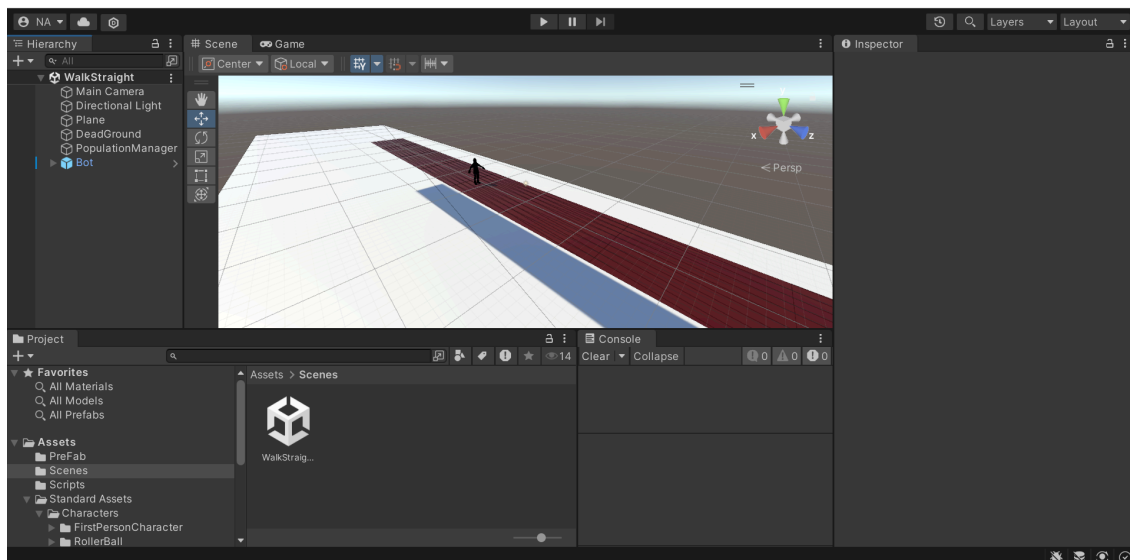


Hafta III

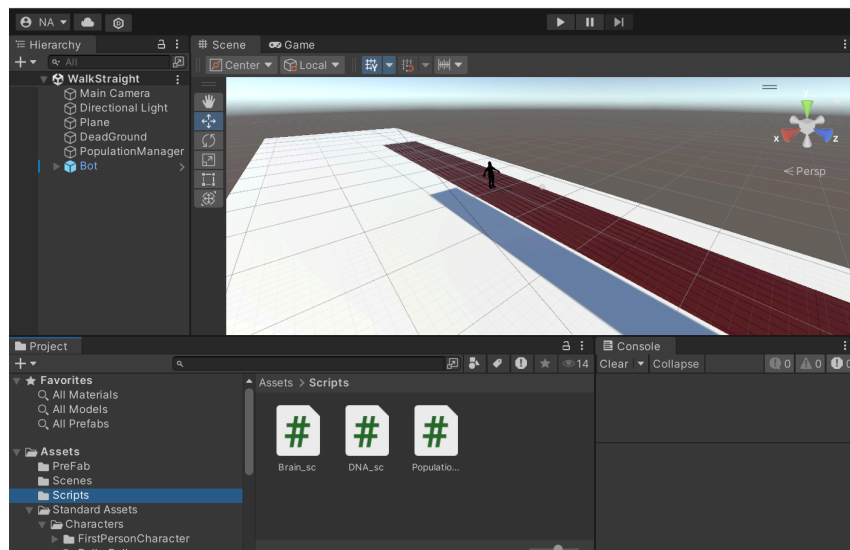
Genetic Programming II

Github link: <https://github.com/Nadi-aa/Genetic-Programming-II>

We start off by creating a new project and naming it accordingly. Then we proceed to download the provided assets and import the package (here Ethan's Package). We replace the default scene and add in the provided scene (WalkStraight).



We create separate folders for Prefab and Scripts. In the scripts folder we add three scripts namely 'DNA_sc', 'Brain_sc', and 'PopulationManager_sc'.

