RogueShooter – All Scripts

Generated: 2025-09-01 11.45 UTC
Files: 37
Scanned: Assets/scripts

RogueShooter – All Scripts

Assets/scripts/AllUnitsList.cs

```
using Mirror;
using UnityEngine;

[DisallowMultipleComponent]
public class FriendlyUnit : NetworkBehaviour {}

[DisallowMultipleComponent]
public class EnemyUnit : NetworkBehaviour {}
```

RogueShooter – All Scripts

Assets/scripts/Debugging/ScreenLogger.cs

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

public class ScreenLogger : MonoBehaviour
{
    static ScreenLogger inst;
    TextMeshProUGUI text;
    readonly Queue<string> lines = new Queue<string>();
    [Range(1,100)] public int maxLines = 100;

    void Awake()
    {
        if (inst != null) { Destroy(gameObject); return; }
        inst = this;
        DontDestroyOnLoad(gameObject);

        // Canvas
        var canvasGO = new GameObject("ScreenLogCanvas");
        var canvas = canvasGO.AddComponent<Canvas>();
        canvas.renderMode = RenderMode.ScreenSpaceOverlay;
        canvas.sortingOrder = 9999;

        // Text
        var tgo = new GameObject("Log");
        tgo.transform.SetParent(canvasGO.transform);
        var rt = tgo.AddComponent<RectTransform>();
        rt.anchorMin = new Vector2(0, 0);
        rt.anchorMax = new Vector2(1, 0);
        rt.pivot = new Vector2(0.5f, 0);
        rt.offsetMin = new Vector2(10, 10);
        rt.offsetMax = new Vector2(-10, 210);

        text = tgo.AddComponent<TextMeshProUGUI>();
        text.fontSize = 18;
        text.textWrappingMode = TextWrappingModes.NoWrap;

        Application.logMessageReceived += HandleLog;
    }

    void OnDestroy() { Application.logMessageReceived -= HandleLog; }

    void HandleLog(string msg, string stack, LogType type)
    {
        string prefix = type == LogType.Error || type == LogType.Exception ? "[ERR]" :
                        type == LogType.Warning ? "[WARN]" : "[LOG]";
        lines.Enqueue($"{System.DateTime.Now:HH:mm:ss} {prefix} {msg}");
        while (lines.Count > maxLines) lines.Dequeue();
        if (text != null) text.text = string.Join("\n", lines);
    }
```

```
}
```

RogueShooter – All Scripts

Assets/scripts/Enemy/EnemyAI.cs

```csharp
using System;
using System.Collections;
using UnityEngine;

public class EnemyAI : MonoBehaviour
{
    public static EnemyAI Instance { get; private set; }
    private float timer;

    void Awake()
    {
        if (Instance != null && Instance != this) { Destroy(gameObject); return; }
        Instance = this;
        DontDestroyOnLoad(gameObject); // valinnainen
    }

    private void Start()
    {
        if (GameModeManager.SelectedMode == GameMode.SinglePlayer)
        {
            TurnSystem.Instance.OnTurnChanged += TurnSystem_OnTurnChanged;
        }

    }
    private void Update()
    {
        // Älä tee mitään co-opissa
        if (GameModeManager.SelectedMode != GameMode.SinglePlayer) return;

        if (TurnSystem.Instance.IsPlayerTurn())
        {
            return;
        }

        timer -= Time.deltaTime;
        if (timer <= 0f)
        {
            TurnSystem.Instance.NextTurn();
        }
    }

    private void TurnSystem_OnTurnChanged(object sender, EventArgs e)
    {
        timer = 2f;
    }

    // UUSI: AI-vuoro koroutiinina (ei NextTurn-kutsua sisällä!)
    [Mirror.Server]
    public IEnumerator RunEnemyTurnCoroutine()
    {
```

```
        Debug.Log("[AI] Enemy turn started");
        yield return new WaitForSeconds(2f);
        Debug.Log("[AI] Enemy turn finished");
    }

}
```

RogueShooter – All Scripts

Assets/scripts/FieldCleaner.cs

```
using System.Linq;
using Mirror;
using UnityEngine;

public class FieldCleaner : MonoBehaviour
{
    public static void ClearAll()
    {

        // Find all friendly and enemy units (also inactive, just in case)
        var friendlies = Resources.FindObjectsOfTypeAll<FriendlyUnit>()
                            .Where(u => u != null && u.gameObject.scene.IsValid());
        var enemies    = Resources.FindObjectsOfTypeAll<EnemyUnit>()
                            .Where(u => u != null && u.gameObject.scene.IsValid());

        foreach (var u in friendlies) Despawn(u.gameObject);
        foreach (var e in enemies)    Despawn(e.gameObject);
    }

    static void Despawn(GameObject go)
    {
        // if server is active, use Mirror's destroy; otherwise normal Unity Destroy
        if (NetworkServer.active)
        {
            NetworkServer.Destroy(go);
        }
        else
        {
            Object.Destroy(go);
        }
    }
}
```

RogueShooter – All Scripts

Assets/scripts/GameLogic/BattleLogic/TurnSystem.cs

```
using System;
using UnityEngine;

public class TurnSystem : MonoBehaviour
{
    public static TurnSystem Instance { get; private set; }

    public event EventHandler OnTurnChanged;
    private int turnNumber = 1;
    private bool isPlayerTurn = true;

    private void Awake()
    {

        // Ensure that there is only one instance in the scene
        if (Instance != null)
        {
            Debug.LogError(" More than one TurnSystem in the scene!" + transform + " " + Instance);
            Destroy(gameObject);
            return;
        }
        Instance = this;
    }

    public void NextTurn()
    {
        // Tarkista pelimoodi
        if (GameModeManager.SelectedMode == GameMode.SinglePlayer)
        {
            Debug.Log("Single Player mode: Proceeding to the next turn.");
            turnNumber++;
            isPlayerTurn = !isPlayerTurn;

            OnTurnChanged?.Invoke(this, EventArgs.Empty);
        }
        else if (GameModeManager.SelectedMode == GameMode.CoOp)
        {
            Debug.Log("Co-Op mode: Proceeding to the next turn.");
            // Tee jotain erityistä CoOp-tilassa
        }
        else if (GameModeManager.SelectedMode == GameMode.Versus)
        {
            Debug.Log("Versus mode: Proceeding to the next turn.");
            // Tee jotain erityistä Versus-tilassa
        }


    }

    public int GetTurnNumber()
```

```
    {
        return turnNumber;
    }

    public bool IsPlayerTurn()
    {
        return isPlayerTurn;
    }

    public void ForcePhase(bool isPlayerTurn, bool incrementTurnNumber)
    {
        if (incrementTurnNumber) turnNumber++;
        this.isPlayerTurn = isPlayerTurn;
        OnTurnChanged?.Invoke(this, EventArgs.Empty);
    }

}
```

RogueShooter – All Scripts

Assets/scripts/GameLogic/CameraController.cs

```
using UnityEngine;
using Unity.Cinemachine;

