using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BallManager : MonoBehaviour

{

    public static BallManager Instance;

    [Header("Ball Configuration")]

    [SerializeField] public GameObject ballObject;

    [SerializeField] public float ballMoveSpeed = 5f;

    [SerializeField] public float shootSpeed = 5f;

    [SerializeField] public float passSpeed = 5f;

    private bool ballIsMoving = false;

    private Vector3 targetPosition;

    private PlayerUnit targetHolder = null;

    public PlayerUnit currentHolder = null;

    private PlayerUnit lastHolder = null;

    private PlayerUnit previousHolder = null;

    public Vector2Int ballGridPosition;

    private Vector2Int previousGridPosition;

    private Vector3 lastHolderPosition;

    private Vector3 lastBallPosition;

    #region Properties

    public PlayerUnit CurrentHolder => currentHolder;

    public PlayerUnit PreviousHolder => previousHolder;

    public PlayerUnit LastHolder => lastHolder;

    public Vector2Int BallGridPosition => ballGridPosition;

    public Vector2Int PreviousGridPosition => previousGridPosition;

    public bool BallIsMoving => ballIsMoving;

    public GameObject BallObject => ballObject;

    public float BallMoveSpeed => ballMoveSpeed;

    #endregion

    #region Unity Methods

    private void Awake()

    {

        if (Instance != null && Instance != this)

        {

            Destroy(gameObject);

            return;

        }

        Instance = this;

    }

    private void Update()

{

    if (!ballIsMoving && currentHolder != null && ballObject != null && GridManager.Instance != null)

    {

        Vector3 desiredPos = GridManager.Instance.GetWorldPosition(currentHolder.gridPosition);

        ballObject.transform.position = Vector3.Lerp(ballObject.transform.position, desiredPos, Time.deltaTime \* ballMoveSpeed);

        if (Vector3.Distance(ballObject.transform.position, desiredPos) < 0.01f)

        {

            ballObject.transform.position = desiredPos;

            SetHolder(currentHolder);

        }

    }

}

    #endregion

    #region Ball Reset Methods

    public void ResetBall()

    {

        if (ballObject != null)

        {

            Vector2Int resetPos = new Vector2Int(6, 10);

            ballObject.transform.position = GridManager.Instance.GetWorldPosition(resetPos);

        }

        currentHolder = null;

        ballGridPosition = Vector2Int.zero;

        ballIsMoving = false;

        Debug.Log("Ball has been reset.");

    }

    public void ResetAfterGoal()

    {

        Vector2Int centerPosition = new Vector2Int(10, 6);

        SetHolder(null);

        SetBallGridPosition(centerPosition);

        currentHolder = null;

        ballIsMoving = false;

        Debug.Log("Reset after goal: Ball placed at center.");

    }

    #endregion

    #region Ball Control Methods

    public void SetHolder(PlayerUnit newHolder)

    {

        if (currentHolder == newHolder) return;

        if (currentHolder != null)

        {

            var temp = currentHolder;

            currentHolder = null;

            temp.LosePossession();

        }

        currentHolder = newHolder;

        if (currentHolder != null)

        {

            ballGridPosition = currentHolder.gridPosition;

            currentHolder.GainPossession();

            Debug.Log($"{currentHolder.name} now holds the ball.");

        }

        else

        {

            Debug.Log("Ball is now free.");

        }

    }

    public void AssignBallToTeam(TeamType team)

    {

        if (ballObject == null)

        {

            Debug.LogError("Ball object is not assigned!");

            return;

        }

        Vector2Int startPosition = team == TeamType.Player ? new Vector2Int(6, 6) : new Vector2Int(14, 6);

        SetBallGridPosition(startPosition);

        DropBall(startPosition);

        Debug.Log($"Ball assigned to {team} team at position {startPosition}.");

    }

    public void DropBall(Vector2Int position)

    {

        if (ballIsMoving)

        {

            Debug.LogWarning("Cannot drop ball while moving.");

            return;

        }

        SetHolder(null);

        ballGridPosition = position;

        if (ballObject != null)

        {

            ballObject.transform.position = GridManager.Instance.GetWorldPosition(position);

        }

    }

    public void SetBallGridPosition(Vector2Int position)

    {

        ballGridPosition = position;

        if (ballObject != null)

        {

            ballObject.transform.position = GridManager.Instance.GetWorldPosition(position);

        }

    }

    public Vector2Int GetBallGridPosition() => ballGridPosition;

    public PlayerUnit GetHolder() => currentHolder;

    public bool IsBallMoving() => ballIsMoving;

    #endregion

    #region Ball Passing & Animation

    public void PassBallTo(PlayerUnit target)

    {

        if (target == null)

        {

            Debug.LogError("PassBallTo: target is null!");

            return;

        }

        if (ballIsMoving)

        {

            Debug.LogWarning("Ball is already moving.");

            return;

        }

        Debug.Log($"Passing ball to {target.name}");

        targetHolder = target;

        targetPosition = GridManager.Instance.GetWorldPosition(target.gridPosition);

        StartCoroutine(MoveBallCoroutine());

    }

    private IEnumerator MoveBallCoroutine()

    {

        ballIsMoving = true;

        while (Vector3.Distance(ballObject.transform.position, targetPosition) > 0.01f)

        {

            ballObject.transform.position = Vector3.MoveTowards(ballObject.transform.position, targetPosition, ballMoveSpeed \* Time.deltaTime);

            yield return null;

        }

        SetHolder(targetHolder);

        targetHolder = null;

        ballIsMoving = false;

        Debug.Log($"Ball passed to {currentHolder.name}.");

    }

    public void SimulateShot(PlayerUnit shooter, Vector2Int goalTile)

    {

        TeamType defendingTeam = (TeamType)shooter.GetOpposingTeamType();

        PlayerUnit goalkeeper = TeamManager.Instance.GetGoalkeeper((TeamManager.TeamType)defendingTeam);

        if (goalkeeper == null)

        {

            Debug.LogWarning("No goalkeeper assigned for team " + defendingTeam + ". Automatically scoring.");

            ScoreManager.Instance.AddGoal(shooter.teamType);

            StartCoroutine(AnimateGoal(goalTile, shooter.teamType, true));

            return;

        }

        MathChallenge.Instance.AskQuestion("goal\_save", goalkeeper, success =>

        {

            StartCoroutine(AnimateGoal(goalTile, shooter.teamType, !success)); // play animation either way

            if (success)

            {

                Debug.Log($"{goalkeeper.name} saved the shot!");

            }

            else

            {

                ScoreManager.Instance.AddGoal(shooter.teamType);

                Debug.Log($"{shooter.name} scored a GOAL!");

            }

        });

    }

    private IEnumerator AnimateGoal(Vector2Int goalTile, TeamType scoringTeam, bool wasGoal)

    {

        Vector3 goalWorldPos = GridManager.Instance.GetWorldPosition(goalTile);

        Transform ballTransform = ballObject.transform;

        Vector3 startPos = ballTransform.position;

        float distance = Vector3.Distance(startPos, goalWorldPos);

        float elapsed = 0f;

        float duration = distance / shootSpeed;

        while (elapsed < duration)

        {

            elapsed += Time.deltaTime;

            float t = elapsed / duration;

            ballTransform.position = Vector3.Lerp(startPos, goalWorldPos, t);

            yield return null;

        }

        ballTransform.position = goalWorldPos;

        if (wasGoal)

        {

            Debug.Log("You scored!");

            yield return new WaitForSeconds(1f);

            ResetAfterGoal();

        }

        else

        {

            Debug.Log("Shot was saved. Ball is deflected.");

            yield return new WaitForSeconds(0.5f);

            DeflectBall(goalTile);

        }

    }

    private void DeflectBall(Vector2Int fromTile)

    {

        List<Vector2Int> nearbyTiles = new List<Vector2Int>();

        for (int x = -1; x <= 1; x++)

        {

            for (int y = -1; y <= 1; y++)

            {

                if (x == 0 && y == 0) continue;

                Vector2Int checkTile = fromTile + new Vector2Int(x, y);

                if (GridManager.Instance.IsValidPosition(checkTile) && !GridManager.Instance.IsOccupied(checkTile))

                {

                    nearbyTiles.Add(checkTile);

                }

            }

        }

        if (nearbyTiles.Count > 0)

        {

            Vector2Int deflectTile = nearbyTiles[Random.Range(0, nearbyTiles.Count)];

            SetHolder(null);

            SetBallGridPosition(deflectTile);

            Debug.Log($"Ball deflected to {deflectTile}");

        }

        else

        {

            Debug.Log("No valid nearby tiles to deflect ball.");

            SetHolder(null);

            SetBallGridPosition(fromTile);

        }

    }

    #endregion

    #region Tracking Last Holder & Position

    public void SetLastHolder(PlayerUnit holder)

    {

        if (holder != null)

        {

            lastHolder = holder;

            lastHolderPosition = holder.transform.position;

            previousHolder = currentHolder;

            previousGridPosition = ballGridPosition;

        }

        else

        {

            lastHolder = null;

            lastHolderPosition = Vector3.zero;

        }

    }

    public void ResetLastHolder()

    {

        lastHolder = null;

        lastHolderPosition = Vector3.zero;

        previousHolder = null;

        previousGridPosition = Vector2Int.zero;

    }

    public void SetLastBallPosition(Vector3 position)

    {

        lastBallPosition = position;

    }

    public Vector3 GetLastHolderPosition() => lastHolderPosition;

    public Vector3 GetLastBallPosition() => lastBallPosition;

    #endregion

}

using UnityEngine;

public class GameStarter : MonoBehaviour

{

    private void Start()

    {

        if (TeamManager.Instance == null || UnitManager.Instance == null)

        {

            Debug.LogError("GameStarter: Missing manager reference!");

            return;

        }

        var playerUnits = TeamManager.Instance.GetTeamUnits(TeamManager.TeamType.Player);

        var cpuUnits = TeamManager.Instance.GetTeamUnits(TeamManager.TeamType.CPU);

        if (playerUnits.Count == 0 || cpuUnits.Count == 0)

        {

            Debug.LogError("GameStarter: Units not initialized properly!");

            return;

        }

        TurnManager.Instance.Initialize(playerUnits, cpuUnits);

        Debug.Log("GameStarter: Game setup complete.");

    }

}

using UnityEngine;

using UnityEngine.UI;

using System.Collections.Generic;

public class GridManager : MonoBehaviour

{

    public static GridManager Instance;

    public GameObject tilePrefab;

    public int width = 21;

    public int height = 13;

    public float tileSize = 1.0f;

    public float tileOffset = 0.0f;

    public float tileHeight = 0.0f;

    public Color tileHighlightColor = new Color(0f, 1f, 0f, 0.4f);

    public Color tileDefaultColor = new Color(1f, 1f, 1f, 1f);

    private Dictionary<Vector2Int, PlayerUnit> tileOccupancy = new Dictionary<Vector2Int, PlayerUnit>();

    public void SetOccupant(Vector2Int position, PlayerUnit unit)

{

    if (unit != null)

        tileOccupancy[position] = unit;

    else

        tileOccupancy.Remove(position);

}

    public PlayerUnit GetOccupant(Vector2Int position)

    {

        tileOccupancy.TryGetValue(position, out PlayerUnit occupant);

        return occupant;

    }

public PlayerUnit GetUnitAtPosition(Vector2Int pos)

{

    return GetOccupant(pos);

}

public bool IsOccupied(Vector2Int position)

{

    return tileOccupancy.ContainsKey(position);

}

    private Dictionary<Vector2Int, Tile> tiles = new Dictionary<Vector2Int, Tile>();

    public static readonly Vector2Int PlayerGoalPosition = new Vector2Int(0, 6);

    public static readonly Vector2Int CPUGoalPosition = new Vector2Int(20, 6);

    public static readonly HashSet<Vector2Int> GoalLineTiles = new HashSet<Vector2Int>