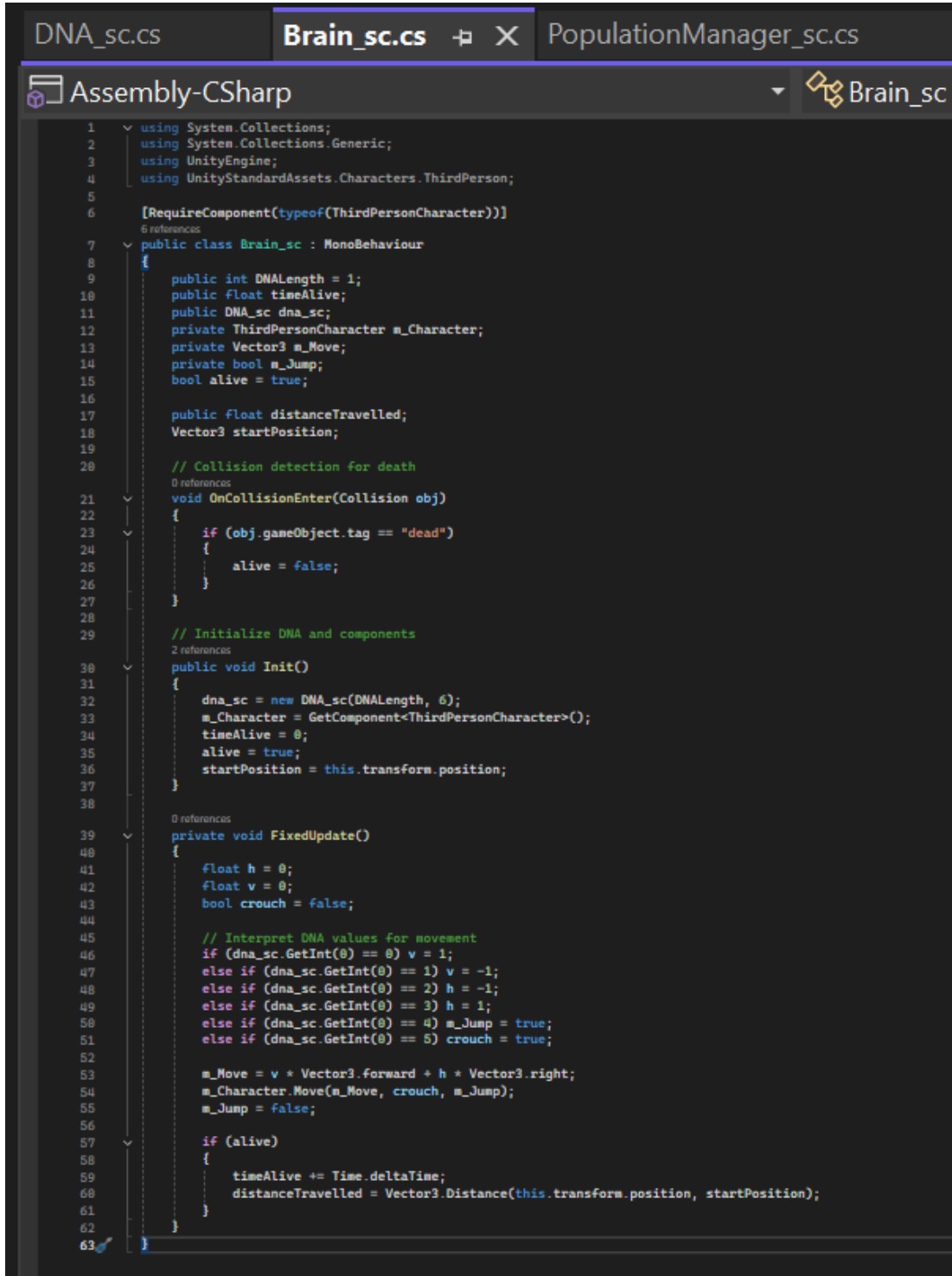


For our project, the “DNA_sc” script simulates genetic material and provides methods for manipulating the genetic data, including mutation, crossover, and initialization. We use different functions like SetRandom(), Mutate(), and Crossover() to initialize, add diversity, and to combine genes.



