Analisis Perbandingan Algoritma Dijkstra, Greedy Best First Search, dan A* dalam Permainan Snake

Andri Nurdianto - 13523145
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: andri070805@gmail.com , 13523145@std.stei.itb.ac.id

***Abstract*—Penelitian ini membandingkan tiga algoritma pencarian jalur popular yaitu Dijkstra, Greedy Best First Search, dan A* dalam konteks permainan Snake otomatis. Masing-masing algoritma diimplementasikan untuk mengendalikan ular secara mandiri, dan diuji melalui pertandingan satu lawan satu dalam beberapa ronde. Dijkstra menjamin jalur terpendek, tetapi lebih lambat, Greedy Best First Search sangat cepat, tetapi lebih berisiko, sedangkan A* memberikan keseimbangan antara kecepatan dan keamanan. Heuristik manhattan distance digunakan untuk Greedy Best First Search dan A*. Hasil percobaan menunjukkan bahwa A* secara umum unggul dalam performa keseluruhan, sementara Greedy Best First Search efektif dalam lingkungan yang terbuka dan sederhana. Penelitian ini menyoroti perbandingan antara optimalitas jalur, efisiensi waktu, dan keselamatan dalam pengambilan keputusan permainan secara real-time.**

Keywords—Dijkstra; Greedy; A*; Snake

I. PENDAHULUAN

Permainan Snake adalah sebuah genre video *game* yang menggunakan ular sebagai karakter yang dimainkan. Pada awalnya, genre permainan ini dipopulerkan oleh video *game arcade* yang bernama Blockade pada tahun 1976 serta diluncurkan oleh Gremlin Industries. Konsep permainan ini memiliki varian pemain tunggal dengan garis kepala dan ekor pada ular yang memanjang.

Pemain diharuskan untuk mengarahkan ular ke makanan supaya ular dapat memanjang dan mendapatkan skor yang banyak. Ular yang dikendalikan dapat bergerak secara vertikal atau horizontal. Permainan berakhir ketika ular berhasil memenuhi seluruh petak atau ular menabrak dirinya sendiri.

Dengan teknologi yang ada, permainan Snake dapat dilakukan dengan *auto player* tanpa memerlukan manusia sebagai pengendalinya. Ular tersebut dijalankan dengan adanya algoritma yang menentukan cara ular tersebut bisa bergerak. Algoritma yang digunakan pada ular tersebut bisa bermancam-macam. Algoritma yang berhubngan dengan *route planning* biasanya digunakan oleh permainan Snake untuk mencari jalur terbaik ke makanan.

Algoritma *route planning* yang dapat digunakan untuk menngendalikan ular secara *auto player* diantaranya Dijkstra, Greedy Best First Search, dan A*. Dijkstra digunakan dengan mencari semua kemungkinan jalur dari kepala ular menuju makanan, sedangkan Greedy Best First Search dan A* digunakan dengan menambahkan heuristik dalam menentukan jalurnya. Ketiga algoritma tersebut dibandingkan untuk menemukan algoritma yang paling efisien pada permainan Snake.

II. LANDASAN TEORI

A. Dijkstra

Algoritma Dijkstra digunakan untuk mencari jalur terpendek dari satu simpul sumber ke semua simpul lain dalam sebuah graf berbobot tanpa bobot negatif. Proses dimulai dengan menetapkan jarak dari simpul awal ke dirinya sendiri sebagai nol, dan ke simpul lainnya sebagai tak hingga. Setiap simpul memiliki label yang terdiri dari jarak saat ini dan simpul pendahulu dalam jalur terpendek.

Langkah pertama adalah memilih simpul awal, lalu memeriksa semua tetangganya yang langsung terhubung. Untuk setiap tetangga, dihitung total biaya dari simpul awal ke tetangga tersebut, dan nilai ini disimpan jika lebih kecil dari nilai sebelumnya. Setelah semua tetangga diperiksa, simpul dengan jarak terkecil yang belum dikunjungi dipilih sebagai simpul berikutnya untuk diproses. Proses ini berulang dari simpul yang sedang diproses, tetangga-tetangganya dievaluasi kembali untuk melihat apakah jalur baru yang melaluinya memberikan jarak yang lebih pendek.

