Graph-Based Pathfinding Optimization for Zombie AI Navigation in DeadZone Uprising

A Discrete Mathematics Approach Using Graph Theory and Shortest-Path Algorithms

Dylan Frederico Ketaren - 13525036
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: dylanfk01@gmail.com , 13525036@std.stei.itb.ac.id

***Abstract*—Graph theory is widely used to model relationships between objects and to solve pathfinding problems in various computational domains. One practical application of graph theory is found in video games, where non-player characters (NPCs) must navigate through an environment while avoiding obstacles. In this study, a graph-based navigation system is implemented in a three-dimensional zombie survival game developed using the Godot Engine. The game environment is represented as a graph in which vertices correspond to navigable grid positions and edges represent valid movements between adjacent locations. To determine an efficient route from a zombie character to the player, the A* (A-Star) pathfinding algorithm is applied using a cost function and heuristic-based search strategy. The implementation is evaluated through comparative navigation scenarios that contrast direct-chase behavior with graph-based pathfinding. Qualitative observations indicate that the graph-based approach produces more structured and obstacle-aware movement compared to direct pursuit. This study demonstrates that graph theory and shortest-path algorithms from discrete mathematics can be effectively applied to game artificial intelligence, bridging theoretical concepts with practical software engineering problems.**

Keywords—graph theory; weighted graph; pathfinding; A* algorithm; game development; Godot Engine; discrete mathematics.

I. INTRODUCTION

Graph theory is one of the fundamental topics in discrete mathematics and provides a powerful abstraction for representing relationships among objects. A graph G = (V, E) consists of a finite set of vertices V and a set of edges E connecting pairs of vertices. Depending on the problem domain, edges may be directed or undirected, and may carry associated numerical weights. These structural properties have enabled graph theory to be applied across a wide range of fields, including transportation networks, communication systems, social network analysis, and computer game development [1].

One important practical application of graph theory is the pathfinding problem. In this context, vertices represent locations within a space, while edges represent traversable connections between adjacent locations. When edge weights are assigned, the resulting structure is a weighted graph that can model distances, movement costs, or travel times. Various graph traversal and search algorithms can then be employed to determine efficient paths between two vertices.

In modern video games, pathfinding is a fundamental component of artificial intelligence (AI) for non-player characters (NPCs). Enemies, allies, and autonomous agents must navigate dynamically through complex three-dimensional environments while responding to player behavior, avoiding obstacles, and pursuing objectives. Without an effective navigation system, NPC behavior is often unrealistic, reducing immersion and overall gameplay quality [2].

This study applies graph theory to the development of DeadZone: Uprising, a three-dimensional zombie survival game implemented using the Godot Engine. The game environment is modeled as a weighted graph in which grid cells act as vertices and traversable connections act as edges, with edge weights defined by Euclidean distance. Zombie characters must locate and pursue the player through this environment. To address the resulting navigation problem, the A* (A-Star) pathfinding algorithm is utilized, which combines an exact path cost function with an admissible heuristic estimate to efficiently guide the search.

The primary objective of this study is to demonstrate how graph representations and shortest-path algorithms from discrete mathematics can be applied in practical game development. Additionally, the study evaluates the behavioral differences between a direct-chase navigation strategy and the graph-based A* approach through qualitative scenario analysis.

The remainder of this paper is organized as follows. Section II presents the theoretical background on graph theory, weighted graphs, and pathfinding algorithms. Section III describes the system design and methodology. Section IV presents the

implementation details and discusses the experimental results. Section V concludes the paper and outlines directions for future work.

II. THEORETICAL BACKGROUND

A. Graph Theory Fundamentals

A graph is a mathematical structure used to represent pairwise relationships between objects. Formally, a graph is defined as G = (V, E), where V is a non-empty finite set of vertices (also called nodes) and E ⊆ V × V is a set of edges connecting pairs of vertices. Graph theory is a foundational topic in discrete mathematics and underpins many algorithms in computer science [1].

Two fundamental categories of graphs are relevant to this study. In an *undirected graph*, each edge (u, v) represents a symmetric relationship, meaning traversal is possible in both directions. In a *directed graph* (digraph), edges are ordered pairs indicating a one-directional relationship. Game navigation graphs may employ either model depending on whether movement between two points is permitted in both directions [3].

Additional structural properties include connectivity, which describes whether a path exists between any two vertices in the graph, and cycles, which occur when a path exists from a vertex back to itself. For navigation applications, a connected graph is generally required to ensure that all regions of the environment remain reachable.

