```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  public class TestMaster : Master
5  {
6      private readonly Dictionary<Player, NeuralNetwork> Testers = new
         Dictionary<Player, NeuralNetwork>();
7      public Rigidbody2D Bot;
8      private float StartTime;
9      private readonly List<NeuralNetwork> Victors = new List<NeuralNetwork>
         ();
10     private readonly int TotalTesters = 120;
11     private float PreviousMax = 0;
12     public LabelText TestersLabel;
13     public LabelText GenerationLabel;
14     public Color32[] Colours = new Color32[5];
15
16
17     protected override void Start()
18     {
19         StartTime = Time.time;
20         InitialiseCamera();
21
22         // Finds the previous generation number
23         NeuralNetwork generation = new NeuralNetwork("NeuralNetwork2");
24         try
25         {
26             Stats.Generation = generation.GetGeneration();
27             GenerationLabel.Start();
28         }
29         catch { }
30
31         // Adds tester bots to the map
32         for (int counter = 0; counter < TotalTesters; counter ++)
33         {
34             Vector3 pos = new Vector3(9, 19);
35             string name = "Bot" + counter ;
36             var bot = Instantiate(Bot, pos, transform.rotation);
37             ChangeColour(counter, bot);
38             bot.name = name;
39             BotMovement move = bot.GetComponent<BotMovement>();
40             move.Tester = true;
41             move.M = this;
42             move.Name = name;
43             move.InputSpeed(Speed);
44             move.InputJumpHeight(JumpHeight);
45             Modify(counter, move);
46             InitialisePlayerNetwork(name, move.GetController());
47         }
48     }
49
50     // Changes the colour based on what neural network spawned the bot
51     private void ChangeColour(int counter , Rigidbody2D bot)
```

```
52      {
53          if (counter > 60)
54          {
55              bot.GetComponent<SpriteRenderer>().color = Colours[1];
56          }
57          else if (counter == 60)
58          {
59              bot.GetComponent<SpriteRenderer>().color = Colours[0];
60          }
61          else if (counter > 20)
62          {
63              bot.GetComponent<SpriteRenderer>().color = Colours[3];
64          }
65          else if (counter == 20)
66          {
67              bot.GetComponent<SpriteRenderer>().color = Colours[2];
68          }
69          else if (counter == 0)
70          {
71              bot.GetComponent<SpriteRenderer>().color = Colours[4];
72          }
73          else
74          {
75              bot.GetComponent<SpriteRenderer>().color = Colours[5];
76          }
77      }
78
79      // Modifies the neural network
80      private void Modify(int counter, BotMovement move)
81      {
82          if (counter > 60)
83          {
84              move.InputNetwork("NeuralNetwork2");
85              move.Modify(1);
86          }
87          else if (counter == 60)
88          {
89              move.InputNetwork("NeuralNetwork2");
90              move.IncreaseGeneration();
91          }
92          else if (counter > 20)
93          {
94              move.InputNetwork("NeuralNetwork1");
95              move.Modify(2);
96          }
97          else if (counter == 20)
98          {
99              move.InputNetwork("NeuralNetwork1");
100             move.IncreaseGeneration();
101         }
102         else if (counter == 0)
103         {
104             move.InputNetwork("NeuralNetwork0");
```

```
105             move.IncreaseGeneration();
106         }
107         else
108         {
109             move.InputNetwork("NeuralNetwork0");
110             move.Modify(3);
111         }
112     }
113 
114     public void Update()
115     {
116         // Restarts the timer if the furthest distance any player has travelled is increased
117         foreach (Player player in Players)
118         {
119             if (player.TotalDistance() > PreviousMax + 1)
120             {
121                 StartTime = Time.time;
122                 PreviousMax = player.TotalDistance();
123             }
124         }
125 
126         // If the maximum distance has not increased in 20 seconds, it is likely that all testers have got stuck
127         if (Time.time - StartTime >= 20)
128         {
129             ResolveTimer();
130         }
131     }
132 
133     // If nobody moves forward for enough time, the game will end
134     private void ResolveTimer()
135     {
136         if (Testers.Count > 3)
137         {
138             // Finds the position of the best 3 players
139             float[] distances = new float[3] { -20, -20, -20 };
140             Player[] bestPlayers = new Player[3];
141             foreach (Player player in Players)
142             {
143                 if (Testers.ContainsKey(player))
144                 {
145                     // If the player is further ahead of any of the bestPlayers, they will become a new bestPlayer
146                     float distance = player.TotalDistance();
147                     if (distance > distances[0])
148                     {
149                         distances[2] = distances[1];
150                         bestPlayers[2] = bestPlayers[1];
151                         distances[1] = distances[0];
152                         bestPlayers[1] = bestPlayers[0];
153                         distances[0] = distance;
154                         bestPlayers[0] = player;
```

```
155                     }
156                     else if (distance > distances[1])
157                     {
158                         distances[2] = distances[1];
159                         bestPlayers[2] = bestPlayers[1];
160                         distances[1] = distance;
161                         bestPlayers[1] = player;