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class DNA_sc
6 {
7     public List<int> genes = new List<int>();
8     public int dnaLength;
9     public int maxValue;
10
11     public DNA_sc(int length, int max)
12     {
13         dnaLength = length;
14         maxValue = max;
15         SetRandom();
16     }
17
18     // Set random values for DNA genes
19     public void SetRandom()
20     {
21         genes.Clear();
22         for (int i = 0; i < dnaLength; i++)
23         {
24             genes.Add(Random.Range(0, maxValue));
25         }
26     }
27
28     // Set a specific gene at position
29     public void SetInt(int pos, int value)
30     {
31         genes[pos] = value;
32     }
33
34     // Retrieve a specific gene value
35     public int GetInt(int pos)
36     {
37         return genes[pos];
38     }
39
40     // Combine two DNA sets from two parents
41     public void Combine(DNA_sc d1, DNA_sc d2)
42     {
43         for (int i = 0; i < dnaLength; i++)
44         {
45             // Combine half genes from each parent
46             genes[i] = i < dnaLength / 2 ? d1.genes[i] : d2.genes[i];
47         }
48     }
49
50     // Mutation method (note: add mutation rate as parameter)
51     public void Mutate(float mutationRate)
52     {
53         for (int i = 0; i < genes.Count; i++)
54         {
55             if (Random.Range(0f, 1f) < mutationRate)
56             {
57                 genes[i] = Random.Range(0, maxValue);
58             }
59         }
60     }
61
62     // Cross over DNA with a partner
63     public DNA_sc Crossover(DNA_sc partner)
64     {
65         DNA_sc child = new DNA_sc(dnaLength, maxValue);
66         int midpoint = Random.Range(0, dnaLength);
67
68         // Genes from both parents up to midpoint
69         for (int i = 0; i < dnaLength; i++)
70         {
71             child.genes[i] = (i > midpoint) ? genes[i] : partner.genes[i];
72         }
73
74         return child;
75     }
76 }
```

Each bot's logic and behavior in the simulation are specified by the "Brain_sc" script. It interprets the DNA as movement orders and connects it to the agent's behavior and survival. We include functions like Init(), FixedUpdate(), and OnCollisionEnter() initializes, controls movement and monitors the state of the bot.



