Gazi Üniversitesi
Fen Bilimleri Dergisi
PART C: TASARIM VE TEKNOLOJİ

Gazi University
Journal of Science
PART C: DESIGN AND TECHNOLOGY

GU J Sci, Part C, 13(1): 187-199 (2025)

Design of Simulation Program for Analysis of Shortest Path Algorithms in Grid-Based Path Planning

İbrahim ŞANLIALP[1*], İbrahim YANDI[1]

[1]*Kırşehir Ahi Evran University, Faculty of Engineering and Architecture, Department of Computer Engineering, Kırşehir, Turkey*

Article Info

Research article
Received: 20/10/2024
Revision: 21/12/2024
Accepted: 25/01/2025

Keywords

Optimal Path Planning
Simulation Program
Shortest Path Algorithms
Grid-based Path Planning
Unity 3D

Makale Bilgisi

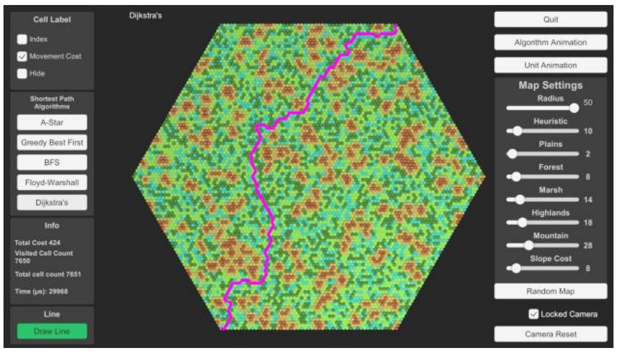
Araştırma makalesi
Başvuru: 20/10/2024
Düzeltme: 21/12/2024
Kabul: 25/01/2025

Anahtar Kelimeler

Optimal Yol Planlama
Simülasyon Programı
En Kısa Yol Algoritmaları
Izgara Tabanlı Yol Planlama
Unity 3D

Graphical/Tabular Abstract (Grafik Özet)

In this study, a simulation program was developed using the Unity 3D game engine and the C# programming language to determine the shortest path between cells with various terrain types and elevation levels on a hexagonal grid-based map. Figure A shows an image of the developed user interface. / Bu çalışmada, altıgen ızgara tabanlı bir harita üzerinde farklı arazi türleri ve yükseklik seviyelerine sahip hücreler arasındaki en kısa yolu belirlemek amacıyla Unity 3D oyun motoru ve C# programlama dili kullanılarak bir simülasyon programı geliştirilmiştir. Şekil A, geliştirilen kullanıcı arayüzüne ait bir görseli göstermektedir.



Info
Total Cost 424
Visited Cell Count 7650
Total cell count 7651
Time (µs): 29968

Map Settings
Radius 50
Heuristic 10
Plains 2
Forest 8
Marsh 14
Highlands 18
Mountain 28
Slope Cost 8

***Figure A**: Designed user interface / **Şekil A:** Tasarlanan kullanıcı arayüzü*

Highlights (Önemli noktalar)

- *The simulation program has been designed for the analysis of shortest path algorithms. / En kısa yol algoritmalarının analizi için bir simülasyon programı tasarlanmıştır.*
- *Heuristic and non-heuristic algorithms were used to find the shortest path, and these algorithms were compared. / En kısa yolu bulmak için için sezgisel ve sezgisel olmayan algoritmaları kullanılmıştır ve bu algoritmalar karşılaştırılmıştır.*
- *Evaluations demonstrate that they can aid in selecting the most appropriate algorithm for optimal path planning. / Değerlendirmeler, optimum yol planlaması için en uygun algoritmanın seçilmesine yardımcı olabileceklerini göstermektedir.*

***Aim (Amaç):** The aim of this study is to analyze algorithms to find the shortest path between cells with various terrain types and elevation levels on a hexagonal grid-based map. / Bu çalışmanın amacı, altıgensel grid tabanlı bir harita üzerinde farklı arazi tipleri ve yükseklik seviyelerine sahip hücreler arasındaki en kısa yolu bulmak için algoritmaları analiz etmektir.*

***Originality (Özgünlük):** The originality of this study lies in the development of a unique simulation program that evaluates various shortest path algorithms on complex hexagonal grid-based maps. / Bu çalışmanın özgünlüğü, karmaşık altıgen ızgara tabanlı haritalar üzerinde çeşitli en kısa yol algoritmalarını değerlendiren özgün bir simülasyon programının geliştirilmesidir.*

***Results (Bulgular):** The results are summarized in three main points: (1) heuristic algorithms demonstrated high performance in terms of computation time and the number of cells visited; (2) the increase in the number of cells visited by the heuristic algorithms was smaller compared to non-heuristic algorithms; (3) heuristic algorithms did not achieve optimum results in terms of traversal cost. / Sonuçlar üç ana noktada özetlenmektedir: (1) Sezgisel algoritmalar, hesaplama süresi ve ziyaret edilen hücre sayısı açısından yüksek performans sergilemiştir; (2) Sezgisel algoritmaların ziyaret ettiği hücre sayısındaki artış, sezgisel olmayan algoritmalara kıyasla daha düşük olmuştur; (3) Sezgisel algoritmalar, geçiş maliyeti açısından optimum sonuçlara ulaşamamıştır.*

***Conclusion (Sonuç):** This study emphasizes the importance of optimal path planning, and its results demonstrate that they can assist in selecting the most suitable algorithm for solving specific pathfinding problems. / Bu çalışma, optimal yol planlamanın önemini vurgulamakta ve sonuçları, belirli yol bulma problemlerinin çözümü için en uygun algoritmanın seçilmesine yardımcı olabileceğini göstermektedir.*

*Corresponding author, e-mail: ibrahim.sanlialp@ahievran.edu.tr

DOI: 10.29109/gujsc.1570730

Gazi Üniversitesi
Fen Bilimleri Dergisi
PART C: TASARIM VE TEKNOLOJİ

Gazi University
Journal of Science
PART C: DESIGN AND TECHNOLOGY

http://dergipark.gov.tr/gujsc

Design of Simulation Program for Analysis of Shortest Path Algorithms in Grid-Based Path Planning

İbrahim ŞANLIALP[1*], İbrahim YANDI[1]

[1]*Kırşehir Ahi Evran University, Faculty of Engineering and Architecture, Department of Computer Engineering, Kırşehir, Turkey*

Article Info

Research article
Received: 20/10/2024
Revision: 21/12/2024
Accepted: 25/01/2025

Keywords

Optimal Path Planning
Simulation Program
Shortest Path Algorithms
Grid-based Path Planning
Unity 3D

Abstract

This study focuses on the analysis of algorithms used to find the shortest path between cells with different terrain types and elevation levels on a map comprising hexagonal cells ranging from 91 to 7651. A simulation program was designed for the analysis and developed using the Unity 3D game engine and the C# programming language. Within the scope of the study, an intelligent agent was incorporated into the simulation. The intelligent agent perceives its environment, evaluates terrain type and elevation factors, and attempts to find the shortest path with the lowest traversal cost between two points based on the selected algorithm. The performance of the algorithms was compared in terms of computation time, the number of cells visited, and traversal cost. The results revealed that the heuristic algorithms demonstrated high performance in computation time and the number of cells visited. However, they did not achieve the same level of success in terms of traversal cost. Furthermore, it was concluded that the increase in the number of cells visited by heuristic algorithms was smaller compared to non-heuristic algorithms. The findings of this study highlight the importance of optimal path planning in determining the most effective algorithm under various conditions and provide valuable contributions to developers for applications requiring efficient navigation in complex environments.