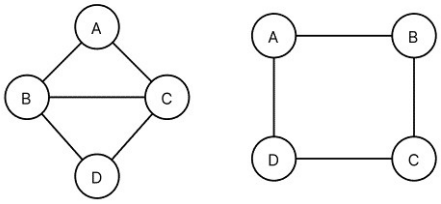


Fig. 1. Simple graph illustrations
sources: Author illustrations

B. Weighted Graphs

A weighted graph extends the basic graph definition by assigning a numerical value, called a weight, to each edge. Formally, a weighted graph is defined as G = (V, E, w), where w: E → ℝ is a weight function that maps each edge to a real number [3].

In game environments, edge weights typically represent the movement cost between two adjacent navigation points. The most common cost metric is Euclidean distance, computed as:

$$d(u, v) = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}$$

where (x_1, y_1, z_1) and (x_2, y_2, z_2) are the three-dimensional coordinates of adjacent vertices u and v, respectively. In the present implementation, movement cost is defined by the distance between connected grid cell.

The use of weighted graphs transforms the navigation problem into a minimum-cost path search problem. Among all paths connecting the start vertex to the destination vertex, the goal is to identify the path whose total edge weight is minimized. This formulation is the basis for classical shortest-path algorithms.

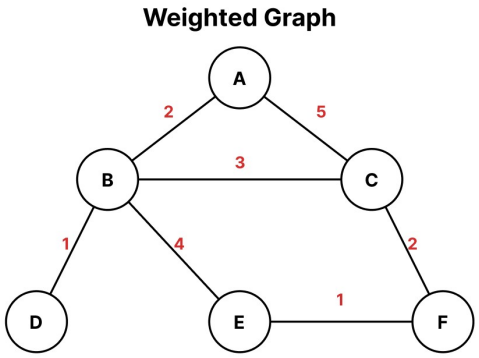


Fig. 2. Weighted graph illustrations
sources: Author illustrations

C. Pathfinding and Shortest-Path Algorithms

Pathfinding is the computational process of determining a route between two locations in a graph. The problem can be formally stated as: given a graph G = (V, E, w), a start vertex s ∈ V, and a goal vertex t ∈ V, find a path P = (s, v_1, v_2, ..., t) such that the total cost Σ w(e) over all edges in P is minimized [4].

Several classical algorithms address this problem with different trade-offs between computational complexity and search efficiency.

Breadth-First Search (BFS) is an uninformed search strategy that explores all vertices at distance k before exploring vertices at distance k+1. BFS guarantees the shortest path in terms of the number of edges in an unweighted graph, but does not account for edge weights [4]. Its time complexity is O(V + E), where V is the number of vertices and E is the number of edges.

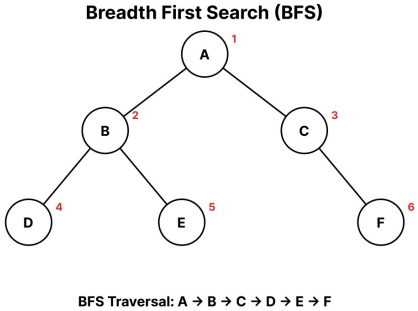


Fig. 3. Breadth first search illustrations
sources: Author illustrations

Dijkstra's Algorithm extends BFS to handle non-negative edge weights. It maintains a priority queue of vertices ordered by their current best-known path cost from the start vertex. At each step, the vertex with the lowest accumulated cost is expanded, and neighboring vertices are relaxed if a shorter path is discovered. Dijkstra's algorithm guarantees optimal solutions for non-negative weight graphs and operates in O((V + E) log V) time when implemented with a binary heap [4].

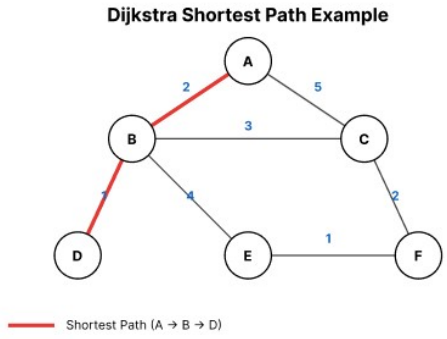


Fig. 4. Dijkstra's Algorithm illustrations
sources: Author illustrations

A* Algorithm augments Dijkstra's approach with a problem-specific heuristic function h(n) that estimates the remaining cost from vertex n to the goal. The evaluation function is:

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

where g(n) is the exact cost from the start vertex to n, and h(n) is the heuristic estimate of the cost from n to the goal. When h(n) is admissible (never overestimates the true cost) and consistent (satisfies the triangle inequality), A* is guaranteed to find an efficient path [5]. In three-dimensional environments, the Euclidean distance between vertex n and the goal serves as a natural admissible heuristic.