```
DNA_sc.cs | Brain_sc.cs | PopulationManager_sc.cs
Assembly-CSharp | Brain_sc

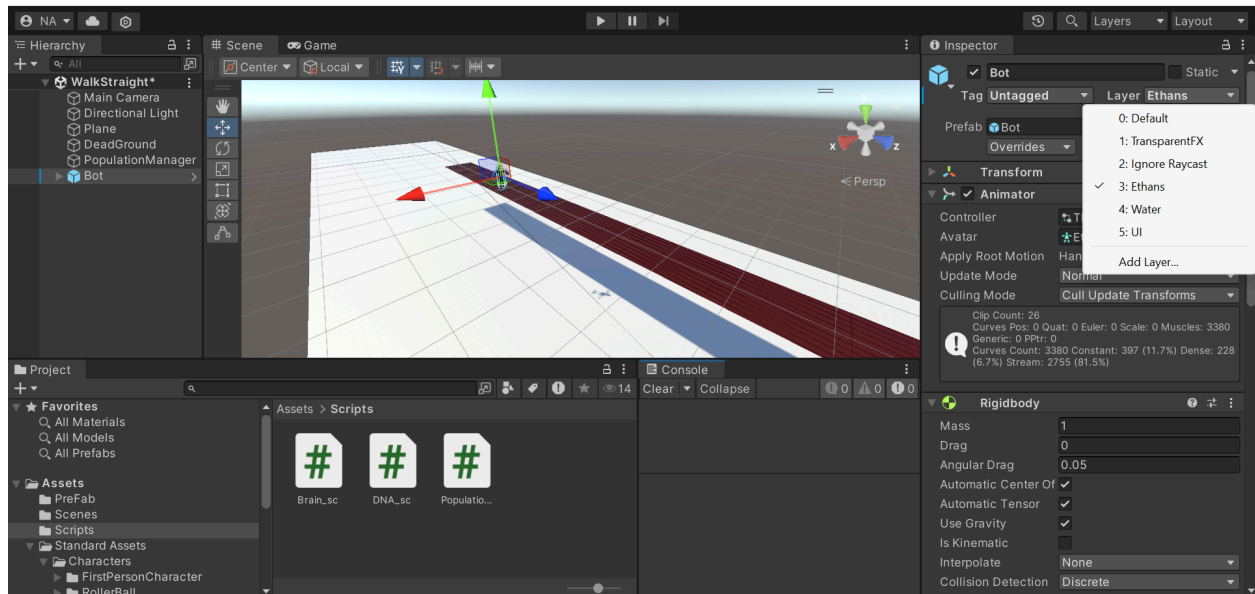
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityStandardAssets.Characters.ThirdPerson;
5
6  [RequireComponent(typeof(ThirdPersonCharacter))]
7  public class Brain_sc : MonoBehaviour
8  {
9      public int DNALength = 1;
10     public float timeAlive;
11     public DNA_sc dna_sc;
12     private ThirdPersonCharacter m_Character;
13     private Vector3 m_Move;
14     private bool m_Jump;
15     bool alive = true;
16
17     public float distanceTravelled;
18     Vector3 startPosition;
19
20     // Collision detection for death
21     void OnCollisionEnter(Collision obj)
22     {
23         if (obj.gameObject.tag == "dead")
24         {
25             alive = false;
26         }
27     }
28
29     // Initialize DNA and components
30     public void Init()
31     {
32         dna_sc = new DNA_sc(DNALength, 6);
33         m_Character = GetComponent<ThirdPersonCharacter>();
34         timeAlive = 0;
35         alive = true;
36         startPosition = this.transform.position;
37     }
38
39     private void FixedUpdate()
40     {
41         float h = 0;
42         float v = 0;
43         bool crouch = false;
44
45         // Interpret DNA values for movement
46         if (dna_sc.GetInt(0) == 0) v = 1;
47         else if (dna_sc.GetInt(0) == 1) v = -1;
48         else if (dna_sc.GetInt(0) == 2) h = -1;
49         else if (dna_sc.GetInt(0) == 3) h = 1;
50         else if (dna_sc.GetInt(0) == 4) m_Jump = true;
51         else if (dna_sc.GetInt(0) == 5) crouch = true;
52
53         m_Move = v * Vector3.forward + h * Vector3.right;
54         m_Character.Move(m_Move, crouch, m_Jump);
55         m_Jump = false;
56
57         if (alive)
58         {
59             timeAlive += Time.deltaTime;
60             distanceTravelled = Vector3.Distance(this.transform.position, startPosition);
61         }
62     }
63 }
```

"PopulationManager_sc" will instantiate copies of the "person" prefab at random positions to form a population. Each "person" will carry the properties and behavior specified in the "DNA_sc" script. So we create an empty Game Object and attach "PopulationManager_sc" to it.

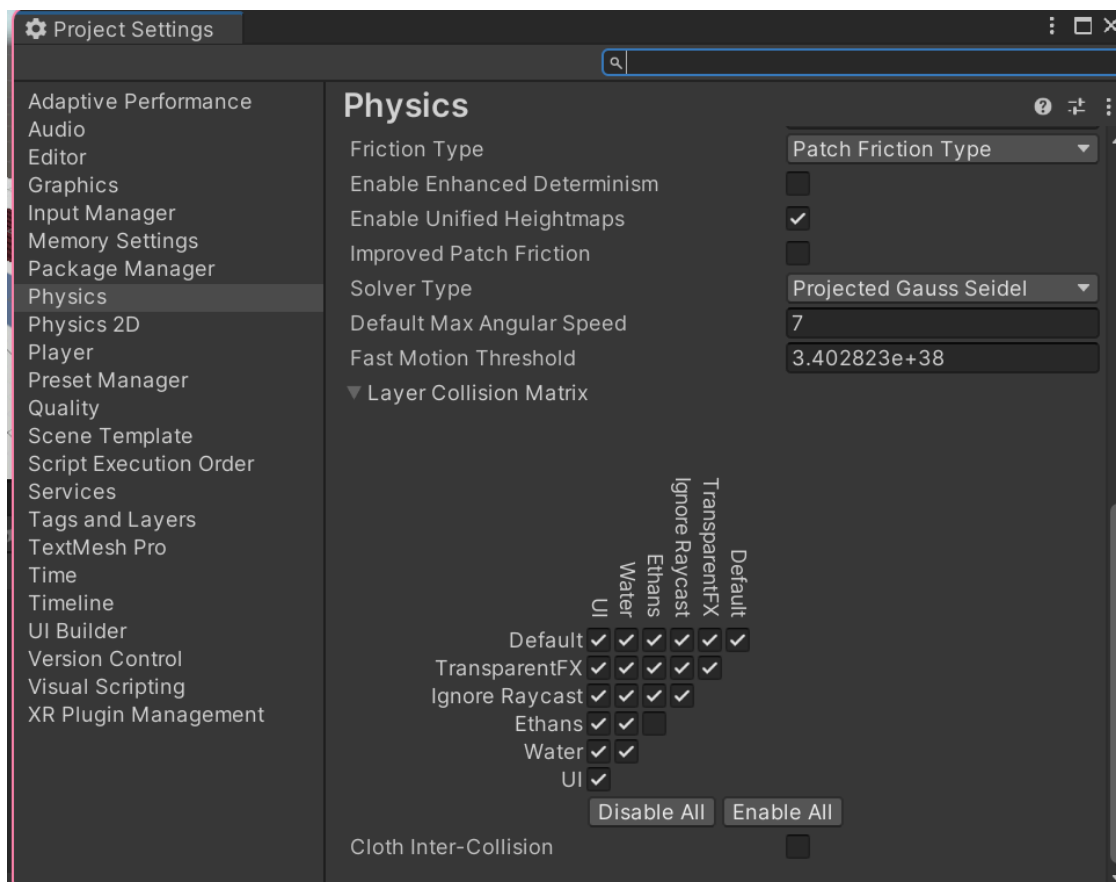
```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System.Linq;
5
6  0 references
7  public class PopulationManager_sc : MonoBehaviour
8  {
9      public GameObject botPrefab;
10     public int populationSize = 50;
11     List<GameObject> population = new List<GameObject>();
12     public static float elapsed = 0;
13     public float trialTime = 5;