// <summary>
// This script controls the camera movement, rotation, and zoom in a Unity game using the Cinemachine package.
// It allows the player to move the camera using WASD keys, rotate it using Q and E keys, and zoom in and out using the mouse scroll wheel.
// The camera follows a target object with a specified offset, and the zoom level is clamped to a minimum and maximum value.
// </summary>
public class CameraController : MonoBehaviour
{
    private const float MIN_FOLLOW_Y_OFFSET = 2f;
    private const float MAX_FOLLOW_Y_OFFSET = 12f;
    [SerializeField] private CinemachineCamera cinemachineCamera;

    private CinemachineFollow cinemachineFollow;
    private Vector3 targetFollowOffset;

    private float moveSpeed = 10f;
    private float rotationSpeed = 100f;
    private float zoomSpeed = 5f;

    private void Start()
    {
        cinemachineFollow = cinemachineCamera.GetComponent<CinemachineFollow>();
        targetFollowOffset = cinemachineFollow.FollowOffset;
    }

    private void Update()
    {
        HandleMovement(moveSpeed);
        HandleRotation(rotationSpeed);
        HandleZoom(zoomSpeed);
    }

    private void HandleMovement(float moveSpeed)
    {
        Vector3 inputMoveDirection = new Vector3(0,0,0);
        if (Input.GetKey(KeyCode.W))
        {
            inputMoveDirection.z = +1f;
        }
        if (Input.GetKey(KeyCode.S))
        {
            inputMoveDirection.z = -1f;
        }
        if (Input.GetKey(KeyCode.A))
        {
            inputMoveDirection.x = -1f;
        }
        if (Input.GetKey(KeyCode.D))
```

```
        {
            inputMoveDirection.x = +1f;
        }

        Vector3 moveVector = transform.forward * inputMoveDirection.z + transform.right * inputMoveDirection.x;
        transform.position += moveSpeed * Time.deltaTime * moveVector;
    }

    private void HandleRotation(float rotationSpeed)
    {
        Vector3 rotationVector = new Vector3(0, 0, 0);
        if (Input.GetKey(KeyCode.Q))
        {
            rotationVector.y = -1f;
        }
        if (Input.GetKey(KeyCode.E))
        {
            rotationVector.y = +1f;
        }

        transform.eulerAngles += rotationSpeed * Time.deltaTime * rotationVector;
    }

    private void HandleZoom(float zoomSpeed)
    {
        float zoomAmount = 1f;
        if(Input.mouseScrollDelta.y > 0)
        {
            targetFollowOffset.y -= zoomAmount;
        }
        if(Input.mouseScrollDelta.y < 0)
        {
            targetFollowOffset.y += zoomAmount;
        }

        targetFollowOffset.y = Mathf.Clamp(targetFollowOffset.y, MIN_FOLLOW_Y_OFFSET, MAX_FOLLOW_Y_OFFSET);
        cinemachineFollow.FollowOffset = Vector3.Lerp(cinemachineFollow.FollowOffset, targetFollowOffset, Time.deltaTime * zoomSpeed);
    }

}
```

RogueShooter – All Scripts

Assets/scripts/GameLogic/MouseWorld.cs

```csharp
using UnityEngine;

/// <summary>
/// This class is responsible for handling mouse interactions in the game world.
/// It provides a method to get the mouse position in the world space based on the camera's perspective.
/// </summary>

public class MouseWorld : MonoBehaviour
{
    private static MouseWorld instance;
    [SerializeField] private LayerMask mousePlaneLayerMask;

    private void Awake()
    {
        instance = this;
    }

    public static Vector3 GetMouseWorldPosition()
    {
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        Physics.Raycast(ray, out RaycastHit raycastHit, float.MaxValue, instance.mousePlaneLayerMask);
        return raycastHit.point;
    }

}
```

Assets/scripts/GameModes/GameModeManager.cs

```
using Mirror;
using UnityEngine;

/// <summary>
/// This class is responsible for managing the game mode and spawning units in the game.
/// It checks if the game is being played online or offline and spawns units accordingly.
/// </summary>
public enum GameMode { SinglePlayer, CoOp, Versus }
public class GameModeManager : MonoBehaviour
{
    public static GameMode SelectedMode { get; private set; } = GameMode.SinglePlayer;

    public static void SetSinglePlayer() => SelectedMode = GameMode.SinglePlayer;
    public static void SetCoOp()         => SelectedMode = GameMode.CoOp;
    public static void SetVersus()       => SelectedMode = GameMode.Versus;


    [Header("Prefabs (all must have NetworkIdentity if used online)")]
    public GameObject playerPrefab;

    public GameObject coOpPlayerPrefab;

    public GameObject enemyPrefab;

    void Start()
    {

        // OFFLINE: Singleplayer -> instanssoi heti
        if (!NetworkClient.isConnected && !NetworkServer.active)
        {
            if (SelectedMode == GameMode.SinglePlayer)
            {
                Debug.Log("Game is offline, spawning singleplayer units.");
                SinglePlayerGameMode();
                return;
            }

            // Jos tänne päädytään ja tila on CoOp/Versus, se tarkoittaa
            // että UI ei vielä käynnistänyt verkkoa. Ei spawnailla mitään.
            Debug.Log("Selected online mode without network active. Waiting for host/client.");
            return;
        }

        // ONLINE: Server hoitaa spawnaamisen, clientit eivät tee mitään Startissa
        if (NetworkServer.active)
        {
            Debug.Log($"Server online in {SelectedMode} mode. Server will spawn enemies.");
            // Pelaaja-objektit tulevat NetworkManager.OnServerAddPlayerissa.
            ServerSpawnInitialForSelectedMode();
        }
```

```
    }

    // ==== UI-KUTSUT ====
    public void OnClickSinglePlayer()
    {
        SelectedMode = GameMode.SinglePlayer;
        // Lataa peliscene jos et jo siellä, tai anna tämän skriptin Start() hoitaa
        // SceneManager.LoadScene("Game");
    }
    public void OnClickCoOp()
    {
        SelectedMode = GameMode.CoOp;
        Debug.Log("SelectedMode set to CoOp. Now start host or client.");
        // Tässä voit näyttää ConnectCanvasin, tai käynnistää suoraan host/clientin:
        // NetworkManager.singleton.StartHost(); // host napille
        // NetworkManager.singleton.StartClient(); // client napille
    }

    public void OnClickVersus()
    {
        SelectedMode = GameMode.Versus;
        // Samoin kuin CoOp
    }

    // ==== OFFLINE (Singleplayer) ====
    void SinglePlayerGameMode()
    {
        SpawnPlayer1UnitsOffline();
        SpawnEnemyUnitsOffline();
    }

    void SpawnPlayer1UnitsOffline()
    {
        Instantiate(playerPrefab, new Vector3(0, 0, 0), Quaternion.identity);
        Instantiate(playerPrefab, new Vector3(2, 0, 0), Quaternion.identity);
    }

    void SpawnEnemyUnitsOffline()
    {
        Instantiate(enemyPrefab, new Vector3(4, 0, 8), Quaternion.identity);
        Instantiate(enemyPrefab, new Vector3(6, 0, 8), Quaternion.identity);
    }

    // <summary>
    /// This method is responsible for spawning units in the game.
    /// In online the host will spawn this units.
    /// </summary>
    private void SpawnPlayer1Units()
    {
        // Create units in the scene
        // Instantiate the player prefab at the specified position and rotation
        Instantiate(playerPrefab, new Vector3(0, 0, 0), Quaternion.identity);
```

