeriksaan tersebut terhadap semua titik di sekitar titik Current apabila Current bukan titik akhir.

2.4 Evaluasi

Evaluasi performa algoritma didasarkan pada dua metriks utama yaitu total waktu eksekusi dan panjang rute yang ditemukan (Damayanti & Akbar, 2024). Perhitungan kedua metriks tersebut dilakukan langsung dari Roblox Studio. Perhitungan waktu eksekusi dilakukan mulai dari fungsi utama dipanggil hingga proses selesai. Panjang rute dihitung dengan cara menjumlahkan jarak dari setiap titik yang dihasilkan algoritma menggunakan fungsi Magnitude (Adamer dkk., 2024). Perhitungan dari kedua metriks ini secara kolektif memberikan gambaran mengenai performa dan tingkat optimal algoritma dalam menemukan jalur (Lawande dkk., 2022).

3. Hasil dan Pembahasan

Pengujian dilakukan langsung dari dalam Roblox Studio. Algoritma dijalankan satu per satu dengan optimisasi yang berbeda – beda. Quick Sort digunakan sebagai metode sorting *default* karena Roblox Luau menyediakan fungsi bawaan berupa *table.sort* yang memanfaatkan metode sorting tersebut. Setiap skenario diuji sebanyak 3 iterasi dengan titik awal dan titik akhir yang sama. Dalam pengujiannya, perangkat yang digunakan hanya membuka Roblox Studio untuk menghindari variabel lain seperti memori yang digunakan oleh aplikasi lain.

Tabel 1. Iterasi Pertama Skenario 64 x 64

No	Modifikasi	Waktu Eksekusi	Panjang Rute
1	Manhattan Distance	18,08ms	130 studs
2	Euclidean Distance	4,73ms	134 studs
3	Chebyshev Distance	21,17ms	130 studs
4	Octile Distance	19,15ms	130 studs
5	Manhattan Distance dan PriorityQueue Sort	6,92ms	130 studs
6	Euclidean Distance dan PriorityQueue Sort	2,59ms	134 studs
7	Chebyshev Distance dan PriorityQueue Sort	9,05ms	130 studs
8	Octile Distance dan PriorityQueue Sort	6,74ms	130 studs

Tabel 2. Iterasi Kedua Skenario 64 x 64

No	Modifikasi	Waktu Eksekusi	Panjang Rute
1	Manhattan Distance	20,06ms	130 studs
2	Euclidean Distance	3,89ms	134 studs
3	Chebyshev Distance	15,29ms	130 studs
4	Octile Distance	16,09ms	130 studs
5	Manhattan Distance dan PriorityQueue Sort	8,86ms	130 studs
6	Euclidean Distance dan PriorityQueue Sort	2,76ms	134 studs
7	Chebyshev Distance dan PriorityQueue Sort	9,45ms	130 studs
8	Octile Distance dan PriorityQueue Sort	9,1ms	130 studs

Tabel 3. Iterasi Ketiga Skenario 64 x 64

No	Modifikasi	Waktu Eksekusi	Panjang Rute
1	Manhattan Distance	19,93ms	130 studs
2	Euclidean Distance	3,95ms	134 studs
3	Chebyshev Distance	18,91ms	130 studs
4	Octile Distance	17,14ms	130 studs
5	Manhattan Distance dan PriorityQueue Sort	8,6ms	130 studs
6	Euclidean Distance dan PriorityQueue Sort	2,03ms	134 studs
7	Chebyshev Distance dan PriorityQueue Sort	7,22ms	130 studs
8	Octile Distance dan PriorityQueue Sort	7,5ms	130 studs

Pada skenario 64 studs x 64 studs, dari ketiga iterasi, waktu eksekusi dengan modifikasi algoritma sorting Min-Heap Priority Queue secara konsisten meningkatkan kecepatan eksekusi algoritma. Modifikasi Euclidean Distance memiliki waktu eksekusi tercepat di semua iterasi, yaitu di rentang 2,03ms hingga 4,73ms. Meskipun begitu, Euclidean Distance memiliki jarak yang lebih panjang dibandingkan fungsi heuristik lainnya. Hasil pengujian lebih lengkapnya untuk skenario 64 studs x 64 studs dapat dilihat pada Tabel 1, Tabel 2, dan Tabel 3.

Tabel 4. Iterasi Pertama Skenario 128 x 128

No	Modifikasi	Waktu Eksekusi	Panjang Rute
1	Manhattan Distance	26,13ms	374 studs
2	Euclidean Distance	8,64ms	427 studs
3	Chebyshev Distance	27,56ms	374 studs
4	Octile Distance	24,48ms	375 studs
5	Manhattan Distance dan PriorityQueue Sort	21,12ms	374 studs
6	Euclidean Distance dan PriorityQueue Sort	7,08ms	427 studs
7	Chebyshev Distance dan PriorityQueue Sort	22,53ms	374 studs
8	Octile Distance dan PriorityQueue Sort	16,73ms	375 studs

Tabel 5. Iterasi Kedua Skenario 128 x 128

No	Modifikasi	Waktu Eksekusi	Panjang Rute
1	Manhattan Distance	22,07ms	374 studs
2	Euclidean Distance	13ms	427 studs
3	Chebyshev Distance	23,87ms	374 studs
4	Octile Distance	22,6ms	375 studs
5	Manhattan Distance dan PriorityQueue Sort	21,16ms	374 studs
6	Euclidean Distance dan PriorityQueue Sort	5,8ms	427 studs
7	Chebyshev Distance dan PriorityQueue Sort	23,23ms	374 studs
8	Octile Distance dan PriorityQueue Sort	26,18ms	375 studs

Tabel 6. Iterasi Ketiga Skenario 128 x 128

No	Modifikasi	Waktu Eksekusi	Panjang Rute
1	Manhattan Distance	20,04ms	374 studs
2	Euclidean Distance	8,05ms	427 studs
3	Chebyshev Distance	21,55ms	374 studs
4	Octile Distance	25,66ms	375 studs
5	Manhattan Distance dan PriorityQueue Sort	23,18ms	374 studs
6	Euclidean Distance dan PriorityQueue Sort	6,87ms	427 studs
7	Chebyshev Distance dan PriorityQueue Sort	20,27ms	374 studs
8	Octile Distance dan PriorityQueue Sort	21,28ms	375 studs

Pada skenario 128 studs x 128 studs, dari ketiga iterasi, ditemukan pula bahwa Min-Heap Priority Queue Sort secara konsisten meningkatkan waktu eksekusi algoritma. Modifikasi Euclidean Distance juga memiliki waktu eksekusi tercepat di semua iterasi, yaitu di rentang 5,8ms dengan Priority Queue hingga 13ms tanpa Priority Queue. Meskipun begitu, Euclidean Distance memiliki perbedaan jarak yang signifikan dibandingkan fungsi heuristik lain. Hasil pengujian lebih lengkapnya untuk skenario 128 studs x 128 studs dapat dilihat pada Tabel 4, Tabel 5, dan Tabel 6.

Tabel 7. Iterasi Pertama Skenario 256 x 256

No	Modifikasi	Waktu Eksekusi	Panjang Rute
1	Manhattan Distance	782,63ms	864 studs
2	Euclidean Distance	345,86ms	1062 studs
3	Chebyshev Distance	432,26ms	864 studs
4	Octile Distance	808,06ms	890 studs
5	Manhattan Distance dan PriorityQueue Sort	224,85ms	864 studs
6	Euclidean Distance dan PriorityQueue Sort	47,92ms	1062 studs
7	Chebyshev Distance dan PriorityQueue Sort	196,93ms	864 studs
8	Octile Distance dan PriorityQueue Sort	191,34ms	890 studs

Tabel 8. Iterasi Kedua Skenario 256 x 256

No	Modifikasi	Waktu Eksekusi	Panjang Rute
1	Manhattan Distance	796,71ms	864 studs
2	Euclidean Distance	347,32ms	1062 studs
3	Chebyshev Distance	460,22ms	864 studs
4	Octile Distance	804,98ms	890 studs
5	Manhattan Distance dan PriorityQueue Sort	193,35ms	864 studs
6	Euclidean Distance dan PriorityQueue Sort	44,85ms	1062 studs
7	Chebyshev Distance dan PriorityQueue Sort	168,08ms	864 studs
8	Octile Distance dan PriorityQueue Sort	191,71ms	890 studs

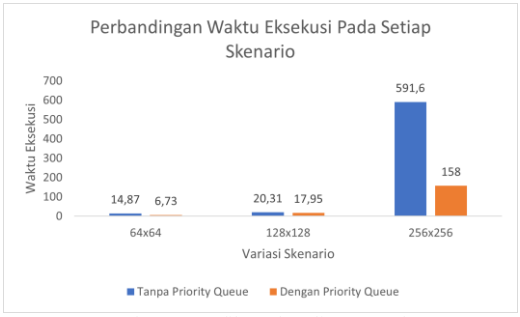
ISSN: 2614-6371 E-ISSN: 2407-070X

Tabel 9. Iterasi Ketiga Skenario 256 x 256

No	Modifikasi	Waktu Eksekusi	Panjang Rute
1	Manhattan Distance	763,15ms	864 studs
2	Euclidean Distance	332,52ms	1062 studs
3	Chebyshev Distance	436,78ms	864 studs
4	Octile Distance	789.02ms	890 studs
5	Manhattan Distance dan PriorityQueue Sort	211,99ms	864 studs
6	Euclidean Distance dan PriorityQueue Sort	42,35ms	1062 studs
7	Chebyshev Distance dan PriorityQueue Sort	189,42ms	864 studs
8	Octile Distance dan PriorityQueue Sort	193,3ms	890 studs

Pada skenario 256 studs x 256 studs, dari ketiga iterasi, mengikuti tren skenario sebelumnya, ditemukan bahwa Min-Heap Priority Queue masih secara konsisten mempercepat waktu eksekusi algoritma. Modifikasi Euclidean Distance masih memiliki waktu eksekusi tercepat di semua iterasi, yaitu di rentang 42,35ms dengan Priority Queue hingga 347,32ms tanpa Priority Queue. Di skenario ini, perbedaan jarak antar fungsi heuristik mulai kelihatan, dengan Euclidean memiliki jarak terjauh sebesar 1062 studs dibandingkan rata – rata jarak algoritma lain sekitar 872 studs. Hasil pengujian lebih lengkapnya untuk skenario 256 studs x 256 studs dapat dilihat pada Tabel 7, Tabel 8, dan Tabel 9.

Dari pengujian ketiga skenario dengan 3 iterasi, nilai rata – rata setiap skenario dihitung dan dapat dilihat bahwa Priority Queue secara konsisten mengurangi waktu eksekusi algoritma yang ditunjukkan pada Gambar 6.



Gambar 6. Grafik Perbandingan Waktu

Tabel 10. Persentase Reduksi Waktu Menggunakan Min – Heap Priority Queue

Fungsi Heuristik	Reduksi 64x64 (%)	Reduksi 128x128 (%)	Reduksi 256x256 (%)	Rata – Rata Heuristik
Manhattan Distance	58,0%	**4,1%**	73,1%	45,1%
Euclidean Disntace	41,3%	33,5%	**86,8%**	**53,9%**
Chebyshev Distance	53,6%	9,5%	58,3%	40,5%
Octile Distance	55,4%	11,7%	76,0%	47,7%
Rata – Rata Skenario	52,1%	**14,7%**	**73,6%**	

Priority Queue juga konsisten mengurangi waktu eksekusi untuk setiap variasi fungsi heuristik yang dapat dilihat lebih lengkap di Tabel 10. Secara umum, optimisasi ini memberikan dampak paling signifikan pada skenario 256 studs x 256 studs, dengan rata – rata reduksi waktu sebesar 73,6% dan penurunan tertinggi sebesar 86,8% pada Euclidean Distance. Secara keseluruhan, Euclidean Distance memiliki rata – rata reduksi waktu tertinggi di semua skenario sebesar 53,9%. Namun, terdapat anomali pada skenario 128x128 yang hanya mengalami rata – rata reduksi waktu sebesar 14,7% dengan nilai terendah 4,1% pada Manhattan Distance. Anomali tersebut terjadi karena kompleksitas labirin pada skenario 128x128 yang relatif sederhana dan menyebabkan jumlah node yang perlu dievaluasi tidak cukup besar untuk memicu *bottleneck* pada metode sorting standar. Hal ini membuktikan bahwa penggunaan Priority Queue lebih efektif diterapkan pada kasus dengan ruang pencarian yang kompleks dan masif.

Panjang rute yang dihasilkan antara fungsi heuristik memiliki perbedaan yang tidak terlalu signifikan, kecuali Euclidean Distance. Dalam 3 skenario di semua iterasi sesuai dengan hasil pada Tabel 1 hingga Tabel 9, Euclidean selalu memiliki jarak yang lebih panjang dibandingkan fungsi heuristik yang lain dan terlihat sangat jelas pada skenario 256 studs x 256 studs. Kombinasi dari Euclidean Distance dan Priority Queue memiliki waktu tercepat diantara optimisasi yang lain, yaitu sebesar 2,03ms di 64x64, 5,8ms di 128x128, dan 42,35 di 256x256, namun menghasilkan rute terpanjang yaitu 134 studs di 64x64, 427 studs di 128x128, dan 1062 studs di 256x256.

Hasil ini sesuai dengan beberapa penelitian terdahulu yang mengatakan bahwa modifikasi fungsi heuristik dan metode *sorting* meningkatkan performa algoritma A*. Penelitian yang dilakukan Foead dkk., membuktikan bahwa perubahan fungsi heuristik dapat meningkatkan kecepatan algoritma hingga 50%, sedangkan penelitian yang dilakukan Kapi dkk., menemukan bahwa perubahan struktur data algoritma menjadi Min-Heap dapat mengurangi kompleksitas waktu algoritma dari O(n) menjadi O(1) (Foead dkk., 2021; Kapi, 2020) .

4. Kesimpulan

Dari data yang didapatkan, dapat disimpulkan bahwa perubahan metode optimisasi algoritma pathfinding dalam lingkungan Roblox berpengaruh pada waktu eksekusi dan panjang rute yang dihasilkan. Penggunaan Min-Heap Priority Queue secara konsisten mengurangi waktu eksekusi, yang membuktikan bahwa metode *sorting* berpengaruh signifikan terhadap performa algoritma. Pengaruh dari variasi fungsi heuristik tidak terlihat pada ukuran skenario yang kecil, namun seiring bertambah besarnya skenario, pengaruh tersebut semakin besar.

Manfaat utama dari hasil penelitian ini adalah membantu pengembang, terutama pengembang dunia virtual atau game pada platform Roblox untuk memberikan sebuah data yang dapat membantu menentukan algoritma dan optimisasi yang ingin digunakan sesuai dengan kebutuhan. Namun, data yang dihasilkan dari penelitian ini memiliki keterbatasan yaitu hanya menggunakan satu algoritma pathfinding, kurangnya variasi skenario labirin selain ukurannya, dan pengujiannya yang hanya dilakukan dalam lingkungan Roblox Luau. Selain itu, penelitian ini juga hanya melakukan pengujian skenario yang statis tidak berubah – ubah, tidak adanya perubahan tinggi pada titik tujuan atau hanya 2 dimensi, dan pengujian dilakukan secara lokal pada Roblox Studio.

Sebagai pengembangan penelitian selanjutnya, dapat mengimplementasikan algoritma pathfinding lain seperti Jump Point Search dan Rectangular Expansion A*, menggunakan algoritma *sorting* yang berbeda, dan menambahkan variasi skenario labirin seperti perubahan area secara *real-time* atau penambahan kapabilitas 3 dimensi. Penting juga melakukan evaluasi yang lebih mendalam, seperti pengaruh *hardware* terhadap algoritma dan evaluasi performa antara *client* dan *server*.

Daftar Pustaka

Adamer, M. F., De Brouwer, E., O'Bray, L., & Rieck, B. (2024). The magnitude vector of images. *Journal of Applied and Computational Topology*, *8*(3), 447–473. https://doi.org/10.1007/s41468-024-00182-9

Al-Aydi, B., & Abu-Naser, S. (2025). *Comparative Study of Traditional and AI-Enhanced Sorting Algorithms: QuickSort, MergeSort, HeapSort, and TimSort*. https://doi.org/10.13140/RG.2.2.13915.84005

Brown, L., Friesen, A., & Jeffrey, A. (2021). *Position Paper: Goals of the Luau Type System*.

Damayanti, Y., & Akbar, Y. (2024). Implementasi Algoritma A* (A-Star) untuk Mencari Rute Terpendek dari Kelurahan Cibubur ke Perpustakaan Nasional Republik Indonesia. *Jurnal Indonesia : Manajemen Informatika Dan Komunikasi*, *5*(3), 3306–3325. https://doi.org/10.35870/jimik.v5i3.1022

Diana, I., & Herdiana, B. (2021). Analisa Penggunaan Nilai Bobot Heuristik yang Berbeda pada Algoritma Weighted A*. *Telekontran : Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan*, *9*(1), 82–93. https://doi.org/10.34010/telekontran.v9i1.5677

Elshahed, A., Mohamed, A. S. A., Aini, F., Aun, J., & Khan, M. (2025). Efficient Pathfinding on Grid Maps: Comparative Analysis of Classical Algorithms and Incremental Line Search. *IEEE Access*, *3*, 465–481. https://doi.org/10.1109/ACCESS.2025.3575168

Foead, D., Ghifari, A., Kusuma, M. B., Hanafiah, N., & Gunawan, E. (2021). A Systematic Literature Review of A* Pathfinding. *Procedia Computer Science*, *179*, 507–514. https://doi.org/10.1016/j.procs.2021.01.034

Hong, Z., Sun, P., Tong, X., Pan, H., Zhou, R., Zhang, Y., Han, Y., Wang, J., Yang, S., & Xu, L. (2021). Improved A-Star Algorithm for Long-Distance Off-Road Path Planning Using Terrain Data Map. *ISPRS International Journal of Geo-Information*, *10*(11), 785. https://doi.org/10.3390/ijgi10110785

Kang, Y., Lee, U., & Lee, S. (2024). Who Makes Popular Content? Information Cues from Content Creators for Users' game Choice: Focusing on User-Created Content Platform "Roblox." *Entertainment Computing*, *50*, 100697. https://doi.org/10.1016/j.entcom.2024.100697

Kapi, A. (2020). A Review on Informed Search Algorithms for Video Games Pathfinding. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*, 2756–2764. https://doi.org/10.30534/ijatcse/2020/42932020

Lawande, S. R., Jasmine, G., Anbarasi, J., & Izhar, L. I. (2022). A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games. *Applied Sciences*, *12*(11), 5499. https://doi.org/10.3390/app12115499

Liu, D. (2023). Research of the Path Finding Algorithm A* in Video Games. *Highlights in Science, Engineering and Technology*, *39*, 763–768. https://doi.org/10.54097/hset.v39i.6642

Lu, J., & Sun, Y. (2024). Application of heuristic function optimization strategy of A* algorithm in path planning of mobile robot. *2024 5th International Conference on Computer Engineering and Application (ICCEA)*, 77–80. https://doi.org/10.1109/ICCEA62105.2024.10603737

Luo, Y. (2024). A fast convergence bidirectional a star path planning algorithm. *Applied and Computational Engineering*. https://api.semanticscholar.org/CorpusID:268831231

Morina, V., & Rafuna, R. (2023). A Comparative Analysis of Pathfinding Algorithms in NPC Movement Systems for Computer Games. *UBT International Conference*.

ISSN: 2614-6371 E-ISSN: 2407-070X

https://knowledgecenter.ubt-uni.net/conference/IC/CS/6

Pardede, S. L., Athallah, F. R., Huda, Y. N., & Zain, F. D. (2022). A Review of Pathfinding in Game Development. *Journal of Computer Engineering*, *1*(01), 47–56. https://doi.org/10.25124/cepat.v1i01.4863

Park, S.-M., & Kim, Y.-G. (2022). A Metaverse: Taxonomy, Components, Applications, and Open Challenges. *IEEE Access*, *10*, 4209–4251. https://doi.org/10.1109/ACCESS.2021.3140175

Rachmawati, D., & Gustin, L. (2020). Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem. *Journal of Physics: Conference Series*, *1566*(1), 012061. https://doi.org/10.1088/1742-6596/1566/1/012061

Riska, S. Y., & Noercholis, A. (2024). PERFORMANCE COMPARISON OF FASTER R-CONVOLUTIONAL NEURAL NETWORK (CNN) AND EFFICIENTNET FOR TRAIN DETECTION UNDER DIVERSE LIGHTING AND IMAGE QUALITY CONDITIONS. *Jurnal Teknik Informatika (Jutif)*, *5*(6), 1811–1821. https://doi.org/10.52436/1.jutif.2024.5.6.3438

Rizvi, D. Q., Rai, H., & Jaiswal, R. (2024, Maret). *Sorting Algorithms in Focus: A Critical Examination of Sorting Algorithm Performance*.

Rospigliosi, P. 'asher.' (2022). Metaverse or Simulacra? Roblox, Minecraft, Meta and the turn to virtual reality for education, socialisation and work. *Interactive Learning Environments*, *30*(1), 1–3. https://doi.org/10.1080/10494820.2022.2022899

Singh, S. (2025, Juni 23). How Many People Play Roblox 2025 [Player Stats]. *DemandSage*. https://www.demandsage.com/how-many-people-play-roblox/

Yan, B., Chen, T., Zhu, X., Yue, Y., Xu, B., & Shi, K. (2020). A Comprehensive Survey and Analysis on Path Planning Algorithms and Heuristic Functions. Dalam K. Arai, S. Kapoor, & R. Bhatia (Ed.), *Intelligent Computing* (hlm. 581–598). Springer International Publishing. https://doi.org/10.1007/978-3-030-52249-0_39

Halaman ini sengaja dikosongkan