BURSA TEKNİK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ (T) BÖLÜMÜ
BİLGİSAYAR OYUNLARDA YAPAY ZEKA

Ödev-10 Raporu

Github: https://github.com/MoussaBane/BOYZ-Unity-Labirent-Projesi

MOUSSA BANE

24435004029

Unity Labirent Projesi: A Algoritması ile Yol Bulma

PROJE HAKKINDA GENEL BILGI

A Algoritmasının Temelleri

A* algoritması, en kısa yolu bulmak için kullanılan sezgisel bir arama algoritmasıdır. Toplam maliyet fonksiyonu f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n) şu şekilde tanımlanır:

- g(n)g(n)g(n): Başlangıç noktasından mevcut düğüme kadar olan maliyet.
- h(n)h(n)h(n): Mevcut düğümden hedefe tahmini maliyet (heuristik).

A* algoritması, hem kesinlikle doğru hem de verimli bir sonuç sunmak için kullanılan güçlü bir tekniktir.

Projenin Hedefleri

- Rastgele bir labirent oluşturmak.

- A* algoritmasını kullanarak labirentin başlangıç ve hedef noktası arasında bir yol bulmak.
- Süreci ve algoritmanın çalışma adımlarını görselleştirmek.

GEREKLI KURULUM VE AYARLAR

Unity Projesinin Oluşturulması

- Unity'de yeni bir 3D proje oluşturun.
- Proje klasörünüzde **Assets** altında yeni bir klasör oluşturun ve ismini LabirentProjesi olarak belirleyin.

Başlangıç Paketi Entegrasyonu

- Verilen **AStarStarter** paketini projenize dahil edin.
- Scenes klasöründen **AStarPath** sahnesini açın.
- Play butonuna basarak labirentin rastgele oluşturulduğunu doğrulayın.

Not: Unity versiyonunuzun en az 2020.3 LTS veya daha yeni bir sürüm olmasına dikkat edin.

KOD VE UYGULAMA ADIMLARI

FindPathAStar Script'inin Oluşturulması

Projenin temel bileşenlerinden biri olan **FindPathAStar** script'i, A* algoritmasının uygulanmasını sağlar. Bu script içinde:

- **Açık ve Kapalı Listeler**: Açık liste, değerlendirilecek düğümleri; kapalı liste ise değerlendirilmiş düğümleri tutar.
- **Başlangıç ve Hedef Tanımı**: Başlangıç ve hedef noktaları rastgele seçilir.
- **Yol Bulma Fonksiyonu**: Algoritmanın her adımda en düşük maliyetli düğümü seçerek ilerlemesi sağlanır.

FindPathStar.cs 5 | PathMarker.cs

Assets > scripts > FindPathStar.cs > FindPathStar

```
using System.Linq;

0 references
public class FindPathStar : MonoBehaviour
{
    17 references
    public Maze maze;
    1 reference
    public Material closedMaterial;
    0 references
    public Material openMaterial;
    8 references
    List<PathMarker> open = new List<PathMarker>();
    3 references
    List<PathMarker> closed = new List<PathMarker>();
    1 reference
    public GameObject start;
    1 reference
    public GameObject end;
    3 references
    public GameObject pathP;

    3 references
    PathMarker goalNode; // The goal node in the maze.
    6 references
    PathMarker startNode; // The start node in the maze.
    4 references
    PathMarker lastPos; // The most recently processed node.
    3 references
    bool done = false; // Flag to indicate if the search is complete.

    // Removes all markers from the maze.
    2 references
    void RemoveAllMarkers()
    {
        GameObject[] markers = GameObject.FindGameObjectsWithTag("marker");
        foreach (GameObject m in markers)
            Destroy(m); // Destroy each marker object.
    }
```