```
        Instantiate(playerPrefab, new Vector3(2, 0, 0), Quaternion.identity);
    }

    private void SpawnPlayer2Units()
    {
        // Create units in the scene
        // Instantiate the player prefab at the specified position and rotation
        Instantiate(coOpPlayerPrefab, new Vector3(4, 0, 0), Quaternion.identity);

        Instantiate(coOpPlayerPrefab, new Vector3(6, 0, 0), Quaternion.identity);
    }
    /// <summary>
    /// This method is responsible for spawning enemy units in the game.
    /// In online the client will spawn this units.
    /// </summary>
    private void SpawnEnemyUnits()
    {
        // Create enemy units in the scene
        // Instantiate the enemy prefab at the specified position and rotation
        Instantiate(enemyPrefab, new Vector3(4, 0, 6), Quaternion.identity);

        Instantiate(enemyPrefab, new Vector3(6, 0, 6), Quaternion.identity);
    }

    // ==== ONLINE (Server only) ====
    [Server]
    void ServerSpawnInitialForSelectedMode()
    {
        switch (SelectedMode)
        {
            case GameMode.CoOp:
                ServerSpawnEnemies();
                break;
            case GameMode.Versus:
                // Versuksessa ei ehkä ole AI-vihollisia -> jätä tyhjäksi tai spawnaa neutraaleja
                break;
            case GameMode.SinglePlayer:
                // Ei pitäisi tapahtua verkossa, mutta varmistetaan
                ServerSpawnEnemies();
                break;
        }
    }

    [Server]
    void ServerSpawnEnemies()
    {
        var e1 = Instantiate(enemyPrefab, new Vector3(4, 0, 6), Quaternion.identity);
        NetworkServer.Spawn(e1);
        var e2 = Instantiate(enemyPrefab, new Vector3(6, 0, 6), Quaternion.identity);
        NetworkServer.Spawn(e2);
    }
```

```
}
```

RogueShooter – All Scripts

Assets/scripts/GithubTestingEmptyScript.cs

```csharp
using UnityEngine;

public class GithubTestingEmptyScript : MonoBehaviour
{
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

RogueShooter – All Scripts

Assets/scripts/Grid/GridDebugObject.cs

```csharp
using UnityEngine;
using TMPro;

// <summary>
// This script is used to display the grid object information in the scene view.
// </summary>

public class GridDebugObject : MonoBehaviour
{
    [SerializeField] private TextMeshPro textMeshPro;
    private GridObject gridObject;
    public void SetGridObject(GridObject gridObject)
    {
        this.gridObject = gridObject;
    }
    private void Update()
    {
        textMeshPro.text = gridObject.ToString();
    }

}
```

Assets/scripts/Grid/GridObject.cs

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

// <summary>
// This class represents a grid object in the grid system.
// It contains a list of units that are present in the grid position.
// It also contains a reference to the grid system and the grid position.
// </summary>

public class GridObject
{
    private GridSystem gridSystem;
    private GridPosition gridPosition;

    private List<Unit> unitList;

    public GridObject(GridSystem gridSystem, GridPosition gridPosition)
    {
        this.gridSystem = gridSystem;
        this.gridPosition = gridPosition;
        unitList = new List<Unit>();
    }

    public override string ToString()
    {
        string unitListString = "";
        foreach (Unit unit in unitList)
        {
            unitListString += unit + "\n";
        }
        return gridPosition.ToString() + "\n" + unitListString;
    }

    public void AddUnit(Unit unit)
    {
        unitList.Add(unit);
    }

    public void RemoveUnit(Unit unit)
    {
        unitList.Remove(unit);
    }

    public List<Unit> GetUnitList()
    {
        return unitList;
    }

    public bool HasAnyUnit()
    {
```

```
        return unitList.Count > 0;
    }

    public Unit GetUnit()
    {
        if (HasAnyUnit())
        {
            return unitList[0];
        } else
        {
            return null;
        }
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Grid/GridPosition.cs

```
using System;
using NUnit.Framework;

// <summary>
// This struct represents a position in a grid system.
// It contains two integer values, x and z, which represent the coordinates of the position in the grid.
// It also contains methods for comparing two GridPosition objects, adding and subtracting them, and converting them to a string representation.
// </summary>

public struct GridPosition:IEquatable<GridPosition>
{
    public int x;
    public int z;

    public GridPosition(int x, int z)
    {
        this.x = x;
        this.z = z;
    }

    public override bool Equals(object obj)
    {
        return obj is GridPosition position &&
        x == position.x
        && z == position.z;
    }

    public bool Equals(GridPosition other)
    {
        return this == other;
    }

    public override int GetHashCode()
    {
        return HashCode.Combine(x, z);
    }

    public override string ToString()
    {
        return $"({x}, {z})";
    }

    public static bool operator ==(GridPosition a, GridPosition b)
    {
        return a.x == b.x && a.z == b.z;
    }

    public static bool operator !=(GridPosition a, GridPosition b)
    {
        return !(a == b);
```

```
    }

    public static GridPosition operator +(GridPosition a, GridPosition b)
    {
        return new GridPosition(a.x + b.x, a.z + b.z);
    }

    public static GridPosition operator -(GridPosition a, GridPosition b)
    {
        return new GridPosition(a.x - b.x, a.z - b.z);
    }

}
```

Assets/scripts/Grid/GridSystem.cs

```csharp
using UnityEngine;

/// <summary>
/// This class represents a grid system in a 2D space.
/// It contains methods to create a grid, convert between grid and world coordinates,
/// and manage grid objects.
/// </summary>

public class GridSystem
{
    private int width;
    private int height;
    private float cellSize;

    private GridObject[,] gridObjectsArray;
    public GridSystem(int width, int height, float cellSize)
    {
        this.width = width;
        this.height = height;
        this.cellSize = cellSize;

        gridObjectsArray = new GridObject[width, height];

        for (int x = 0; x< width; x++)
        {
            for(int z = 0; z < height; z++)
            {
                GridPosition gridPosition = new GridPosition(x, z);
                gridObjectsArray[x, z] = new GridObject(this, gridPosition);
            }
        }
    }

/// Purpose: This method converts grid coordinates (x, z) to world coordinates.
/// It multiplies the grid coordinates by the cell size to get the world position.
    public Vector3 GetWorldPosition(GridPosition gridPosition)
    {
        return new Vector3(gridPosition.x, 0, gridPosition.z )* cellSize;
    }

/// Purpose: This is used to find the grid position of a unit in the grid system.
/// It is used to check if the unit is within the bounds of the grid system.
/// It converts the world position to grid coordinates by dividing the world position by the cell size.
    public GridPosition GetGridPosition(Vector3 worldPosition)
    {
        return new GridPosition( Mathf.RoundToInt(worldPosition.x/cellSize), Mathf.RoundToInt(worldPosition.z/cellSize));
    }

/// Purpose: This method creates debug objects in the grid system for visualization purposes.
/// It instantiates a prefab at each grid position and sets the grid object for that position.
```

```
    public void CreateDebugObjects(Transform debugPrefab)
    {
        for (int x = 0; x< width; x++)
        {
            for(int z = 0; z < height; z++)
            {
                GridPosition gridPosition = new GridPosition(x, z);
                Transform debugTransform = GameObject.Instantiate(debugPrefab, GetWorldPosition(gridPosition), Quaternion.identity);
                GridDebugObject gridDebugObject = debugTransform.GetComponent<GridDebugObject>();
                gridDebugObject.SetGridObject(GetGridObject(gridPosition));
            }
        }
    }

/// Purpose: This method returns the grid object at a specific grid position.
/// It is used to get the grid object for a specific position in the grid system.
    public GridObject GetGridObject(GridPosition gridPosition)
    {
        return gridObjectsArray[gridPosition.x, gridPosition.z];
    }

/// Purpose: This method checks if a grid position is valid within the grid system.
/// It checks if the x and z coordinates are within the bounds of the grid width and height.
    public bool IsValidGridPosition(GridPosition gridPosition)
    {
        return  gridPosition.x >= 0 &&
                gridPosition.x < width &&
                gridPosition.z >= 0 &&
                gridPosition.z < height;
    }

    public int GetWidth()
    {
        return width;
    }
    public int GetHeight()
    {
        return height;
    }

}
```

