BURSA TEKNİK ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ (T) BÖLÜMÜ

BİLGİSAYAR OYUNLARDA YAPAY ZEKA

Ödev-4 Raporu

Github: https://github.com/MoussaBane/BOYZ-Genetik-Programlama-III

MOUSSA BANE

24435004029

Proje Tanımı

Bu projede, genetik programlama tekniklerinin ve sensör kullanımının 3D bir ortamda zeki ajanlar yaratmak için uygulanmasını keşfetmeyi amaçlıyoruz. Projemizin temeli, karakter hareketlerini platformlar üzerinde simüle etmek için temel bir yapı sağlayan **StayOnPlatformStarter** çerçevesine dayanmaktadır.

Ortam, karakterlerin serbestçe hareket edebileceği güvenli bir alan olarak belirlenen bir **Plane** ve tehlikeli bir bölgeyi temsil eden **Cube** olmak üzere iki tür platformdan oluşmaktadır. Karakterlerin ana hedefi, mümkün olduğunca uzun süre Plane platformunda kalmak ve Cube platformundan kaçınmaktır. Karakterler, ileri hareket edebilir ve sola veya sağa dönebilir; bu, genetik kodlamaları tarafından etkilenen temel hareket davranışlarını göstermektedir.

Genetik programlama yaklaşımının bir parçası olarak, en uzun süre hayatta kalan karakterler genetik özelliklerini sonraki nesillere aktaracak; bu da hareket stratejilerinin ardışık nesiller boyunca geliştirilmesini sağlayacaktır.

Gereksinimler

- **Unity**: Projeyi geliştirmek için Unity oyun motoru.
- **C# Scriptleri**: Botların hareketi ve DNA yönetimi için C# dilinde yazılmış scriptler.
- **3D Model**: Bot prefab'ı olarak kullanılacak bir 3D model (önceden sağlanmış).

1. Proje Kurulumu ve Sahne Ayarları

- **Yeni bir Unity 3D Projesi Oluşturun**:
  - Unity'yi açın ve yeni bir 3D proje oluşturun.

- Projeye uygun bir isim verin ve dosya konumunu belirleyin.

➢ **Gerekli Asset'leri İçe Aktarın**:

- StayOnPlatformStarter adlı varlıkları projeye aktarın.

➢ **Sahneyi Ayarlayın**:

- StayOnPlatform adlı sahneyi açın.
- Sahnenin içinde iki platform bulunmalı:
  - **Plane** nesnesi (etiketi "platform").
  - **Cube** nesnesi (etiketi "dead").
- Plane üzerinde botlar hareket edecek, Cube platformuna düşen botlar ise “ölecek” ve yeni nesile geçilecek.

2. Bot Nesnesi ve "Göz" Oluşturma

➢ **Bot Nesnesini Oluşturun**:

- **Hierarchy** panelinde sağ tıklayın ve **3D Object > Capsule** seçin. Bu nesne bizim bot karakterimiz olacak.
- Capsule nesnesinin adını **Bot** olarak değiştirin.
- **Inspector** panelinde **Rigidbody** bileşeni ekleyin.
- **Constraints** bölümünde **Rotation** için x, y, ve z eksenlerini kilitleyin. Bu, botun platform üzerinde sabit kalmasını sağlar.

➢ **"Göz" Nesnesini Oluşturun**:

- Bot’un üzerinde bir "göz" işlevi görecek bir küp ekleyin.
- Bot nesnesinin üzerine sağ tıklayın ve **3D Object > Cube** seçin.
- Küpün adını **Eyes** olarak değiştirin.
- Eyes nesnesinin boyutunu ve konumunu Bot’un üst kısmında olacak şekilde ayarlayın. Bu nesne botun "görüş alanı" olarak çalışacak.

3. DNA Script’ini Oluşturma (DNA_sc.cs)

Bu script, botun genetik özelliklerini yani DNA'sını tanımlar.

➢ **Scripts** adlı bir klasör oluşturun.