FindPathStar.cs 5 | PathMarker.cs

Assets > scripts > FindPathStar.cs > FindPathStar

```
public class FindPathStar : MonoBehaviour

    // Begins the A* search process by initializing start and goal nodes.
    1 reference
    void BeginSearch()
    {
        done = false;
        RemoveAllMarkers(); // Clear existing markers.
        List<MapLocation> locations = new List<MapLocation>(); // List of available locations in the maze.
        for (int z = 1; z < maze.depth - 1; z++)
        {
            for (int x = 1; x < maze.depth - 1; x++)
            {
                if (maze.map[x, z] != 1) // Check if the location is not a wall.
                    locations.Add(new MapLocation(x, z));
            }

            locations.Shuffle(); // Randomize the locations.

            // Set up the start node.
            Vector3 startLocation = new Vector3(locations[0].x * maze.scale, 0, locations[0].z * maze.scale);
            startNode = new PathMarker(new MapLocation(locations[0].x, locations[0].z), 0, 0, 0,
            Instantiate(start, startLocation, Quaternion.identity), null);

            // Set up the goal node.
            Vector3 goalLocation = new Vector3(locations[1].x * maze.scale, 0, locations[1].z * maze.scale);
            goalNode = new PathMarker(new MapLocation(locations[1].x, locations[1].z), 0, 0, 0,
            Instantiate(end, goalLocation, Quaternion.identity), null);

            // Clear open and closed lists, and initialize with the start node.
            open.Clear();
            closed.Clear();
            open.Add(startNode);
            lastPos = startNode;
        }
    }
```

FindPathStar.cs 5 ● | PathMarker.cs

Assets > scripts > FindPathStar.cs > FindPathStar > Search

```
7   public class FindPathStar : MonoBehaviour
65      // Performs the A* search logic for the current node.
        1 reference
66      void Search(PathMarker thisNode)
67      {
68          if (thisNode == null) return; // Ensure the current node is not null.
69
70          if (thisNode.Equals(goalNode)) { done = true; return; } // End search if goal is reached.
71
72          // Evaluate all possible neighboring nodes.
73          foreach (MapLocation dir in maze.directions)
74          {
75              MapLocation neighbor = dir + thisNode.location; // Calculate neighbor location.
76              if (maze.map[neighbor.x, neighbor.z] == 1) continue; // Skip walls.
77              if (neighbor.x < 1 || neighbor.x >= maze.width || neighbor.z < 1 || neighbor.z >= maze.depth) continue;
78              // Skip out-of-bounds locations.
79              if (IsClosed(neighbor)) continue; // Skip if the neighbor is already in the closed list.
80
81              // Calculate G, H, and F values for the neighbor.
82              float G = Vector2.Distance(thisNode.location.ToVector(), neighbor.ToVector()) + thisNode.G;
83              float H = Vector2.Distance(neighbor.ToVector(), goalNode.location.ToVector());
84              float F = G + H;
85
86              // Instantiate a marker for the neighbor.
87              GameObject pathBlock = Instantiate(pathP, new Vector3(neighbor.x * maze.scale, 0, neighbor.z * maze.scale),
88              Quaternion.identity);
89              TextMesh[] values = pathBlock.GetComponentsInChildren<TextMesh>();
90              // Assign G, H, and F values to the marker for visualization.
91              values[0].text = "G: " + G.ToString("0.00");
92              values[1].text = "H: " + H.ToString("0.00");
93              values[2].text = "F: " + F.ToString("0.00");
94
95              // Add the neighbor to the open list if it's not already there or update its values.
96              if (!UpdateMarker(neighbor, G, H, F, thisNode))
97                  open.Add(new PathMarker(neighbor, G, H, F, pathBlock, thisNode));
98          }
```

FindPathStar.cs 5 | PathMarker.cs

Assets > scripts > FindPathStar.cs > FindPathStar > Search

```
7   public class FindPathStar : MonoBehaviour
66      void Search(PathMarker thisNode)
99          open = open.OrderBy(p => p.F).ThenBy(n => n.H).ToList<PathMarker>();
100         PathMarker pm = (PathMarker)open.ElementAt(0);
101         closed.Add(pm); // Move the node from open to closed list.
102         open.RemoveAt(0);
103         pm.marker.GetComponent<Renderer>().material = closedMaterial; // Update material to indicate closed status.
104         lastPos = pm; // Update the last processed node.
105     }
106
107     // Checks if a location is in the closed list.
        1 reference
108     bool IsClosed(MapLocation marker)
109     {
110         foreach (PathMarker p in closed)
111         {
112             if (p.location.Equals(marker)) return true;
113         }
114         return false;
115     }
116
117     // Updates the marker in the open list if it already exists.
        1 reference
118     bool UpdateMarker(MapLocation pos, float g, float h, float f, PathMarker prt)
119     {
120         foreach (PathMarker p in open)
121         {
122             if (p.location.Equals(pos))
123             {
124                 p.G = g; // Update G value.
125                 p.H = h; // Update H value.
126                 p.F = f; // Update F value.
127                 p.parent = prt; // Update parent.
128                 return true;
129             }
130         }
131         return false;
```