Assets/scripts/Grid/GridSystemVisual.cs

```
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

/// <summary>
/// This class is responsible for visualizing the grid system in the game.
/// It creates a grid of visual objects that represent the grid positions.
/// </summary>

public class GridSystemVisual : MonoBehaviour
{

    public static GridSystemVisual Instance { get; private set; }

    /// Purpose: This prefab is used to create the visual representation of each grid position.
    [SerializeField] private Transform gridSystemVisualSinglePrefab;

    /// Purpose: This array holds the visual objects for each grid position.
    private GridSystemVisualSingle [,] gridSystemVisualSingleArray;

    private void Awake()
    {

        ///  Purpose: Ensure that there is only one instance in the scene
        if (Instance != null)
        {
            Debug.LogError("More than one GridSystemVisual in the scene!" + transform + " " + Instance);
            Destroy(gameObject);
            return;
        }

        Instance  = this;
    }

    private void Start()
    {
        gridSystemVisualSingleArray = new GridSystemVisualSingle[LevelGrid.Instance.GetWidth(), LevelGrid.Instance.GetHeight()];

        /// Purpose: Create a grid of visual objects that represent the grid positions.
        /// It instantiates a prefab at each grid position and sets the grid object for that position.
        for (int x = 0 ;x < LevelGrid.Instance.GetWidth(); x++)
        {
            for (int z = 0; z < LevelGrid.Instance.GetHeight(); z++)
            {
                GridPosition gridPosition = new(x, z);
                Transform gridSystemVisualSingleTransform = Instantiate(gridSystemVisualSinglePrefab, LevelGrid.Instance.GetWorldPosition(gridPosition), Quaternion.identity);

                gridSystemVisualSingleArray[x, z] = gridSystemVisualSingleTransform.GetComponent<GridSystemVisualSingle>();
            }
        }
```

```
    }

    private void Update()
    {
        UpdateGridVisuals();
    }

    public void HideAllGridPositions()
    {
        for (int x = 0 ;x < LevelGrid.Instance.GetWidth(); x++)
        {
            for (int z = 0; z < LevelGrid.Instance.GetHeight(); z++)
            {
                gridSystemVisualSingleArray[x, z].Hide();
            }
        }
    }

    public void ShowGridPositionList(List< GridPosition> gridPositionList)
    {
        HideAllGridPositions();
        foreach (GridPosition gridPosition in gridPositionList)
        {
            gridSystemVisualSingleArray[gridPosition.x, gridPosition.z].Show();
        }
    }

    private void UpdateGridVisuals()
    {
        HideAllGridPositions();
        Unit selectedUnit = UnitActionSystem.Instance.GetSelectedUnit();
        if (selectedUnit == null) return;

        BaseAction selectedAction = UnitActionSystem.Instance.GetSelectedAction();
        ShowGridPositionList(
            selectedAction.GetValidGridPositionList());

    }
}
```

RogueShooter – All Scripts

Assets/scripts/Grid/GridSystemVisualSingle.cs

```csharp
using UnityEngine;

/// <summary>
/// This class is responsible for visualizing a single grid position in the game.
/// It contains a MeshRenderer component that is used to show or hide the visual representation of the grid position.
/// </summary>
public class GridSystemVisualSingle : MonoBehaviour
{
    [SerializeField] private MeshRenderer meshRenderer;

    public void Show()
    {
        meshRenderer.enabled = true;
    }
    public void Hide()
    {
        meshRenderer.enabled = false;
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Grid/LevelGrid.cs

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

/// <summary>
/// This class is responsible for managing the grid system in the game.
/// It creates a grid of grid objects and provides methods to interact with the grid.
/// </summary>

public class LevelGrid : MonoBehaviour
{
    public static LevelGrid Instance { get; private set; }
    [SerializeField] private Transform debugPrefab;
    private GridSystem gridSystem;
    private void Awake()
    {

        // Ensure that there is only one instance in the scene
        if (Instance != null)
        {
            Debug.LogError("LevelGrid: More than one LevelGrid in the scene!" + transform + " " + Instance);
            Destroy(gameObject);
            return;
        }
        Instance = this;

        gridSystem = new GridSystem(10, 10, 2f);
        gridSystem.CreateDebugObjects(debugPrefab);
    }

    public void AddUnitAtGridPosition(GridPosition gridPosition, Unit unit)
    {
        GridObject gridObject = gridSystem.GetGridObject(gridPosition);
        gridObject.AddUnit(unit);
    }

    public List<Unit> GetUnitListAtGridPosition(GridPosition gridPosition)
    {
        GridObject gridObject = gridSystem.GetGridObject(gridPosition);
        if (gridObject != null)
        {
            return gridObject.GetUnitList();
        }
        return null;
    }

    public void RemoveUnitAtGridPosition(GridPosition gridPosition, Unit unit)
    {
        GridObject gridObject = gridSystem.GetGridObject(gridPosition);
        gridObject.RemoveUnit(unit);
    }
```

```
public void UnitMoveToGridPosition(GridPosition fromGridPosition, GridPosition toGridPosition, Unit unit)
{
    RemoveUnitAtGridPosition(fromGridPosition, unit);
    AddUnitAtGridPosition(toGridPosition, unit);
}

public GridPosition GetGridPosition(Vector3 worldPosition)
{
    return gridSystem.GetGridPosition(worldPosition);
}

public Vector3 GetWorldPosition(GridPosition gridPosition)
{
    return gridSystem.GetWorldPosition(gridPosition);
}

public bool IsValidGridPosition(GridPosition gridPosition)
{
    return gridSystem.IsValidGridPosition(gridPosition);
}

public int GetWidth()
{
    return gridSystem.GetWidth();
}
public int GetHeight()
{
    return gridSystem.GetHeight();
}

public bool HasAnyUnitOnGridPosition(GridPosition gridPosition)
{
    GridObject gridObject = gridSystem.GetGridObject(gridPosition);
    return gridObject.HasAnyUnit();
}

public Unit GetUnitAtGridPosition(GridPosition gridPosition)
{
    GridObject gridObject = gridSystem.GetGridObject(gridPosition);
    return gridObject.GetUnit();
}

public void ClearAllOccupancy()
{
    for (int x = 0; x < gridSystem.GetWidth(); x++)
    {
        for (int z = 0; z < gridSystem.GetHeight(); z++)
        {
            var gridPosition = new GridPosition(x, z);
            var gridObject = gridSystem.GetGridObject(gridPosition);
            var list = gridObject.GetUnitList();
```

```
                list?.Clear();
            }
        }
    }

    public void RebuildOccupancyFromScene()
    {
        ClearAllOccupancy();
        var units = Object.FindObjectsByType<Unit>(FindObjectsSortMode.None);
        foreach (var u in units)
        {
            var gp = GetGridPosition(u.transform.position);
            AddUnitAtGridPosition(gp, u);
        }
    }

}
```