A comparative summary of these algorithms with respect to key properties is presented in Table I.

TABLE I. Comparison of Pathfinding Algorithms
sources: Author illustrations

Algorithm	Edge Weights	Heuristic	Optimality	Time Complexity
BFS	Unweighted	No	yes (unweighted)	O(V + E)
Dijkstra	Yes	No	Yes	O((V+E) log V)
A*	Yes	Yes	Yes	O(E log V) typical

D. Application of Graph Theory in Game Navigation

The application of graph theory to game AI navigation is well-established in the literature. Navigation meshes (NavMesh) and grid-based graphs are the two most common spatial representations used in commercial and research game engines [2].

In a grid-based graph, traversable grid cells are connected through graph edges representing valid movements.. This representation allows pathfinding algorithms to operate on a finite, manageable graph rather than on continuous three-dimensional space. The main limitation of grid-based graphs is coverage granularity: paths are restricted to the predefined vertices, which may not represent all navigable positions accurately [6].

The A* algorithm is the dominant choice for game pathfinding due to its balance between optimality and efficiency. By focusing the search toward the goal using a heuristic, A* typically explores fewer vertices than Dijkstra's algorithm in practice, making it more suitable for real-time game environments where computation must complete within a single rendering frame [5].

In this study, a custom grid-based graph is constructed for the DeadZone: Uprising environment, and A* is applied directly over this graph structure. This approach maintains transparency and educational clarity, as the graph construction and search logic are explicitly implemented rather than delegated to an engine abstraction layer.

III. SYSTEM DESIGN AND METHODOLOGY

A. Research Methodology

This study adopts an implementation-based approach to investigate the practical application of graph theory in game AI development. A three-dimensional zombie survival game, DeadZone: Uprising, serves as the experimental platform. The zombie navigation system is modeled as a graph search problem and solved using the A* pathfinding algorithm.

The development process is organized into four main stages: (1) modeling the game environment as a weighted graph, (2) defining navigation vertices and edges, (3) implementing the A* pathfinding algorithm, and (4) evaluating the resulting navigation behavior through qualitative scenario comparison.

The visual assets used in the prototype, including character models and animations, were obtained from publicly available asset libraries. The primary contribution of this work lies in the implementation and evaluation of graph-based navigation techniques rather than asset creation.

B. Graph Representation of the Game Environment

To apply graph theory to the game world, the three-dimensional environment is represented as a weighted undirected graph G = (V, E, w). The game environment is discretized into a two-dimensional navigation grid. Each grid cell is represented as a graph vertex, while connections between adjacent cells form graph edges. Each pair of spatially adjacent grid cells is connected by an edge. Edge weights are determined by grid traversal costs. Horizontal and

vertical movements are assigned a cost of 1, while diagonal movements incur a cost of approximately 1.414, corresponding to the square root of two.The graph representation transforms the zombie navigation problem into a shortest-path search problem. The zombie's position and the player's position are converted into grid coordinates, which serve as the start and goal vertices for pathfinding, and the player's current position is similarly mapped to its closest vertex. The pathfinding module then computes a minimum-cost path between these two vertices.

A conceptual illustration of the navigation graph is shown in Fig.5. Vertices represent traversable grid cells distributed across the game environment, and edges connect adjacent grid cells.

Fig. 5. Experimental navigation environment with obstacles and traversable grid cells.
sources: Author illustrations

C. A Pathfinding Process*

The A* algorithm operates on the navigation graph to determine an efficient path from the zombie's position to the player's position. The algorithm maintains two data structures throughout its execution: an *open set*, containing vertices that have been discovered but not yet fully evaluated, and a *closed set*, containing vertices whose efficient path cost has been finalized.

At each iteration, the vertex n with the smallest f(n) = g(n) + h(n) value is extracted from the open set. All neighbors of n are examined: for each neighbor m, a tentative path cost g'(m) = g(n) + w(n, m) is computed. If this cost is lower than the previously recorded g(m), the record is updated and m is inserted (or re-prioritized) in the open set. The Euclidean distance from m to the goal vertex serves as the admissible heuristic h(m). The process terminates when the goal vertex is extracted from the open set, at which point the efficient path is reconstructed by tracing parent pointers.

