```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using TMPro;

public class GameManager : MonoBehaviour
{
    public static GameManager Instance { get; private set; }
    public List<Blocks> allBlocks;
    public List<Blocks> firstBlocks;
    public List<GameObject> darkerBlocks;
    public GameObject character;

    public Animator loseAnim;
    public Animator catAnim;
    public GameObject loseLogo;
    public Animator winAnim;
    public GameObject winLogo;

    public bool didFinished;
    private bool isInputEnabled = true;
    public Blocks CurrentBlock { get; private set; }

    public int clickCount = 0;


    public TextMeshProUGUI winCounterText;
    public TextMeshProUGUI loseCounterText;

    private int winCounter = 0;
    private int loseCounter = 0;

    void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
        }
        else
        {
            Destroy(gameObject);
        }
    }

    private void Update()
    {
        if (CurrentBlock.isEscape)
        {
            didFinished = true;
        }
    }

    void Start()
    {
        StartCoroutine(StartCat(0.01f));
        ActivateRandomDarkerBlocks();
        EnableInput();
        UpdateWinLoseCounters();
    }

    public void ClickControl()
    {
        foreach (Blocks block in allBlocks)
        {
            PolygonCollider2D collider = 
              block.GetComponent<PolygonCollider2D>();
            if (collider != null)
            {
                collider.enabled = false;
            }
        }

        StartCoroutine(EnableClick(0.6f));
    }

    public void MovesCounter()
    {
        clickCount++;

        if (clickCount >= 2)
        {
            clickCount = 0;
            if (!didFinished)
            {
                MoveCatTowardsFinishLine();
            }
            else
            {
                GoToFinishLine();
            }
        }
    }

    IEnumerator EnableClick(float delay)
    {
        yield return new WaitForSeconds(delay);

        foreach (Blocks block in allBlocks)
        {
            PolygonCollider2D collider = 
              block.GetComponent<PolygonCollider2D>();
            if (collider != null)
            {
                collider.enabled = true;
            }
```

```csharp
        }
    }

    public void DisableInputForSeconds(float seconds)
    {
        isInputEnabled = false;
        StartCoroutine(EnableInputAfterDelay(seconds));
    }

    private IEnumerator EnableInputAfterDelay(float seconds)
    {
        yield return new WaitForSeconds(seconds);
        EnableInput();
    }

    private void EnableInput()
    {
        isInputEnabled = true;
    }

    public bool IsInputEnabled()
    {
        return isInputEnabled;
    }

    public void WinControl()
    {
        if (CurrentBlock.adjacentBlocks.Count == 0)
        {
            Won();
        }
    }

    void SetCatToRandomBlock()
    {
        if (allBlocks.Count == 0 || character == null)
        {
            Debug.LogError("Blocks list is empty or character is not assigned.");
            return;
        }

        List<Blocks> nonEscapeBlocks = firstBlocks.FindAll(block => !block.isEscape && !block.isDarker && block.isFirst);

        if (nonEscapeBlocks.Count == 0)
        {
            Debug.LogError("No available blocks for the cat to start.");
            return;
        }

        int randomIndex = Random.Range(0, nonEscapeBlocks.Count);
        CurrentBlock = nonEscapeBlocks[randomIndex];
        character.transform.position = CurrentBlock.transform.position;

        Debug.Log($"Kedi başlangıç bloğuna yerleştirildi: {CurrentBlock.name}");
    }

    public void ActivateRandomDarkerBlocks()
    {
        if (darkerBlocks == null || darkerBlocks.Count == 0)
        {
            Debug.LogWarning("DarkerBlocks listesi boş.");
            return;
        }

        int randomCount = Random.Range(3, 16);
        List<GameObject> randomDarkerBlocks = new List<GameObject>(darkerBlocks);

        for (int i = 0; i < randomCount && randomDarkerBlocks.Count > 0; i++)
        {
            int randomIndex = Random.Range(3, randomDarkerBlocks.Count);
            GameObject blockToActivate = randomDarkerBlocks[randomIndex];
            blockToActivate.SetActive(true);
            randomDarkerBlocks.RemoveAt(randomIndex);
        }

        Debug.Log($"{randomCount} adet DarkerBlock aktif edildi.");
    }

    public void GoToFinishLine()
    {
        GameObject finishLine = CurrentBlock.GetComponent<Blocks>().finishLine;

        Vector3 direction = finishLine.transform.position - character.transform.position;

        if (Mathf.Abs(direction.y) > Mathf.Abs(direction.x))
        {
            if (direction.y > 0)
            {
                catAnim.SetTrigger("UpAnim");
            }
            else
            {
                catAnim.SetTrigger("DownAnim");
            }
        }
        else
        {
            if (direction.x > 0)
            {
                catAnim.SetTrigger("RightAnim");
            }
            else
            {
                catAnim.SetTrigger("WalkAnim");
            }
        }