RogueShooter – All Scripts

Assets/scripts/Helpers/AuthorityHelper.cs

```
using Mirror;

public static class AuthorityHelper
{
    /// <summary>
    /// Checks if the given NetworkBehaviour has local control.
    /// Prevents the player from controlling the object if they are not the owner.
    /// </summary>
    public static bool HasLocalControl(NetworkBehaviour netBehaviour)
    {
        return NetworkClient.isConnected && !netBehaviour.isOwned;
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Menu/BackButtonUI.cs

```
using UnityEngine;
using UnityEngine.UI;

public class BackButtonUI : MonoBehaviour
{

    // Serialized fields
    [Header("Canvas References")]
    [SerializeField] private GameObject connectCanvas; // this (self)
    [SerializeField] private GameObject gameModeSelectCanvas; // Hiden on start

    [Header("Buttons")]
    [SerializeField] private Button backButton;

    private void Awake()
    {

        // Add button listener
        backButton.onClick.AddListener(BackButton_OnClick);
    }

    private void BackButton_OnClick()
    {
        Debug.Log("Back button clicked.");
        // Hide the connect canvas and show the game mode select canvas
        connectCanvas.SetActive(false);
        gameModeSelectCanvas.SetActive(true);
    }

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

RogueShooter – All Scripts

Assets/scripts/Menu/GameModeSelectUI.cs

```
using UnityEngine;
using UnityEngine.UI;

public class GameModeSelectUI : MonoBehaviour
{
    // Serialized fields
    [Header("Canvas References")]
    [SerializeField] private GameObject gameModeSelectCanvas; // this (self)
    [SerializeField] private GameObject connectCanvas;        // Hiden on start

    // UI Elements
    [Header("Buttons")]
    [SerializeField] private Button coopButton;
    [SerializeField] private Button pvpButton;

    private void Awake()
    {
        // Ensure the game mode select canvas is active and connect canvas is inactive at start
        gameModeSelectCanvas.SetActive(true);
        connectCanvas.SetActive(false);

        // Add button listeners
        // coopButton.onClick.AddListener(() => OnModeSelected("Co-op"));
        // pvpButton.onClick.AddListener(() => OnModeSelected("PvP"));
        coopButton.onClick.AddListener(OnClickCoOp);
        pvpButton.onClick.AddListener(OnClickPvP);
    }

    private void OnModeSelected(string mode)
    {
        // Clear the field of existing units
        FieldCleaner.ClearAll();
        // UnitActionSystem.Instance?.SetSelectedUnit(null);
        StartCoroutine(ResetGridNextFrame());

        Debug.Log($"{mode} mode selected.");
        // Hide the game mode select canvas and show the connect canvas
        gameModeSelectCanvas.SetActive(false);
        connectCanvas.SetActive(true);
        // Additional logic for handling mode selection can be added here

        // Set the selected game mode in GameModeManager
        if (mode == "Co-op")
        {
            GameModeManager.SetCoOp();
        }
        else
        {
            GameModeManager.SetVersus();
        }
```

```
    }

    public void OnClickCoOp()
    {
        GameModeManager.SetCoOp();
        OnSelected();
    }

    public void OnClickPvP()
    {
        GameModeManager.SetVersus();
        OnSelected();
    }

    public void OnSelected()
    {
        FieldCleaner.ClearAll();
        StartCoroutine(ResetGridNextFrame());
        gameModeSelectCanvas.SetActive(false);
        connectCanvas.SetActive(true);
    }

    private System.Collections.IEnumerator ResetGridNextFrame()
    {
        yield return new WaitForEndOfFrame();
        var lg = LevelGrid.Instance;
        if (lg != null) lg.RebuildOccupancyFromScene();
    }

}
```

RogueShooter – All Scripts

Assets/scripts/Oneline/Authentication.cs

```csharp
using System;
using Unity.Services.Authentication;
using Unity.Services.Core;
using UnityEngine;

/// <summary>
/// This class is responsible for handling the authentication process using Unity Services.
/// It initializes the Unity Services and signs in the user anonymously.
/// </summary>

public class Authentication : MonoBehaviour
{

    async void Start()
    {
        try
        {
            await UnityServices.InitializeAsync();
            await AuthenticationService.Instance.SignInAnonymouslyAsync();
            Debug.Log("Logged into Unity, player ID: " + AuthenticationService.Instance.PlayerId);
        }
        catch (Exception e)
        {
            Debug.LogError(e);
        }
    }

}
```

RogueShooter – All Scripts

Assets/scripts/Oneline/Connect.cs

```
using UnityEngine;
using TMPro;
using Mirror;
using Utp;

/// <summary>
/// This class is responsible for connecting to the Unity Relay service.
/// It provides methods to host a game and join a game as a client.
/// </summary>
public class Connect : MonoBehaviour
{
    [SerializeField] private GameNetworkManager nm; // vedä tämä Inspectorissa
    [SerializeField] private TMP_InputField ipField;

    void Awake()
    {
        // find the NetworkManager in the scene if not set in Inspector
        if (!nm) nm = NetworkManager.singleton as GameNetworkManager;
        if (!nm) nm = FindFirstObjectByType<GameNetworkManager>();
        if (!nm) Debug.LogError("[Connect] GameNetworkManager not found in scene.");
    }

    // HOST (LAN): ei Relaytä
    public void HostLAN()
    {
        nm.StartStandardHost(); // tämä asettaa useRelay=false ja käynnistää hostin
    }

    // CLIENT (LAN): ei Relaytä
    public void ClientLAN()
    {
        // Jos syötekenttä puuttuu/tyhjä → oletus localhost (sama kone)
        string ip = (ipField != null && !string.IsNullOrWhiteSpace(ipField.text))
                    ? ipField.text.Trim()
                    : "localhost"; // tai 127.0.0.1

        nm.networkAddress = ip;   // <<< TÄRKEIN KOHTA
        nm.JoinStandardServer();  // useRelay=false ja StartClient()
    }

    public void Host()
    {
        if (!nm)
        {
            Debug.LogError("[Connect] GameNetworkManager not found in scene.");
            return;
        }

        nm.StartRelayHost(2, null);
    }
```

```
    public void Client ()
    {
        if (!nm)
        {
            Debug.LogError("[Connect] GameNetworkManager not found in scene.");
            return;
        }

        nm.JoinRelayServer();
    }

}
```