➢ Bu klasöre sağ tıklayın ve **Create > C# Script** seçin. Dosyanın adını **DNA_sc** olarak verin ve aşağıdaki kodu ekleyin:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DNA_sc
{
    // List to hold the genes (integer values representing traits)
    List<int> genes = new List<int>();

    // Length of the DNA, determining how many genes it will have
    int dnaLength = 0;

    // Maximum possible value for each gene
    int maxValues = 0;

    // Constructor to initialize DNA length and maximum gene values
    public DNA_sc(int l, int v)
    {
        dnaLength = l;        // Set the length of the DNA
        maxValues = v;        // Set the maximum gene value
        SetRandom();          // Initialize genes randomly
    }

    // Sets random values for each gene in the DNA sequence
    public void SetRandom()
    {
        genes.Clear();        // Clear any existing genes
        for (int i = 0; i < dnaLength; i++)
        {
            // Add a random integer between 0 and maxValues for each gene
            genes.Add(Random.Range(0, maxValues));
        }
    }

    // Sets a specific gene's value at the given position
    public void SetInt(int pos, int value)
    {
        genes[pos] = value;  // Set the gene at position 'pos' to 'value'
    }

    // Gets the value of a gene at a specified position
    public int GetGene(int pos)
    {
        return genes[pos];   // Return the gene at position 'pos'
    }

    // Combines genes from two parent DNA objects (d1 and d2) to create this DNA
    public void Combine(DNA_sc d1, DNA_sc d2)
    {
        for (int i = 0; i < dnaLength; i++)
        {
            // If within the first half of the DNA length, take gene from d1
            if (i < dnaLength / 2.0)
            {
                int c = d1.genes[i];
                genes[i] = c;
            }
            // Otherwise, take gene from d2 for the second half
            else
            {
                int c = d2.genes[i];
                genes[i] = c;
            }
        }
    }

    // Mutates a random gene by assigning it a new random value
    public void Mutate()
    {
        // Select a random gene and set it to a new random value
        genes[Random.Range(0, dnaLength)] = Random.Range(0, maxValues);
    }
}
```

4. Brain Script'ini Oluşturma (Brain_sc.cs)

Bu script, DNA'ya bağlı olarak botun hareketini ve çarpışma kontrolünü sağlar.

- **Scripts** klasörüne sağ tıklayın, yeni bir **C# Script** oluşturun ve adını **Brain_sc** koyun.
- Aşağıdaki kodları ekleyin:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// Class that defines the behavior of a bot in the game
public class Brain_sc : MonoBehaviour
{
    // Length of the DNA for the bot (number of genes)
    int DNALength = 2;

    // Time the bot has been alive
    public float timeAlive;

    // Reference to the bot's DNA script
    public DNA_sc dna_sc;

    // Reference to the GameObject representing the bot's eyes
    public GameObject eyes;

    // Indicates whether the bot is alive
    bool isAlive = true;

    // Indicates whether the bot can see the ground
    bool canSeeGround = true;

    // Variables to control movement and turning
    float turn = 0;
    float move = 0;

    // Called when the bot collides with another object
    void OnCollisionEnter(Collision other)
    {
        // If the object has the "dead" tag, the bot is considered dead
        if (other.gameObject.tag == "dead")
```

```
        // If the object has the "dead" tag, the bot is considered dead
        if (other.gameObject.tag == "dead")
        {
            isAlive = false; // Mark the bot as not alive
        }
    }

    // Initialize the bot's DNA and reset its state
    public void Init()
    {
        // Initialize DNA with DNALength and 3 possible values for each gene
        dna_sc = new DNA_sc(DNALength, 3);
        timeAlive = 0; // Reset time alive
        isAlive = true; // Set the bot as alive
    }

    // Update is called once per frame
    private void Update()
    {
        // If the bot is not alive, exit the update function
        if (!isAlive) return;

        // Draw a ray in the forward direction from the bot's eyes for visualization
        Debug.DrawRay(eyes.transform.position, eyes.transform.forward * 10, Color.red, 10);
        canSeeGround = false; // Reset ground visibility

        RaycastHit hit; // Variable to store raycast hit information

        // Perform a raycast to check if there is ground ahead
        if (Physics.Raycast(eyes.transform.position, eyes.transform.forward * 10, out hit))
        {
            // If the raycast hits an object with the "platform" tag, set canSeeGround to true
            if (hit.collider.gameObject.tag == "platform")
            {
                canSeeGround = true;
            }
        }
```

```
        // Update the time the bot has been alive based on the population manager's elapsed time
        timeAlive = PopulationManager_sc.elapsed;

        // If the bot can see the ground, read the first gene to determine movement
        if (canSeeGround)
        {
            // Move forward, turn left, or turn right based on the gene value
            if (dna_sc.GetGene(0) == 0) move = 1; // Move forward
            else if (dna_sc.GetGene(0) == 1) turn = -90; // Turn left
            else if (dna_sc.GetGene(0) == 2) turn = 90; // Turn right
        }
        // If the bot cannot see the ground, read the second gene for movement
        else
        {
            if (dna_sc.GetGene(1) == 0) move = 1; // Move forward
            else if (dna_sc.GetGene(1) == 1) turn = -90; // Turn left
            else if (dna_sc.GetGene(1) == 2) turn = 90; // Turn right
        }

        // Move the bot forward by translating it along the z-axis
        this.transform.Translate(0, 0, move * 0.1f);
        // Rotate the bot around the y-axis based on the turn value
        this.transform.Rotate(0, turn, 0);
    }
}
```