    {

        new Vector2Int(0, 5), new Vector2Int(0, 6), new Vector2Int(0, 7),

        new Vector2Int(20, 5), new Vector2Int(20, 6), new Vector2Int(20, 7)

    };

    private void Awake()

    {

        if (Instance == null)

            Instance = this;

        else

            Destroy(gameObject);

    }

    private void Start()

    {

        GenerateGrid();

    }

    void GenerateGrid()

    {

        for (int x = 0; x < width; x++)

        {

            for (int z = 0; z < height; z++)

            {

                Vector2Int gridPos = new Vector2Int(x, z);

                Vector3 position = GetWorldPosition(gridPos);

                GameObject tileObj = Instantiate(tilePrefab, position, Quaternion.identity, transform);

                tileObj.name = $"Tile\_{x}\_{z}";

                Tile tile = tileObj.GetComponent<Tile>();

                tile.tilePosition = gridPos;

                tiles.Add(gridPos, tile);

            }

        }

    }

    public Vector3 GetWorldPosition(Vector2Int gridPosition)

    {

        return new Vector3(gridPosition.x \* tileSize, tileHeight, gridPosition.y \* tileSize);

    }

    public Tile GetTile(Vector2Int position, bool createIfMissing = false)

    {

        if (tiles.TryGetValue(position, out Tile tile))

        {

            return tile;

        }

        else if (createIfMissing)

        {

            Vector3 worldPosition = GetWorldPosition(position); // ✅ Centered

            GameObject tileObj = Instantiate(tilePrefab, worldPosition, Quaternion.identity, transform);

            tileObj.name = $"Tile\_{position.x}\_{position.y}";

            tile = tileObj.GetComponent<Tile>();

            tile.tilePosition = position;

            tiles[position] = tile;

            return tile;

        }

        return null;

    }

    public List<Vector2Int> GetLineBetweenPoints(Vector2Int start, Vector2Int end)

{

    List<Vector2Int> points = new List<Vector2Int>();

    int dx = Mathf.Abs(end.x - start.x);

    int dy = Mathf.Abs(end.y - start.y);

    int sx = start.x < end.x ? 1 : -1;

    int sy = start.y < end.y ? 1 : -1;

    int err = dx - dy;

    int x = start.x;

    int y = start.y;

    while (x != end.x || y != end.y)

    {

        points.Add(new Vector2Int(x, y));

        int e2 = 2 \* err;

        if (e2 > -dy)

        {

            err -= dy;

            x += sx;

        }

        if (e2 < dx)

        {

            err += dx;

            y += sy;

        }

    }

    return points;

}

    public List<Tile> GetGoalLineTiles(TeamType teamType)

    {

        List<Tile> goalLineTiles = new List<Tile>();

        int goalX = teamType == TeamType.Player ? 0 : width - 1;

        for (int z = 5; z <= 7; z++)

        {

            Vector2Int pos = new Vector2Int(goalX, z);

            if (tiles.TryGetValue(pos, out Tile tile))

            {

                goalLineTiles.Add(tile);

            }

        }

        return goalLineTiles;

    }

    // Removed duplicate GetUnitAtPosition(Vector2Int pos) to resolve ambiguity and compile error.

    public Tile GetGoalTile(TeamType teamType)

    {

        Vector2Int goalPosition = teamType == TeamType.Player ? PlayerGoalPosition : CPUGoalPosition;

        return GetTile(goalPosition);

    }

    public Tile GetTile(Vector2Int position)

    {

        tiles.TryGetValue(position, out Tile tile);

        return tile;

    }

    public Tile GetTileAtPosition(Vector2Int pos)

    {

        return GetTile(pos);

    }

    public Tile GetTileAtWorldPosition(Vector3 worldPosition, bool snapToGrid = true)

    {

        Vector2Int gridPosition = snapToGrid

            ? new Vector2Int(Mathf.RoundToInt(worldPosition.x), Mathf.RoundToInt(worldPosition.z))

            : new Vector2Int(Mathf.FloorToInt(worldPosition.x), Mathf.FloorToInt(worldPosition.z));

        return GetTile(gridPosition);

    }

    public Vector2Int GetGridPosition(Vector3 worldPosition, bool snapToGrid = true)

    {

        return snapToGrid

            ? new Vector2Int(Mathf.RoundToInt(worldPosition.x), Mathf.RoundToInt(worldPosition.z))

            : new Vector2Int(Mathf.FloorToInt(worldPosition.x), Mathf.FloorToInt(worldPosition.z));

    }

    public bool IsValidPosition(Vector2Int position) => tiles.ContainsKey(position);

    public bool IsValidWorldPosition(Vector3 worldPosition, bool snapToGrid = true)

    {

        Vector2Int gridPosition = snapToGrid

            ? new Vector2Int(Mathf.RoundToInt(worldPosition.x), Mathf.RoundToInt(worldPosition.z))

            : new Vector2Int(Mathf.FloorToInt(worldPosition.x), Mathf.FloorToInt(worldPosition.z));

        return IsValidPosition(gridPosition);

    }

    public Vector2Int GetShotTarget(TeamType teamType)

    {

        return teamType == TeamType.Player ? CPUGoalPosition : PlayerGoalPosition;

    }

    public void HighlightTile(Vector2Int position, bool highlight)

    {

        if (tiles.TryGetValue(position, out Tile tile))

        {

            tile.Highlight(highlight);

        }

        else

        {

            Debug.LogWarning($"Tile at position {position} not found for highlighting!");

        }

    }

    public void HighlightTiles(List<Vector2Int> positions, bool highlight)

    {

        foreach (Vector2Int pos in positions)

        {

            HighlightTile(pos, highlight);

        }

    }

    public void HighlightTilesInRange(Vector2Int center, int range, bool highlight)

    {

        for (int x = -range; x <= range; x++)

        {

            for (int z = -range; z <= range; z++)

            {

                Vector2Int pos = center + new Vector2Int(x, z);

                if (IsValidPosition(pos))

                {

                    HighlightTile(pos, highlight);

                }

            }

        }

    }

    public List<Tile> GetAllTiles()

    {

        return new List<Tile>(tiles.Values);

    }

    public void ClearHighlights()

    {

        foreach (Tile tile in tiles.Values)

        {

            tile.Highlight(false);

        }

    }

    public void ClearAllHighlights()

    {

        ClearHighlights();

    }

    public List<PlayerUnit> GetAdjacentUnits(Vector2Int position)

    {

        List<PlayerUnit> adjacentUnits = new List<PlayerUnit>();

        Vector2Int[] directions = { Vector2Int.up, Vector2Int.down, Vector2Int.left, Vector2Int.right };

        foreach (Vector2Int dir in directions)

        {

            Vector2Int adjacentPos = position + dir;

            PlayerUnit unit = GetUnitAtPosition(adjacentPos);

            if (unit != null)

            {

                adjacentUnits.Add(unit);

            }

        }

        return adjacentUnits;

    }

}

using UnityEngine;

public class InputManager : MonoBehaviour

{

    void Update()

    {

        if (Input.GetMouseButtonDown(0)) // mouse or touchscreen tap

        {

            if (Camera.main == null)

            {

                Debug.LogWarning("Camera.main is null!");

                return;

            }

            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out RaycastHit hit))

            {

                Tile tile = hit.collider.GetComponent<Tile>();

                if (tile != null)

                {

                    if (TurnManager.Instance == null)

                    {

                        Debug.LogWarning("TurnManager.Instance is null!");

                        return;

                    }

                    PlayerUnit currentUnit = TurnManager.Instance.GetCurrentUnit();

                    if (currentUnit != null && currentUnit.teamType == TeamType.Player)

                    {

                        currentUnit.TryMove(tile.tilePosition);

                    }

                }

            }

        }

    }

}

using UnityEngine;

public enum TeamType

{

    Player,

    CPU

}

public class MatchInitializer : MonoBehaviour

{

    public GameObject ballPrefab;

    public Transform startPosition;

    public TeamManager teamManager; // reference in inspector or via code

    public static MatchInitializer Instance;

    private void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

    }

    void Start()

    {

        GameObject ball = Instantiate(ballPrefab, startPosition.position, Quaternion.identity);

        BallManager.Instance.ballObject = ball;  // assign GameObject, not Transform

        // You need to specify which team gets initial possession; replace TeamType.Home with your desired team

        StartCoroutine(SetInitialPossession(TeamType.Player));

    }

    public System.Collections.IEnumerator SetInitialPossession(TeamType team)

    {

        yield return new WaitForSeconds(0.5f); // give enough time for spawning

        PlayerUnit lastPlayer = teamManager.GetLastPlayer(); // <-- your own method to get the player you want

        if (lastPlayer != null)

        {

            BallManager.Instance.SetHolder(lastPlayer);

        }

        else

        {

            Debug.LogWarning("No player found for initial possession.");

        }

    }

    public void InitializeMatch()

    {

        // Additional initialization logic if needed

        Debug.Log("Match initialized with ball and teams.");

    }

    private void OnValidate()

    {

        if (ballPrefab == null)

        {

            Debug.LogError("Ball prefab is not assigned in MatchInitializer.");

        }

        if (startPosition == null)

        {

            Debug.LogError("Start position is not assigned in MatchInitializer.");

        }

        if (teamManager == null)

        {

            Debug.LogError("TeamManager reference is not assigned in MatchInitializer.");

        }

    }

}

using System;

using UnityEngine;

public class MathChallenge : MonoBehaviour

{

    public static MathChallenge Instance;

    public MathChallengeUI ui;

    public static Action<bool> OverrideCallback = null;

    void Awake()

    {

        if (Instance == null)

        {

            Instance = this;

            DontDestroyOnLoad(gameObject);

        }

        else

        {

            Destroy(gameObject);

        }

    }

    private void Start()

    {

        if (ui == null)

        {

            ui = FindFirstObjectByType<MathChallengeUI>();

            if (ui == null)

                Debug.LogError("[MathChallenge] MathChallengeUI not found in the scene!");

        }

    }

    public void AskQuestion(string actionType, PlayerUnit player, Action<bool> onComplete, PlayerUnit target = null)

    {

        if (ui == null)

        {

            Debug.LogError("[MathChallenge] Cannot ask question: UI is null.");

            onComplete?.Invoke(false);

            return;

        }

        int difficulty = GetDifficulty(actionType, player, target);

        int a = UnityEngine.Random.Range(1, difficulty + 1);

        int b = UnityEngine.Random.Range(1, difficulty + 1);

        int correctAnswer = a + b;

        string question = $"What is {a} + {b}?";

        Debug.Log($"[MathChallenge] Asking '{actionType}' question to {player.name}: {question} (Difficulty: {difficulty})");

        if (player.isCPU)

        {

            float delay = Mathf.Clamp(1.5f - (player.Speed \* 0.05f), 0.2f, 1.5f);

            StartCoroutine(SimulateCPUAnswer(player, actionType, target, correctAnswer, delay, onComplete));

        }

        else

        {

            ui.ShowQuestion(question, correctAnswer, success =>

            {

                onComplete?.Invoke(success);

            });

        }

    }

    private int GetDifficulty(string action, PlayerUnit player, PlayerUnit target = null)

    {

        int baseDifficulty = 10;

        int leagueModifier = 0;

        switch (action.ToLower())

        {

            case "shoot":

                baseDifficulty += Mathf.RoundToInt((10 - player.Accuracy) \* 0.7f);

                if (target != null)

                    baseDifficulty += Mathf.RoundToInt(target.Defense \* 0.5f);

                break;

            case "pass":

                float passBonus = (player.Accuracy \* 0.5f) + (player.Speed \* 0.3f);

                baseDifficulty -= Mathf.RoundToInt(passBonus);

                break;

            case "advance":

                float advanceBonus = (player.Speed \* 0.4f) + (player.Attack \* 0.4f);

                baseDifficulty -= Mathf.RoundToInt(advanceBonus);

                if (target != null)

                    baseDifficulty += Mathf.RoundToInt(target.Defense \* 0.6f);

                break;

            case "tackle":

                if (target != null)

                    baseDifficulty += Mathf.RoundToInt((target.Defense - player.Attack) \* 0.6f);

                break;

            case "intercept":

                if (target != null)