Setiap simpul yang telah diproses dan semua tetangganya diperiksa akan diberi status permanen, artinya jarak terpendek ke simpul tersebut telah ditemukan. Proses ini diulangi sampai semua simpul diberi label permanen. Hasil akhirnya adalah daftar jarak minimum dari simpul awal ke semua simpul lain, serta jalur terpendek yang dapat ditelusuri kembali melalui catatan simpul pendahulu masing-masing. [1]

Algoritma ini menjamin jalur terpendek dengan memilih simpul dengan jarak terkecil di setiap langkah, dan memastikan bahwa jalur tersebut tidak dapat diperpendek lebih lanjut setelah diberi label permanen. Pendekatan ini efisien dan dapat digunakan pada berbagai jenis graf selama tidak ada bobot negatif pada sisi-sisinya. Dengan kompleksitas waktu yang relatif tinggi, terutama jika graf yang digunakan besar atau padat. Dalam konteks permainan seperti Snake, Dijkstra akan mencoba semua kemungkinan arah pergerakan sebelum menentukan jalur terbaik menuju makanan, sehingga lebih aman, tetapi bisa kurang efisien dari segi waktu, terutama saat

berhadapan dengan banyak simpul atau ketika waktu reaksi menjadi faktor penting.

B. *Greedy Best First Search*

Greedy Best First Search (GBFS) adalah algoritma pencarian yang hanya mempertimbangkan nilai heuristik h(n) dari setiap simpul, dengan asumsi bahwa simpul dengan nilai h yang lebih rendah lebih dekat ke tujuan. Pada setiap langkahnya, GBFS memilih dan mengeksplorasi simpul yang belum diperluas dan memiliki nilai h terkecil. Karena hanya mengandalkan estimasi heuristik tanpa mempertimbangkan biaya dari simpul awal, GBFS tidak menjamin solusi optimal dan bersifat *satisficing* untuk menemukan solusi, meskipun bukan yang terbaik.

Dalam perilakunya, GBFS sensitif terhadap akurasi dan bentuk fungsi heuristik. Algoritma ini bisa terjebak pada daerah yang disebut *heuristic plateau* atau *local minima*, yaitu bagian graf di mana banyak simpul memiliki nilai heuristik yang sama atau justru meningkat sehingga tidak ada panduan yang jelas ke arah solusi. Untuk memahami perilaku pencariannya lebih lanjut, diperkenalkan konsep seperti *high-water mark*, yang menunjukkan batas tertinggi nilai heuristik yang harus dilewati untuk mencapai solusi, serta *apex*, yaitu nilai maksimum heuristik pada jalur dari simpul awal ke suatu simpul tertentu.

GBFS sebenarnya bukan satu algoritma tunggal, melainkan keluarga algoritma yang berbeda berdasarkan strategi *tie-breaking* yaitu cara memilih di antara simpul-simpul dengan nilai heuristik yang sama. Dalam studi lebih lanjut, ruang pencarian dibagi ke dalam struktur yang disebut *benches* (bangku), yaitu kelompok simpul dengan karakteristik heuristik tertentu. GBFS mengeksplorasi satu *bench* sepenuhnya sebelum berpindah ke bench berikutnya dengan nilai heuristik yang lebih rendah. Setelah keluar dari satu bench, algoritma tidak akan kembali lagi ke bench tersebut.

Dalam analisis yang lebih mendalam, ditemukan juga konsep *bottleneck states* yaitu simpul yang wajib dilalui untuk mencapai tujuan dari simpul awal, serta craters, yaitu area dalam bench di mana semua simpul memiliki nilai heuristik lebih rendah dan memaksa GBFS mengeksplorasi seluruh area tersebut sebelum dapat kembali naik ke jalur solusi. Dengan analisis ini, GBFS dapat dipahami secara struktural simpul mana yang pasti diperluas, mana yang mungkin diperluas tergantung tie-breaking, dan mana yang tidak akan pernah disentuh, terlepas dari strategi pemilihan. [2]