5. PopulationManager Script'i Oluşturma (PopulationManager_sc.cs)

Bu script, popülasyon yönetimini sağlar ve yeni nesiller oluşturur.

- **Scripts** klasörüne sağ tıklayın, yeni bir **C# Script** oluşturun ve adını **PopulationManager_sc** koyun.
- Aşağıdaki kodları ekleyin:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;

// Class that manages the population of bots in the simulation
public class PopulationManager_sc : MonoBehaviour
{
    // Prefab of the bot that will be instantiated
    public GameObject botPrefab;

    // Total number of bots in the population
    public int populationSize = 50;

    // List to hold all bots in the population
    List<GameObject> population = new List<GameObject>();

    // Static variable to track the elapsed time
    public static float elapsed = 0;

    // Time duration for each trial before breeding new bots
    public float trialTime = 5;

    // Current generation number
    int generation = 1;

    // GUIStyle for displaying statistics on the screen
    GUIStyle guiStyle = new GUIStyle();

    // Method to display GUI elements
    void OnGUI()
    {
        guiStyle.fontSize = 25; // Set font size for the GUI
        guiStyle.normal.textColor = Color.white; // Set font color to white
```

```
    // Method to display GUI elements
    void OnGUI()
    {
        guiStyle.fontSize = 25; // Set font size for the GUI
        guiStyle.normal.textColor = Color.white; // Set font color to white

        // Create a group for the stats box
        GUI.BeginGroup(new Rect(10, 10, 250, 150));
        GUI.Box(new Rect(0, 0, 140, 140), "Stats", guiStyle); // Box for stats

        // Display current generation, elapsed time, and population size
        GUI.Label(new Rect(10, 25, 200, 30), "Gen: " + generation, guiStyle);
        GUI.Label(new Rect(10, 50, 200, 30), string.Format("Time: {0:0.00}", elapsed), guiStyle);
        GUI.Label(new Rect(10, 75, 200, 30), "Population: " + population.Count, guiStyle);
        GUI.EndGroup(); // End the stats group
    }

    // Use this for initialization
    void Start()
    {
        // Create initial population of bots
        for (int i = 0; i < populationSize; i++)
        {
            // Generate a random starting position for the bot
            Vector3 startingPos = new Vector3(
                this.transform.position.x + Random.Range(-2, 2),
                this.transform.position.y,
                this.transform.position.z + Random.Range(-2, 2)
            );