FindPathStar.cs 5 | PathMarker.cs

Assets > scripts > FindPathStar.cs > FindPathStar > Search

```
7   public class FindPathStar : MonoBehaviour
134     // Retrieves the path from the goal to the start by following parent links.
        1 reference
135     void GetPath()
136     {
137         RemoveAllMarkers(); // Clear existing markers.
138         PathMarker begin = lastPos; // Start from the last processed node.
139         while (!startNode.Equals(begin) && begin != null)
140         {
141             // Instantiate a path marker for each node in the path.
142             Instantiate(pathP, new Vector3(begin.location.x * maze.scale, 0, begin.location.z * maze.scale), Quaternion.identity);
143             begin = begin.parent; // Move to the parent node.
144         }
145         // Instantiate a marker for the start node.
146         Instantiate(pathP, new Vector3(startNode.location.x * maze.scale, 0, startNode.location.z * maze.scale), Quaternion.identity);
147     }
148
149     // Handles user input to control the search and path visualization.
        0 references
150     void Update()
151     {
152         if (Input.GetKeyDown(KeyCode.P)) BeginSearch(); // Start a new search.
153         if (Input.GetKeyDown(KeyCode.C) && !done) Search(lastPos); // Continue the search step-by-step.
154         if (Input.GetKeyDown(KeyCode.M)) GetPath(); // Visualize the complete path.
155     }
156
157 }
```

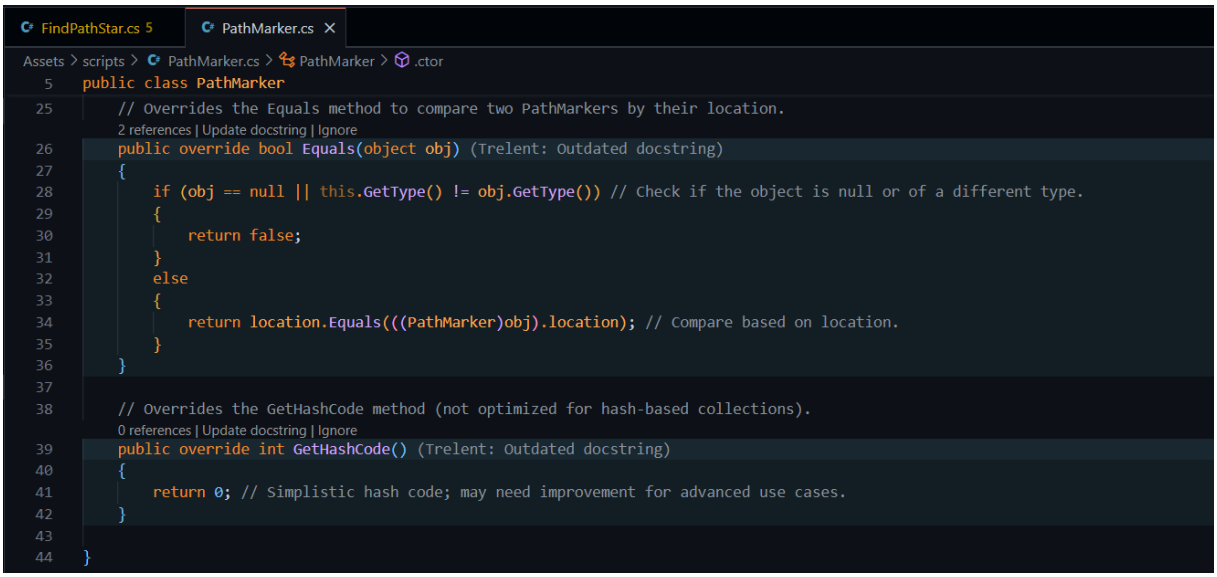
PathMarker Sınıfının Tanımı

PathMarker sınıfı, bir düğümün konumu, maliyet bilgileri (G, H, F), görsel temsilcisi ve ebeveyn düğüm bilgilerini içerir. Bu sınıf, algoritmanın yol takibini kolaylaştırır.