Greedy Best First Search lebih mengutamakan kecepatan menuju tujuan dibandingkan jaminan optimalitas jalur. Algoritma ini menggunakan heuristik untuk menentukan node mana yang harus dieksplorasi berikutnya, tanpa mempertimbangkan total jarak yang telah dilalui. Pendekatan ini membuat GBFS sangat cepat dan efisien di ruang terbuka, tetapi juga berisiko tinggi karena bisa salah memilih jalur jika hanya berdasarkan perkiraan jarak ke tujuan. Dalam permainan Snake, hal ini bisa menyebabkan *auto player* dengan GBFS terjebak di sudut atau menabrak tubuh sendiri jika tidak ada pengecekan yang memadai.

C. *A**

A* (A-Star) adalah algoritma pencarian informatif yang mengombinasikan dua komponen utama g(n) yang mewakili jarak dari simpul awal ke simpul saat ini, dan h(n) sebagai estimasi jarak dari simpul saat ini ke tujuan. Penjumlahan dari kedua komponen ini, yaitu f(n) = g(n) + h(n), menjadi dasar dalam menentukan simpul mana yang akan dieksplorasi selanjutnya. Heuristik h(n) sangat menentukan arah pencarian, sehingga A* akan lebih cepat menemukan solusi jika estimasi h(n) mendekati nilai sebenarnya. Karena alasan ini, kualitas dan desain fungsi heuristik sangat berpengaruh terhadap efisiensi algoritma.

Dalam penerapannya, A* sering kali mampu menemukan jalur yang optimal, tetapi membutuhkan memori yang besar dan waktu proses yang tinggi terutama pada graf atau peta yang sangat luas. Oleh karena itu, pengembangan algoritma turunan seperti HPA* (Hierarchical Pathfinding A*) digunakan untuk mengurangi kompleksitas dan mempercepat pencarian dengan membagi peta besar menjadi wilayah-wilayah yang lebih kecil dan saling terhubung. Selain itu, A* juga dimodifikasi untuk kasus multi-agent, pencarian bidirectional, serta ditingkatkan dengan metode *swarm intelligence* atau *heuristic directional* untuk meningkatkan efisiensi tanpa mengorbankan akurasi.

Secara keseluruhan, meskipun A* memiliki keterbatasan, dengan modifikasi yang tepat algoritma ini tetap sangat relevan dalam menyelesaikan berbagai masalah pathfinding. [3]

D. *Heuristik*

Heuristik bekerja berdasarkan pengetahuan yang tidak lengkap, dugaan yang masuk akal, dan intuisi untuk memperkirakan langkah selanjutnya yang paling bermanfaat. Cara kerjanya adalah dengan memberikan panduan keputusan dalam suatu proses pencarian, misalnya dengan memilih simpul atau langkah yang terlihat paling dekat ke tujuan. Heuristik juga dapat berupa strategi penyederhanaan masalah, penghilangan opsi yang dianggap tidak menjanjikan, atau pemanfaatan pengalaman masa lalu untuk mempercepat pencarian. Dalam praktiknya, heuristik sering digunakan untuk meningkatkan kinerja sistem, bukan untuk mendapatkan solusi yang dijamin benar, tetapi solusi yang “cukup baik” dalam waktu yang wajar. [4]

Heuristik memiliki empat dimensi utama: ketidakpastian hasil, dasar pengetahuan yang tidak lengkap, peningkatan kinerja, dan panduan pengambilan keputusan. Dengan demikian, heuristik adalah pendekatan pemecahan masalah yang menyeimbangkan antara kepraktisan dan ketidaksempurnaan, dan sangat berguna dalam domain di mana solusi optimal sulit ditemukan secara langsung atau tidak layak dihitung secara menyeluruh.

Heuristik bukanlah lawan dari algoritma dalam pengertian formal, melainkan alternatif strategi yang muncul dari keterbatasan sumber daya komputasi atau waktu. Dalam banyak kasus nyata, solusi algoritmik mungkin ada tetapi terlalu mahal secara komputasional. Di sinilah heuristik memainkan peran penting sebagai alat bantu pragmatis mengorbankan kepastian dan optimalitas demi efisiensi dan kelayakan praktis. Sebuah heuristik dapat berupa saran, aturan informal, atau strategi eksperimental yang muncul dari

pengalaman sebelumnya atau intuisi domain tertentu. Dengan demikian, heuristik memberikan kerangka kerja eksperimental dalam pengembangan kecerdasan buatan, memungkinkan peneliti menyesuaikan strategi pemecahan masalah sesuai dengan karakteristik masalah yang dihadapi.