14     int generation = 1;
15     GUIStyle guiStyle = new GUIStyle();
16
17     0 references
18     void OnGUI()
19     {
20         guiStyle.fontSize = 25;
21         guiStyle.normal.textColor = Color.white;
22
23         GUI.BeginGroup(new Rect(10, 10, 250, 150));
24         GUI.Box(new Rect(0, 0, 140, 140), "Stats", guiStyle);
25         GUI.Label(new Rect(10, 25, 200, 30), "Gen: " + generation, guiStyle);
26         GUI.Label(new Rect(10, 50, 200, 30), string.Format("Time: {0:0.00}", elapsed), guiStyle);
27         GUI.Label(new Rect(10, 75, 200, 30), "Population: " + population.Count, guiStyle);
28         GUI.EndGroup();
29     }
30
31     0 references
32     void Start()
33     {
34         // Instantiate population
35         for (int i = 0; i < populationSize; i++)
36         {
37             Vector3 startingPos = new Vector3(
38                 this.transform.position.x + Random.Range(-2, 2),
39                 this.transform.position.y,
40                 this.transform.position.z + Random.Range(-2, 2)
41             );
42             GameObject b = Instantiate(botPrefab, startingPos, this.transform.rotation);
43             b.GetComponent<Brain_sc>().Init();
44             population.Add(b);
45         }
46
47     2 references
48     GameObject Breed(GameObject parent1, GameObject parent2)
49     {
50         Vector3 startingPos = new Vector3(
51             this.transform.position.x + Random.Range(-2, 2),
52             this.transform.position.y,
53             this.transform.position.z + Random.Range(-2, 2)
54         );
55         GameObject offspring = Instantiate(botPrefab, startingPos, this.transform.rotation);
56         Brain_sc b = offspring.GetComponent<Brain_sc>();
57
58         // Mutate or combine genes
59         if (Random.Range(0, 100) < 1) // mutate with a small probability
60         {
61             b.Init();
62             b.dna_sc.Mutate(0.01f); // Example mutation rate
63         }
64         else
65         {
66             b.Init();
67         }
68     }
69 }
```

42 % | No issues found

In order to assure that the collision does take place we add a new layer to the bot object from the inspector. We add our own layer and name it "Ethans".



After that we go to the project settings > Physics and disable the Ethans x Ethans layer to prevent collision.



Summary:

In this simulation, a group of bots learns to move smarter over time by passing down successful traits to the next generation. Starting with random movement, they gradually get better at navigating their environment as each generation builds on the last.

The screenshots of game play:

