```
 1  using System.Collections;
 2  using System.Collections.Generic;
 3  using UnityEngine;
 4  using UnityEngine.SceneManagement;
 5  using TMPro;
 6
 7  public class GameManager : MonoBehaviour
 8  {
 9      public static GameManager Instance { get; private set; }
10      public List<Blocks> allBlocks;
11      public List<Blocks> firstBlocks;
12      public List<GameObject> darkerBlocks;
13      public GameObject character;
14
15      public Animator loseAnim;
16      public Animator catAnim;
17      public GameObject loseLogo;
18      public Animator winAnim;
19      public GameObject winLogo;
20
21      public bool didFinished;
22      private bool isInputEnabled = true;
23      public Blocks CurrentBlock { get; private set; }
24
25      public int clickCount = 0;
26
27
28      public TextMeshProUGUI winCounterText;
29      public TextMeshProUGUI loseCounterText;
30
31      private int winCounter = 0;
32      private int loseCounter = 0;
33
34      void Awake()
35      {
36          if (Instance == null)
37          {
38              Instance = this;
39          }
40          else
41          {
42              Destroy(gameObject);
43          }
44      }
45
46      private void Update()
47      {
48          if (CurrentBlock.isEscape)
49          {
50              didFinished = true;
51          }
52      }
53
```

```
54      void Start()
55      {
56          StartCoroutine(StartCat(0.01f));
57          ActivateRandomDarkerBlocks();
58          EnableInput();
59          UpdateWinLoseCounters();
60      }
61
62      public void ClickControl()
63      {
64          foreach (Blocks block in allBlocks)
65          {
66              PolygonCollider2D collider =
                  block.GetComponent<PolygonCollider2D>();
67              if (collider != null)
68              {
69                  collider.enabled = false;
70              }
71          }
72
73          StartCoroutine(EnableClick(0.6f));
74      }
75
76      public void MovesCounter()
77      {
78          clickCount++;
79
80          if (clickCount >= 2)
81          {
82              clickCount = 0;
83              if (!didFinished)
84              {
85                  MoveCatTowardsFinishLine();
86              }
87              else
88              {
89                  GoToFinishLine();
90              }
91          }
92      }
93
94      IEnumerator EnableClick(float delay)
95      {
96          yield return new WaitForSeconds(delay);
97
98          foreach (Blocks block in allBlocks)
99          {
100             PolygonCollider2D collider =
                  block.GetComponent<PolygonCollider2D>();
101             if (collider != null)
102             {
103                 collider.enabled = true;
104             }
```

```
105         }
106     }
107
108     public void DisableInputForSeconds(float seconds)
109     {
110         isInputEnabled = false;
111         StartCoroutine(EnableInputAfterDelay(seconds));
112     }
113
114     private IEnumerator EnableInputAfterDelay(float seconds)
115     {
116         yield return new WaitForSeconds(seconds);
117         EnableInput();
118     }
119
120     private void EnableInput()
121     {
122         isInputEnabled = true;
123     }
124
125     public bool IsInputEnabled()
126     {
127         return isInputEnabled;
128     }
129
130     public void WinControl()
131     {
132         if (CurrentBlock.adjacentBlocks.Count == 0)
133         {
134             Won();
135         }
136     }
137
138     void SetCatToRandomBlock()
139     {
140         if (allBlocks.Count == 0 || character == null)
141         {
142             Debug.LogError("Blocks list is empty or character is not
                assigned.");
143             return;
144         }
145
146         List<Blocks> nonEscapeBlocks = firstBlocks.FindAll(block => !
              block.isEscape && !block.isDarker && block.isFirst);
147
148         if (nonEscapeBlocks.Count == 0)
149         {
150             Debug.LogError("No available blocks for the cat to
                start.");
151             return;
152         }
153
154         int randomIndex = Random.Range(0, nonEscapeBlocks.Count);
```

```
155         CurrentBlock = nonEscapeBlocks[randomIndex];
156         character.transform.position = CurrentBlock.transform.position;
157 
158         Debug.Log($"Kedi başlangıç bloğuna yerleştirildi:
              {CurrentBlock.name}");
159     }
160 
161     public void ActivateRandomDarkerBlocks()
162     {
163         if (darkerBlocks == null || darkerBlocks.Count == 0)
164         {
165             Debug.LogWarning("DarkerBlocks listesi boş.");
166             return;
167         }
168 
169         int randomCount = Random.Range(3, 16);
170         List<GameObject> randomDarkerBlocks = new List<GameObject>
              (darkerBlocks);
171 
172         for (int i = 0; i < randomCount && randomDarkerBlocks.Count >
              0; i++)
173         {
174             int randomIndex = Random.Range(3,
                  randomDarkerBlocks.Count);
175             GameObject blockToActivate = randomDarkerBlocks
                  [randomIndex];
176             blockToActivate.SetActive(true);
177             randomDarkerBlocks.RemoveAt(randomIndex);
178         }
179 
180         Debug.Log($"{randomCount} adet DarkerBlock aktif edildi.");
181     }
182 
183     public void GoToFinishLine()
184     {
185         GameObject finishLine = CurrentBlock.GetComponent<Blocks>
              ().finishLine;
186 
187         Vector3 direction = finishLine.transform.position -
              character.transform.position;
188 
189         if (Mathf.Abs(direction.y) > Mathf.Abs(direction.x))
190         {
191             if (direction.y > 0)
192             {
193                 catAnim.SetTrigger("UpAnim");
194             }
195             else
196             {
197                 catAnim.SetTrigger("DownAnim");
198             }
199         }
200         else
```

```
201         {
202             if (direction.x > 0)
203             {
204                 catAnim.SetTrigger("RightAnim");
205             }
206             else
207             {
208                 catAnim.SetTrigger("WalkAnim");
209             }
210         }
211 