Heuristik dapat berperan baik sebagai mekanisme aktif maupun pasif dalam proses pengambilan keputusan. Heuristik aktif secara langsung memilih langkah selanjutnya dalam proses pencarian, seperti fungsi evaluasi dalam permainan catur. Sementara itu, heuristik pasif bertindak sebagai filter atau penyaring opsi sebelum keputusan diambil. Kedua jenis ini bersama-sama memperlihatkan bahwa heuristik memiliki fungsi sentral dalam membimbing arah pencarian di ruang masalah yang kompleks. Penekanan pada kemampuan heuristik untuk mengurangi jumlah pilihan yang harus dievaluasi secara eksplisit menjadikannya strategi penting dalam bidang seperti perencanaan otomatis, sistem pakar, dan pembuktian teorema.

E. Manhattan Distance

Manhattan Distance dijelaskan sebagai salah satu metode perhitungan jarak yang digunakan sebagai heuristi dari sebuah algoritma. Manhattan Distance, yang juga dikenal sebagai *city block distance*, menghitung total jarak antara dua titik dengan menjumlahkan nilai absolut dari selisih masing-masing atribut (koordinat) antara dua titik tersebut. Secara matematis, jika dua titik berada di ruang berdimensi-d, maka jaraknya adalah jumlah dari $|x_1 - y_1| + |x_2 - y_2| + ... + |xd - yd|$. Metode ini mirip dengan cara seseorang berjalan di sepanjang jalan berbentuk kotak pada peta kota. [5]

Manhattan Distance sering digunakan sebagai fungsi heuristik dalam algoritma pencarian jalur seperti A* dan Greedy Best First Search, karena sifatnya yang sederhana, efisien, dan cukup akurat dalam kondisi tertentu terutama pada lingkungan grid atau peta kotak-kotak di mana pergerakan dibatasi hanya ke arah vertikal dan horizontal.

III. IMPLEMENTASI

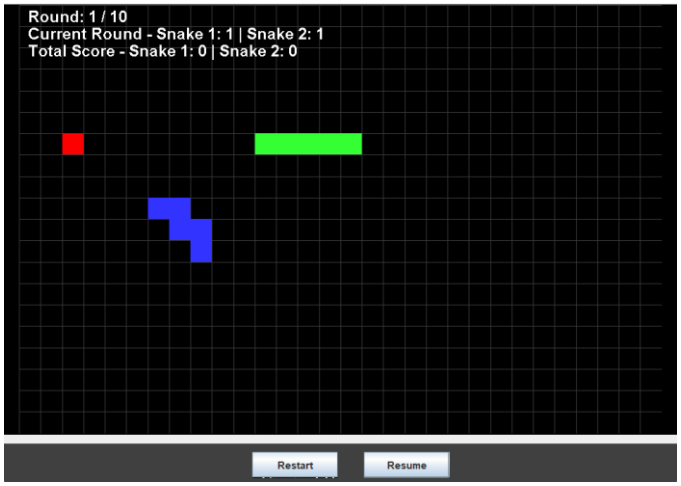
Implementasi dilakukan dengan menggunakan 2 ular yang masing-masing memiliki algoritma yang berbeda. Percobaan akan terbagi menjadi 3 bagian yaitu Dijkstra melawan Greedy Best First Search, Dijkstra melawan A*, dan Greedy Best First Seach melawan A*.

Implementasi dilakukan dengan repository yang ada pada pranala berikut https://github.com/Arsh-Jafri/dijkstra-Snake-game dan dilakukan modifikasi seperti penambahan algoritma Greedy Best First Search dan A*, perhitungan skor, dan opsi permainan *auto player* vs *auto player*.

Akan dilakukan 10 percobaan untuk masing-masing skenario. Skor akan dihitung berdasarkan banyaknya makanan yang berhasil dimakan, permainan akan berhenti ketika salah satu ular menabarak. Hasil dari pertandingan tersebut akan dianilisis untuk melihat perbandingan dari setiap algoritma yang dipakai.

A. Dijkstra melawan Greedy Best First Search

Percobaan pertama dengan menggunakan algoritma Dijkstra dan Greedy Best First Search. Ular yang menggunakan algoritma Dijkstra akan memakai warna hijau dan berperan sebagai *Snake* 1 sedangkan ular dengan algoritma Greedy Best First Search akan memakai warna biru dan berperan sebagai *Snake* 2.



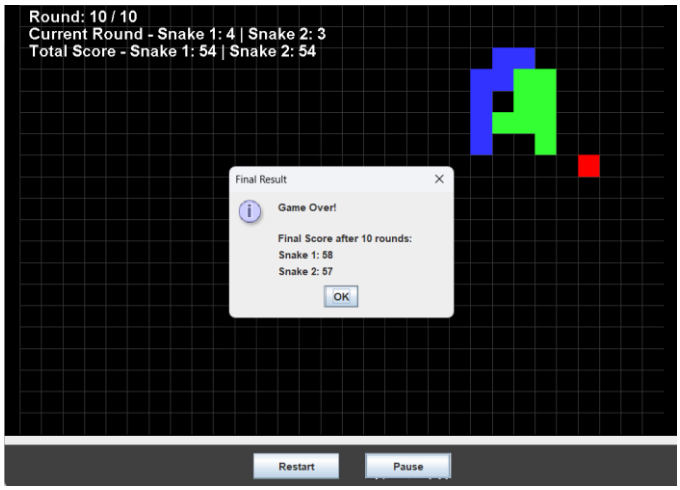
Gambar 1. Ular algoritma Dijkstra melawan ular algortima Greedy Best First Search

Sumber: dokumentasi pribadi

Rekapitulasi skor dapat dilihat pada tabel berikut:

Ronde	Skor Ular Algoritma Dijkstra	Skor Ular Algoritma Greedy Best First Search
1	4	2
2	2	2
3	6	8
4	2	2
5	4	2
6	1	2
7	21	22
8	13	13
9	1	1
10	4	3
Skor Total	58	57

Tabel 1. Perolehan skor algoritma Dijkstra dan algoritma Greedy Best First Search



Gambar 2. Hasil pertandingan ular algoritma Dijkstra melawan ular algoritma Greedy Best First Search

Sumber: dokumentasi pribadi

Pertandingan antara algoritma Dijkstra dan Greedy best First Search memperlihatkan pertandingan yang cukup kompetitif dengan selisih 1 poin, yaitu 58 untuk Dijkstra dan 57 untuk Greedy Best First Search. Dijkstra yang mengeksplorasi jalur secara menyeluruh dapat diimbangi oleh Greedy Best First Search yang menggunakan manhattan distance sebagai heuristiknya. Pertandingan ini menunjukkan bahwa kedua algoritma memiliki karakteristik berbeda, tetapi seimbang dalam praktiknya.

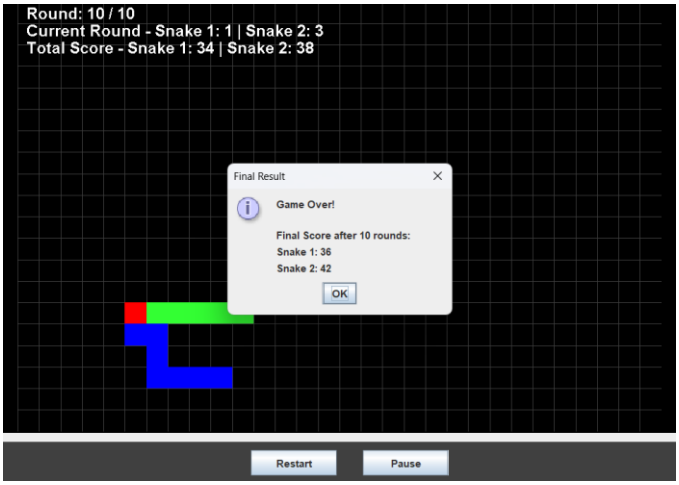
*B. Dijkstra melawan A**

Percobaan kedua dengan menggunakan algoritma Dijkstra dan A*. Ular yang menggunakan algoritma Dijkstra akan memakai warna hijau dan berperan sebagai *Snake* 1 sedangkan ular dengan algoritma A* akan memakai warna biru dan berperan sebagai *Snake* 2.