RogueShooter – All Scripts

Assets/scripts/Oneline/CoopTurnCoordinator.cs

```csharp
using System.Collections;
using System.Collections.Generic;
using Mirror;
using UnityEngine;

public enum TurnPhase { Players, Enemy }

public class CoopTurnCoordinator : NetworkBehaviour
{
    public static CoopTurnCoordinator Instance { get; private set; }

    [SyncVar] public TurnPhase phase = TurnPhase.Players;
    [SyncVar] public int turnNumber = 1;
    [SyncVar] public int endedCount = 0;
    [SyncVar] public int requiredCount = 0; // päivitetään kun pelaajia liittyy/lähtee

    // Server only: ketkä ovat jo painaneet End Turn tässä kierrossa
    private readonly HashSet<uint> endedPlayers = new HashSet<uint>();

    void Awake()
    {
        if (Instance != null && Instance != this) { Destroy(gameObject); return; }
        Instance = this;
    }

    public override void OnStartServer()
    {
        base.OnStartServer();
        ResetTurnState();
        // jos haluat lukita kahteen pelaajaan protoa varten:
        if (GameModeManager.SelectedMode == GameMode.CoOp) requiredCount = 2;
        Debug.Log($"[TURN][SERVER] Start, requiredCount={requiredCount}");

    }

    [Server]
    public void ServerUpdateRequiredCount(int playersNow)
    {
        requiredCount = Mathf.Max(1, playersNow); // Co-opissa yleensä 2
        // jos yksi poistui kesken odotuksen, tarkista täyttyikö ehto nyt
        TryAdvanceIfReady();
    }

    [Server]
    public void ServerPlayerEndedTurn(uint playerNetId)
    {
        if (phase != TurnPhase.Players) return;         // ei lasketa jos ei pelaajavuoro
        if (!endedPlayers.Add(playerNetId)) return;     // älä laske tuplia

        endedCount = endedPlayers.Count;
```

```
        Debug.Log($"[TURN][SERVER] Player {playerNetId} ended. {endedCount}/{requiredCount}");
        RpcUpdateWaiting(endedCount, requiredCount);      // UI:lle "odotetaan X/Y"

        TryAdvanceIfReady();
    }

    [Server]
    void TryAdvanceIfReady()
    {
        if (phase == TurnPhase.Players && endedPlayers.Count >= Mathf.Max(1, requiredCount))
        {
            Debug.Log("[TURN][SERVER] All players ready → enemy turn");
            StartCoroutine(ServerEnemyTurnThenNextPlayers());
        }
    }

    [Server]
    private IEnumerator ServerEnemyTurnThenNextPlayers()
    {
        phase = TurnPhase.Enemy;
        RpcTurnPhaseChanged(phase, turnNumber);

        // Silta Unit-luokalle (AP-logiikka jne.)
        if (TurnSystem.Instance != null)
            TurnSystem.Instance.ForcePhase(isPlayerTurn: false, incrementTurnNumber: false);

        // Aja SP-AI uudelleenkäyttönä
        yield return RunEnemyAI();

        // Takaisin pelaajille + uusi turn-numero
        turnNumber++;
        ResetTurnState();
        if (TurnSystem.Instance != null)
            TurnSystem.Instance.ForcePhase(isPlayerTurn: true, incrementTurnNumber: false);

        RpcTurnPhaseChanged(phase, turnNumber);
    }

    [Server]
    IEnumerator RunEnemyAI()
    {
        if (EnemyAI.Instance != null)
            yield return EnemyAI.Instance.RunEnemyTurnCoroutine();
        else
            yield return new WaitForSeconds(2f); // fallback, ettei ketju katkea
    }

    [Server]
    void ResetTurnState()
    {
        Debug.Log("[TURN][SERVER] ResetTurnState");
        phase = TurnPhase.Players;
```

```
        endedPlayers.Clear();
        endedCount = 0;

        // nollaa kaikilta pelaajilta 'hasEndedThisTurn'
        foreach (var kvp in NetworkServer.connections)
        {
            var id = kvp.Value.identity;
            if (!id) continue;
            var pc = id.GetComponent<PlayerController>();
            if (pc) pc.ServerSetHasEnded(false);  // <<< TÄRKEIN RIVI
        }

        RpcUpdateWaiting(endedCount, requiredCount);
    }

    // ---- Client-notifikaatiot UI:lle ----
    [ClientRpc]
    void RpcTurnPhaseChanged(TurnPhase newPhase, int turnNo)
    {
        // Päivitä clientin paikallinen TurnSystem → laukaisee OnTurnChanged
        bool isPlayers = newPhase == TurnPhase.Players;
        if (TurnSystem.Instance != null)
            TurnSystem.Instance.ForcePhase(isPlayers, incrementTurnNumber: false);
    }

    [ClientRpc]
    void RpcUpdateWaiting(int have, int need)
    {
        // UI: "Waiting for teammate: have/need"
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Oneline/GameNetworkManager.cs

```
using System;
using System.Collections.Generic;

using Mirror;

using UnityEngine;
using Unity.Services.Relay.Models;

namespace Utp
{
 public class GameNetworkManager : NetworkManager
 {
  private bool enemiesSpawned;

  // --- Lisää luokan alkuun kentät ---
  [Header("Co-op squad prefabs")]
  public GameObject unitHostPrefab;      // -> UnitSolo
  public GameObject unitClientPrefab;    // -> UnitSolo Player 2

  [Header("Enemy spawn (Co-op)")]
  public GameObject enemyPrefab;

  [Header("Spawn positions (world coords on your grid)")]
  public Vector3[] hostSpawnPositions = {
   new Vector3(0, 0, 0),
   new Vector3(2, 0, 0),
  };
  public Vector3[] clientSpawnPositions = {
   new Vector3(0, 0, 6),
   new Vector3(2, 0, 6),
  };
  public Vector3[] enemySpawnPositions = {
   new Vector3(4, 0, 8),
   new Vector3(6, 0, 8),
  };
  // --- -------------------------------

  private UtpTransport utpTransport;

  /// <summary>
  /// Server's join code if using Relay.
  /// </summary>
  public string relayJoinCode = "";

  public override void Awake()
  {
   base.Awake();

   utpTransport = GetComponent<UtpTransport>();
```

```
  string[] args = System.Environment.GetCommandLineArgs();
  for (int key = 0; key < args.Length; key++)
  {
   if (args[key] == "-port")
   {
    if (key + 1 < args.Length)
    {
     string value = args[key + 1];

     try
     {
      utpTransport.Port = ushort.Parse(value);
     }
     catch
     {
      UtpLog.Warning($"Unable to parse {value} into transport Port");
     }
    }
   }
  }
}

public override void OnStartServer()
{
 base.OnStartServer();
 enemiesSpawned = false;
 Debug.Log("[NM] OnStartServer() called. Mode=" + GameModeManager.SelectedMode);

 if (GameModeManager.SelectedMode == GameMode.CoOp)
 {
  ServerSpawnEnemies();
 }
}

/// <summary>
/// Get the port the server is listening on.
/// </summary>
/// <returns>The port.</returns>
public ushort GetPort()
{
 return utpTransport.Port;
}

/// <summary>
/// Get whether Relay is enabled or not.
/// </summary>
/// <returns>True if enabled, false otherwise.</returns>
public bool IsRelayEnabled()
{
 return utpTransport.useRelay;
}
```