FindPathStar.cs 5 | PathMarker.cs

Assets > scripts > PathMarker.cs > PathMarker > .ctor

```
    22 references
5   public class PathMarker
6   {
        12 references
7       public MapLocation location;
        3 references
8       public float G;
        3 references
9       public float H;
        3 references
10      public float F;
        2 references
11      public GameObject marker;
        3 references
12      public PathMarker parent;
13
14      // Constructor to initialize a PathMarker with its properties.
        3 references
15      public PathMarker(MapLocation l, float g, float h, float f, GameObject marker, PathMarker p)
16      {
17          location = l;
18          G = g;
19          H = h;
20          F = f;
21          this.marker = marker;
22          parent = p;
23      }
```



Arama Fonksiyonlarının Implementasyonu

BeginSearch fonksiyonu:

- Başlangıç ve hedef noktalarını rastgele seçer.
- Açık ve kapalı listeleri temizler.

Search fonksiyonu:

- Açık listedeki en düşük maliyetli düğümü seçerek komşu düğümleri değerlendirir.
- Hedef düğüm bulunduğunda aramayı durdurur.

GetPath fonksiyonu:

- Son düğümden başlangıç düğümüne doğru ebeveyn bağlantıları takip ederek yolu işaretler.

TEST VE ÇALISTIRMA

Labirent Oluşturulması

- Sahnedeki **Maze** nesnesini seçin.
- Recursive script'i altında width ve depth değerlerini 10 olarak ayarlayın.
- Play butonuna basarak rastgele bir labirent oluşturun.

Başlangıç ve Hedef Noktalarının Seçimi

- P tuşuna basarak başlangıç ve hedef noktalarını oluşturun.
- Seçilen noktaların görsel olarak işaretlendiğini doğrulayın.

A Algoritmasının Çalıştırılması

- C tuşuna basarak algoritmayı adım adım çalıştırın.
- Hedefe ulaşıldığında, M tuşuna basarak bulunan yolu görselleştirin.

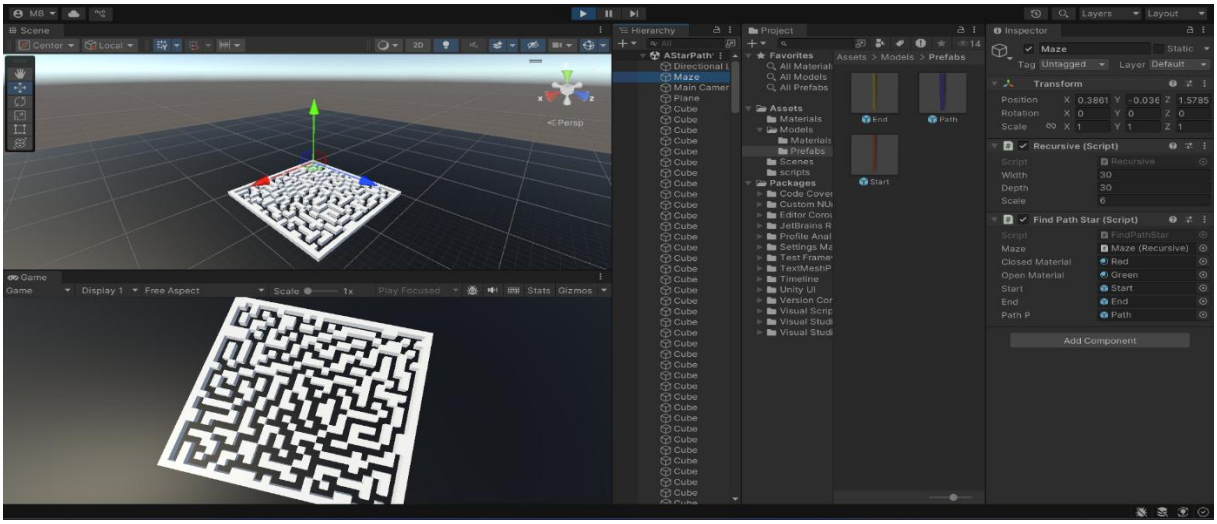
SONUÇ VE ÇALISTIRILMIS HALI

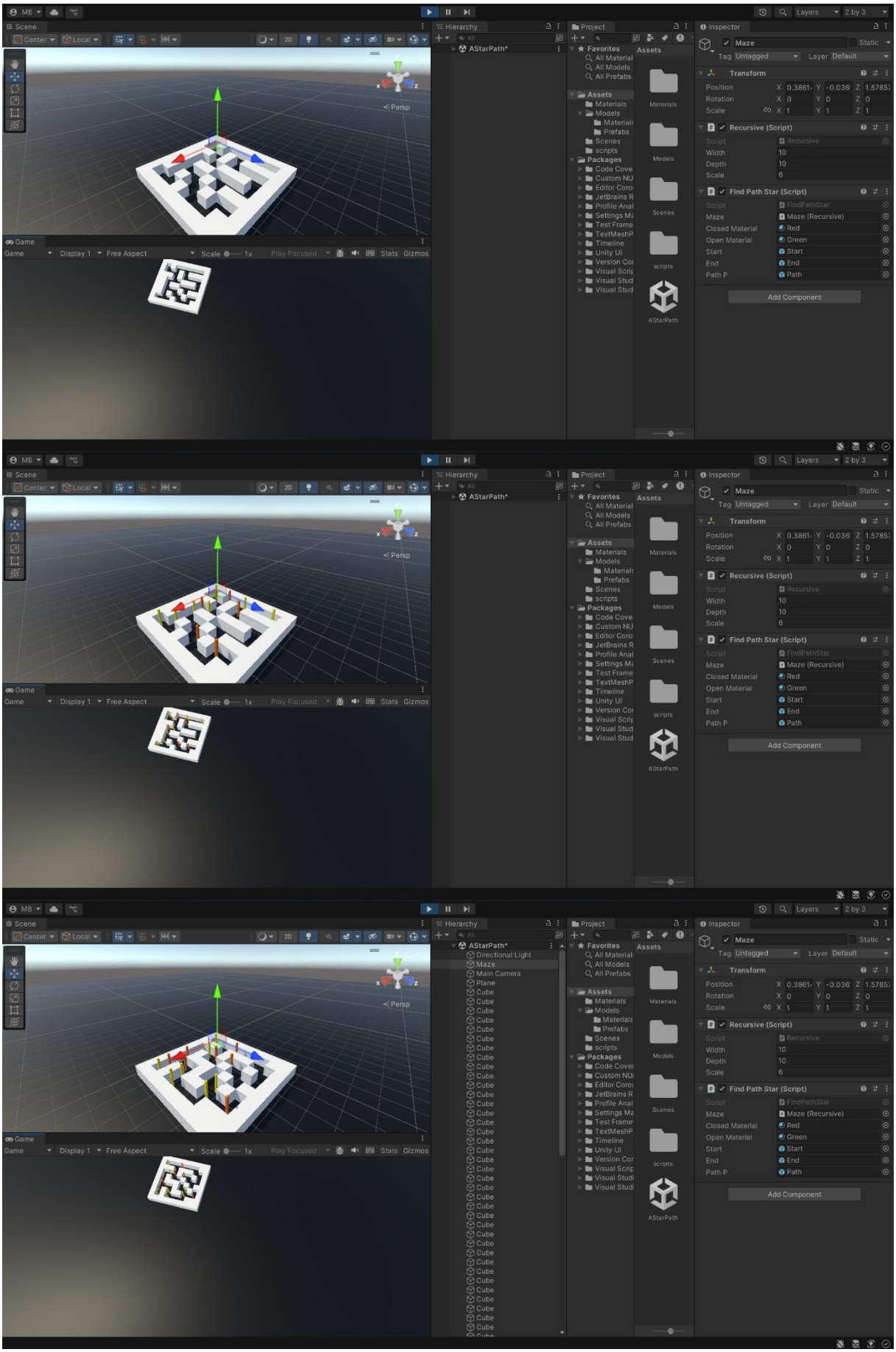
Algoritmanın İşleyişi

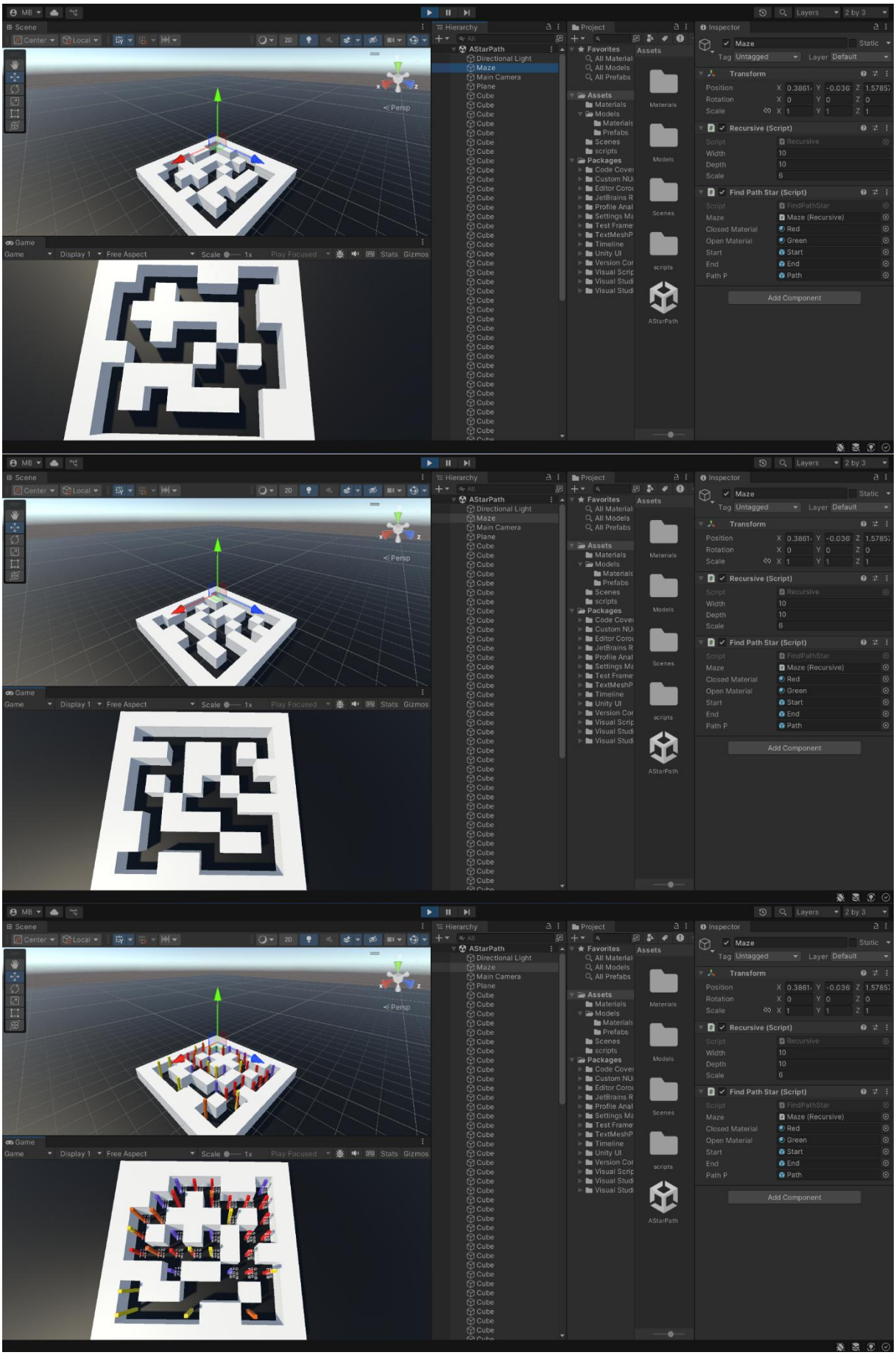
Bu proje, A* algoritmasının etkinliğini ve doğruluğunu başarıyla sergiler. Rastgele oluşturulan her labirentte, başlangıç ve hedef noktaları arasındaki en kısa yol etkin bir şekilde bulunur.