The pseudocode for the A* procedure as applied in this implementation is conceptually described in Algorithm 1 below.

Algorithm 1: a* pathfinding on navigation graph

Step	Operation
1	Initialize open set with start vertex s; g(s) = 0; f(s) = h(s)
2	While open set is not empty:
3	n ← vertex in open set with minimum f(n)
4	If n = goal, reconstruct and return path
5	Move n to closed set
6	For each neighbor m of n:
7	If m in closed set, skip
8	g'(m) = g(n) + w(n, m)
9	If g'(m) < g(m): update g(m), f(m); record parent(m) = n
10	If open set empty with no path found, return failure

sources: Author illustrations

In practical game environments, A* typically explores fewer vertices than Dijkstra's algorithm because the heuristic guides the search toward the target location.

D. Zombie Navigation System Design

The zombie navigation system is structured as a modular pipeline consisting of three primary components: the Player entity, the Pathfinding Module (AStarPathfinder), and the Zombie entity. The system workflow is as follows:

Player Position → AStarPathfinder Module → Path node sequence → Zombie Movement

The **Player** entity represents the moving target. Its position is continuously tracked by nearby zombie entities, triggering path recomputation whenever the player's position changes beyond a defined threshold distance.

The **AStarPathfinder Module** receives the current position of the zombie and the current position of the player as inputs. It maps these positions to the nearest graph vertices, executes the A* search, and returns the resulting path as an ordered sequence of grid cells.

The **Zombie Entity** consumes the path node sequence and moves toward each successive path node in order. Upon reaching a path node, the zombie advances to the next one in the sequence. This continues until the zombie reaches the player's vicinity or a path update is triggered.

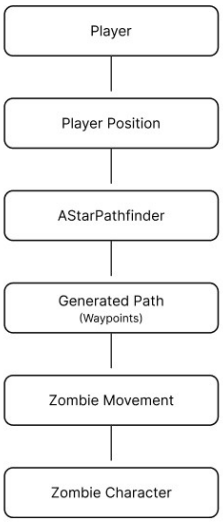


Fig. 6. Architecture of the graph-based zombie navigation system. The AStarPathfinder module computes navigation paths that are subsequently followed by zombie entities.
sources: Author illustrations

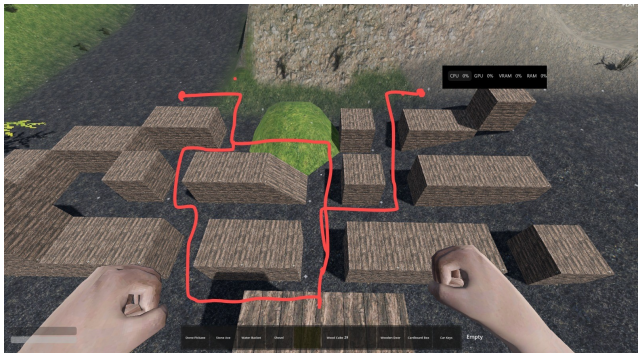


Fig. 7. Simple system of the zombie navigation pipeline. The AStarPathfinder module decouples the path computation logic from the zombie movement logic, facilitating modular extension.
sources: Author illustrations

E. Experimental Scenarios

To evaluate the proposed navigation system, two test scenarios are prepared and compared within the game environment.

Scenario 1: Direct Chase. In this baseline approach, the zombie computes a direction vector pointing from its current position directly toward the player's position and moves along that vector each frame. No graph representation is used. This approach is simple to implement but cannot account for obstacles or determine alternative routes.

Scenario 2: Graph-Based Pathfinding. In this approach, the zombie navigation is driven by the AStarPathfinder module. The zombie follows the path node sequence produced by A* rather than pursuing the player directly. The navigation behavior is observed across multiple test configurations involving different obstacle placements and player positions.

In both scenarios, the observable outcomes of interest are: whether zombies successfully reach the player, whether zombies become blocked by obstacles, and the overall quality and naturalness of the navigation paths produced.

IV. IMPLEMENTATION AND RESULTS

A. Development Environment

The game was developed using Godot Engine 4. The game environment consists of a three-dimensional survival world populated by zombie entities and a controllable player character. The implementation focuses on integrating the custom graph-based pathfinding system into the zombie navigation behavior.