212         LeanTween.move(character, finishLine.transform.position, 0.3f)
213             .setEase(LeanTweenType.easeInOutQuad)
214             .setOnComplete(() => { });
215 
216         if (loseAnim != null && didFinished)
217         {
218             loseCounter++;
219             UpdateWinLoseCounters();
220             DisableInputForSeconds(3f);
221             loseLogo.SetActive(true);
222             loseAnim.SetTrigger("Win");
223             ResetGame();
224             return;
225         }
226     }
227 
228     public void Won()
229     {
230         if (winAnim != null)
231         {
232             winCounter++;
233             UpdateWinLoseCounters();
234             DisableInputForSeconds(3f);
235             winLogo.SetActive(true);
236             winAnim.SetTrigger("Win");
237             ResetGame();
238             return;
239         }
240     }
241 
242     private void ResetGame()
243     {
244         didFinished = false;
245         clickCount = 0;
246         CurrentBlock = null;
247 
248 
249         foreach (Blocks block in allBlocks)
250         {
251             block.ResetBlock();
252         }
253 
```

```
254         SetCatToRandomBlock();
255         ActivateRandomDarkerBlocks();
256         loseLogo.SetActive(false);
257         winLogo.SetActive(false);
258         EnableInput();
259         SetCatToRandomBlock();
260     }
261 
262 
263     private void UpdateWinLoseCounters()
264     {
265         if (winCounterText != null)
266         {
267             winCounterText.text = $"Wins: {winCounter}";
268         }
269         if (loseCounterText != null)
270         {
271             loseCounterText.text = $"Losses: {loseCounter}";
272         }
273     }
274 
275     public IEnumerator StartCat(float delay)
276     {
277         yield return new WaitForSeconds(delay);
278         SetCatToRandomBlock();
279     }
280 
281     public Blocks FindClosestFinishLine()
282     {
283         Blocks closestFinishLine = null;
284         float shortestDistance = Mathf.Infinity;
285 
286         foreach (Blocks block in allBlocks)
287         {
288             if (block.isEscape)
289             {
290                 float distance = Vector3.Distance                    ↵
                      (character.transform.position,                  ↵
                      block.transform.position);
291                 if (distance < shortestDistance)
292                 {
293                     shortestDistance = distance;
294                     closestFinishLine = block;
295                 }
296             }
297         }
298 
299         return closestFinishLine;
300     }
301 
302     public List<Blocks> lastBlocks = new List<Blocks>();
303     public int memoryLimit = 10;
304 
```

```
305     public void MoveCatTowardsFinishLine()
306     {
307         Debug.Log("Calisti");
308         Blocks targetBlock = FindClosestFinishLine();
309 
310         if (targetBlock == null)
311         {
312             Debug.LogWarning("No finish line block found.");
313             return;
314         }
315 
316         Blocks bestNextBlock = null;
317         float shortestDistance = Mathf.Infinity;
318 
319         List<Blocks> validAdjacentBlocks = new List<Blocks>();
320 
321         foreach (Blocks adjacentBlock in CurrentBlock.adjacentBlocks)
322         {
323             if (adjacentBlock.canCame)
324             {
325                 validAdjacentBlocks.Add(adjacentBlock);
326                 float distanceToTarget = Vector3.Distance
                      (adjacentBlock.transform.position,
                      targetBlock.transform.position);
327 
328                 if (distanceToTarget < shortestDistance && !
                      lastBlocks.Contains(adjacentBlock))
329                 {
330                     shortestDistance = distanceToTarget;
331                     bestNextBlock = adjacentBlock;
332                 }
333             }
334         }
335 
336         if (bestNextBlock == null && validAdjacentBlocks.Count > 0)
337         {
338             List<Blocks> filteredBlocks = validAdjacentBlocks.FindAll
                  (block => !lastBlocks.Contains(block));
339 
340             if (filteredBlocks.Count > 0)
341             {
342                 bestNextBlock = filteredBlocks[Random.Range(0,
                      filteredBlocks.Count)];
343             }
344             else
345             {
346                 bestNextBlock = validAdjacentBlocks[Random.Range(0,
                      validAdjacentBlocks.Count)];
347             }
348         }
349 
350 
351         if (bestNextBlock != null)
```

```
352         {
353             lastBlocks.Add(bestNextBlock);
354 
355 
356             if (lastBlocks.Count > memoryLimit)
357             {
358                 lastBlocks.RemoveAt(0);
359             }
360 
361 
362             MoveCatToBlock(bestNextBlock);
363         }
364         else
365         {
366             Debug.LogWarning("No valid adjacent block found for the cat ↪
                to move.");
367         }
368     }
369 
370     private void MoveCatToBlock(Blocks targetBlock)
371     {
372         Vector3 direction = targetBlock.transform.position -            ↪
              character.transform.position;
373 
374         if (Mathf.Abs(direction.y) > Mathf.Abs(direction.x))
375         {
376             if (direction.y > 0)
377             {
378                 catAnim.SetTrigger("UpAnim");
379             }
380             else
381             {
382                 catAnim.SetTrigger("DownAnim");
383             }
384         }
385         else
386         {
387             if (direction.x > 0)
388             {
389                 catAnim.SetTrigger("RightAnim");
390             }
391             else
392             {
393                 catAnim.SetTrigger("WalkAnim");
394             }
395         }
396 
397 
398         LeanTween.move(character, targetBlock.transform.position, 0.3f)
399             .setEase(LeanTweenType.easeInOutQuad)
400             .setOnComplete(() =>
401             {
402                 CurrentBlock = targetBlock;
```

```
403                     WinControl();
404                 });
405         }
406     }
407
408
409
```