        LeanTween.move(character, finishLine.transform.position, 0.3f)
            .setEase(LeanTweenType.easeInOutQuad)
            .setOnComplete(() => { });

        if (loseAnim != null && didFinished)
        {
            loseCounter++;
            UpdateWinLoseCounters();
            DisableInputForSeconds(3f);
            loseLogo.SetActive(true);
            loseAnim.SetTrigger("Win");
            ResetGame();
            return;
        }
    }

    public void Won()
    {
        if (winAnim != null)
        {
            winCounter++;
            UpdateWinLoseCounters();
            DisableInputForSeconds(3f);
            winLogo.SetActive(true);
            winAnim.SetTrigger("Win");
            ResetGame();
            return;
        }
    }

    private void ResetGame()
    {
        didFinished = false;
        clickCount = 0;
        CurrentBlock = null;


        foreach (Blocks block in allBlocks)
        {
            block.ResetBlock();
        }

        SetCatToRandomBlock();
        ActivateRandomDarkerBlocks();
        loseLogo.SetActive(false);
        winLogo.SetActive(false);
        EnableInput();
        SetCatToRandomBlock();
    }


    private void UpdateWinLoseCounters()
    {
        if (winCounterText != null)
        {
            winCounterText.text = $"Wins: {winCounter}";
        }
        if (loseCounterText != null)
        {
            loseCounterText.text = $"Losses: {loseCounter}";
        }
    }

    public IEnumerator StartCat(float delay)
    {
        yield return new WaitForSeconds(delay);
        SetCatToRandomBlock();
    }

    public Blocks FindClosestFinishLine()
    {
        Blocks closestFinishLine = null;
        float shortestDistance = Mathf.Infinity;

        foreach (Blocks block in allBlocks)
        {
            if (block.isEscape)
            {
                float distance = Vector3.Distance
                  (character.transform.position,
                  block.transform.position);
                if (distance < shortestDistance)
                {
                    shortestDistance = distance;
                    closestFinishLine = block;
                }
            }
        }

        return closestFinishLine;
    }

    public List<Blocks> lastBlocks = new List<Blocks>();
    public int memoryLimit = 10;

    public void MoveCatTowardsFinishLine()
    {
        Debug.Log("Calisti");
        Blocks targetBlock = FindClosestFinishLine();

        if (targetBlock == null)
        {
            Debug.LogWarning("No finish line block found.");
            return;
        }

        Blocks bestNextBlock = null;
        float shortestDistance = Mathf.Infinity;

        List<Blocks> validAdjacentBlocks = new List<Blocks>();

        foreach (Blocks adjacentBlock in CurrentBlock.adjacentBlocks)
        {
            if (adjacentBlock.canCame)
            {
                validAdjacentBlocks.Add(adjacentBlock);
                float distanceToTarget = Vector3.Distance(adjacentBlock.transform.position, targetBlock.transform.position);

                if (distanceToTarget < shortestDistance && !lastBlocks.Contains(adjacentBlock))
                {
                    shortestDistance = distanceToTarget;
                    bestNextBlock = adjacentBlock;
                }
            }
        }

        if (bestNextBlock == null && validAdjacentBlocks.Count > 0)
        {
            List<Blocks> filteredBlocks = validAdjacentBlocks.FindAll(block => !lastBlocks.Contains(block));

            if (filteredBlocks.Count > 0)
            {
                bestNextBlock = filteredBlocks[Random.Range(0, filteredBlocks.Count)];
            }
            else
            {
                bestNextBlock = validAdjacentBlocks[Random.Range(0, validAdjacentBlocks.Count)];
            }
        }


        if (bestNextBlock != null)
        {
            lastBlocks.Add(bestNextBlock);


            if (lastBlocks.Count > memoryLimit)
            {
                lastBlocks.RemoveAt(0);
            }


            MoveCatToBlock(bestNextBlock);
        }
        else
        {
            Debug.LogWarning("No valid adjacent block found for the cat to move.");
        }
    }

    private void MoveCatToBlock(Blocks targetBlock)
    {
        Vector3 direction = targetBlock.transform.position - character.transform.position;

        if (Mathf.Abs(direction.y) > Mathf.Abs(direction.x))
        {
            if (direction.y > 0)
            {
                catAnim.SetTrigger("UpAnim");
            }
            else
            {
                catAnim.SetTrigger("DownAnim");
            }
        }
        else
        {
            if (direction.x > 0)
            {
                catAnim.SetTrigger("RightAnim");
            }
            else
            {
                catAnim.SetTrigger("WalkAnim");
            }
        }


        LeanTween.move(character, targetBlock.transform.position, 0.3f)
            .setEase(LeanTweenType.easeInOutQuad)
            .setOnComplete(() =>
            {
                CurrentBlock = targetBlock;
                WinControl();
            });
    }
}
```