Rekapitulasi skor dapat dilihat pada tabel berikut:

Ronde	Skor Ular Algoritma Dijkstra	Skor Ular Algoritma A*
1	5	5
2	3	3
3	7	6
4	8	11
5	3	1
6	1	1
7	4	10
8	2	0
9	1	1
10	2	4
Skor Total	36	42

Tabel 2. Perolehan skor algoritma Dijkstra dan algoritma A*



Gambar 3. Hasil pertandingan ular algoritma Dijkstra melawan ular algoritma A*

Sumber: dokumentasi pribadi

Pertandingan antara algoritma Dijkstra dan A* menunjukan hasil yang cukup signifikan. A* unggul 6 poin dengan skor 42 sedangkan Dijkstra hanya berhasil mendapatkan 36 poin. Perolehan skor yang signifikan terjadi pada ronde 7 dengan Dijkstra hanya mendapat 4 poin, sedangkan A* mendapatkan 10 poin. Dari sini dapat dilihat kekuatan pada A* yang mempertimbangkan eksplorasi tidak hanya secara menyeluruh, tetapi juga dengan mempertimbangkan jarak menuju ke tujuan. A* dibantu heuristic Manhattan distance dalam praktiknya. Kemenangan A* dalam pertandingan ini menunjukan algoritma tersebut lebih adaptif dalam permainan berabsis grid khususnya permainan *Snake*, tanpa harus mengorbankan banyak jalur untuk dieksplorasi

*C. Greedy Best First Search melawan A**

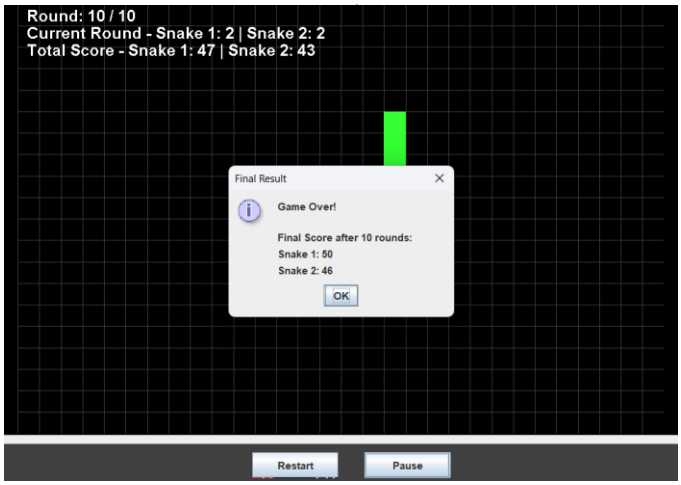
Percobaan ketiga dengan menggunakan algoritma Greedy Best First Search dan A*. Ular yang menggunakan algoritma Dijkstra akan memakai warna hijau dan berperan sebagai *Snake* 1 sedangkan ular dengan algoritma A* akan memakai warna biru dan berperan sebagai *Snake* 2.

Rekapitulasi skor dapat dilihat pada tabel berikut:

Ronde	Skor Ular Algoritma Greedy Best First Search	Skor Ular Algoritma A*
1	10	7
2	2	2
3	6	8
4	8	4
5	4	3
6	3	8

7	5	3
8	4	3
9	5	5
10	3	3
Skor Total	50	46

Tabel 3. Perolehan skor algoritma algoritma Greedy Best First Search dan algoritma A*



Gambar 4. Hasil pertandingan ular algoritma Greedy Best First Search melawan ular algoritma A*