                {

                    // Target = the passer. Interceptor = player

                    baseDifficulty += Mathf.RoundToInt((target.Accuracy - player.Defense) \* 0.7f);

                }

                else

                {

                    baseDifficulty += Mathf.RoundToInt((10 - player.Defense) \* 0.8f);

                }

                break;

            default:

                Debug.LogWarning($"[MathChallenge] Unknown action '{action}', using default difficulty.");

                break;

        }

        baseDifficulty += leagueModifier;

        return Mathf.Clamp(baseDifficulty, 3, 25);

    }

    private System.Collections.IEnumerator SimulateCPUAnswer(PlayerUnit cpu, string action, PlayerUnit target, int correctAnswer, float delay, Action<bool> callback)

    {

        yield return new WaitForSeconds(delay);

        float successChance = CalculateCPUSuccessChance(cpu, action, target);

        float roll = UnityEngine.Random.Range(0f, 1f);

        bool isCorrect = roll <= successChance;

        int chosenAnswer = isCorrect ? correctAnswer : correctAnswer + UnityEngine.Random.Range(-2, 3);

        Debug.Log($"[MathChallenge] CPU '{cpu.name}' answered: {chosenAnswer} (Correct: {correctAnswer})");

        Debug.Log($"[MathChallenge] Action: {action} → Chance: {successChance \* 100:F1}% | Rolled: {roll:F2} → {(isCorrect ? "✅ Success" : "❌ Fail")}");

        cpu.UseActionPoints(0); // Optional

        callback?.Invoke(isCorrect);

    }

    private float CalculateCPUSuccessChance(PlayerUnit cpu, string action, PlayerUnit target)

    {

        float chance = 0.5f;

        switch (action.ToLower())

        {

            case "shoot":

                chance = Mathf.Clamp01((cpu.Accuracy / 10f) - ((target != null ? target.Defense : 0) \* 0.05f));

                break;

            case "pass":

                chance = Mathf.Clamp01((cpu.Accuracy \* 0.5f + cpu.Speed \* 0.3f) / 10f);

                break;

            case "advance":

                chance = Mathf.Clamp01(((cpu.Speed \* 0.4f + cpu.Attack \* 0.4f) / 10f) - ((target?.Defense ?? 0) \* 0.05f));

                break;

            case "tackle":

                chance = target != null

                    ? Mathf.Clamp01(((cpu.Attack - target.Defense) \* 0.1f) + 0.5f)

                    : 0.5f;

                break;

            case "intercept":

                float defenseFactor = cpu.Defense \* 0.5f;

                float targetAccuracyPenalty = (target != null ? target.Accuracy \* 0.4f : 0f);

                chance = Mathf.Clamp01((defenseFactor - targetAccuracyPenalty) / 10f + 0.5f);

                break;

            default:

                chance = Mathf.Clamp01(cpu.Accuracy / 10f);

                break;

        }

        return Mathf.Clamp(chance, 0.1f, 0.95f);

    }

}

using System;

using TMPro;

using UnityEngine;

using UnityEngine.UI;

public class MathChallengeUI : MonoBehaviour

{

    public static MathChallengeUI Instance;

    [Header("UI References")]

    public GameObject panel;

    public TextMeshProUGUI questionText;

    public TextMeshProUGUI timerText; // 🔸 Add this in your UI and assign it in the inspector

    public Button[] answerButtons;

    [Header("Timer Settings")]

    public float timeLimit = 10f;

    private int correctAnswer;

    private Action<bool> callback;

    private float remainingTime;

    private bool isTiming;

    void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

        if (panel != null)

            panel.SetActive(false);

    }

    void Update()

    {

    if (isTiming)

    {

        remainingTime -= Time.deltaTime;

        timerText.text = $"Time: {Mathf.CeilToInt(remainingTime)}";

        if (remainingTime <= 0)

        {

            isTiming = false;

            timerText.gameObject.SetActive(false); // Hide when time ends

            AutoFail();

        }

        }

    }

    /// <summary>

    /// Public method to show a math question with automatic answer button setup.

    /// Called by MathChallenge.

    /// </summary>

    public void ShowQuestion(string question, int result, Action<bool> onAnswered)

    {

        correctAnswer = result;

        callback = onAnswered;

        panel.SetActive(true);

        questionText.text = question;

        SetupAnswerButtons(correctAnswer);

        remainingTime = timeLimit;

        isTiming = true;

    }

    private void SetupAnswerButtons(int correct)

    {

        if (answerButtons == null || answerButtons.Length == 0)

        {

            Debug.LogError("[MathChallengeUI] Answer buttons not assigned!");

            return;

        }

        int[] answers = new int[answerButtons.Length];

        int correctIndex = UnityEngine.Random.Range(0, answers.Length);

        answers[correctIndex] = correct;

        for (int i = 0; i < answers.Length; i++)

        {

            if (i == correctIndex) continue;

            int wrong;

            do

            {

                wrong = correct + UnityEngine.Random.Range(-10, 11);

            } while (wrong == correct || Array.Exists(answers, a => a == wrong));

            answers[i] = wrong;

        }

        for (int i = 0; i < answerButtons.Length; i++)

        {

            int answer = answers[i];

            TMP\_Text btnText = answerButtons[i].GetComponentInChildren<TMP\_Text>();

            if (btnText != null) btnText.text = answer.ToString();

            answerButtons[i].onClick.RemoveAllListeners();

            answerButtons[i].onClick.AddListener(() => OnAnswerClicked(answer == correct));

        }

    }

    private void OnAnswerClicked(bool isCorrect)

    {

        FinishChallenge(isCorrect);

    }

    private void AutoFail()

    {

        Debug.Log("[MathChallengeUI] Time's up! Answer marked incorrect.");

        FinishChallenge(false);

    }

    private void FinishChallenge(bool success)

{

    isTiming = false;

    panel.SetActive(false);

    timerText.gameObject.SetActive(false); // Hide timer when question ends

    callback?.Invoke(success);

}

}

using UnityEngine;

using System.Collections.Generic;

public class PassSelector : MonoBehaviour

{

    public static PassSelector Instance;

    public enum TeamType

    {

        Player,

        CPU

    }

    public TeamType currentTeam = TeamType.Player;

    public TeamType CurrentTeam

    {

        get { return currentTeam; }

        set

        {

            currentTeam = value;

            Debug.Log($"Current team set to: {currentTeam}");

        }

    }

    public enum PositionType

    {

        Goalkeeper,

        Field

    }

    public PositionType currentPosition = PositionType.Field;

    private List<Tile> highlightedPassTiles = new List<Tile>();

    public bool isSelectingPassTarget = false;

    private void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

    }

    public void StartPassSelection()

    {

        PlayerUnit currentUnit = TurnManager.Instance.GetCurrentUnit();

        if (currentUnit == null || (TeamType)currentUnit.teamType != TeamType.Player)

        {

            Debug.LogWarning("PassSelector: No player unit or not player's turn.");

            return;

        }

        isSelectingPassTarget = true;

        HighlightPassTargets(currentUnit);

        Debug.Log("Select a teammate to pass to.");

    }

    // Highlights all valid pass targets (teammates) for the current unit

    void HighlightPassTargets(PlayerUnit currentUnit)

    {

        ClearHighlights();

        // Get teammates excluding self

        List<PlayerUnit> teammates = TeamManager.Instance.GetTeamMembers((TeamManager.TeamType)currentUnit.teamType);

        teammates.Remove(currentUnit);

        foreach (PlayerUnit teammate in teammates)

        {

            Tile tile = GridManager.Instance.GetTileAtPosition(teammate.gridPosition);

            if (tile != null)

            {

                tile.Highlight(true, Color.green);

                highlightedPassTiles.Add(tile);

            }

        }

    }

    void ClearHighlights()

    {

        foreach (Tile tile in highlightedPassTiles)

        {

            tile.Highlight(false);

        }

        highlightedPassTiles.Clear();

    }

    private void Update()

    {

        if (!isSelectingPassTarget) return;

        if (Input.GetMouseButtonDown(0))

        {

            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out RaycastHit hit))

            {

                Tile clickedTile = hit.collider.GetComponent<Tile>();

                if (clickedTile != null && highlightedPassTiles.Contains(clickedTile))

                {

                    OnPassTargetSelected(clickedTile);

                }

                else

                {

                    Debug.Log("Clicked tile not a valid pass target.");

                }

            }

        }

        if (Input.GetMouseButtonDown(1) || Input.GetKeyDown(KeyCode.Escape))

        {

            CancelPassSelection();

        }

    }

    void OnPassTargetSelected(Tile tile)

    {

        PlayerUnit currentUnit = TurnManager.Instance.GetCurrentUnit();

        if (currentUnit == null) return;

        selectedTargetPlayer = TeamManager.Instance.GetPlayerAtPosition(tile.tilePosition);

        if (selectedTargetPlayer == null)

        {

            Debug.LogError("PassSelector: No player unit at selected tile.");

            CancelPassSelection();

            return;

        }

        isSelectingPassTarget = false;

        ClearHighlights();

        Debug.Log($"Selected pass target: {selectedTargetPlayer.name}");

        // ✅ New: Start the challenge

        StartCoroutine(StartMathChallengeRoutine());

    }

    private PlayerUnit selectedTargetPlayer;

    private System.Collections.IEnumerator StartMathChallengeRoutine()

    {

        if (MathChallenge.Instance == null)

        {

            Debug.LogError("MathChallenge instance is missing!");

            yield break;

        }

        // Start the math challenge for the pass

        MathChallenge.Instance.AskQuestion("pass", selectedTargetPlayer, (bool success) =>

        {

            if (success)

            {

                // Pass succeeded, execute pass logic

                TurnManager.Instance.PassBallTo(selectedTargetPlayer);

                Debug.Log($"Pass to {selectedTargetPlayer.name} succeeded.");

            }

            else

            {

                // Pass failed, handle failure logic

                Debug.Log("Pass failed due to incorrect answer.");

            }

        });

    }

    void CancelPassSelection()

    {

        ClearHighlights();

        isSelectingPassTarget = false;

        Debug.Log("Pass selection canceled.");

    }

    public void EndPassSelection()

    {

        isSelectingPassTarget = false;

        ClearHighlights();

        Debug.Log("Pass selection ended.");

    }

    public void ResetPassSelector()

    {

        isSelectingPassTarget = false;

        ClearHighlights();

        currentTeam = TeamType.Player;

        currentPosition = PositionType.Field;

        Debug.Log("PassSelector reset.");

    }

}

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

public enum PositionType { Goalkeeper, Field }

public class PlayerUnit : MonoBehaviour

{

    public TeamType teamType;

    public PositionType positionType;

    public Vector2Int gridPosition;

    public Vector2Int startingGridPosition;

    public int Speed = 5;

    public int Accuracy = 5;

    public int Defense = 5;

    public int Attack = 5;

    private int baseDefense;

    public int maxActionPoints = 3;

    private int currentActionPoints;

    public int CurrentActionPoints

    {

        get => currentActionPoints;

        set

        {

            currentActionPoints = Mathf.Clamp(value, 0, maxActionPoints);

            Debug.Log($"{gameObject.name} now has {currentActionPoints} AP.");

            if (currentActionPoints == 0 && isPlayerControlled)

                TurnManager.Instance?.EndUnitTurn();

        }

    }

    public Vector2Int GetGridPosition() => gridPosition;

    public bool isPlayerControlled = false;

    private bool isDefending = false;

    public bool IsDefending() => isDefending;

    public Color moveHighlightColor = Color.green;

    public Color passHighlightColor = Color.yellow;

    public Color shootHighlightColor = Color.red;

    public Color tackleHighlightColor = Color.blue;

    public Vector2Int CurrentTilePosition { get; set; }

    public bool HasMovedThisTurn { get; private set; } = false;

    private Tile currentTile;