Izgara Tabanlı Yol Planlamasında En Kısa Yol Algoritmalarının Analizi İçin Simülasyon Programı Tasarımı

Makale Bilgisi

Araştırma makalesi
Başvuru: 20/10/2024
Düzeltme: 21/12/2024
Kabul: 25/01/2025

Anahtar Kelimeler

Optimal Yol Planlama
Simülasyon Programı
En Kısa Yol Algoritmaları
Izgara Tabanlı Yol Planlama
Unity 3D

Öz

Bu çalışma, 91 ila 7651 arasında değişen altıgen hücrelerden oluşan bir haritada farklı arazi tipleri ve yükseklik seviyelerine sahip hücreler arasındaki en kısa yolu bulmak için kullanılan algoritmaların analizine odaklanmaktadır. Analiz için bir simülasyon programı tasarlanmış ve Unity 3D oyun motoru ile C# programlama dili kullanılarak geliştirilmiştir. Çalışma kapsamında simülasyona bir akıllı ajan entegre edilmiştir. Akıllı ajan, çevresini algılar, arazi türü ve yükseklik faktörlerini değerlendirir ve seçilen algoritmaya göre iki nokta arasında en düşük geçiş maliyetine sahip en kısa yolu bulmaya çalışır. Algoritmaların performansı, hesaplama süresi, ziyaret edilen hücre sayısı ve geçiş maliyeti açısından karşılaştırılmıştır. Sonuçlar, sezgisel algoritmaların hesaplama süresi ve ziyaret edilen hücre sayısı açısından yüksek performans gösterdiğini ortaya koymuştur. Ancak, geçiş maliyetleri açısından aynı başarıyı sağlayamadıkları görülmüştür. Ayrıca, sezgisel algoritmalar tarafından ziyaret edilen hücre sayısındaki artışın sezgisel olmayan algoritmalara kıyasla daha küçük olduğu sonucuna varılmıştır. Bu çalışmanın bulguları, çeşitli koşullar altında en etkili algoritmayı belirlemede optimum yol planlamanın önemini vurgulamakta ve karmaşık ortamlarda verimli gezinme gerektiren uygulamalar için geliştiricilere değerli katkılar sunmaktadır.

1. INTRODUCTION (GİRİŞ)

Optimization is a discipline that is commonly used to address complex problems in a variety of application areas [1]. Map-based pathfinding problems have a wide range of applications and play a crucial role in game development, optimization, and artificial intelligence research [2, 3]. However, pathfinding can be resource-intensive, especially when dealing with complex maps [4]. Algorithms such as A-Star and Dijkstra’s are commonly used; however, on large, intricate maps, they can be

*Corresponding author, e-mail: ibrahim.sanlialp@ahievran.edu.tr
DOI: 10.29109/gujsc.1570730

computationally demanding. For example, while A-Star is highly efficient in finding optimal paths, it incurs higher processing costs. Dijkstra's algorithm, though it consistently finds the shortest path by exploring all nodes, requires significant computational power, especially when using large datasets [5]. As a result, researchers have developed optimization techniques to reduce these costs in game development, optimization, and artificial intelligence research.

Unity [6] is a versatile and innovative game engine that supports real-time 3D animations and provides interactive content to users [7]. It enables seamless integration of movements, environmental elements, and user interfaces into game objects [8] through drag-and-drop or as programmable variables in C# [9]. Furthermore, hexagonal grid maps are more information-rich than traditional square grids. This framework adapts to a variety of scene requirements and offers efficiency, flexibility, and homogeneity [2, 10].

Several studies have focused on grid-based pathfinding problems. In one such study, Barbour emphasized the efficiency of hexagonal grids and proposed a new pathfinding technique that improves runtime and reduces algorithmic complexity [11]. This method enhances the degree of movement of a unit while reducing the costs typically associated with other techniques. Bailey et al. conducted a path-length analysis for grid-based path planning and demonstrated that as node connectivity increases, the percentage difference between a grid path and the real shortest path decreases [12]. In another study, Yang et al. addressed the issue of origin-destination matrix estimation by developing a hexagon-based dynamic graph convolutional network that generates distinct hexagon-based road graphs throughout different time periods [13].

The purpose of this study is to analyze algorithms to find the shortest path between cells with various terrain types and elevation levels on a hexagonal grid-based map. Unity 3D game engine and C# are utilized during the development of the simulation program. The proposed program finds neighborhood connections of each hexagonal cell, which, in turn, defines the movement costs between them. In addition, it helps identify efficient pathfinding strategies for hexagonal grid environments. The study makes a significant contribution by introducing a simulation tool that evaluates various shortest path algorithms on complex hexagonal grid-based maps.

The second part of the study is about the hexagonal grid-based map structure, and general information about shortest path algorithms is given in Section 3. Section 4 explains the design of the simulation program developed within this study. The analysis and results are explained in Section 5, followed by a discussion in Section 6. The last section presents the conclusions.

2. HEXAGONAL GRID-BASED MAP STRUCTURE (ALTIGEN AĞ TABANLI HARİTA YAPISI)

Advances in computer graphics and game development have enabled new spatial representation and design possibilities. A prominent example is the hexagonal grid-based map, which uses six-sided polygons to achieve more efficient area coverage and enhanced visual appeal [2]. This technique offers a significant alternative to conventional square grid layouts by providing more accurate distance metrics and facilitating smooth user interactions [14, 15].

Hexagonal geometry is suitable for analyzing map usage in experimental research. Hexagons efficiently cover large areas without gaps. Their symmetrical structure enhances spatial tasks like estimating object positions, measuring distances, and determining directions [16]. Figure 1 shows a map design based on a hexagonal grid and a single row path.

The simulation program developed using the Unity 3D game engine features a map structure composed of hexagonal cells arranged in a regular 3D grid. Each cell is connected to six neighboring cells, forming a hexagonal pattern. The neighboring cell information is assigned to each cell, simplifying the tracking and management of connections and transitions. For example, when moving between cells, identifying adjacent cells enables pathfinding algorithms to operate more efficiently [13].