The AStarPathfinder class was implemented as a dedicated GDScript module within the Godot project. This module encapsulates the graph structure (vertex list, adjacency representation, and edge weights), the A* search procedure, and the point retrieval interface.

B. Navigation Graph Construction

The navigation graph is generated implicitly from a uniform grid representation of the game environment. Each traversable grid cell corresponds to a graph vertex and neighboring cells are connected through graph edges. Each grid node is registered as a vertex in the graph. Edges are established between vertices whose Grid traversal costs are used as edge weights, with different costs assigned to orthogonal and diagonal movements.

This construction process ensures that the graph accurately reflects the navigable topology of the game environment. Regions blocked by static obstacles are excluded from the edge set, so the pathfinding module cannot produce routes that pass through impassable geometry.

The resulting graph covers the playable area of the game level, providing sufficient vertex density to enable smooth navigation along corridor-like and open-space paths.

C. Pathfinding Implementation

The AStarPathfinder module is invoked by each zombie entity at regular intervals or whenever the player's position changes significantly. Given the start and goal positions, the module identifies the nearest graph vertices, runs the A* search, and returns the computed path node sequence.

The zombie movement system then iterates over the returned sequence, computing a movement direction toward the current target path node. Upon arrival at a path node (within a defined proximity threshold), the zombie advances to the next path node. This path-following behavior continues until the path is exhausted or a new path is requested.

D. Comparison of Navigation Behaviors

The behavioral differences between direct-chase and graph-based navigation are qualitatively summarized in Table III.

TABLE II. Property Comparison of Navigation Approaches

Property	Direct Chase	Graph-Based (A*)
Implementation Complexity	Low	Moderate
Obstacle Handling	None	Supported via graph edges
Path Optimality	N/A (direct vector)	Optimal path under the graph representation
Navigation Realism	Low (stuck at obstacles)	Higher (path node-guided)
Computational Cost	Very low (one vector op)	Higher (graph search)
Scalability to Complex Levels	Poor	Good

sources: Author illustrations

E. Experimental Results and Discussion

Qualitative evaluation of the two navigation scenarios reveals clear behavioral differences between the direct-chase and graph-based approaches.

In Scenario 1 (direct chase), zombies consistently moved in a straight line toward the player's position. When an obstacle was located between the zombie and the player, the zombie pressed continuously against the obstacle surface without finding an alternative route. This behavior frequently resulted in the zombie becoming effectively stationary despite continued movement computation.

In Scenario 2 (graph-based pathfinding), the navigation system generated structured path node sequences that guided zombie movement around obstacles. Zombies were observed to navigate along corridors, bypass barriers, and reach the player's position through indirect but traversable paths. The overall movement appeared more purposeful and contextually appropriate to the game environment.

Observations from the experimental scenarios also indicate that the quality of navigation is sensitive to the density and placement of grid cells. In regions with sparse vertex coverage, the computed paths may not closely follow the ideal route. This suggests that careful graph construction is an important factor in deployment quality, consistent with observations in the broader game AI literature [6].

The modular architecture of the AStarPathfinder module supports future extensions. Dynamic obstacle registration, variable edge weights based on terrain type, and multi-agent path coordination are all feasible additions within the existing framework. These extensions align with established directions in game AI research [2][6].

Overall, the experimental results confirm that representing the game environment as a weighted graph and applying the A* algorithm produces navigation behavior that is more obstacle-aware to the direct-chase baseline. The implementation demonstrates that core graph theory concepts from discrete mathematics translate directly into practical and effective game AI solutions.

Table III. Qualitative Experimental Results

Scenario	Obstacle Present	Reach Player	Observed Behavior
Direct Chase	Yes	No	Stuck at obstacle
Graph-Based (A*)	Yes	Yes	Alternative path generated
Direct Chase	No	Yes	Straight-line movement
Graph-Based (A*)	No	Yes	Structured and obstacle-aware movement