    // Shortcuts

    public bool HasBall => BallManager.Instance && BallManager.Instance.currentHolder == this;

    public bool IsGoalkeeper => positionType == PositionType.Goalkeeper;

    public bool IsFieldPlayer => positionType == PositionType.Field;

    public bool IsPlayer => teamType == TeamType.Player;

    public bool isCPU => teamType == TeamType.CPU;

    public bool IsTurn => TurnManager.Instance?.GetCurrentUnit() == this;

    public bool IsValidPosition => GridManager.Instance?.IsValidPosition(gridPosition) ?? false;

    public float ModifyDefenseFactor() => Defense \* 0.6f;

    public void Initialize(TeamType team, PositionType position, Vector2Int startPos)

    {

        teamType = team;

        positionType = position;

        startingGridPosition = startPos;

        SetGridPosition(startPos);

        baseDefense = Defense;

        CurrentActionPoints = maxActionPoints;

        HasMovedThisTurn = false;

    }

    public void StartTurn()

    {

        CurrentActionPoints = maxActionPoints;

        HasMovedThisTurn = false;

        isDefending = false;

        Defense = baseDefense;

        Debug.Log($"{gameObject.name} ({teamType}) turn started with {CurrentActionPoints} AP.");

    }

    public void SetPlayerControlled(bool isControlled)

    {

        isPlayerControlled = isControlled;

        if (isControlled)

        {

            TurnManager.Instance?.SetCurrentTurn(teamType);

        }

    }

    public void SetGridPosition(Vector2Int newPosition)

    {

    gridPosition = newPosition;

    Vector3 worldPos = GridManager.Instance.GetWorldPosition(newPosition);

    worldPos.y += 0.5f; // Ensure it sits on top of the tile

    transform.position = worldPos;

        // Clear old tile

    GridManager.Instance.SetOccupant(gridPosition, null);

// Update internal grid position

    gridPosition = newPosition;

    transform.position = GridManager.Instance.GetWorldPosition(newPosition);

// Mark new tile

    GridManager.Instance.SetOccupant(newPosition, this);

    }

    public void SetGridPosition(int x, int y)

    {

        SetGridPosition(new Vector2Int(x, y));

    }

    public void ResetToStartPosition()

    {

        SetGridPosition(startingGridPosition);

        ResetTurnState();

    }

    public void ResetTurnState()

    {

        CurrentActionPoints = maxActionPoints;

        HasMovedThisTurn = false;

        isDefending = false;

    }

    public void Pass(PlayerUnit targetUnit)

{

    if (targetUnit == null || !HasBall || targetUnit.teamType != teamType || !HasActionPoints(1))

    {

        Debug.Log($"{name} failed pass check.");

        return;

    }

    Debug.Log($"{name} attempts to pass to {targetUnit.name}.");

    List<Vector2Int> path = GridManager.Instance.GetLineBetweenPoints(gridPosition, targetUnit.gridPosition);

    PlayerUnit interceptor = null;

    foreach (var point in path)

    {

        List<PlayerUnit> adjacentUnits = GridManager.Instance.GetAdjacentUnits(point);

        foreach (var unit in adjacentUnits)

        {

            if (unit.teamType != teamType && unit.IsDefending())

            {

                interceptor = unit;

                Debug.Log($"{unit.name} is intercepting the pass at {point}!");

                break;

            }

        }

        if (interceptor != null) break;

    }

    void ResolvePass()

    {

        BallManager.Instance?.SetHolder(targetUnit);

        UseActionPoints(1);

    }

    void ResolveInterception()

    {

        BallManager.Instance?.SetHolder(interceptor);

        UseActionPoints(1);

        Debug.Log($"{interceptor.name} intercepted the pass!");

    }

    MathChallenge mathChallenge = FindFirstObjectByType<MathChallenge>();

    if (interceptor != null && mathChallenge)

    {

        Debug.Log($"{interceptor.name} attempts interception!");

        mathChallenge.AskQuestion("intercept", interceptor, success =>

        {

            if (success)

                ResolveInterception();

            else

                ResolvePass();

        }, this);

    }

    else if (mathChallenge)

    {

        mathChallenge.AskQuestion("pass", this, success =>

        {

            if (success)

                ResolvePass();

            else

                BallManager.Instance?.DropBall(gridPosition);

            UseActionPoints(0);

        });

    }

    else

    {

        Debug.LogError("No MathChallenge instance found.");

    }

}

    public void Shoot(Vector2Int shotTarget)

    {

        if (!HasBall || !HasActionPoints(2) || !IsValidTurn()) return;

        if (!GridManager.Instance || !GridManager.Instance.IsValidPosition(shotTarget)) return;

        if (ManhattanDistance(gridPosition, shotTarget) > 10) return;

        MathChallenge mathChallenge = FindFirstObjectByType<MathChallenge>();

        if (mathChallenge)

        {

            mathChallenge.AskQuestion("shoot", this, success =>

            {

                if (success)

                {

                    if (GridManager.GoalLineTiles.Contains(shotTarget))

                        BallManager.Instance?.SimulateShot(this, shotTarget);

                    else

                        BallManager.Instance?.DropBall(shotTarget);

                }

                else BallManager.Instance?.DropBall(gridPosition);

                UseActionPoints(2);

            });

        }

        else Debug.LogError("No MathChallenge instance found.");

    }

    public void Advance()

    {

        if (!HasActionPoints(2) || HasMovedThisTurn || !IsValidTurn() || !IsFieldPlayer) return;

        PlayerUnit adjacentDefender = GridManager.Instance.GetAdjacentUnits(gridPosition)

            .FirstOrDefault(unit => unit != this && unit.teamType != teamType && unit.IsDefending());

        if (adjacentDefender == null) return;

        MathChallenge mathChallenge = FindFirstObjectByType<MathChallenge>();

        if (mathChallenge)

        {

            mathChallenge.AskQuestion("advance", this, success =>

            {

                if (success)

                {

                    Vector2Int direction = GetTeamMovementDirection();

                    Vector2Int currentPos = gridPosition;

                    for (int step = 1; step <= 3; step++) // fixed to 3-step advance

                    {

                        Vector2Int nextPos = currentPos + direction;

                        if (!GridManager.Instance.IsValidPosition(nextPos)) break;

                        Tile nextTile = GridManager.Instance.GetTileAtPosition(nextPos);

                        if (nextTile == null || nextTile.isOccupied) break;

                        currentPos = nextPos;

                    }

                    SetGridPosition(currentPos);

                    HasMovedThisTurn = true;

                    // Check if player picked up the ball after moving

                    if (!HasBall && BallManager.Instance?.currentHolder == null &&

                        gridPosition == BallManager.Instance.ballGridPosition)

                    {

                        BallManager.Instance?.SetHolder(this);

                    }

                    UseActionPoints(2);

                }

                else

                {

                    UseActionPoints(2); // still consume AP on failure

                }

            }, adjacentDefender);

        }

    }

    public void AttemptTackle(PlayerUnit targetUnit)

    {

        if (targetUnit == null || !targetUnit.HasBall || !IsAdjacent(targetUnit)) return;

        MathChallenge mathChallenge = FindFirstObjectByType<MathChallenge>();

        if (mathChallenge)

        {

            mathChallenge.AskQuestion("tackle", this, success =>

            {

                if (success)

                {

                    targetUnit.LosePossession();

                    GainPossession();

                }

                UseActionPoints(1);

            });

        }

        else Debug.LogError("No MathChallenge instance found.");

    }

    public void Defend()

    {

        if (!HasActionPoints(1)) return;

        isDefending = true;

        Defense += 3;

        UseActionPoints(1);

        Debug.Log($"{name} is now defending. Defense increased to {Defense}.");

    }

    public bool TryMove(Vector2Int targetPos)

    {

        if (!IsValidTurn()) return false;

        int distance = ManhattanDistance(gridPosition, targetPos);

        if (!HasActionPoints(1) || distance > 3 || !GridManager.Instance.IsValidPosition(targetPos)) return false;

        var adjacentEnemies = GridManager.Instance.GetAdjacentUnits(gridPosition)

            .Where(unit => unit.teamType != this.teamType && unit.IsDefending());

        if (adjacentEnemies.Any())

        {

            Debug.Log($"{name} is adjacent to a defending opponent and must use Advance.");

            return false;

        }

        if (IsGoalkeeper)

        {

            int gridWidth = 3;

            if (!IsWithinGoalZone(targetPos, gridWidth)) return false;

        }

        if (!GridManager.Instance.IsValidPosition(targetPos) || GridManager.Instance.IsOccupied(targetPos))

        {

    Debug.Log($"{name} cannot move to {targetPos}, tile is invalid or already occupied.");

    return false;

        }

        SetGridPosition(targetPos);

        if (BallManager.Instance != null &&

            BallManager.Instance.currentHolder == null &&

            BallManager.Instance.ballGridPosition == gridPosition)

        {

            BallManager.Instance.SetHolder(this);

        }

        UseActionPoints(1);

        HasMovedThisTurn = true;

        return true;

    }

    public void UseActionPoints(int amount) => CurrentActionPoints -= amount;

    public bool HasActionPoints(int amount) => CurrentActionPoints >= amount;

    public bool IsValidTurn() => TurnManager.Instance?.GetCurrentUnit() == this;

    public void GainPossession() => BallManager.Instance?.SetHolder(this);

    public void LosePossession() => BallManager.Instance?.DropBall(gridPosition);

    public bool IsAdjacent(PlayerUnit other) => ManhattanDistance(gridPosition, other.gridPosition) == 1;

    public int ManhattanDistance(Vector2Int a, Vector2Int b) => Mathf.Abs(a.x - b.x) + Mathf.Abs(a.y - b.y);

    public Vector2Int GetTeamMovementDirection()

    {

        return teamType == TeamType.Player ? Vector2Int.up :

            teamType == TeamType.CPU ? Vector2Int.down : Vector2Int.zero;

    }

    public TeamType GetOpposingTeamType()

    {

        return teamType == TeamType.Player ? TeamType.CPU : TeamType.Player;

    }

    public bool IsBlockingOpponentAt(Vector2Int pos)

    {

        PlayerUnit opponent = GridManager.Instance?.GetUnitAtPosition(pos);

        return opponent != null && opponent.teamType != this.teamType && opponent.IsDefending();

    }

    public bool IsWithinGoalZone(Vector2Int targetPos, int gridWidth)

    {

        int zoneWidth = 3;

        int teamSideStartX = (teamType == TeamType.Player) ? 0 : gridWidth - zoneWidth - 2;

        int teamSideEndX = (teamType == TeamType.CPU) ? zoneWidth : gridWidth - 2;

        return targetPos.x >= teamSideStartX && targetPos.x <= teamSideEndX;

    }

    public int ModifyDifficulty(int baseDifficulty)

    {

        return positionType == PositionType.Goalkeeper ? baseDifficulty + 3 : baseDifficulty;

    }

    public void SetStartingPosition(Vector2Int pos)

    {

        startingGridPosition = pos;

        SetGridPosition(pos);

    }

}

using UnityEngine;

using TMPro;

using Unity.VisualScripting;

public class ScoreManager : MonoBehaviour

{

    public static ScoreManager Instance;

    public int playerScore = 0;

    public int cpuScore = 0;

    public TextMeshProUGUI scoreText;

    private void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

    }

    System.Collections.IEnumerator ResetAfterGoal(TeamType scoringTeam)

    {

        yield return new WaitForSeconds(0.8f); // small delay

        BallManager.Instance.ResetBall();  // resets ball to center (or hidden temporarily)

        yield return new WaitForSeconds(0.8f); // optional extra wait

        TeamManager.Instance.ResetTeams(); // moves all units back

        // Assign ball to non-scoring team using SetHolder()