```
/// <summary>
/// Ensures Relay is disabled. Starts the server, listening for incoming connections.
/// </summary>
public void StartStandardServer()
{
 utpTransport.useRelay = false;
 StartServer();
}

/// <summary>
/// Ensures Relay is disabled. Starts a network "host" - a server and client in the same application
/// </summary>
public void StartStandardHost()
{
 if (utpTransport == null)
 {
  utpTransport = GetComponent<UtpTransport>();
  if (utpTransport == null)
  {
   Debug.LogError("[NM] UtpTransport puuttuu samalta GameObjectilta!");
   return;
  }
 }

 utpTransport.useRelay = false;
 StartHost();
}

/// <summary>
/// Gets available Relay regions.
/// </summary>
///
public void GetRelayRegions(Action<List<Region>> onSuccess, Action onFailure)
{
 utpTransport.GetRelayRegions(onSuccess, onFailure);
}

/// <summary>
/// Ensures Relay is enabled. Starts a network "host" - a server and client in the same application
/// </summary>
public void StartRelayHost(int maxPlayers, string regionId = null)
{
 if (utpTransport == null)
 {
  utpTransport = GetComponent<UtpTransport>();
  if (utpTransport == null)
  {
   Debug.LogError("[NM] UtpTransport puuttuu samalta GameObjectilta!");
   return;
  }
 }
```

```
 if (utpTransport == null)
 {
  utpTransport = GetComponent<UtpTransport>();
  if (utpTransport == null)
  {
   Debug.LogError("[NM] UtpTransport puuttuu samalta GameObjectilta!");
   return;
  }
 }

 utpTransport.useRelay = true;
 utpTransport.AllocateRelayServer(maxPlayers, regionId,
 (string joinCode) =>
 {
  relayJoinCode = joinCode;
  Debug.LogError($"Relay join code: {joinCode}");
  StartHost();
 },
 () =>
 {
  UtpLog.Error($"Failed to start a Relay host.");
 });
}

/// <summary>
/// Ensures Relay is disabled. Starts the client, connects it to the server with networkAddress.
/// </summary>
public void JoinStandardServer()
{
 utpTransport.useRelay = false;
 StartClient();
}

/// <summary>
/// Ensures Relay is enabled. Starts the client, connects to the server with the relayJoinCode.
/// </summary>
public void JoinRelayServer()
{
 utpTransport.useRelay = true;
 utpTransport.ConfigureClientWithJoinCode(relayJoinCode,
 () =>
 {
  StartClient();
 },
 () =>
 {
  UtpLog.Error($"Failed to join Relay server.");
 });
}

public override void OnValidate()
{
```

```
  base.OnValidate();
  if (utpTransport == null)
   utpTransport = GetComponent<UtpTransport>();
}

/// <summary>
/// Tämä metodi spawnaa jokaiselle clientille oman Unitin ja tekee siitä heidän ohjattavan yksikkönsä.
/// </summary>
public override void OnServerAddPlayer(NetworkConnectionToClient conn)
{
 /*
 Transform startPos = GetStartPosition(); // Spawn position (valinnainen)
 Vector3 spawnPosition = startPos != null ? startPos.position : Vector3.zero;

 GameObject player = Instantiate(playerPrefab, spawnPosition, Quaternion.identity);

 NetworkServer.AddPlayerForConnection(conn, player); // antaa authorityn clientille
 */

 // 1) Luo Mirrorin "player object" (EmptySquad) playerPrefab-slotista:
 // 1) luo player-object (EmptySquad)
 if (playerPrefab == null)
 {
  Debug.LogError("[NM] Player Prefab (EmptySquad) puuttuu!");
  return;
 }
 base.OnServerAddPlayer(conn);

 // 2) päätä host vs client
 bool isHost = conn.connectionId == 0;

 GameObject unitPrefab = isHost ? unitHostPrefab : unitClientPrefab;
 Vector3[] spawnPoints = isHost ? hostSpawnPositions : clientSpawnPositions;

 if (unitPrefab == null)
 {
  Debug.LogError($"[NM] {(isHost ? "unitHostPrefab" : "unitClientPrefab")} puuttuu!");
  return;
 }
 if (spawnPoints == null || spawnPoints.Length == 0)
 {
  Debug.LogError($"[NM] {(isHost ? "hostSpawnPositions" : "clientSpawnPositions")} ei ole asetettu!");
  return;
 }

 foreach (var pos in spawnPoints)
 {
  var go = Instantiate(unitPrefab, pos, Quaternion.identity);
  NetworkServer.Spawn(go, conn); // authority tälle pelaajalle
 }

 if (GameModeManager.SelectedMode == GameMode.CoOp && !enemiesSpawned)
```

```
   {
     Debug.Log("Spawning enemies for Co-op.");
     ServerSpawnEnemies();
   }
   var coord = CoopTurnCoordinator.Instance;
   if (coord != null)
     coord.ServerUpdateRequiredCount(NetworkServer.connections.Count);

}

[Server]
void ServerSpawnEnemies()
{
   Debug.Log("[NM] ServerSpawnEnemies() called.");
   if (!enemyPrefab || enemySpawnPositions == null || enemySpawnPositions.Length == 0)
   {
     Debug.LogWarning("[NM] EnemyPrefab/positions puuttuu");
     return;
   }

   foreach (var pos in enemySpawnPositions)
   {
     var e = Instantiate(enemyPrefab, pos, Quaternion.identity);
     NetworkServer.Spawn(e);
     Debug.Log("Enemy spawned at " + pos);
   }

   enemiesSpawned = true;
}

public override void OnServerDisconnect(NetworkConnectionToClient conn)
{
   base.OnServerDisconnect(conn);
   // päivitä pelaajamäärä koordinaattorille
   var coord = CoopTurnCoordinator.Instance;
   if (coord != null)
     coord.ServerUpdateRequiredCount(NetworkServer.connections.Count);
}
}
}
```

RogueShooter – All Scripts

Assets/scripts/PlayerUnits/EmptySquad.cs

```
using Mirror;
using UnityEngine;

/// <summary>
/// GameNetorkManager is required to have a NetworkManager component.
/// This is an empty class just to satisfy that requirement.
///
public class EmptySquad : MonoBehaviour
{

}
```

Assets/scripts/PlayerUnits/Unit.cs

```
using Mirror;
using System;
using UnityEngine;

/// <summary>
///     This class represents a unit in the game.
///     Actions can be called on the unit to perform various actions like moving or shooting.
///     The class inherits from NetworkBehaviour to support multiplayer functionality.
/// </summary>
public class Unit : NetworkBehaviour
{

    private const int ACTION_POINTS_MAX = 2;

    public static event EventHandler OnAnyActionPointsChanged;

    [SerializeField] private bool isEnemy;

    private GridPosition gridPosition;
    private MoveAction moveAction;
    private SpinAction spinAction;

    private BaseAction[] baseActionsArray;

    private int actionPoints = ACTION_POINTS_MAX;

    private void Awake()
    {
        moveAction = GetComponent<MoveAction>();
        spinAction = GetComponent<SpinAction>();
        baseActionsArray = GetComponents<BaseAction>();
    }

    private void Start()
    {
        if (LevelGrid.Instance != null)
        {
            gridPosition = LevelGrid.Instance.GetGridPosition(transform.position);
            LevelGrid.Instance.AddUnitAtGridPosition(gridPosition, this);
        }