            // Instantiate the bot and initialize its brain
            GameObject bot = Instantiate(botPrefab, startingPos, this.transform.rotation);
            bot.GetComponent<Brain_sc>().Init(); // Initialize bot's brain
            population.Add(bot); // Add bot to the population list
        }
    }
```

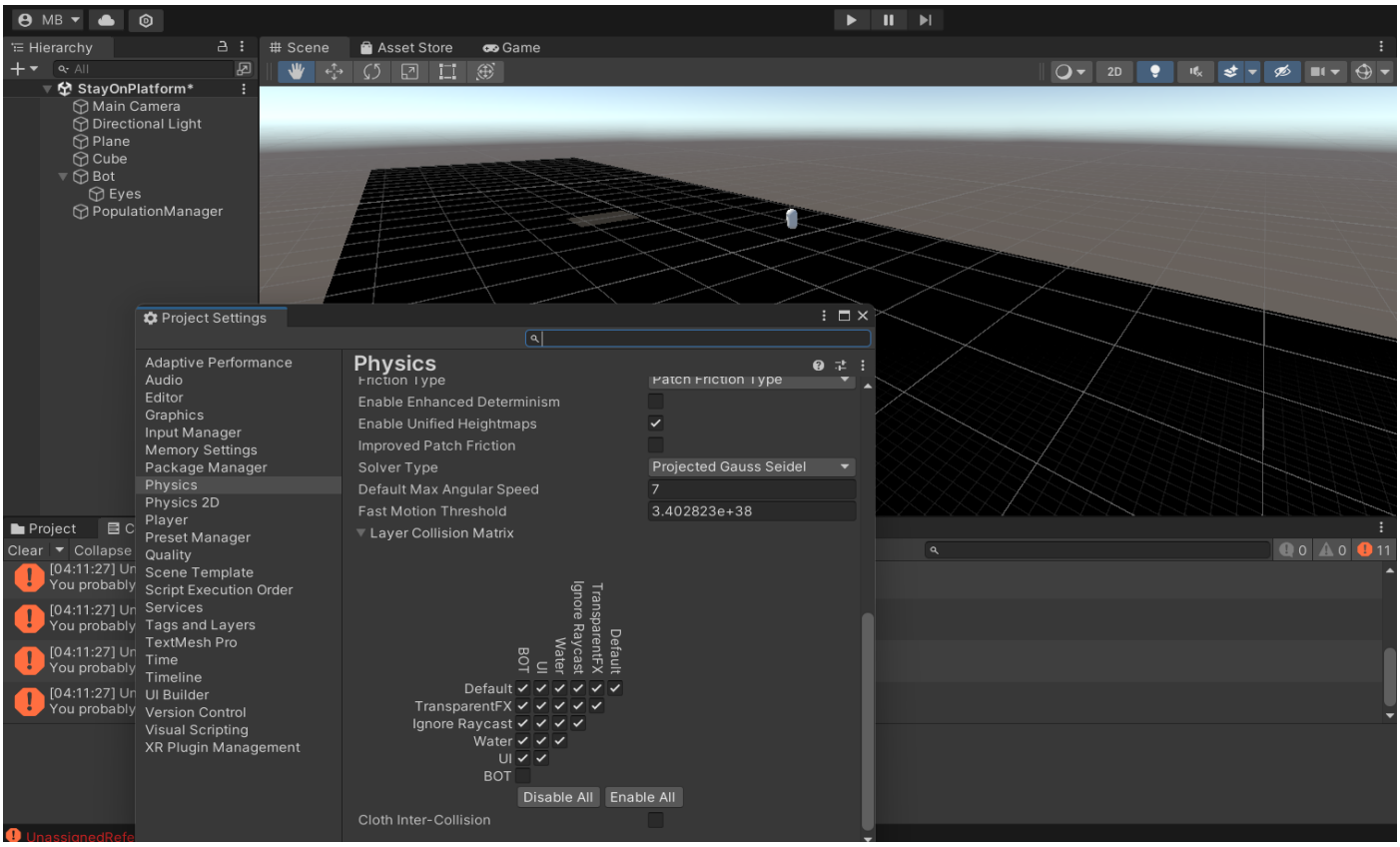
```
    // Method to breed two parent bots and create an offspring
    GameObject Breed(GameObject parent1, GameObject parent2)
    {
        // Generate a random starting position for the offspring
        Vector3 startingPos = new Vector3(
            this.transform.position.x + Random.Range(-2, 2),
            this.transform.position.y,
            this.transform.position.z + Random.Range(-2, 2)
        );

        // Instantiate the offspring bot
        GameObject offspring = Instantiate(botPrefab, startingPos, this.transform.rotation);
        Brain_sc b = offspring.GetComponent<Brain_sc>();
        b.Init(); // Initialize the brain of the offspring

        // Mutate or combine parents' DNA
        if (Random.Range(0, 100) == 1) // 1 in 100 chance to mutate
        {
            b.dna_sc.Mutate(); // Mutate the DNA of the offspring
        }
        else
        {
            // Combine DNA from both parents
            b.dna_sc.Combine(
                parent1.GetComponent<Brain_sc>().dna_sc,
                parent2.GetComponent<Brain_sc>().dna_sc
            );
        }
        return offspring; // Return the created offspring
    }
```

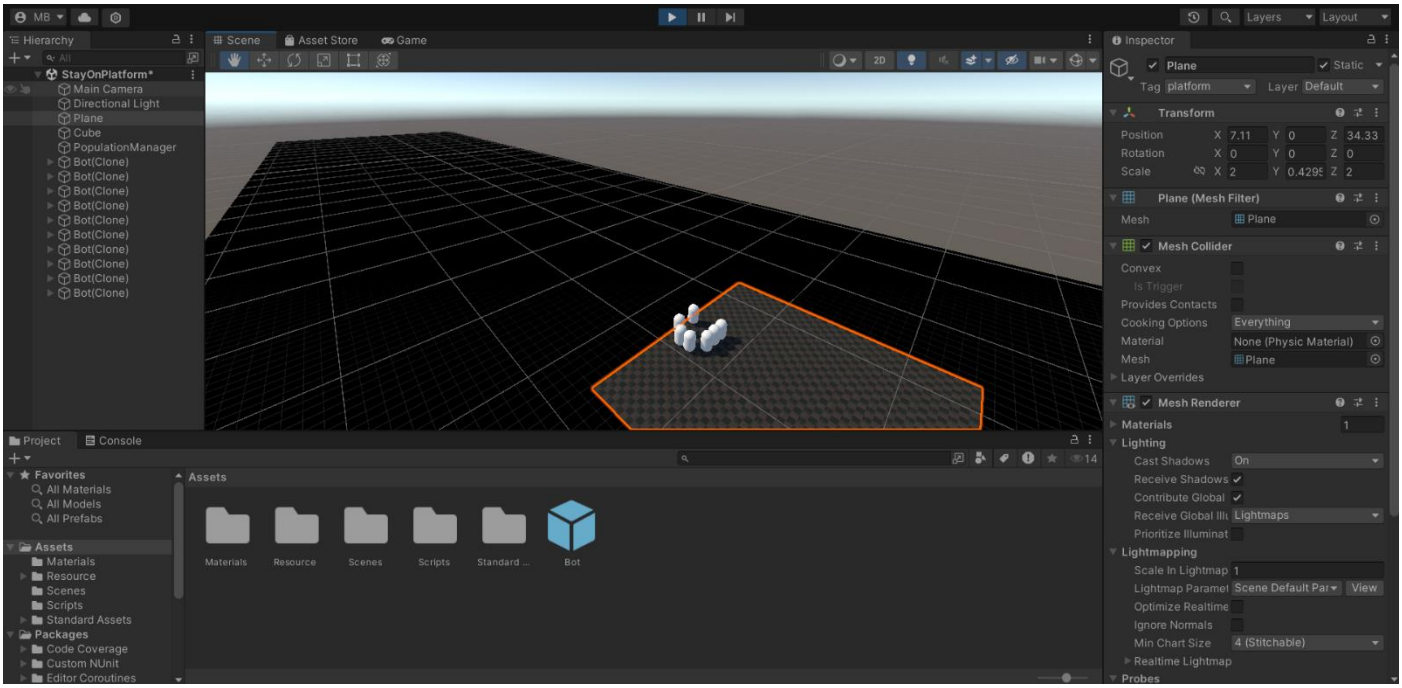
```
    // Method to breed a new population based on the performance of the current population
    void BreedNewPopulation()
    {
        // Sort the population by time alive (from shortest to longest)
        List<GameObject> sortedList = population.OrderBy(o => o.GetComponent<Brain_sc>().timeAliv
e).ToList();
        population.Clear(); // Clear the current population list

        // Create new bots from the top performers in the sorted list
        for (int i = (int)(sortedList.Count / 2.0f) - 1; i < sortedList.Count - 1; i++)
        {
            population.Add(Breed(sortedList[i], sortedList[i + 1])); // Breed parents
            population.Add(Breed(sortedList[i + 1], sortedList[i])); // Breed in reverse order
        }

        // Destroy all previous bots in the sorted list
        for (int i = 0; i < sortedList.Count; i++)
        {
            Destroy(sortedList[i]); // Clean up the previous population
        }

        generation++; // Increment the generation counter
    }

    // Update is called once per frame
    void Update()
    {
        elapsed += Time.deltaTime; // Update elapsed time
                                   // If the trial time has elapsed, breed a new population
        if (elapsed >= trialTime)
        {
            BreedNewPopulation(); // Breed new bots based on performance
            elapsed = 0; // Reset the elapsed time
        }
    }
}
```

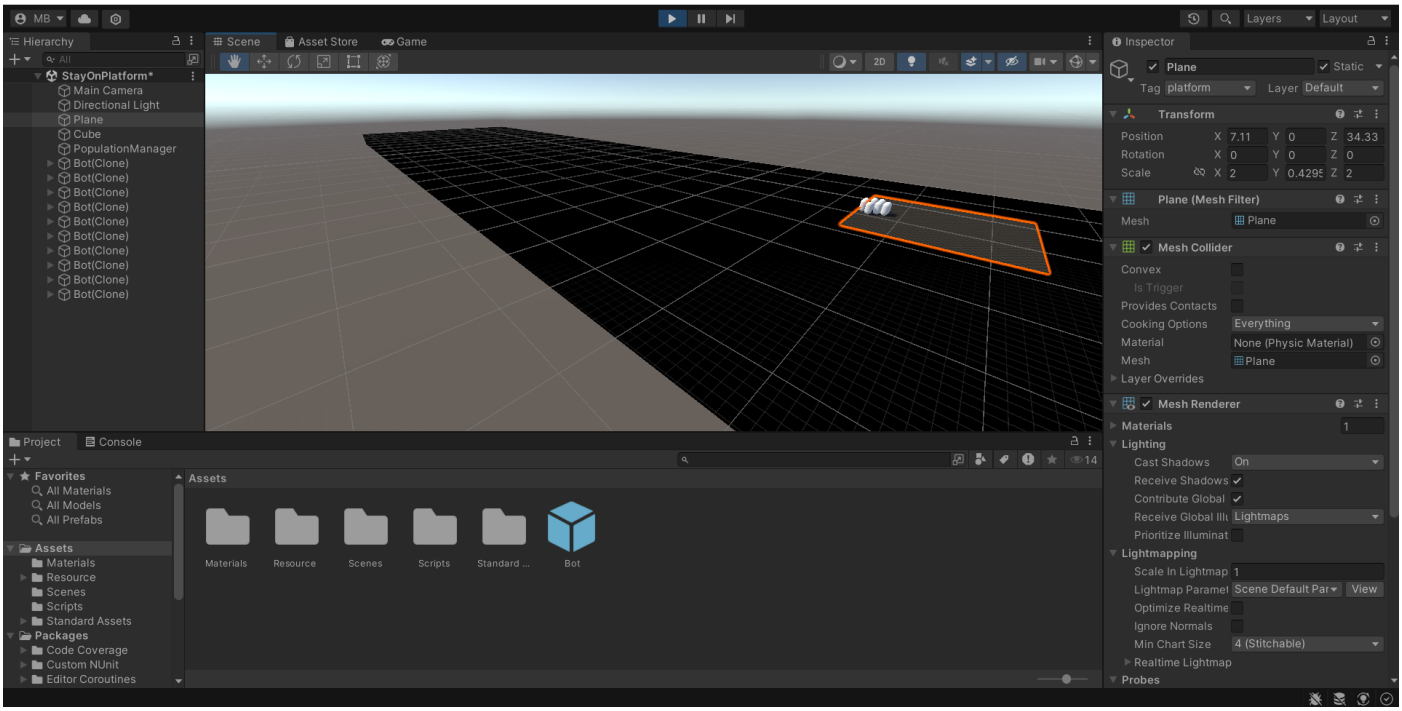
6. Unity'de Ayarlamaları Yapma

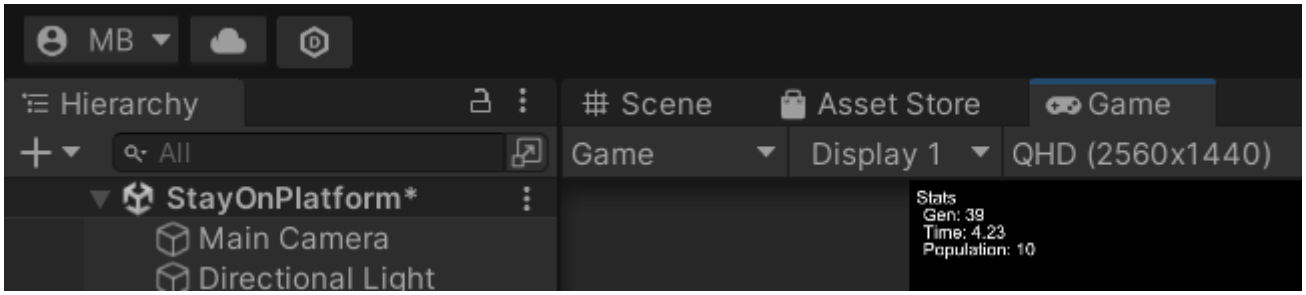
- **PopulationManager Nesnesi Oluşturun**:
  - **Hierarchy** penceresinde boş bir nesne oluşturun ve adını **PopulationManager** olarak değiştirin.
  - **PopulationManager_sc** script'ini bu nesneye ekleyin.
  - PopulationManager içindeki **botPrefab** değişkenine bot prefab'ini atayın.
- **Bot Prefab'i Oluşturun**:
  - **Hierarchy** penceresindeki Bot nesnesini **Project** paneline sürükleyin, böylece bir prefab olarak kaydedilsin.
  - Bot prefab'ini **PopulationManager**'ın **botPrefab** alanına sürükleyerek ekleyin.

7. Oyunun Bazı Ekran Gönüleri :






MB
Hierarchy
Scene
Asset Store
Game
Inspector
StayOnPlatform*
Main Camera
Directional Light
Plane
Cube
PopulationManager
Bot(Clone)
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Materials
Resource
Scenes
Scripts
Standard Assets
Packages
Code Coverage
Custom NUnit
Editor Coroutines
Standard ...
Bot
Plane
Static
Tag platform
Layer Default
Transform
Position
Rotation
Scale
Plane (Mesh Filter)
Mesh
Mesh Collider
Convex
Is Trigger
Provides Contacts
Cooking Options
Everything
Material
None (Physic Material)
Layer Overrides
Mesh Renderer
Materials
Lighting
Cast Shadows
On
Receive Shadows
Lightmapping
Scale In Lightmap
Optimize Realtime
Ignore Normals
Min Chart Size
4 (Stitchable)
Realtime Lightmap
Probes


MB
Hierarchy
Scene
Asset Store
Game
All
Game
Display 1
QHD (2560x1440)
StayOnPlatform*
Main Camera
Directional Light
Stats
Gen: 39
Time: 4.23
Population: 10