        TeamType nextPossession = (scoringTeam == TeamType.Player) ? TeamType.CPU : TeamType.Player;

        // TODO: Replace the line below with the correct parameters for your GetLastPlayer method

        PlayerUnit nextHolder = TeamManager.Instance.GetLastPlayer();

        if (nextHolder == null)

        {

            Debug.LogError("No player unit found for next possession!");

            yield break; // exit if no holder found

        }

        BallManager.Instance.SetHolder(null);

        // Set the next team to play

        TurnManager.Instance.SetCurrentTurn(nextPossession);

// Restart the turn sequence

        TurnManager.Instance.ResetTurnOrder();

    // TurnManager.Instance.SetCurrentTurn(nextPossession); // optional if your system uses this

    }

    public void AddGoal(TeamType scoringTeam)

    {

        if (scoringTeam == TeamType.Player)

            playerScore++;

        else

            cpuScore++;

        UpdateScoreUI();

        StartCoroutine(ResetAfterGoal(scoringTeam));

    }

    void UpdateScoreUI()

    {

        scoreText.text = $"Player: {playerScore}  CPU: {cpuScore}";

    }

    public void ResetAfterGoal()

    {

        BallManager.Instance.ResetBall();

    }

    //if player score or cpu score increase by 1, place all units on eh feild back to thier orginal tile lcoations

}

using System.Collections.Generic;

using UnityEngine;

using System.Linq;

public class TeamManager : MonoBehaviour

{

    public static TeamManager Instance;

    public enum TeamType { Player, CPU }

    public enum PositionType { Goalkeeper, Field }

    public TeamType currentTeam = TeamType.Player;

    public PositionType currentPosition = PositionType.Field;

    // Updated spawn positions

    private readonly Vector2Int playerGoalkeeperPosition = new Vector2Int(1, 6);

    private readonly Vector2Int cpuGoalkeeperPosition = new Vector2Int(19, 6);

    public Vector2Int[] playerFieldPositions = new Vector2Int[]

    {

        new Vector2Int(1, 6), // GK

        new Vector2Int(5, 3),

        new Vector2Int(5, 6),

        new Vector2Int(5, 10),

        new Vector2Int(9, 5),

        new Vector2Int(9, 7)

    };

    public Vector2Int[] cpuFieldPositions = new Vector2Int[]

    {

        new Vector2Int(19, 6), // GK

        new Vector2Int(14, 3),

        new Vector2Int(14, 6),

        new Vector2Int(14, 10),

        new Vector2Int(11, 5),

        new Vector2Int(11, 7)

    };

    public List<PlayerUnit> playerUnits = new List<PlayerUnit>();

    public List<PlayerUnit> cpuUnits = new List<PlayerUnit>();

    public PlayerUnit playerGoalkeeper;

    public PlayerUnit cpuGoalkeeper;

    void Awake()

    {

        if (Instance == null)

        {

            Instance = this;

            DontDestroyOnLoad(gameObject);

        }

        else

        {

            Destroy(gameObject);

            return;

        }

    }

    public void InitializeTeams(List<PlayerUnit> playerUnits, List<PlayerUnit> cpuUnits)

    {

        this.playerUnits = playerUnits;

        this.cpuUnits = cpuUnits;

        // Ensure goalkeepers are set

        if (playerGoalkeeper == null)

            playerGoalkeeper = GetGoalkeeper(TeamType.Player);

        if (cpuGoalkeeper == null)

            cpuGoalkeeper = GetGoalkeeper(TeamType.CPU);

    }

    public PlayerUnit GetGoalkeeper(TeamType teamType)

    {

        foreach (var unit in GetTeamUnits(teamType))

        {

            if ((TeamManager.PositionType)unit.positionType == TeamManager.PositionType.Goalkeeper)

                return unit;

        }

        return null;

    }

    public List<PlayerUnit> GetTeamMembers(TeamType team)

    {

        return new List<PlayerUnit>(team == TeamType.Player ? playerUnits : cpuUnits);

    }

    public List<PlayerUnit> GetAllUnits()

    {

        List<PlayerUnit> allUnits = new List<PlayerUnit>();

        allUnits.AddRange(playerUnits);

        allUnits.AddRange(cpuUnits);

        return allUnits;

    }

    public PlayerUnit GetPlayerAtPosition(Vector2Int position)

    {

        foreach (var unit in GetAllUnits())

        {

            if (unit.gridPosition == position)

                return unit;

        }

        return null;

    }

    public List<PlayerUnit> GetTeamUnits(TeamType teamType)

    {

        return teamType == TeamType.Player ? playerUnits : cpuUnits;

    }

    public PlayerUnit GetFirstPlayer()

    {

        return playerUnits.Count > 0 ? playerUnits[0] : null;

    }

    public static TeamType GetTeamAtPosition(Vector2Int position)

    {

        if (position == new Vector2Int(0, 6)) return TeamType.Player;

        if (position == new Vector2Int(19, 6)) return TeamType.CPU;

        return TeamType.Player; // Default fallback

    }

    public PlayerUnit GetLastPlayer()

    {

        return playerUnits.Count > 0 ? playerUnits[playerUnits.Count - 1] : null;

    }

    public void RegisterUnits(List<PlayerUnit> units)

{

    playerUnits.Clear();

    cpuUnits.Clear();

    foreach (var unit in units)

    {

        if ((TeamManager.TeamType)unit.teamType == TeamType.Player)

        {

            playerUnits.Add(unit);

            if ((PositionType)unit.positionType == PositionType.Goalkeeper)

                playerGoalkeeper = unit;

        }

        else // CPU

        {

            cpuUnits.Add(unit);

            if ((PositionType)unit.positionType == PositionType.Goalkeeper)

                cpuGoalkeeper = unit;

        }

    }

}

    public int AdjustDifficultyWithGoalkeeper(PlayerUnit isgoalkeeper, int baseDifficulty)

    {

        return baseDifficulty;

    }

    public void ResetTeams()

{

    foreach (var unit in UnitManager.Instance.allUnits)

    {

        unit.ResetToStartPosition();

    }

    }

}

using UnityEngine;

using System.Collections.Generic;

public class TileSelector : MonoBehaviour

{

    public static TileSelector Instance;

    private List<Tile> highlightedTiles = new List<Tile>();

    private bool isSelectingMove = false;

    private int moveRange = 3;

    public int MoveRange

    {

        get { return moveRange; }

        set { moveRange = value; }

    }

    public int MoveRangeMin = 1;

    public int MoveRangeMax = 3;

    public int MoveRangeDefault = 3;

    private void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

    }

    // Called by UI Move button to start move selection

    public void OnMoveButtonClicked()

    {

        if (TurnManager.Instance == null)

        {

            Debug.LogWarning("TileSelector: TurnManager not initialized.");

            return;

        }

        PlayerUnit currentUnit = TurnManager.Instance.GetCurrentUnit();

        if (currentUnit == null || currentUnit.teamType != TeamType.Player)

        {

            Debug.LogWarning("TileSelector: No player unit or not player's turn.");

            return;

        }

        if (currentUnit.HasMovedThisTurn)

        {

            Debug.LogWarning("TileSelector: Unit already moved this turn.");

            return;

        }

        StartMoveSelection();

    }

    // Called to start move selection process

    public void StartMoveSelection()

    {

        if (TurnManager.Instance == null) return;

        PlayerUnit currentUnit = TurnManager.Instance.GetCurrentUnit();

        if (currentUnit == null || currentUnit.teamType != TeamType.Player)

        {

            Debug.Log("No player unit or not player's turn.");

            return;

        }

        isSelectingMove = true;

        HighlightValidTiles(currentUnit);

        Debug.Log($"Select a tile within {moveRange} tiles to move.");

    }

    void HighlightValidTiles(PlayerUnit unit)

{

    ClearHighlights();

    Vector2Int pos = unit.gridPosition;

    foreach (Tile tile in FindObjectsByType<Tile>(FindObjectsSortMode.None))

    {

        int dist = Mathf.Abs(tile.tilePosition.x - pos.x) + Mathf.Abs(tile.tilePosition.y - pos.y);

        if (dist <= moveRange && dist > 0) // ✅ exclude self

        {

            tile.Highlight(true);

            highlightedTiles.Add(tile);

        }

    }

}

    void ClearHighlights()

    {

        foreach (Tile tile in highlightedTiles)

        {

            tile.Highlight(false);

        }

        highlightedTiles.Clear();

    }

    private void Update()

    {

        if (!isSelectingMove) return;

        if (Input.GetMouseButtonDown(0))

        {

            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out RaycastHit hit))

            {

                Tile clickedTile = hit.collider.GetComponent<Tile>();

                if (clickedTile != null && highlightedTiles.Contains(clickedTile))

                {

                    OnTileSelected(clickedTile);

                }

                else

                {

                    Debug.Log("Clicked tile not valid.");

                }

            }

        }

    }

    void OnTileSelected(Tile tile)

    {

        PlayerUnit currentUnit = TurnManager.Instance.GetCurrentUnit();

        if (currentUnit == null) return;

        bool moved = currentUnit.TryMove(tile.tilePosition);

        if (moved)

        {

            ClearHighlights();

            if (!currentUnit.HasActionPoints(1)) // ✅ Still has AP, re-highlight!

            {

                StartMoveSelection(); // 👈 This also highlights and sets the flag again

            }

            Debug.Log($"Moved to {tile.tilePosition}. Remaining AP: {currentUnit.CurrentActionPoints}");

        }

        else if (currentUnit.HasActionPoints(0))

        {

            Debug.Log("Move completed, no more AP left.");

            isSelectingMove = false; // End selection

        }

        else

        {

            Debug.Log("Move failed — likely out of AP or invalid tile.");

        }

    }

    public void OnTileSelectedFromInput(Tile tile)

    {

        if (!isSelectingMove) return;

        OnTileSelected(tile); // Reuse your actual handler

    }

}

using System;

using System.Collections.Generic;

using UnityEngine;

public class Tile : MonoBehaviour

{

    public Vector2Int tilePosition;

    private Renderer rend;

    private Color originalColor;

    public Color highlightColor = Color.cyan;

    public bool isPassTarget = false;

    // 🆕 Occupancy Tracking

    public bool isOccupied = false;

    // Make sure the Unit class is defined somewhere in your project, or define a placeholder here if missing.