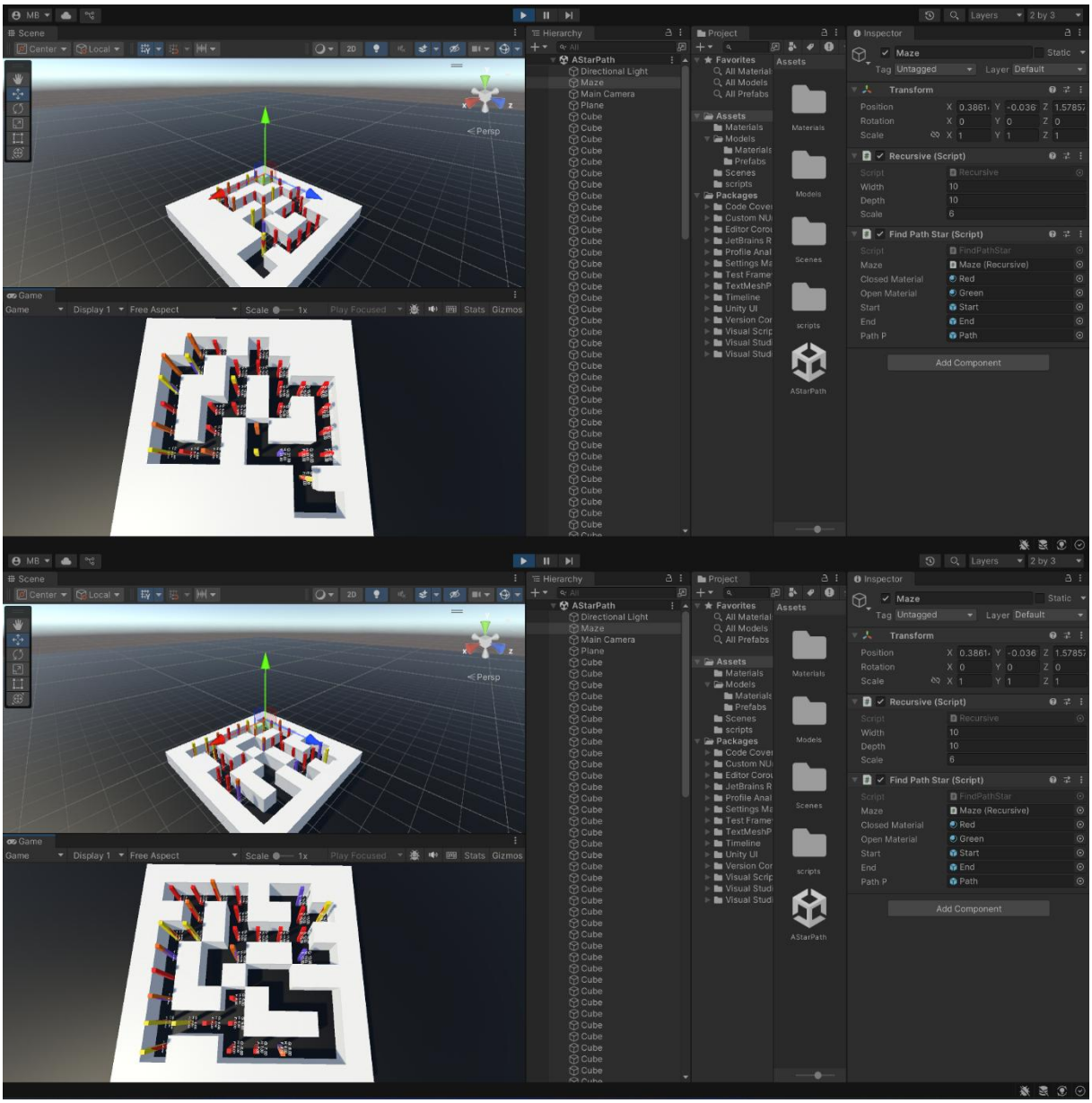
Projenin öne çıkan noktaları şunlardır:

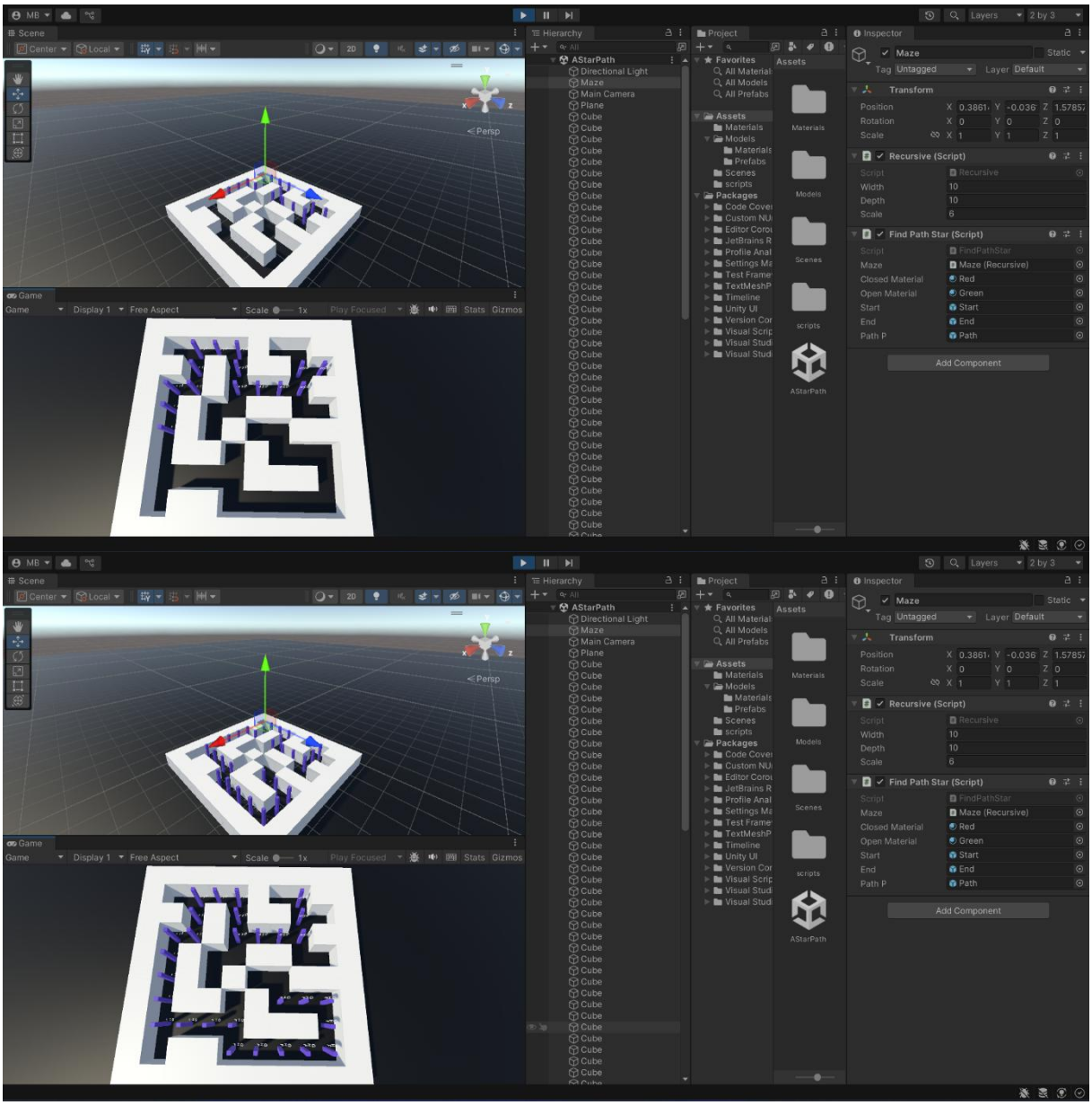
- **Hızlı Arama:** A* algoritması, minimum maliyetli yolu seçerek hedefe ulaşmayı en kısa sürede tamamlar.
- **Görselleştirme:** Yol bulma sürecindeki aşamalar, açık ve kapalı listelerdeki renkli görsel işaretçiler ile net bir şekilde sunulmuştur.
- **Dinamik Labirentler:** Algoritmanın rastgele oluşan her labirentte başarıyla çalışması, esnekliğini kanıtlar.




Scene
Hierarchy
Project
Inspector
Game
AStarPath*
Maze
Transform
Recursive (Script)
Find Path Star (Script)
Add Component
Persp
Directional Light
Main Camera
Plane
Cube


Scene
Persp
Game
Display 1
Free Aspect
Scale 1x
Play Focused
Stats
Gizmos
Hierarchy
AStarPath
Directional Light
Maze
Main Camera
Plane
Project
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Models
Prefabs
Scenes
scripts
Packages
AStarPath
Inspector
Maze
Static
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Recursive (Script)
Script Recursive
Width 10
Depth 10
Scale 6
Find Path Star (Script)
Script FindPathStar
Maze Maze (Recursive)
Closed Material Red
Open Material Green
Start Start
End End
Path P Path
Add Component


Scene
Hierarchy
Project
Inspector
Game
AStarPath
Directional Light
Maze
Main Camera
Plane
Persp
Display 1
Free Aspect
Scale
Play Focused
Stats
Gizmos
Assets
Materials
Models
Scenes
scripts
Transform
Position
Rotation
Scale
Recursive (Script)
Width
Depth
Find Path Star (Script)
Maze (Recursive)
Closed Material
Red
Open Material
Green
Start
End
Path
Add Component


Scene
Hierarchy
Project
Inspector
Game
Persp
AStarPath
Directional Light
Maze
Main Camera
Plane
Assets
Materials
Models
Scenes
scripts
Packages
Favorites
Transform
Position
Rotation
Scale
Recursive (Script)
Width
Depth
Find Path Star (Script)
Maze (Recursive)
Closed Material
Open Material
Red
Green
Start
End
Path
Add Component