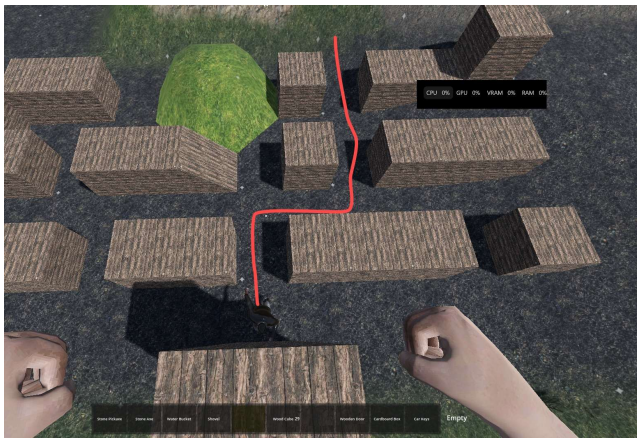


Fig. 8. Example navigation paths produced by graph-based A* (solid line) versus direct chase (dashed line). The graph-based path routes around the obstacle, while the direct-chase vector is blocked.
sources: Author illustrations

V. CONCLUSION AND FUTURE WORK

A. Conclusion

This study demonstrates the effective application of graph theory to the design and implementation of a navigation system for zombie AI characters in DeadZone: Uprising, a three-dimensional survival game developed with the Godot Engine. By representing the game environment as a weighted graph of grid cells, the zombie navigation problem is transformed into a shortest-path search problem, which is solved using the A* pathfinding algorithm.

The A* algorithm evaluates candidate vertices using the combined cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the exact path cost from the start and $h(n)$ is an admissible Euclidean distance heuristic. The resulting path is converted

into a sequence of path nodes that guides zombie movement toward the player in a structured, obstacle-aware manner.

Qualitative comparison between direct-chase and graph-based navigation demonstrates that the graph-based approach produces more obstacle-aware and contextually appropriate movement behavior. Zombies following graph-based paths can navigate around obstacles and traverse complex environments, behaviors that the direct-chase baseline cannot support.

This study confirms that discrete mathematics concepts, and graph theory in particular, are not merely abstract theoretical tools but have direct and practical application in software engineering contexts such as game development. The mapping between mathematical formalism and implementable algorithms is clear and direct, making graph-based game AI a valuable domain for applied discrete mathematics education.

B. Future Work

Several directions for future development are identified. First, the navigation graph can be extended to cover larger and more complex game levels, including multi-story environments that require vertical edge traversal. Second, dynamic graph updates can be integrated to handle destructible obstacles or temporarily blocked paths, requiring efficient incremental graph modification.

Third, additional pathfinding algorithms, including Dijkstra's algorithm and bidirectional A*, may be implemented and compared with the current A* approach in terms of path quality and computational performance across varied graph topologies. Fourth, multi-agent navigation coordination can be investigated to prevent zombie entities from converging on identical paths, improving the collective realism of the enemy swarm behavior.

Finally, the navigation graph construction process may be automated through procedural placement based on obstacle geometry, reducing the manual effort required when deploying the system to new game levels.

VIDEO LINK

https://youtu.be/l8t0LxYezfo

ACKNOWLEDGMENT

The author would like to acknowledge the creators of the TPS Character Demo asset available through the Godot Asset Library, which provided character models and animations used in the prototype demonstration.

REFERENCES

[1] R. Diestel, Graph Theory, 5th ed. Berlin: Springer, 2017.

[2] M. Buckland, Programming Game AI by Example. Plano, TX: Wordware Publishing, 2005.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. Cambridge, MA: MIT Press, 2009.

[4] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Hoboken, NJ: Pearson, 2020.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE Trans. Syst. Sci. Cybern., vol. 4, no. 2, pp. 100–107, Jul. 1968.

[6] D. Champandard, Behavioral Mathematics for Game AI. Boston, MA: Charles River Media, 2009.

[7] D. E. Knuth, The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd ed. Reading, MA: Addison-Wesley, 1997.

[8] K. H. Rosen, Discrete Mathematics and Its Applications, 8th ed. New York, NY: McGraw-Hill Education, 2019.

[9] R. Munir, Matematika Diskrit, 6th ed. Bandung, Indonesia: Informatika, 2019.

[10] Godot Engine Documentation, "Navigation and Pathfinding," Godot Engine. [Online]. Available: https://docs.godotengine.org. [Accessed: Jun.18,2026].

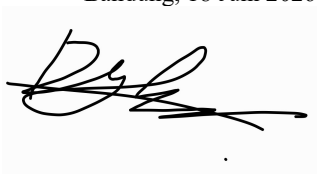
[11] N. J. Nilsson, Principles of Artificial Intelligence. Palo Alto, CA: Tioga Publishing Company, 1980.

[12] S. Rabin, AI Game Programming Wisdom 4. Boston, MA: Charles River Media, 2008.

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, 18 Juni 2026



Dylan Frederico Ketaren 13525036