    // public Unit occupyingUnit = null;

    public MonoBehaviour occupyingUnit = null; // Replace 'MonoBehaviour' with the correct type when 'Unit' is available.

    private void Awake()

    {

        rend = GetComponent<Renderer>();

        originalColor = rend.material.color;

    }

    public void Highlight(bool enable, Color? customColor = null)

    {

        if (enable)

        {

            rend.material.color = customColor ?? highlightColor;

            isPassTarget = true;

        }

        else

        {

            rend.material.color = originalColor;

            isPassTarget = false;

        }

    }

    private void OnMouseDown()

    {

    if (TurnManager.Instance == null)

    {

        Debug.LogWarning("TurnManager.Instance is null!");

        return;

    }

    var currentUnit = TurnManager.Instance.GetCurrentUnit();

    if (currentUnit == null || currentUnit.teamType != TeamType.Player) return;

    if (isPassTarget)

    {

        TurnManager.Instance.TryPassToTile(this);

    }

    else

    {

        if (isOccupied)

        {

            Debug.Log("Cannot move here — tile is already occupied!");

            return;

        }

        bool moved = currentUnit.TryMove(tilePosition);

        if (!moved)

        {

            Debug.Log("Move failed. Occupancy unchanged.");

        }

    }

    }

    public void ClearHighlight()

    {

        rend.material.color = originalColor;

        isPassTarget = false;

    }

}

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

public class TurnManager : MonoBehaviour

{

    public static TurnManager Instance;

    public TeamType currentTeam = TeamType.Player;

    private List<PlayerUnit> playerUnits = new List<PlayerUnit>();

    private List<PlayerUnit> cpuUnits = new List<PlayerUnit>();

    private int currentUnitIndex = 0;

    public TeamType CurrentTurn { get; private set; }

    private void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

    }

    private void Start()

    {

        if (TeamManager.Instance != null)

        {

            Initialize(TeamManager.Instance.playerUnits, TeamManager.Instance.cpuUnits);

        }

        else

        {

            Debug.LogError("[TurnManager] TeamManager.Instance is null! Cannot initialize.");

        }

    }

    public void Initialize(List<PlayerUnit> players, List<PlayerUnit> cpus)

    {

        if (players == null || cpus == null || players.Count == 0 || cpus.Count == 0)

        {

            Debug.LogError("[TurnManager] Initialization failed: Missing player or CPU units.");

            return;

        }

        playerUnits = players;

        cpuUnits = cpus;

        currentTeam = TeamType.Player;

        currentUnitIndex = 0;

        Debug.Log("[TurnManager] Initialized. Starting Player Turn...");

        StartTeamTurn();

    }

    public void StartTeamTurn()

{

    if (currentTeam == TeamType.Player)

    {

        List<PlayerUnit> teamUnits = GetCurrentTeamUnits();

        if (teamUnits.Count == 0)

        {

            Debug.LogWarning("[TurnManager] No units found for current player team.");

            SwitchTeam();

            return;

        }

        currentUnitIndex = 0;

        StartUnitTurn(teamUnits[currentUnitIndex]);

    }

    else if (currentTeam == TeamType.CPU)

    {

        if (CPUUnitLogic.Instance != null)

        {

            CPUUnitLogic.Instance.TakeTeamTurn();

        }

        else

        {

            Debug.LogError("[TurnManager] CPUUnitLogic instance missing.");

            SwitchTeam();

        }

    }

}

    public void SetCurrentTurn(TeamType team)

    {

        currentTeam = team;

        Debug.Log($"[TurnManager] Current turn set to: {currentTeam}");

    }

    private void StartUnitTurn(PlayerUnit unit)

    {

        if (unit == null)

        {

            Debug.LogError("[TurnManager] Null unit passed to StartUnitTurn.");

            return;

        }

        // Only player units use this

        if (unit.teamType == TeamType.Player)

        {

            Debug.Log($"[TurnManager] Starting turn for {unit.name} ({unit.teamType})");

            unit.StartTurn();

            unit.isPlayerControlled = true;

            TileSelector.Instance?.StartMoveSelection();

        }

        else

        {

            Debug.LogWarning("[TurnManager] StartUnitTurn called for CPU unit, but CPU turn handled separately.");

        }

    }

public void OnCPUTeamTurnEnded()

{

    Debug.Log("[TurnManager] CPU Team turn ended. Switching to Player.");

    SwitchTeam();

}

    public void EndUnitTurn()

{

    if (currentTeam == TeamType.CPU)

    {

        Debug.LogWarning("[TurnManager] Ignored EndUnitTurn during CPU team turn.");

        return;

    }

    PlayerUnit currentUnit = GetCurrentUnit();

    if (currentUnit != null) currentUnit.isPlayerControlled = false;

    List<PlayerUnit> teamUnits = GetCurrentTeamUnits();

    currentUnitIndex++;

    if (currentUnitIndex >= teamUnits.Count)

    {

        SwitchTeam();

    }

    else

    {

        StartUnitTurn(teamUnits[currentUnitIndex]);

    }

}

    private void SwitchTeam()

    {

        currentTeam = (currentTeam == TeamType.Player) ? TeamType.CPU : TeamType.Player;

        Debug.Log($"[TurnManager] Switching to {currentTeam} team.");

        StartTeamTurn();

    }

    public PlayerUnit GetCurrentUnit()

    {

        List<PlayerUnit> teamUnits = GetCurrentTeamUnits();

        if (currentUnitIndex >= 0 && currentUnitIndex < teamUnits.Count)

        {

            return teamUnits[currentUnitIndex];

        }

        return null;

    }

    private List<PlayerUnit> GetCurrentTeamUnits()

    {

        return (currentTeam == TeamType.Player) ? playerUnits : cpuUnits;

    }

    // Called by button/UI or game logic to initiate a pass

    public void OnPassButtonClicked()

    {

        if (PassSelector.Instance == null) return;

        if (!PassSelector.Instance.isSelectingPassTarget)

        {

            PassSelector.Instance.StartPassSelection();

        }

    }

    public void TryPassToTile(Tile tile)

    {

        if (PassSelector.Instance == null) return;

        if (!PassSelector.Instance.isSelectingPassTarget)

        {

            PassSelector.Instance.StartPassSelection();

        }

    }

    public void StartPassChallenge(PlayerUnit targetPlayer)

    {

        if (targetPlayer == null)

        {

            Debug.LogError("[TurnManager] Target player for pass challenge is null.");

            return;

        }

        if (MathChallengeUI.Instance == null)

        {

            Debug.LogError("[TurnManager] MathChallengeUI instance is missing.");

            return;

        }

        string question = "What is 2 + 2?"; // Replace with dynamic logic

        int difficulty = 1;

        Debug.Log($"[TurnManager] Starting pass challenge to {targetPlayer.name}");

        MathChallengeUI.Instance.ShowQuestion(

            question,

            difficulty,

            (bool success) =>

            {

                if (success)

                {

                    PassBallTo(targetPlayer);

                }

                else

                {

                    EndUnitTurn();

                }

            }

        );

    }

    public void PassBallTo(PlayerUnit targetPlayer)

    {

        if (BallManager.Instance == null)

        {

            Debug.LogError("[TurnManager] BallManager instance is missing!");

            return;

        }

        BallManager.Instance.PassBallTo(targetPlayer);

        StartCoroutine(WaitForBallMovement());

    }

    private IEnumerator WaitForBallMovement()

    {

        while (BallManager.Instance != null && BallManager.Instance.IsBallMoving())

        {

            yield return null;

        }

        Debug.Log("[TurnManager] Ball movement complete. Resuming...");

        EndUnitTurn();

    }

    public void ResetTurnOrder()

{

    Debug.Log("[TurnManager] Resetting turn order.");

    currentUnitIndex = 0;

    StartTeamTurn(); // respects whatever `currentTeam` is already set to

}

}

using UnityEngine;

using UnityEngine.UI;

public class UIActionPanel : MonoBehaviour

{

    public Button moveButton;

    public Button endTurnButton;

    public Button passButton;

    public Button shootButton;

    public Button advanceButton;

    public Button defendButton;

    public Button tackleButton;

    private PlayerUnit currentUnit;

    private readonly int movementRange = 3;

    private Color moveHighlightColor = new Color(0f, 1f, 0f, 0.4f);

    private bool isSelectingPassTarget = false;

    private void Awake()

    {

        if (moveButton == null || endTurnButton == null ||

            passButton == null || shootButton == null ||

            advanceButton == null || defendButton == null ||

            tackleButton == null)

        {

            Debug.LogError("UIActionPanel: Not all buttons are assigned in the inspector.");

        }

    }

    private void Start()

    {

        moveButton.onClick.AddListener(OnMoveClicked);

        endTurnButton.onClick.AddListener(OnEndTurnClicked);

        passButton.onClick.AddListener(OnPassClicked);

        shootButton.onClick.AddListener(OnShootClicked);

        advanceButton.onClick.AddListener(OnAdvanceClicked);

        defendButton.onClick.AddListener(OnDefendClicked);

        tackleButton.onClick.AddListener(OnTackleClicked);

        UpdateCurrentUnit(TurnManager.Instance.GetCurrentUnit());

    }

    private void Update()

    {

        if (isSelectingPassTarget && Input.GetMouseButtonDown(0))

        {

            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out RaycastHit hit))

            {

                var targetUnit = hit.collider.GetComponent<PlayerUnit>();

                if (targetUnit != null && currentUnit != null && targetUnit.teamType == currentUnit.teamType && targetUnit != currentUnit)

                {

                    isSelectingPassTarget = false;

                    ClearHighlights();

                    currentUnit.Pass(targetUnit);

                    UpdateCurrentUnit(currentUnit);

                }

            }

        }

        if (TurnManager.Instance == null) return;

        PlayerUnit unit = TurnManager.Instance.GetCurrentUnit();

        if (unit != null && unit != currentUnit)

        {

            UpdateCurrentUnit(unit);

        }

    }

    private void UpdateCurrentUnit(PlayerUnit unit)

    {

        currentUnit = unit;

        GridManager.Instance.ClearAllHighlights();

        bool isPlayerTurn = currentUnit != null && currentUnit.teamType == TeamType.Player;

        moveButton.interactable = isPlayerTurn && currentUnit.CurrentActionPoints > 0;

        endTurnButton.interactable = isPlayerTurn;

        passButton.interactable = isPlayerTurn && currentUnit.CurrentActionPoints > 0;

        shootButton.interactable = isPlayerTurn && currentUnit.CurrentActionPoints > 1;

        advanceButton.interactable = isPlayerTurn && currentUnit.CurrentActionPoints > 0;

        defendButton.interactable = isPlayerTurn && currentUnit.CurrentActionPoints > 0;

        tackleButton.interactable = isPlayerTurn && currentUnit.CurrentActionPoints > 1;

        if (isPlayerTurn)

        {

            HighlightMovableTiles();

        }

    }

    private void HighlightMovableTiles()

    {

        if (currentUnit == null) return;

        Vector2Int origin = currentUnit.gridPosition;

        for (int x = -movementRange; x <= movementRange; x++)

        {

            for (int y = -movementRange; y <= movementRange; y++)

            {

                Vector2Int testPos = origin + new Vector2Int(x, y);

                int manhattanDistance = Mathf.Abs(x) + Mathf.Abs(y);

                if (manhattanDistance <= movementRange)

                {

                    Tile tile = GridManager.Instance.GetTileAtPosition(testPos);

                    if (tile != null)

                    {

                        tile.Highlight(true, moveHighlightColor);

                    }

                }

            }

        }

    }

    private void OnMoveClicked()

{

    if (currentUnit == null || currentUnit.CurrentActionPoints <= 0) return;

    Debug.Log("Move mode activated.");

    TileSelector.Instance?.StartMoveSelection();

}

    private void OnEndTurnClicked()

    {

        TurnManager.Instance.EndUnitTurn();

        UpdateCurrentUnit(TurnManager.Instance.GetCurrentUnit());

    }

    private void OnPassClicked()

    {

        if (currentUnit == null || currentUnit.CurrentActionPoints <= 0) return;

        isSelectingPassTarget = true;

        HighlightTeammates(currentUnit);

    }

    private void HighlightTeammates(PlayerUnit unit)

    {

        if (unit == null) return;

        TeamManager teamManager = TeamManager.Instance;

        foreach (var teammate in teamManager.GetTeamUnits((TeamManager.TeamType)unit.teamType))

        {

            if (teammate != unit)

            {

                Tile tile = GridManager.Instance.GetTileAtPosition(teammate.gridPosition);

                if (tile != null)

                {

                    tile.Highlight(true, new Color(0f, 0.5f, 1f, 0.4f));

                }

            }

        }

    }

    private void OnShootClicked()

    {

        if (currentUnit == null || currentUnit.CurrentActionPoints < 2) return;

        Vector2Int shotTarget = GridManager.Instance.GetShotTarget(currentUnit.teamType);

        currentUnit.Shoot(shotTarget);

        UpdateCurrentUnit(currentUnit);

    }

    private void OnAdvanceClicked()

    {

        if (currentUnit == null || currentUnit.CurrentActionPoints < 1) return;

        currentUnit.Advance();

        UpdateCurrentUnit(currentUnit);

    }

    private void OnDefendClicked()

    {

        if (currentUnit == null || currentUnit.CurrentActionPoints < 1) return;

        currentUnit.Defend();

        UpdateCurrentUnit(currentUnit);

    }

    private void OnTackleClicked()

    {

        if (currentUnit == null || currentUnit.CurrentActionPoints < 2) return;

        var target = FindTackleTarget();

        if (target == null)

        {

            Debug.Log("No valid target to tackle.");

            return;

        }

        currentUnit.AttemptTackle(target);

        UpdateCurrentUnit(currentUnit);

    }

    public PlayerUnit FindTackleTarget()

    {

        PlayerUnit ballHolder = BallManager.Instance?.currentHolder;

        if (ballHolder != null && currentUnit.IsAdjacent(ballHolder))

        {

            return ballHolder;

        }

        TeamManager teamManager = TeamManager.Instance;

        foreach (var opponent in teamManager.GetTeamUnits((TeamManager.TeamType)currentUnit.GetOpposingTeamType()))

        {

            if (currentUnit.IsAdjacent(opponent) && opponent.HasBall)

            {

                return opponent;

            }

        }

        return null;

    }

    public void ClearHighlights()

    {

        GridManager.Instance.ClearAllHighlights();

    }

}

using System.Collections.Generic;

using System.Linq;

using JetBrains.Annotations;

using UnityEngine;

public class UnitManager : MonoBehaviour

{

    public static UnitManager Instance;

    public GameObject playerUnitPrefab;

    public GameObject cpuUnitPrefab;

    public List<PlayerUnit> allUnits = new List<PlayerUnit>();

    private readonly Vector2Int[] playerTeamPositions = new Vector2Int[]

    {

        new Vector2Int(1, 6), // GK

        new Vector2Int(5, 3),

        new Vector2Int(5, 6),

        new Vector2Int(5, 10),

        new Vector2Int(9, 5),

        new Vector2Int(9, 7)

    };

    private readonly Vector2Int[] cpuTeamPositions = new Vector2Int[]

    {

        new Vector2Int(19, 6), // GK

        new Vector2Int(14, 3),

        new Vector2Int(14, 6),

        new Vector2Int(14, 10),

        new Vector2Int(11, 5),

        new Vector2Int(11, 7)

    };

    public Vector2Int gridPosition;

    public Vector2Int spawnPosition;

    private void Awake()

    {

        if (Instance == null)

            Instance = this;

        else

            Destroy(gameObject);

    }

    private void Start()

    {

        SpawnInitialUnits();

    }

    public void SpawnInitialUnits()

    {

        allUnits.Clear();

        for (int i = 0; i < playerTeamPositions.Length; i++)

        {

            var team = TeamManager.TeamType.Player;

            var position = (i == 0) ? TeamManager.PositionType.Goalkeeper : TeamManager.PositionType.Field;

            SpawnUnit(playerUnitPrefab, playerTeamPositions[i], team, position);

        }

        for (int i = 0; i < cpuTeamPositions.Length; i++)

        {

            var team = TeamManager.TeamType.CPU;

            var position = (i == 0) ? TeamManager.PositionType.Goalkeeper : TeamManager.PositionType.Field;

            SpawnUnit(cpuUnitPrefab, cpuTeamPositions[i], team, position);

        }

        TeamManager.Instance?.RegisterUnits(allUnits);

    }

    void SpawnUnit(GameObject prefab, Vector2Int gridPos, TeamManager.TeamType team, TeamManager.PositionType position)

    {

        float unitYOffset = 0.5f; // Adjust this value to sit units just above the tile

        Vector3 worldPos = GridManager.Instance.GetWorldPosition(gridPos) + Vector3.up \* unitYOffset;

        GameObject unitObj = Instantiate(prefab, worldPos, Quaternion.identity);

        PlayerUnit unit = unitObj.GetComponent<PlayerUnit>();

        unit.Initialize((TeamType)team, (PositionType)position, gridPos);

        allUnits.Add(unit);

    }

}

using UnityEngine;

public class IdleState : AIState

{

    public IdleState(PlayerUnit unit) : base(unit) { }

    public override void EnterState()

    {

        Debug.Log($"{unit.name} enters Idle State.");

    }

    public override void UpdateState()

    {

        Debug.Log($"{unit.name} is idling...");

    }

}

using UnityEngine;

public class TackleState : AIState

{

    private bool isDone = false;

    private PlayerUnit target;

    public TackleState(PlayerUnit unit, PlayerUnit target) : base(unit)

    {

        this.target = target;

    }

    public override void EnterState()

    {

        if (target == null || !target.HasBall || !unit.IsAdjacent(target))

        {

            Debug.Log($"{unit.name} cannot tackle — invalid target.");

            isDone = true;

            return;

        }

        Debug.Log($"{unit.name} attempts to tackle {target.name}.");

        unit.AttemptTackle(target);

        isDone = true;

    }

    public override void UpdateState() { }

    public override bool IsComplete => isDone || unit.CurrentActionPoints <= 0;

}

using UnityEngine;

public class DefendState : AIState

{

    private bool isDone = false;

    public DefendState(PlayerUnit unit) : base(unit) { }

    public override void EnterState()

    {

        if (unit.CurrentActionPoints <= 0)

        {

            isDone = true;

            return;

        }

        unit.Defend();

        isDone = true;

    }

    public override void UpdateState() { }

    public override bool IsComplete => isDone;

}

using UnityEngine;

public class ChaseBallState : AIState

{

    private bool isDone = false;

    public ChaseBallState(PlayerUnit unit) : base(unit) { }

    public override void EnterState()

    {

        Debug.Log($"{unit.name} enters ChaseBall State.");

    }

    public override void UpdateState()

    {

        if (isDone || unit.CurrentActionPoints <= 0)

        {

            isDone = true;

            return;

        }

        Vector2Int ballPos = BallManager.Instance?.ballGridPosition ?? unit.gridPosition;

        Vector2Int dirToBall = ballPos - unit.gridPosition;

        Vector2Int target = unit.gridPosition + new Vector2Int(

            Mathf.Clamp(dirToBall.x, -1, 1),

            Mathf.Clamp(dirToBall.y, -1, 1)

        );

        if (unit.TryMove(target))

        {

            Debug.Log($"{unit.name} moved toward the ball.");

        }

        else

        {

            Debug.Log($"{unit.name} failed to move.");

            unit.CurrentActionPoints = 0;

        }

        isDone = true;

    }

    public override bool IsComplete => isDone || unit.CurrentActionPoints <= 0;

}

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

public class CPUUnitLogic : MonoBehaviour

{

    public static CPUUnitLogic Instance;

    private List<PlayerUnit> cpuUnits;

    private int currentUnitIndex = 0;

    private bool isTakingTurn = false;

    private void Awake()

    {

        if (Instance == null) Instance = this;

        else Destroy(gameObject);

    }

    public void TakeTeamTurn()

    {

        if (isTakingTurn) return;

        cpuUnits = TeamManager.Instance?.cpuUnits;

        if (cpuUnits == null || cpuUnits.Count == 0)

        {

            Debug.LogWarning("CPUUnitLogic: No CPU units available to take turn.");

            EndTeamTurn();

            return;

        }

        currentUnitIndex = 0;

        isTakingTurn = true;

        StartCoroutine(TakeNextUnitTurn());

    }

    private IEnumerator TakeNextUnitTurn()

    {

        if (currentUnitIndex >= cpuUnits.Count)

        {

            EndTeamTurn();

            yield break;

        }

        PlayerUnit unit = cpuUnits[currentUnitIndex];

        currentUnitIndex++;

        if (unit == null)

        {

            Debug.LogWarning("CPUUnitLogic: null unit found in cpuUnits list.");

            yield return TakeNextUnitTurn();

            yield break;

        }

        AIStateMachine ai = unit.GetComponent<AIStateMachine>();

        if (ai == null)

        {

            Debug.LogWarning($"CPUUnitLogic: AIStateMachine missing on unit {unit.name}.");

            yield return TakeNextUnitTurn();

            yield break;

        }

        unit.CurrentActionPoints = unit.maxActionPoints; // Reset action points

        Debug.Log($"CPUUnitLogic: Starting turn for {unit.name} with {unit.CurrentActionPoints} AP.");

        yield return ai.ExecuteTurn();

        // Optional short wait before next unit

        yield return new WaitForSeconds(0.2f);

        yield return TakeNextUnitTurn();

    }

    private void EndTeamTurn()

    {

        isTakingTurn = false;

        Debug.Log("CPUUnitLogic: CPU team turn ended.");

        TurnManager.Instance?.OnCPUTeamTurnEnded();

    }

    // Example simple logic to find a pass target (returns first teammate not self)

    public PlayerUnit FindBestPassTarget(PlayerUnit passer)

    {

        if (cpuUnits == null) return null;

        foreach (var unit in cpuUnits)

        {

            if (unit != null && unit != passer)

                return unit;

        }

        return null;

    }

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public abstract class BTNode

{

    public enum NodeState { Success, Failure, Running }

    public NodeState State { get; protected set; } = NodeState.Running;

    public abstract NodeState Evaluate();

}

public class BTSelector : BTNode

{

    private List<BTNode> children;

    public BTSelector(List<BTNode> nodes) { children = nodes; }

    public override NodeState Evaluate()

    {

        foreach (var node in children)

        {

            var result = node.Evaluate();

            if (result == NodeState.Success) return State = NodeState.Success;

            if (result == NodeState.Running) return State = NodeState.Running;

        }

        return State = NodeState.Failure;

    }

}

public class BTSequence : BTNode

{

    private List<BTNode> children;

    public BTSequence(List<BTNode> nodes) { children = nodes; }

    public override NodeState Evaluate()

    {

        foreach (var node in children)

        {

            var result = node.Evaluate();

            if (result != NodeState.Success) return State = result;

        }

        return State = NodeState.Success;

    }

}

// Condition Nodes

public class HasBallNode : BTNode

{

    PlayerUnit unit;

    public HasBallNode(PlayerUnit unit) { this.unit = unit; }

    public override NodeState Evaluate() => unit.HasBall ? NodeState.Success : NodeState.Failure;

}

public class IsNearGoalNode : BTNode

{

    PlayerUnit unit;

    public IsNearGoalNode(PlayerUnit unit) { this.unit = unit; }

    public override NodeState Evaluate()

    {

        Vector2Int goalPos = ((unit.teamType == TeamType.CPU) ? new Vector2Int(0, 5) : new Vector2Int(20, 5));

        return Vector2Int.Distance(unit.gridPosition, goalPos) <= 2 ? NodeState.Success : NodeState.Failure;

    }

}

public class HasActionPointsNode : BTNode

{

    PlayerUnit unit; int requiredAP;

    public HasActionPointsNode(PlayerUnit unit, int ap) { this.unit = unit; requiredAP = ap; }

    public override NodeState Evaluate() => unit.CurrentActionPoints >= requiredAP ? NodeState.Success : NodeState.Failure;

}

public class IsAdjacentToEnemyWithBallNode : BTNode

{

    PlayerUnit unit;

    public IsAdjacentToEnemyWithBallNode(PlayerUnit unit) { this.unit = unit; }

    public override NodeState Evaluate()

    {

        PlayerUnit enemyWithBall = BallManager.Instance?.currentHolder;

        if (enemyWithBall != null && enemyWithBall.teamType != unit.teamType)

            return unit.IsAdjacent(enemyWithBall) ? NodeState.Success : NodeState.Failure;

        return NodeState.Failure;

    }

}

// Action Nodes - these run coroutines and use MathChallenge

public class ShootActionNode : BTNode

{

    PlayerUnit unit;

    Vector2Int goalPos;

    bool isRunning = false;

    bool finishedSuccess = false;

    public ShootActionNode(PlayerUnit unit)

    {

        this.unit = unit;

        goalPos = ((unit.teamType == TeamType.CPU) ? new Vector2Int(0, 5) : new Vector2Int(20, 5));

    }

    public override NodeState Evaluate()

    {

        if (!isRunning)

        {

            isRunning = true;

            CoroutineRunner.Instance.StartCoroutine(ShootRoutine());

            return NodeState.Running;

        }

        return finishedSuccess ? NodeState.Success : NodeState.Failure;

    }

    private IEnumerator ShootRoutine()

    {

        // Ask math challenge, difficulty depending on distance

        float dist = Vector2Int.Distance(unit.gridPosition, goalPos);

        int difficulty = Mathf.Clamp(Mathf.RoundToInt(dist \* 2), 1, 5);

        bool? result = null;

        MathChallenge.OverrideCallback = (success) => result = success;

        MathChallenge.Instance.AskQuestion(difficulty.ToString(), unit, MathChallenge.OverrideCallback, null);

        while (result == null) yield return null;

        if (result == true)

        {

            unit.Shoot(goalPos);

            unit.UseActionPoints(2);

            finishedSuccess = true;

        }

        else

        {

            finishedSuccess = false;

        }

        MathChallenge.OverrideCallback = null;

        isRunning = false;

    }

}

public class PassActionNode : BTNode

{

    PlayerUnit unit;

    PlayerUnit target;

    bool isRunning = false;

    bool finishedSuccess = false;

    public PassActionNode(PlayerUnit unit, PlayerUnit target)

    {

        this.unit = unit;

        this.target = target;

    }

    public override NodeState Evaluate()

    {

        if (target == null) return NodeState.Failure;

        if (!isRunning)

        {

            isRunning = true;

            CoroutineRunner.Instance.StartCoroutine(PassRoutine());

            return NodeState.Running;

        }

        return finishedSuccess ? NodeState.Success : NodeState.Failure;

    }

    private IEnumerator PassRoutine()

    {

        // Distance-based difficulty

        float dist = Vector2Int.Distance(unit.gridPosition, target.gridPosition);

        int difficulty = Mathf.Clamp(Mathf.RoundToInt(dist \* 2), 1, 5);

        bool? result = null;

        MathChallenge.OverrideCallback = (success) => result = success;

        MathChallenge.Instance.AskQuestion(difficulty.ToString(), unit, MathChallenge.OverrideCallback, target);

        while (result == null) yield return null;

        if (result == true)

        {

            unit.Pass(target);

            unit.UseActionPoints(1);

            finishedSuccess = true;

        }

        else

        {

            finishedSuccess = false;

        }

        MathChallenge.OverrideCallback = null;

        isRunning = false;

    }

}

public class AdvanceActionNode : BTNode

{

    PlayerUnit unit;

    bool isRunning = false;

    bool finishedSuccess = false;

    public AdvanceActionNode(PlayerUnit unit) { this.unit = unit; }

    public override NodeState Evaluate()

    {

        if (!isRunning)

        {

            isRunning = true;

            CoroutineRunner.Instance.StartCoroutine(AdvanceRoutine());

            return NodeState.Running;

        }

        return finishedSuccess ? NodeState.Success : NodeState.Failure;

    }

    private IEnumerator AdvanceRoutine()

    {

        Vector2Int goalPos = ((unit.teamType == TeamType.CPU) ? new Vector2Int(0, 5) : new Vector2Int(20, 5));

        Vector2Int direction = (goalPos - unit.gridPosition);

        Vector2Int moveTarget = unit.gridPosition;

        if (Mathf.Abs(direction.x) > Mathf.Abs(direction.y))

            moveTarget.x += (direction.x > 0) ? 1 : -1;

        else if (direction.y != 0)

            moveTarget.y += (direction.y > 0) ? 1 : -1;

        bool moved = unit.TryMove(moveTarget);

        if (moved)

        {

            unit.UseActionPoints(2);

            finishedSuccess = true;

        }

        else

        {

            finishedSuccess = false;

        }

        yield return null;

        isRunning = false;

    }

}

public class TackleActionNode : BTNode

{

    PlayerUnit unit;

    PlayerUnit target;

    bool isRunning = false;

    bool finishedSuccess = false;

    public TackleActionNode(PlayerUnit unit, PlayerUnit target)

    {

        this.unit = unit;

        this.target = target;

    }

    public override NodeState Evaluate()

    {

        if (target == null) return NodeState.Failure;

        if (!isRunning)

        {

            isRunning = true;

            CoroutineRunner.Instance.StartCoroutine(TackleRoutine());

            return NodeState.Running;

        }

        return finishedSuccess ? NodeState.Success : NodeState.Failure;

    }

    private IEnumerator TackleRoutine()

    {

        int difficulty = 3; // fixed difficulty or based on some stat

        bool? result = null;

        MathChallenge.OverrideCallback = (success) => result = success;

        MathChallenge.Instance.AskQuestion(difficulty.ToString(), unit, MathChallenge.OverrideCallback, target);

        while (result == null) yield return null;

        if (result == true)

        {

            unit.AttemptTackle(target);

            unit.UseActionPoints(1);

            finishedSuccess = true;

        }

        else

        {

            finishedSuccess = false;

        }

        MathChallenge.OverrideCallback = null;

        isRunning = false;

    }

}

public class IdleActionNode : BTNode

{

    PlayerUnit unit;

    bool isDone = false;

    public IdleActionNode(PlayerUnit unit) { this.unit = unit; }

    public override NodeState Evaluate()

    {

        if (!isDone)

        {

            unit.CurrentActionPoints = 0;  // End turn immediately

            isDone = true;

            return NodeState.Success;

        }

        return NodeState.Success;

    }

}

using UnityEngine;

public class ShootOrPassState : AIState

{

    private bool isDone = false;

    public ShootOrPassState(PlayerUnit unit) : base(unit) { }

    public override void EnterState()

    {

        Debug.Log($"{unit.name} enters ShootOrPass State.");

        if (unit.CurrentActionPoints <= 0 || !unit.HasBall)

        {

            isDone = true;

            return;

        }

        if (IsNearGoal(unit) && unit.CurrentActionPoints >= 2)

        {

            Vector2Int goalPos = GetGoalPosition(unit);

            unit.Shoot(goalPos);

            isDone = true;

        }

        else

        {

            PlayerUnit target = FindBestPassTarget();

            if (target != null)

            {

                unit.Pass(target);

                isDone = true;

            }

            else

            {

                unit.Advance(); // Will use math challenge to attempt movement

                isDone = true;

            }

        }

    }

    public override void UpdateState() { }

    private bool IsNearGoal(PlayerUnit u)

    {

        return Vector2Int.Distance(u.gridPosition, GetGoalPosition(u)) <= 2;

    }

    private Vector2Int GetGoalPosition(PlayerUnit u)

    {

        return (u.teamType == TeamType.CPU) ? new Vector2Int(0, 6) : new Vector2Int(20, 6);

    }

    private PlayerUnit FindBestPassTarget()

    {

        var teammates = TeamManager.Instance.GetTeamMembers((TeamManager.TeamType)unit.teamType);

        PlayerUnit bestTarget = null;

        float bestScore = float.MinValue;

        foreach (var teammate in teammates)

        {

            if (teammate == unit || teammate.CurrentActionPoints <= 0) continue;

            float distance = Vector2Int.Distance(unit.gridPosition, teammate.gridPosition);

            if (distance > 5) continue;

            float score = (3 - distance) + teammate.CurrentActionPoints \* 0.5f;

            if (score > bestScore)

            {

                bestScore = score;

                bestTarget = teammate;

            }

        }

        return bestTarget;

    }

    public override bool IsComplete => isDone;

}

using UnityEngine;

public abstract class AIState

{

    protected PlayerUnit unit;

    public AIState(PlayerUnit unit)

    {

        this.unit = unit;

    }

    public virtual void EnterState() { }

    public virtual void ExitState() { }

    public abstract void UpdateState();

    public virtual bool IsComplete => true;

}

using UnityEngine;

public class CoroutineRunner : MonoBehaviour

{

    public static CoroutineRunner Instance { get; private set; }

    private void Awake()

    {

        if (Instance == null) {

            Instance = this;

            DontDestroyOnLoad(gameObject);

        } else {

            Destroy(gameObject);

        }

    }

}

using UnityEngine;

using System.Collections;

using System.Collections.Generic;

public class AIStateMachine : MonoBehaviour

{

    public PlayerUnit unit;

    private BTNode behaviorTree;

    private void Awake()

    {

        unit = GetComponent<PlayerUnit>();

    }

    private void Start()

    {

        BuildBehaviorTree();

    }

    public void Init(PlayerUnit u)

    {

        unit = u;

    }

    private void BuildBehaviorTree()

    {

        var cpuLogic = CPUUnitLogic.Instance;

        if (cpuLogic == null)

        {

            Debug.LogError("AIStateMachine: CPUUnitLogic instance missing!");

            behaviorTree = null;

            return;

        }

        PlayerUnit passTarget = cpuLogic.FindBestPassTarget(unit);

        PlayerUnit enemyWithBall = BallManager.Instance?.currentHolder;

        // Defensive fallback: if no pass target, use Idle instead of PassActionNode to avoid null ref

        BTNode passNode = passTarget != null

            ? new PassActionNode(unit, passTarget)

            : (BTNode)new IdleActionNode(unit);

        behaviorTree = new BTSelector(new List<BTNode>

        {

            new BTSequence(new List<BTNode>

            {

                new HasBallNode(unit),

                new IsNearGoalNode(unit),

                new HasActionPointsNode(unit, 2),

                new ShootActionNode(unit)

            }),

            new BTSequence(new List<BTNode>

            {

                new HasBallNode(unit),

                new HasActionPointsNode(unit, 1),

                passNode

            }),

            new BTSequence(new List<BTNode>

            {

                new HasBallNode(unit),

                new HasActionPointsNode(unit, 2),

                new AdvanceActionNode(unit)

            }),

            new BTSequence(new List<BTNode>

            {

                new IsAdjacentToEnemyWithBallNode(unit),

                new HasActionPointsNode(unit, 1),

                new TackleActionNode(unit, enemyWithBall)

            }),

            new IdleActionNode(unit)

        });

    }

    // Coroutine that drives the AI's turn execution step-by-step

    public IEnumerator ExecuteTurn()

    {

        if (unit == null)

        {

            Debug.LogError("AIStateMachine: Unit is null!");

            yield break;

        }

        unit.CurrentActionPoints = unit.maxActionPoints; // Reset AP

        BuildBehaviorTree();

        while (unit.CurrentActionPoints > 0)

        {

            if (behaviorTree == null)

            {

                Debug.LogError("AIStateMachine: Behavior tree not built.");

                yield break;

            }

            var result = behaviorTree.Evaluate();

            if (result == BTNode.NodeState.Failure)

            {

                Debug.Log($"{unit.name}: No valid actions left or finished turn.");

                break;

            }

            // Wait for animations, effects, or user feedback

            yield return new WaitForSeconds(0.3f);

            // Rebuild tree after action in case situation changed

            BuildBehaviorTree();

        }

        Debug.Log($"{unit.name}: Turn complete.");

        yield break;

    }

}

using UnityEngine;

public class RepositionState : AIState

{

    private bool isDone = false;

    public RepositionState(PlayerUnit unit) : base(unit) { }

    public override void EnterState()

    {

        Debug.Log($"{unit.name} is repositioning...");

        Vector2Int center = new Vector2Int(10, 6); // field center

        Vector2Int targetPos = unit.teamType == TeamType.CPU

            ? new Vector2Int(Random.Range(10, 14), Random.Range(4, 8))

            : new Vector2Int(Random.Range(6, 10), Random.Range(4, 8));

        if (unit.TryMove(targetPos))

        {

            Debug.Log($"{unit.name} moved to reposition.");

        }

        else

        {

            Debug.Log($"{unit.name} failed to reposition.");

        }

        isDone = true;

    }

    public override void UpdateState() { }

    public override bool IsComplete => isDone || unit.CurrentActionPoints <= 0;

}