Pertandingan antara algoritma Greedy Best First Search dan A* menunjukan hasil GBFS yang unggul tipis dengan selisih 4 poin dengan perolehan skor 50 untuk GBFS dan 46 untuk A*. Secara teori A* lebih unggul karena memepertimbangkan total biaya perjalanan, tetapi GBFS dapat memanfaatkan keunggulan kecepatan menggunakan heuristiknya saja. Dalam permainan grid seperti *Snake* kecepatan lebih diunggulkan dibandingan dengan mengeksplorasi seluruh grid terlebih dahulu, terlebih permainan yang dilakukan tidak memiliki *obstacle* sebagai tantangan tambahan. Hal itu bisa menjadi alas an mengapa GBFS lebih unggul dibandingkan A* dalam kasus ini. Namun, perbedaan skor yang tidak terlalu jauh menunjukkan bahwa A* konsisten dalam performanya meskipun tetap melihat seluruh jalur saat mengeksplorasi.

IV. Analisis dan Pembahasan

Berdasarkan hasil percobaan antara algoritma Dijkstra melawan Greedy Best First Search (GBFS), dapat diamati bahwa kedua algoritma memiliki performa yang cukup seimbang. Dijkstra unggul tipis dengan skor akhir 58 dibandingkan 57 dari GBFS. Hal ini menunjukkan bahwa meskipun Dijkstra mengeksplorasi seluruh jalur untuk menemukan yang terpendek, pendekatan heuristik dari GBFS cukup kompetitif dalam konteks permainan Snake. Namun, dari sisi efisiensi, GBFS memiliki keunggulan karena tidak perlu menghitung jarak dari titik awal ke semua simpul lain, melainkan hanya memprioritaskan simpul yang dianggap paling dekat ke makanan berdasarkan heuristik.

Pertandingan antara Dijkstra dan A* menghasilkan skor akhir 36 untuk Dijkstra dan 42 untuk A*. Ini menunjukkan bahwa A* cenderung lebih unggul dalam memaksimalkan jumlah makanan yang dikonsumsi dalam kondisi grid yang sama. Hal ini dapat dijelaskan oleh fakta bahwa A* menggabungkan keunggulan Dijkstra (g(n)) dan GBFS (h(n)) dalam penilaian simpul, sehingga memberikan hasil pencarian yang lebih seimbang antara eksplorasi dan estimasi tujuan. A* mampu menemukan jalur yang optimal dengan lebih cepat dibandingkan Dijkstra yang cenderung eksploratif penuh.

Sementara itu, pada percobaan antara GBFS dan A*, hasil akhir menunjukkan GBFS meraih skor 50 dan A* memperoleh skor 46. Meskipun sebelumnya A* mengungguli Dijkstra, pada percobaan ini A* kalah dari GBFS. Hal ini bisa jadi disebabkan oleh sifat greedy dari GBFS yang lebih agresif dalam mengejar makanan, yang dalam konteks permainan Snake dengan peta kecil dan makanan tunggal, justru dapat memberikan keuntungan. A* yang mempertimbangkan total biaya cenderung lebih berhati-hati, sehingga mungkin kalah cepat dari GBFS dalam beberapa skenario.

Analisis ini menunjukkan bahwa tidak ada satu algoritma yang secara mutlak lebih unggul dalam semua kondisi. Dijkstra menjamin jalur terpendek, tetapi cenderung lebih lambat, A* memberikan solusi optimal dengan keseimbangan efisiensi dan akurasi, sementara GBFS menawarkan kecepatan, tetapi bisa mengorbankan optimalitas. Dalam konteks permainan Snake, yang lebih mengutamakan kecepatan reaksi terhadap posisi makanan, agresivitas GBFS bisa memberikan keunggulan dalam banyak kasus meskipun tidak konsisten.

Selain itu, perlu diperhatikan bahwa permainan akan berakhir saat salah satu ular menabrak, bukan saat waktu habis atau skor tertentu tercapai. Hal ini berarti bahwa algoritma yang mampu menavigasi ruang dengan aman sekaligus cepat menuju makanan akan lebih unggul. Dari hasil percobaan, terlihat bahwa GBFS dan A* relatif seimbang dalam hal kecepatan, tetapi A* cenderung lebih aman, sedangkan GBFS bisa berisiko jika heuristik mengarah ke jalur sempit.