162                     }
163                     else if (distance > distances[2])
164                     {
165                         distances[2] = distance;
166                         bestPlayers[2] = player;
167                     }
168                 }
169             }
170 
171             Victors.Add(Testers[bestPlayers[2]]);
172             Victors.Add(Testers[bestPlayers[1]]);
173             Victors.Add(Testers[bestPlayers[0]]);
174         }
175         else if (Testers.Count == 3)
176         {
177             // Organises the final 3 players
178             List<Player> remaining = new List<Player>(Testers.Keys);
179             if (remaining[0].TotalDistance() >= remaining[1].TotalDistance() ↵
                    && remaining[0].TotalDistance() >= remaining[2].TotalDistance ↵
                   ())
180             {
181                 if (remaining[1].TotalDistance() >= remaining                ↵
                      [2].TotalDistance())
182                 {
183                     Victors.Add(Testers[remaining[2]]);
184                     Victors.Add(Testers[remaining[1]]);
185                     Victors.Add(Testers[remaining[0]]);
186                 }
187                 else
188                 {
189                     Victors.Add(Testers[remaining[1]]);
190                     Victors.Add(Testers[remaining[2]]);
191                     Victors.Add(Testers[remaining[0]]);
192                 }
193             }
194             else if (remaining[1].TotalDistance() >= remaining               ↵
                   [0].TotalDistance() && remaining[1].TotalDistance() >=         ↵
                   remaining[2].TotalDistance())
195             {
196                 if (remaining[0].TotalDistance() >= remaining                ↵
                      [2].TotalDistance())
197                 {
198                     Victors.Add(Testers[remaining[2]]);
199                     Victors.Add(Testers[remaining[0]]);
200                     Victors.Add(Testers[remaining[1]]);
201                 }
```

```
202                 else
203                 {
204                     Victors.Add(Testers[remaining[0]]);
205                     Victors.Add(Testers[remaining[2]]);
206                     Victors.Add(Testers[remaining[1]]);
207                 }
208             }
209             else
210             {
211                 if (remaining[0].TotalDistance() >= remaining
                      [1].TotalDistance())
212                 {
213                     Victors.Add(Testers[remaining[1]]);
214                     Victors.Add(Testers[remaining[0]]);
215                     Victors.Add(Testers[remaining[2]]);
216                 }
217                 else
218                 {
219                     Victors.Add(Testers[remaining[0]]);
220                     Victors.Add(Testers[remaining[1]]);
221                     Victors.Add(Testers[remaining[2]]);
222                 }
223             }
224         }
225         else if (Testers.Count == 2)
226         {
227             // Organises the final 2 players
228             List<Player> remaining = new List<Player>(Testers.Keys);
229             if (remaining[0].TotalDistance() >= remaining[1].TotalDistance
                  ())
230             {
231                 Victors.Add(Testers[remaining[1]]);
232                 Victors.Add(Testers[remaining[0]]);
233             }
234             else
235             {
236                 Victors.Add(Testers[remaining[0]]);
237                 Victors.Add(Testers[remaining[1]]);
238             }
239         }
240         else if (Testers.Count == 1)
241         {
242             List<Player> remaining = new List<Player>(Testers.Keys);
243             Victors.Add(Testers[remaining[0]]);
244         }
245 
246         Save();
247         // Repeats the test
248         if (Arena)
249         {
250             ChangeScene.ChangeToScene(4);
251         }
252         else
```

```
253         {
254             ChangeScene.ChangeToScene(3);
255         }
256     }
257 
258     protected override void ResolveDeath(string name)
259     {
260         try
261         {
262             // Updates the number of Testers
263             TestersLabel.Change();
264         }
265         catch { }
266         if (Testers.Count <= 3)
267         {
268             // Adds the player to Victors
269             Victors.Add(Testers[GetPlayer(name)]);
270         }
271         // Removes the player from Testers
272         Testers.Remove(GetPlayer(name));
273         if (Testers.Count == 1)
274         {
275             foreach(Player player in Players)
276             {
277                 if (Testers.ContainsKey(player))
278                 {
279                     Victors.Add(Testers[player]);
280                 }
281             }
282             Save();
283             // Repeats the test
284             if (Arena)
285             {
286                 ChangeScene.ChangeToScene(4);
287             }
288             else
289             {
290                 ChangeScene.ChangeToScene(3);
291             }
292         }
293     }
294 
295     // Saves the best neural networks
296     private void Save()
297     {
298         for (int counter = 0; counter < Victors.Count; counter ++)
299         {
300             Victors[counter].Save( "NeuralNetwork" + counter );
301         }
302     }
303 
304     // Adds the testers
305     private void InitialisePlayerNetwork(string name, NeuralNetwork net)
```

```
306     {
307         if (Players.Count == 0)
308         {
309             if (Arena)
310             {
311                 ArenaCounter = 4;
312             }
313         }
314         Player player = new Player(name, ChunkLength);
315         Players.Add(player);
316         Testers.Add(player, net);
317     }
318 }
```