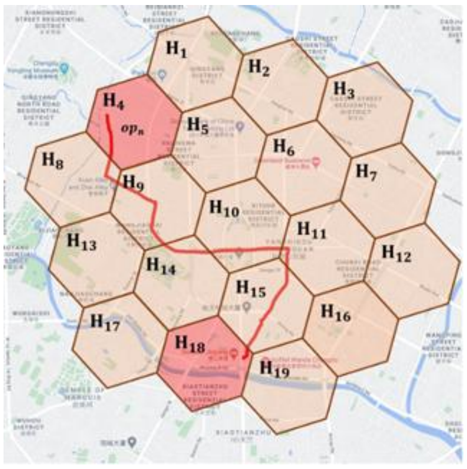


Figure 1. Hexagonal grid-based map design (Altıgen ızgara tabanlı harita tasarımı) [13]

2.1. Cube Coordinates (Küp Koordinatları)

The hexagonal grid structure consists of three main axes: x, y, and z. Unlike square grids, which have two axes, hexagonal grids use a symmetrical coordinate system. There is a symmetrical relationship between these axes [17]:

$$x + y + z = 0 \tag{1}$$

The equations describing the relationship between the coordinates and column widths is given:

$$x = t - r/2 \tag{2}$$

$$y = -(t - r/2 + r) \tag{3}$$

$$z = r \tag{4}$$

where t represents the number of columns and width of the hexagonal grid, r represents the number of rows, and height, x, y, and z are the coordinates of the cube; the sum of these three coordinates is 0 [2].

Cube coordinate system enables more effective management of hexagonal grid structures, improves algorithm performance, and enhances user experience. In addition, the flexibility and simplicity of cube coordinate calculations play key roles in creating and managing hexagonal maps. As a result, cube coordinates are applied to the cells when designing the hexagonal grid-based map in this study. Figure 2 illustrates the cube coordinate system used in this study [18].

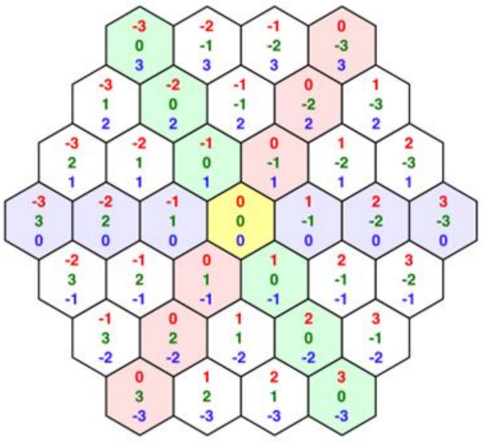


Figure 2. Cube coordinate system used in the study (Çalışmada kullanılan küp koordinat sistemi) [18]

3. SHORTEST PATH ALGORITHM (EN KISA YOL ALGORİTMASI)

A computer application employs a shortest path algorithm, along with a plotting component, to determine the shortest route between two points, from the source to the destination. Shortest path algorithms are essential to identify the shortest and most optimal paths. Many critical applications, including video games, robotics, GPS, and simulations, depend on these algorithms [19]. This study discusses the shortest path algorithms in simulation program design. The shortest path problems are solved using A-Star, Greedy Best-First Search, Breadth-First Search, Floyd-Warshall, and Dijkstra's algorithms. In the study, A-Star and Greedy Best-First Search represent heuristic algorithms, while Breadth-First Search, Floyd-Warshall and Dijkstra's algorithms represent non-heuristic algorithms. These algorithms are explained in order.

3.1. A-Star Algorithm (A-Yıldız Algoritması)

The A-Star algorithm is a popular heuristic search algorithm in pathfinding and graph traversal because it efficiently determines the shortest route between two nodes [20, 21]. This algorithm evaluates positions within the search space to identify the optimal path from the beginning point to the target node [22]. A-Star employs an evaluation function to guide its search. The function is defined as [23]:

$$f(n) = g(n) + h(n) \tag{5}$$

where $g(n)$ represents the cost from the starting point to the current node n, and $h(n)$ is the estimated cost from the current node n to the goal node; $f(n)$ is the total estimated cost of the path through node n [24]. The objective of the A-Star algorithm is to find a path that minimizes $f(n)$. The A-Star algorithm employs a heuristic function to calculate the cost h of moving from the current node to the goal node and ensures that the search is both accurate and efficient using the heuristic function $h(n)$ [25].

3.2. Greedy Best-First Search Algorithm (En İyi Öncelikli Arama Algoritması Algoritması)

Greedy Best-First Search (GBFS) algorithm is a heuristic search algorithm used to explore paths in a search space, which is often represented as a tree [26]. It prioritizes nodes based on an evaluation function, selecting the most favorable nodes first to efficiently reach the goal [27]. When exhaustive exploration is too costly, the GBFS algorithm applies a greedy heuristic to narrow down the number of paths, frequently discarding less promising ones [26, 28].

Let $\langle S, s_I, S_g, scor\rangle$ be a state space, where S is a finite collection of states, *scor* is the successor function, $s_I \in S$ is the beginning state, and $Sg \subseteq S$ is the set of target states. The algorithm defines the state space and applies the heuristic function h to the states in S. The GBFS algorithm generates the initial state s_I, iteratively expands the generated but unexpanded states, and stops when expanding a goal state from Sg [29]. The preference is to expand goal states from Sg if they are generated; otherwise, it expands the state s with the minimum $h(s)$ among all generated but unexpanded states. In GBFS, state s is "opened" when it is generated for the first time and "closed" when it is expanded. The GBFS process involves generating successors of the current state and adding those that are not yet open or closed to the open list. Once a state is expanded, it is transferred from an open list to a closed list [28].

3.3. Breadth First Search Algorithm (Sığ Öncelikli Arama Algoritması)

The Breadth First Search (BFS) algorithm gradually broadens the solution space and investigates all possible outcomes at each stage [30]. BFS is commonly used to discover the shortest path from a single source in an unweighted graph [31]. BFS operates by visiting all nodes at level n before progressing to the next nodes at level $n+1$. The search begins at the root node and traverses nodes from left to right at each level, advancing to subsequent levels in a systematic manner until a solution is discovered [32,33].

One notable advantage of BFS is its ability to avoid deadlocks; it inevitably finds a solution if it exists. In addition, when multiple solutions are present, the BFS ensures that the minimal solution is identified. However, a significant drawback of this method is its substantial memory requirement, as it needs to store all nodes within the search tree. In addition, BFS can be time-intensive because it must evaluate all nodes at level n before proceeding to find a solution at level $n+1$[30,32,33].

3.4. Floyd-Warshall Algorithm (Floyd-Warshall Algoritması

The Floyd-Warshall algorithm is a dynamic method for finding the shortest paths between all pairs of nodes in a directed graph [34,35]. This algorithm solves the problem by using previous solutions that are interconnected, which allows for multiple possibilities. Moreover, Floyd-Warshall algorithm allows for the presence of negative weights on edges, provided there are no negative weight cycles in the graph [35,36].

Given a graph $G=(V,E)$, where V and E represent the set of vertices and the set of edges with weights, respectively, the algorithm calculates the minimum weight path between each pair of vertices. The weights are denoted by $w(e)$. The sum of edge weights along a path gives the total path weight [36,37]. This algorithm creates a distance matrix M, with each entry representing the shortest distance from vertex i to vertex j. $M[i][j]$ is initially set to the weight of edge (i,j) if one exists or infinity otherwise, with the exception of $M[i][i]$, which is set to zero. The algorithm then iteratively updates this matrix by taking each vertex as an intermediate point and testing whether a shorter path exists through that intermediate vertex [37].

3.5. Dijkstra’s Algorithm (Dijkstra Algoritması)

Dijkstra's algorithm solves the problem of determining the shortest path between two nodes in a graph, and it was introduced by Holland in 1970 [19]. This algorithm is designed to find the optimal path. As it searches for the minimum-cost path by evaluating all possible routes starting from the initial point, the search area expands outward in concentric circles. Therefore, it suffers from low search efficiency and extended search times, particularly when the distance to the destination is significant. Both Dijkstra's algorithm and the A-Star algorithm are among the most commonly used shortest-route optimization methods [39,40].

4. DESIGN OF SIMULATION PROGRAM (SİMÜLASYON PROGRAMI TASARIMI)

The simulation program design consists of five stages. In the first stage, the definition and algorithm of the simulation program are created. In the second stage, the terrain structure and starting and ending points are modeled as hexagonal maps using Unity 3D. In the third stage, the shortest path algorithms are programmed using the C# programming language. In the final stage, the user interface is designed, and the simulation program is tested. Each stage is presented sequentially.

4.1. Simulation Program Definition (Simülasyon Programı Tanımı)

In the developed simulation program, a geographical area is represented by a map comprising hexagonal cells. This map consists of cells representing plains, forests, swamps, plateaus, mountain terrains and elevation levels. Each cell on the map is connected to other cells through neighborhood ties, and these ties determine the cost of movement between cells. An intelligent agent integrated into this hexagonal map structure is added to the proposed program. The intelligent agent evaluates its environment by calculating it according to terrain type and elevation parameters and attempts to find the shortest path between two points using the selected shortest pathfinding algorithm. The proposed program solves the problem of finding the shortest path between two given points in this way. The developed simulation program helps determine the most effective pathfinding strategies for the complex structure of hexagonal cells and evaluates the performance of the shortest pathfinding algorithms. The workflow diagram of the developed simulation program is shown in Figure 3.

4.2. Map Modeling (Harita Modelleme)

A variety of map sizes are used at this stage, each revealing different terrain types and slope costs. The map sizes in the simulation program consist of 91, 331, 721, 1261, 1951, 2791, 3781, 4921, 6211, and 7651 cells. In addition, the maps include slope costs. A hexagonal map is constructed by arranging hexagonal cells in a regular grid in 3D space, where each cell is connected to six neighboring cells. The hexagonal map containing the terrain types of the cells created for this study is shown in Figure 4.

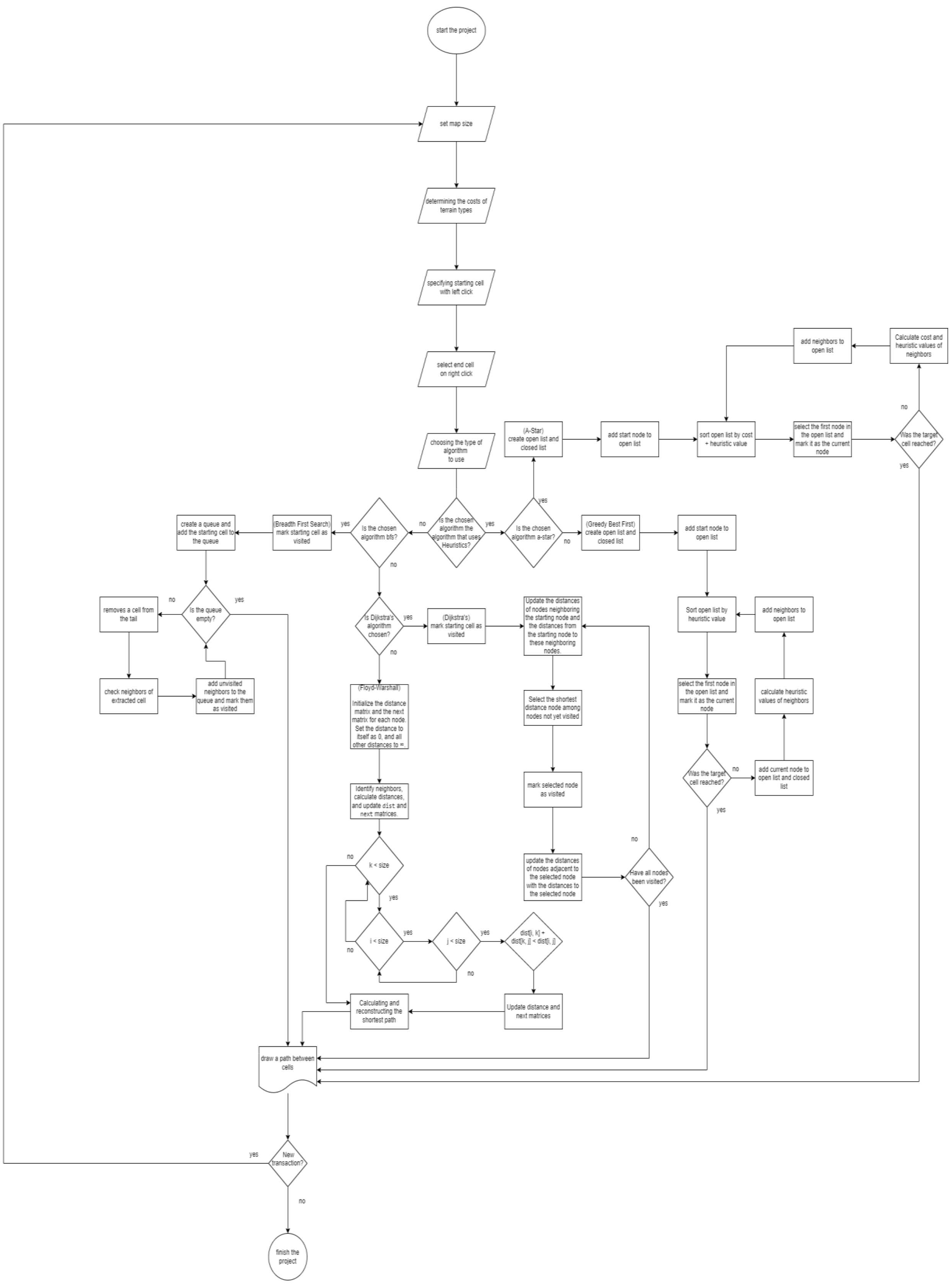

start the project
set map size
determining the costs of terrain types
specifying starting cell with left click
select end cell on right click
choosing the type of algorithm to use
Is the chosen algorithm the algorithm that uses Heuristics?
Is the chosen algorithm bfs?
(Breadth First Search) mark starting cell as visited
create a queue and add the starting cell to the queue
Is the queue empty?
removes a cell from the tail
check neighbors of extracted cell
add unvisited neighbors to the queue and mark them as visited
Is Dijkstra's algorithm chosen?
(Dijkstra's) mark starting cell as visited
Update the distances of nodes neighboring the starting node and the distances from the starting node to these neighboring nodes.
Select the shortest distance node among nodes not yet visited
mark selected node as visited
update the distances of nodes adjacent to the selected node with the distances to the selected node
Have all nodes been visited?
(Floyd-Warshall) Initialize the distance matrix and the next matrix for each node. Set the distance to itself as 0, and all other distances to ∞
Identify neighbors, calculate distances, and update dist and next matrices.
k < size
i < size
j < size
dist[i, k] + dist[k, j] < dist[i, j]
Update distance and next matrices
Calculating and reconstructing the shortest path
Is the chosen algorithm a-star?
(A-Star) create open list and closed list
add start node to open list
sort open list by cost + heuristic value
select the first node in the open list and mark it as the current node
Was the target cell reached?
Calculate cost and heuristic values of neighbors
add neighbors to open list
(Greedy Best First) create open list and closed list
add start node to open list
Sort open list by heuristic value
select the first node in the open list and mark it as the current node
Was the target cell reached?
add current node to open list and closed list
calculate heuristic values of neighbors
add neighbors to open list
draw a path between cells
New transaction?
finish the project
yes
no

Figure 3. Flow chart of the simulation program (Simülasyon programının akış şeması)

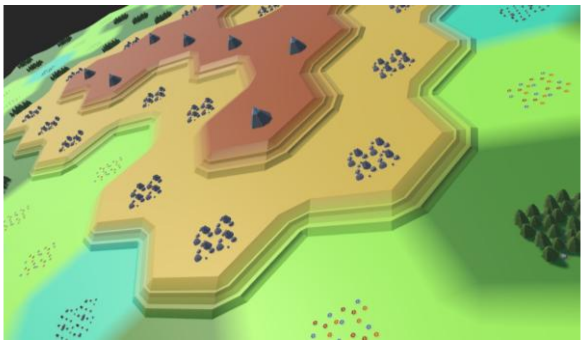


Figure 4. Experimental setup and workflow (Deney tasarımı ve iş akışı)

4.3. Intelligent Agent (Akıllı Ajan)

The intelligent agent is programmed using the C# programming language in Unity and used to perform simulations on a hexagonal map. This intelligent agent perceives its environment, evaluates terrain type and elevation factors, and attempts to find the shortest path between two points using the selected shortest path algorithm. The representations of the starting and ending points are shown in Figure 5. Here, the starting point represents the intelligent agent model, depicted in the red hexagonal cell. The other end point represents the tent model and is in the blue hexagonal cell.

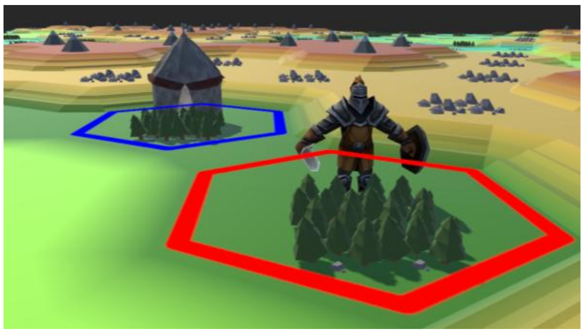


Figure 5. Displaying representations of starting and ending points (Başlangıç ve bitiş noktalarının temsillerinin görüntülenmesi)

To allow the created intelligent agent model to work, an animator is added to the model in Unity, and it is allowed to work on the paths obtained by associating it with the shortest path algorithms.

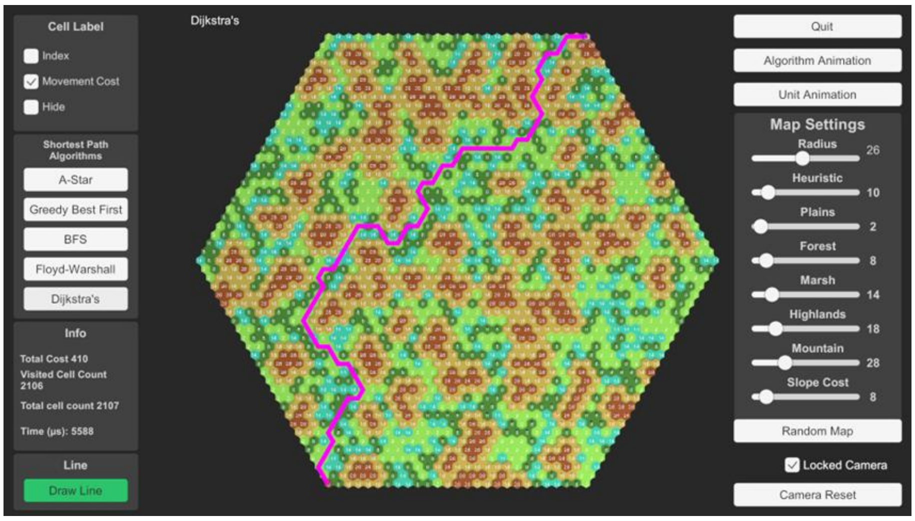
4.4. User Interface (Kullanıcı Arayüzü)

The user interface is designed for the developed simulation program. This interface is developed using the C# programming language and Unity 3D game engine. With the help of the developed user interface, users can visualize the analysis of the shortest path algorithms and perform simulations. An image taken from the developed user interface unit is shown in Figure 6.

The user interface of the simulation program is composed of several control units, including the cell labeling panel, the shortest path algorithms panel, the information panel, the line panel, the algorithm and navigated cells animation, the parameter editing required to create a hexagonal map, and an option for exiting the simulation program.

The cell labeling panel provides features for displaying the index and movement cost of cells, where the index uniquely identifies each cell, and the movement cost represents the terrain-dependent traversal cost. These values can be toggled using a checkbox control; selecting the "Hide" option ensures that the information remains concealed when not needed. Furthermore, the shortest path algorithm panel allows users to choose from a variety of algorithms, including A-Star, GBFS, BFS, Floyd-Warshall, and Dijkstra’s.

The starting point is defined by clicking the left mouse button, whereas the ending point is selected using the right mouse button, enabling the computation of the shortest path based on the chosen points and algorithm. In addition, the information panel offers comprehensive details about the simulation, such as the total cost of the calculated path, the number of cells visited during the algorithm’s execution, the total number of cells on the map, and the execution times of the algorithms measured in microseconds. Moreover, the interface includes a button that visually represents the cells visited by the algorithm. It also has a button that animates the movement of the intelligent agent character as it transitions from the starting to the ending cell. To enhance usability, the camera reset control repositions the camera to its initial alignment with the intelligent agent character, while the random map button generates a new hexagonal map based on the specified radius, thereby introducing dynamic map variations.



Info
Total Cost 410
Visited Cell Count 2106
Total cell count 2107
Time (μs): 5588

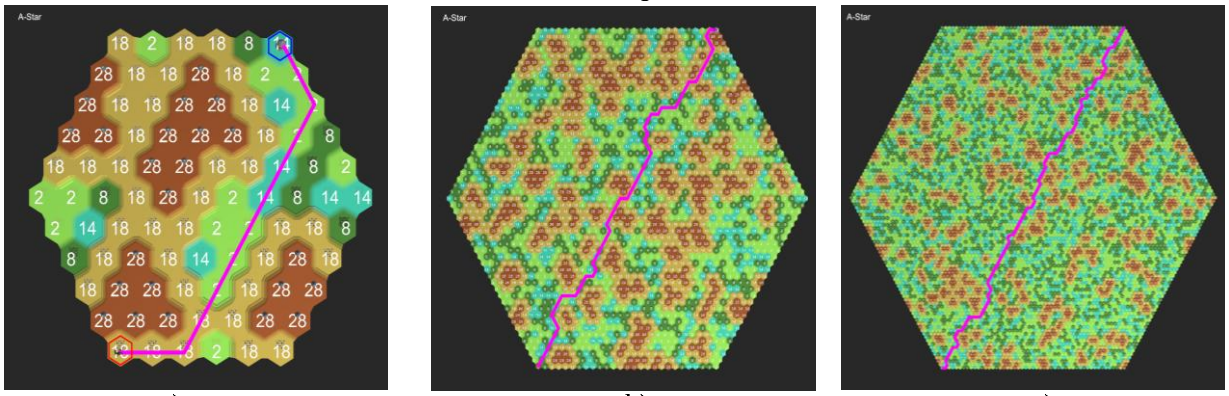
Map Settings
Radius 26
Heuristic 10
Plains 2
Forest 8
Marsh 14
Highlands 18
Mountain 28
Slope Cost 8

Figure 6. Design of the simulation interface (Simülasyon arayüzünün tasarımı)

By integrating these features into a cohesive design, the user interface provides an interactive simulation environment that seamlessly combines visualization and analysis, facilitating an in-depth exploration of various pathfinding algorithms.

5. ANALYSIS AND RESULTS (ANALİZ VE BULGULAR)

The process of creating hexagonal cells in Unity 3D begins by representing each cell with 6 triangles. These triangles form each side of the hexagon, ensuring the geometric integrity of the hexagonal shape. The map creation process follows a spiral pattern from the center outwards. This arrangement is performed based on the radius value determined by the user. In other words, the radius determined by the user directly affects the size of the map and the number of hexagons. In the simulation, the radius values are limited to between 5 and 50, and the analyses were carried out in this range. Tests conducted with different radius values for the performance of the algorithms are presented in Figure 7.



a) b) c)

Figure 7. Testing the performance of algorithms with different radius: A-star a) radius 5 b) radius 26 c) radius 50 (Farklı yarıçaplarla algoritmaların performansının test edilmesi: A-Star a) yarıçap 5 b) yarıçap 26 c) yarıçap 50)

The analyses were carried out in the form of scenario-based performance comparisons for different terrain types, elevation levels, and difficulty levels. The performance of each algorithm was measured 100 times. This study was developed and analyzed in the Unity 2021.3.16f1 environment on a computer: Windows 10, Intel i7, 2.2-GHz CPU, and 32-GB RAM.

Tests were conducted to analyze the shortest path algorithms on maps comprising hexagonal grid-based cells. These tests are scenario-based. In this scenario, the distance between the starting and ending points on the cube grid maps is calculated as the maximum distance. The distance was calculated using the Manhattan metric [41]. The performance results were comparatively analyzed for various radii. All algorithms were compared on maps with an equal number of cells, and each test was run for 100 iterations. Table 1 lists the number of cells visited by the algorithms, the shortest path cost, and the computation time results.

Table 1. Comparison of the algorithm results for 100 runs (100 çalıştırma için algoritma sonuçlarının karşılaştırılması)

	A - Star			Greedy Best First Search			Breadth First Search			Floyd-Warshall			Dijkstra's		
Cell Count	**Cells Visited**	**Cost**	**Time**	**Cells Visited**	**Cost**	**Time**	**Cells Visited**	**Cost**	**Time**	**Cells Visited**	**Cost**	**Time**	**Cells Visited**	**Cost**	**Time**
91	54	238	145	31	242	97	91	242	901	91	222	6.503	90	228	512
331	101	364	410	61	472	174	331	472	5.701	331	260	243.233	330	272	883
721	150	502	633	91	524	309	721	650	25.141	721	372	2.187.507	720	372	1.795
1.261	185	324	938	121	650	500	1.261	524	84.210	1.261	224	10.782.611	1.260	228	2.734
1.951	293	428	2.303	151	792	783	1.951	792	216.216	1.951	314	37.462.657	1.950	314	4.564
2.791	337	520	2.559	181	914	1.392	2.791	914	461.616	2.791	350	101.338.671	2.790	350	6.911
3.781	365	764	3.028	211	1.006	1.290	3.781	1.166	863.089	3.781	472	246.229.475	3.780	472	9.532
4.921	424	654	4.731	241	1.166	2.229	4.921	1.214	1.430.888	4.921	424	532.024.244	4.920	476	16.562
6.211	451	764	5.811	271	1.074	2.666	6.211	1.006	2.334.501	6.211	476	1.062.641.112	6.210	402	26.792
7.651	502	626	5.954	301	1.214	4.633	7.651	1.074	3.540.228	7.651	402	1.933.918.809	7.650	424	28.933

The comparison of the algorithms in terms of transition cost is shown in Figure 8. A comparison of the algorithms in terms of computation time is shown in Figure 9. Figures 8 and 9 give the average values obtained over 100 runs of the algorithms.

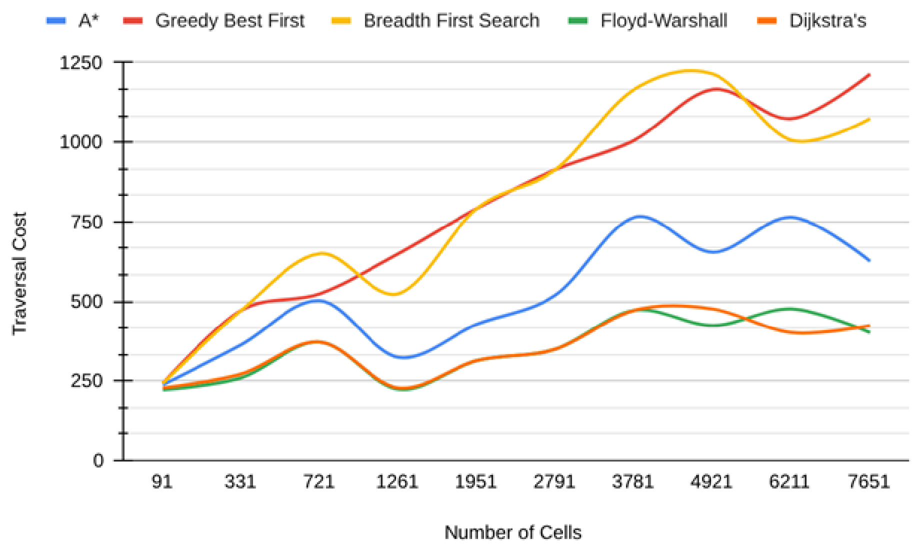


Figure 8. Algorithm comparison of traversal cost (Geçiş maliyetinin algoritma karşılaştırması)

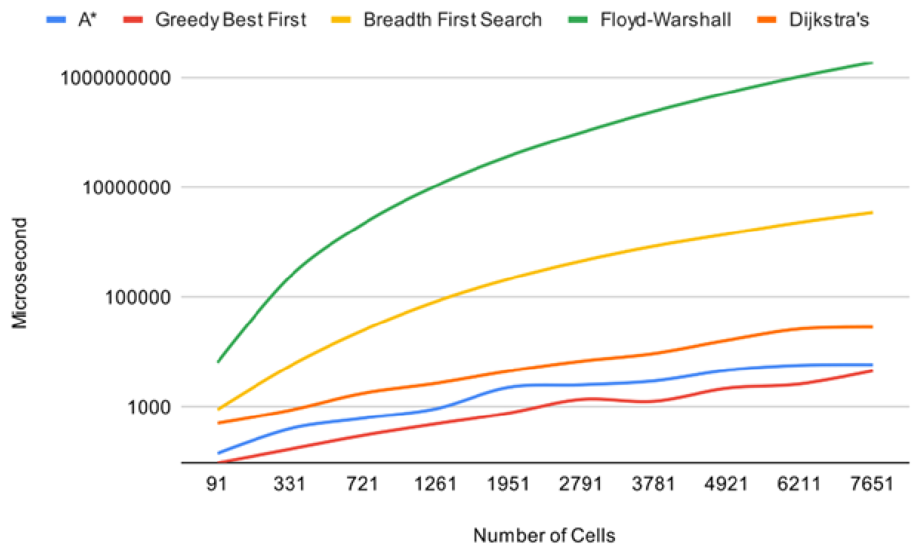


Figure 9. Algorithm comparison of computation times (Hesaplama sürelerinin algoritma karşılaştırması)

In the analyzed algorithms, the number of cells visited increases as the radius expands. However, the increase in the number of cells visited by the heuristic algorithms is smaller than that of the non-heuristic algorithms. Table 1 and Figures 8 and 9 demonstrate that the Floyd-Warshall and Dijkstra algorithms are the most efficient in terms of traversal cost. However, Floyd-Warshall is the least efficient with respect to computation time. The GBFS algorithm is the fastest algorithm in terms of computation time but incurs the highest traversal cost. Similarly, BFS is another algorithm with high traversal costs and visits a large number of cells. In contrast, both A-Star and GBFS visit fewer cells in the generated maps. Among these algorithms, however, A-Star identifies a path with a nearly optimal traversal cost.

6. DISCUSSION (TARTIŞMA)

The A-Star algorithm demonstrates balanced performance by effectively finding the shortest paths while maintaining reasonable computation times. Although the number of cells visited increases with larger radii, the resulting path cost remains relatively low. Its computation time, while moderate, increases proportionally with problem size. In contrast, the Greedy Best-First Search (GBFS) algorithm reaches the target quickly by visiting fewer cells. However, it does not guarantee optimal path costs, making it more suitable for scenarios where fast but suboptimal solutions are acceptable. On the other hand, the Breadth-First Search (BFS) algorithm explores all possible paths, leading to the visitation of a large number of cells. Consequently, it consumes substantial computational resources and exhibits long running times, particularly for larger radii. Due to its exhaustive search nature, the resulting path costs tend to be higher, rendering BFS inefficient for large-scale problems. The Floyd-Warshall algorithm, while capable of determining the shortest paths between all node pairs by visiting a fixed number of cells, suffers from extremely long computation times, especially as problem size or radii increase. Despite its optimal or near-optimal path costs, this characteristic makes it impractical for solving large-scale problems. Lastly, Dijkstra's algorithm systematically evaluates all possible paths by incorporating movement costs and effectively finds the shortest path with the least overall cost. This property makes it a practical solution for addressing large-scale problems.

7. CONCLUSION (SONUÇ)

This study examines algorithms used to find the shortest path between cells with different terrain types and elevation levels on a map consisting of varying numbers of hexagonal cells. The developed simulation program features a unique interface that allows the comparison of shortest path algorithms in a simulated environment. The performance of these algorithms are compared at different radii and problem sizes. As the radius expands, the number of cells visited by each algorithm increases. The analysis results of shortest path algorithms can be summarized in three main points: (1) heuristic algorithms demonstrated high performance in terms of computation time and the number of cells visited; (2) the increase in the number of cells visited by the heuristic algorithms was smaller compared to non-heuristic algorithms; (3) heuristic algorithms did not achieve optimum results in terms of traversal cost.

In conclusion, these evaluations demonstrate that they can assist in selecting the most appropriate algorithm for solving specific pathfinding problems. When considering performance criteria, it becomes evident that a balance must be established between traversal cost and computation time. The algorithm should be chosen based on the requirements of the target problem. Therefore, this study contributes to identifying the most effective algorithm under different conditions and emphasizes the importance of optimal path planning in determining the best algorithm for various scenarios. Additionally, this study provides valuable insights for developers working on applications that require efficient navigation in complex environments.

ACKNOWLEDGMENTS (TEŞEKKÜR)

This study was supported by Kırşehir Ahi Evran University Scientific Research Projects Coordination Unit (Project Number: MMF.A4.24.004). / Bu çalışma Kırşehir Ahi Evran Üniversitesi Bilimsel Araştırma Projeleri Birimi tarafından desteklenmiştir (Proje Numarası: MMF.A4.24.004).

DECLARATION OF ETHICAL STANDARDS (ETİK STANDARTLARIN BEYANI)

The author of this article declares that the materials and methods they use in their work do not require ethical committee approval and/or legal-specific permission.

Bu makalenin yazarı çalışmalarında kullandıkları materyal ve yöntemlerin etik kurul izni ve/veya yasal-özel bir izin gerektirmediğini beyan ederler.

AUTHORS' CONTRIBUTIONS (YAZARLARIN KATKILARI)

***İbrahim ŞANLIALP*:** He directed the study, designed the simulation program, carried out experiments, analyzed the results, wrote and revised the manuscript.

Çalışmayı yönlendirmiş, simülasyon programını tasarlamış, deneyleri gerçekleştirmiş, sonuçları analiz etmiş, makaleyi yazmış ve gözden geçirmiştir.

***İbrahim YANDI*:** He developed the simulation program and significantly contributed to performing experiments and writing.

Simulasyon programını geliştirmiş ve deneylerin gerçekleştirilmesi ile makale yazımına önemli ölçüde katkıda bulunmuştur.

CONFLICT OF INTEREST (ÇIKAR ÇATIŞMASI)

There is no conflict of interest in this study.

Bu çalışmada herhangi bir çıkar çatışması yoktur.

REFERENCES (KAYNAKLAR)

[1] Ruszczynski, A. (2011). Nonlinear optimization. Princeton university press.

[2] Xing, H., Chai, M., Song, Y. (2024). Artificial intelligence pathfinding based on Unreal Engine 5 hexagonal grid map. In 2024 IEEE 4th International Conference on Neural Networks, Information and Communication (NNICE), 1708-1711.

[3] Adaixo, M. C. G. (2014). Influence Map-Based Pathfinding Algorithms in Video Games, M.S. Thesis, Universidade da Beira Interior.

[4] 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.

[5] Johner, R., Lanaia, A., Dornberger, R., Hanne, T. (2022). Comparing the Pathfinding Algorithms A*, Dijkstra's, Bellman-Ford, Floyd-Warshall, and Best First Search for the Paparazzi Problem. In Congress on Intelligent Systems: Proceedings of CIS, 561-576.

[6] Real-Time 3D Development Platform, Unity, https://unity.com/products/unity-engine

[7] Jeong-Shick, Y. (2023). Unity: A Powerful Tool for 3D Computer Animation Production. Journal of the Korea Computer Graphics Society, 29(3), 45-57.

[8] Boyraz G, Kırcı, P. (2019). 3D Game Design with UNITY 3D Game Simulator. International Journal of Multidisciplinary Studies and Innovative Technologies, 3(2), 225-229.

[9] Gunes, M., Dilipak, H. (2020). Shortest Path Approach in Pedestrian Transfers Application in Unity. Gazi Mühendislik Bilimleri Dergisi, 6(2), 111-119.

[10] Zhang, X., Zhang, X. (2022). Based on Navmesh to implement AI intelligent pathfinding in three-dimensional maps in UE4. In Proceedings of the 2022 5th International Conference on Algorithms, Computing and Artificial Intelligence, 1-5.

[11] Barbour Jr, R. D. (2008). Reduction of complexity in path finding using grid-based methods. Faculty of Graduate Studies and Research, University of Regina.

[12] Bailey, J. P., Nash, A., Tovey, C. A., Koenig, S. (2021). Path-length analysis for grid-based path planning. Artificial Intelligence, 301, 103560.

[13] Yang, Y., Zhang, S., Zhang, C., James, J. Q. (2021). Origin-destination matrix prediction via hexagon-based generated graph. In 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), 1399-1404.

[14] Wüthrich, C. A., Stucki, P. (1991). An algorithmic comparison between square-and hexagonal-based grids. CVGIP: Graphical Models and Image Processing, 53(4), 324-339.

[15] Duszak, P. (2022). SLAM on the Hexagonal Grid. Sensors, 22(16), 6221.

[16] Edler, D., Keil, J., Bestgen, A. K., Kuchinke, L., Dickmann, F. (2019). Hexagonal map grids–an experimental study on the performance in memory of object locations. Cartography and Geographic Information Science, 46(5), 401-411.

[17] Her, I. (1995). Geometric transformations on the hexagonal grid. IEEE Transactions on Image Processing, 4(9), 1213-1222.

[18] Hex Map 1, Hexagonal Grid, https://catlikecoding.com/unity/tutorials/hex-map/part-1/

[19] Rafiq, A., Kadir, T. A. A., Ihsan, S. N. (2020). Pathfinding algorithms in game development. In IOP Conference Series: Materials Science and Engineering, 769(1), 012021.

[20] Yan, Y. (2023). Research on the A Star Algorithm for Finding Shortest Path. Highlights in Science, Engineering and Technology, 46, 154-161.

[21] Wayahdi, M. R., Ginting, S. H. N., Syahputra, D. (2021). Greedy, A-Star, and Dijkstra's algorithms in finding shortest path. International Journal of Advances in Data and Information Systems, 2(1), 45-52.

[22] Deng, Z., Wang, D. (2023). Research on Parking Path Planing Based on A-Star Algorithm. Journal of New Media, 5(1).

[23] Candra, A., Budiman, M. A., Pohan, R. I. (2021). Application of a-star algorithm on pathfinding game. In Journal of Physics: Conference Series (IOP Publishing), 1898(1), 012047.

[24] Saian, P. O. N. (2016). Optimized A-Star algorithm in hexagon-based environment using parallel bidirectional search. In 2016 IEEE 8th International Conference on Information Technology and Electrical Engineering (ICITEE), 1-5.

[25] Zhang, H., Tao, Y., Zhu, W. (2023). Global path planning of unmanned surface vehicle

based on improved A-Star algorithm. Sensors, 23(14), 6647.

[26] Frăsinaru, C., Răschip, M. (2019). Greedy best-first search for the optimal-size sorting network problem. Procedia Computer Science, 159, 447-454.

[27] Heusner, M. (2019). Search behavior of greedy best-first search (Doctoral dissertation, University_of_Basel).

[28] Heusner, M., Keller, T., Helmert, M. (2017). Understanding the search behaviour of greedy best-first search. In Proceedings of the International Symposium on Combinatorial Search, 8(1), 47-55.

[29] Heusner, M., Keller, T., Helmert, M. (2018). Best-case and worst-case behavior of greedy best-first search. International Joint Conferences on Artificial Intelligence, 1463-1470.

[30] Lina, T. N., Rumetna, M. S. (2021). Comparison analysis of breadth first search and depth limited search algorithms in sudoku game. Bulletin of Computer Science and Electrical Engineering, 2(2), 74-83.

[31] Fayed, H. A., Atiya, A. F. (2013). A mixed breadth-depth first strategy for the branch and bound tree of Euclidean k-center problems. Computational Optimization and Applications, 54, 675-703.

[32] Sihotang, J. (2020). Analysis Of Shortest Path Determination By Utilizing Breadth First Search Algorithm. Jurnal Info Sains: Informatika dan Sains, 10(2), 1-5.

[33] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2022). Introduction to algorithms. MIT press.

[34] Pandika, I. K. L. D., Irawan, B., Setianingsih, C. (2018). Apllication of optimization heavy traffic path with floyd-warshall algorithm. In 2018 IEEE International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), 57-62.

[35] Mirino, A. E. (2017). Best routes selection using Dijkstra and Floyd-Warshall algorithm. In 2017 IEEE 11th International Conference on Information & Communication Technology and System (ICTS), 155-158.

[36] Hougardy, S. (2010). The Floyd–Warshall algorithm on graphs with negative cycles. Information Processing Letters, 110(8-9), 279-281.

[37] Azis, H., Lantara, D., Salim, Y. (2018). Comparison of Floyd-Warshall algorithm and greedy algorithm in determining the shortest route. In 2018 IEEE 2nd East Indonesia conference on computer and information technology (EIConCIT), 294-298).

[38] Noto, M., Sato, H. (2000). A method for the shortest path search by extended Dijkstra algorithm. In Smc 2000 conference proceedings. 2000 IEEE international conference on systems, man and cybernetics, 3, 2316-2320.

[39] Salem, I. E., Mijwil, M. M., Abdulqader, A. W., Ismaeel, M. M. (2022). Flight-schedule using Dijkstra's algorithm with comparison of routes findings. International Journal of Electrical and Computer Engineering, 12(2), 1675.

[40] Cui, X., Shi, H. (2011). A*-based pathfinding in modern computer games. International Journal of Computer Science and Network Security, 11(1), 125-130.

[41] Du, D. Z., Kleitman, D. J. (1990). Diameter and radius in the Manhattan metric. Discrete & computational geometry, 5, 351-356.