Secara keseluruhan, pemilihan algoritma terbaik untuk permainan Snake sangat bergantung pada konteks tujuan permainan apakah mengutamakan keselamatan, kecepatan, atau kombinasi keduanya. A* terbukti sebagai algoritma paling seimbang dari ketiganya, sementara GBFS bisa unggul dalam pertandingan cepat dengan kondisi ruang terbuka karena agresif langsung menuju ke makanan, dan Dijkstra tetap menjadi algoritma yang andal untuk eksplorasi penuh, tetapi kurang efisien dalam konteks *real time* game seperti Snake.

V. Kesimpulan

Penelitian ini menunjukkan bahwa setiap algoritma memiliki keunggulan dan kelemahan masing-masing ketika diterapkan pada permainan Snake. Algoritma Dijkstra secara konsisten menghasilkan jalur terpendek, tetapi waktu pencariannya cenderung lebih lama. Di sisi lain, Greedy Best First Search menunjukkan performa yang sangat cepat dengan pendekatan heuristik, meskipun tidak selalu menghasilkan jalur optimal. A* menjadi algoritma yang paling seimbang karena mampu memadukan ketepatan dan efisiensi waktu.

Dari hasil percobaan dalam sepuluh ronde untuk setiap pasangan algoritma, A* secara umum memperoleh skor tertinggi dibandingkan Dijkstra maupun GBFS. Greedy Best First Search juga beberapa kali unggul atas A* dalam skenario tertentu yang menekankan kecepatan menuju makanan. Hal ini membuktikan bahwa konteks lingkungan dan strategi pergerakan sangat memengaruhi performa tiap algoritma.

Permainan Snake yang berakhir saat salah satu ular menabrak memberikan tantangan tersendiri dalam menentukan algoritma yang paling efektif. A* cenderung lebih aman, sementara GBFS bisa unggul dalam pertandingan cepat tetapi berisiko tinggi. Dijkstra tetap relevan untuk skenario yang membutuhkan kepastian jalur terpendek dan aman, meskipun kurang cocok dalam permainan yang memerlukan respon cepat.

Secara keseluruhan, A* direkomendasikan sebagai algoritma terbaik dalam konteks permainan Snake yang kompleks dan dinamis. Namun, GBFS dapat digunakan ketika efisiensi waktu lebih diutamakan, dan Dijkstra bisa tetap berguna dalam lingkungan yang tidak terlalu padat dan membutuhkan eksplorasi menyeluruh.

ACKNOWLEDGMENT

Terima kasih kepada Allah SWT karena atas nikmat dan rahmat-Nya. Penulis dapat menyelesaikan makalah berjudul "Analisis Perbandingan Algoritma Dijkstra, Greedy Best First Search, dan A* dalam Permainan Snake" dengan baik. Selain itu, tidak lupa saya ucapkan terima kasih kepada dosen mata kuliah Strategi Algoritma, Teknik Informatika ITB Semester II 2024/2025 K3 Kampus Jatinangor, Monterico Adrian, S.T., M.T. Penulis juga berterima kasih kepada seluruh sumber yang dijadikan referensi pada makalah ini.

REFERENSI

[1] Javaid, A. (2013). Understanding Dijkstra Algorithm. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.2340905

[2] Heusner, M., Keller, T., & Helmert, M. (2021). Understanding the search behaviour of greedy Best-First Search. Proceedings of the International Symposium on Combinatorial Search, 8(1), 47–55. https://doi.org/10.1609/socs.v8i1.18425

[3] 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

[4] Romanycia, M. H. J., Pelletier F. J. (1985). What is A heuristic?.

[5] Suwanda, R., Syahputra, Z., & Zamzami, E. M. (2020). Analysis of Euclidean distance and Manhattan distance in the K-Means Algorithm for variations number of Centroid K. Journal of Physics Conference Series, 1566(1), 012058. https://doi.org/10.1088/1742-6596/1566/1/012058

PERNYATAAN

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

Bandung, 24 Juni 2025

Andri Nurdianto - 13523145