        TurnSystem.Instance.OnTurnChanged += TurnSystem_OnTurnChanged;
    }

    private void Update()
    {

        GridPosition newGridPosition = LevelGrid.Instance.GetGridPosition(transform.position);
        if (newGridPosition != gridPosition)
        {
```

```
                LevelGrid.Instance.UnitMoveToGridPosition(gridPosition, newGridPosition, this);
                gridPosition = newGridPosition;
        }
    }

    public MoveAction GetMoveAction()
    {
        return moveAction;
    }

    public SpinAction GetSpinAction()
    {
        return spinAction;
    }

    public GridPosition GetGridPosition()
    {
        return gridPosition;
    }

    public Vector3 GetWorldPosition()
    {
        return transform.position;
    }

    public BaseAction[] GetBaseActionsArray()
    {
        return baseActionsArray;
    }

    public bool TrySpendActionPointsToTakeAction(BaseAction baseAction)
    {
        if (CanSpendActionPointsToTakeAction(baseAction))
        {
            SpendActionPoints(baseAction.GetActionPointsCost());
            return true;
        }
        return false;
    }

    public bool CanSpendActionPointsToTakeAction(BaseAction baseAction)
    {
        if (actionPoints >= baseAction.GetActionPointsCost())
        {
            // actionPoints -= baseAction.GetActionPointsCost();
            return true;
        }
        return false;
    }

    private void SpendActionPoints(int amount)
    {
```

```
        actionPoints -= amount;

        OnAnyActionPointsChanged?.Invoke(this, EventArgs.Empty);
    }

    public int GetActionPoints()
    {
        return actionPoints;
    }

    /// <summary>
    ///     This method is called when the turn changes. It resets the action points to the maximum value.
    /// </summary>
    private void TurnSystem_OnTurnChanged(object sender, EventArgs e)
    {
        if ((isEnemy && !TurnSystem.Instance.IsPlayerTurn()) ||
        (!isEnemy && TurnSystem.Instance.IsPlayerTurn()))
        {
            actionPoints = ACTION_POINTS_MAX;
            OnAnyActionPointsChanged?.Invoke(this, EventArgs.Empty);
        }
    }

    public bool IsEnemy()
    {
        return isEnemy;
    }

    public void Damage()
    {
        Debug.Log(transform + " took damage");
    }

    void OnDestroy()
    {
        if (LevelGrid.Instance != null)
        {
            gridPosition = LevelGrid.Instance.GetGridPosition(transform.position);
            LevelGrid.Instance.RemoveUnitAtGridPosition(gridPosition, this);
        }
    }
}
```

RogueShooter – All Scripts

Assets/scripts/PlayerUnits/UnitActions/BaseAction.cs

```
using UnityEngine;
using Mirror;
using System;
using System.Collections.Generic;

/// <summary>
/// Base class for all unit actions in the game.
/// This class inherits from NetworkBehaviour and provides common functionality for unit actions.
/// </summary>
public abstract class BaseAction : NetworkBehaviour
{
    protected Unit unit;
    protected bool isActive;
    protected Action onActionComplete;

    protected virtual void Awake()
    {
        unit = GetComponent<Unit>();
    }

    public abstract string GetActionName();

    public abstract void TakeAction(GridPosition gridPosition, Action onActionComplete);

    public virtual bool IsValidGridPosition(GridPosition gridPosition)
    {
        List<GridPosition> validGridPositionsList = GetValidGridPositionList();
        return validGridPositionsList.Contains(gridPosition);
    }

    public abstract List<GridPosition> GetValidGridPositionList();

    public virtual int GetActionPointsCost()
    {
        return 1;
    }

    protected void ActionStart(Action onActionComplete)
    {
        isActive = true;
        this.onActionComplete = onActionComplete;
    }

    protected void ActionComplete()
    {
        isActive = false;
        onActionComplete();
    }
}
```

Assets/scripts/PlayerUnits/UnitActions/MoveAction.cs

```
using System;
using System.Collections.Generic;
using Mirror;
using UnityEngine;

/// <summary>
/// The MoveAction class is responsible for handling the movement of a unit in the game.
/// It allows the unit to move to a target position, and it calculates valid move grid positions based on the unit's current position.
/// </summary>

public class MoveAction : BaseAction
{
    [SerializeField] private Animator unitAnimator;
    [SerializeField] private int maxMoveDistance = 4;
    private Vector3 targetPosition;

    protected override void Awake()
    {
        base.Awake();
        targetPosition = transform.position;
    }

    private void Update()
    {
        if(AuthorityHelper.HasLocalControl(this)) return;
        if(!isActive) return;
        Vector3 moveDirection = (targetPosition - transform.position).normalized;

        float stoppingDistance = 0.2f;
        if (Vector3.Distance(transform.position, targetPosition) > stoppingDistance)
        {

            // Move towards the target position
            // Vector3 moveDirection = (targetPosition - transform.position).normalized;
            float moveSpeed = 4f;
            transform.position += moveSpeed * Time.deltaTime * moveDirection;

            // Rotate towards the target position
            float rotationSpeed = 10f;
            transform.forward = Vector3.Lerp(transform.forward, moveDirection, Time.deltaTime * rotationSpeed);

            unitAnimator.SetBool("IsRunning", true);

        }
        else
        {
            unitAnimator.SetBool("IsRunning", false);
            ActionComplete();

        }
```

```
    }

    public override void TakeAction(GridPosition gridPosition, Action onActionComplete )
    {
        ActionStart(onActionComplete);

        targetPosition = LevelGrid.Instance.GetWorldPosition(gridPosition);

    }


    public override List<GridPosition> GetValidGridPositionList()
    {
        List<GridPosition> validGridPositionList = new();

        GridPosition unitGridPosition = unit.GetGridPosition();

        for (int x = - maxMoveDistance; x <= maxMoveDistance; x++)
        {
            for (int z = -maxMoveDistance; z <= maxMoveDistance; z++)
            {
                GridPosition offsetGridPosition = new(x, z);
                GridPosition testGridPosition = unitGridPosition + offsetGridPosition;

                // Check if the test grid position is within the valid range and not occupied by another unit
                if(!LevelGrid.Instance.IsValidGridPosition(testGridPosition) ||
                unitGridPosition == testGridPosition ||
                LevelGrid.Instance.HasAnyUnitOnGridPosition(testGridPosition)) continue;

                validGridPositionList.Add(testGridPosition);
                // Debug.Log($"Testing grid position: {testGridPosition}");

            }

        }

        return validGridPositionList;
    }

    public  override string GetActionName()
    {
        return "Move";
    }
}
```

Assets/scripts/PlayerUnits/UnitActions/ShootAction.cs

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

public class ShootAction : BaseAction
{
    private enum State
    {
        Aiming,
        Shooting,
        Cooloff
    }

    private State state;
    private int maxShootDistance = 7;

    private float stateTimer;
    private Unit targetUnit;
    private bool canShootBullet;

    // Update is called once per frame
    void Update()
    {
        if (!isActive) return;

        stateTimer -= Time.deltaTime;
        switch (state)
        {
            case State.Aiming:
                // Rotate towards the target position
                Vector3 aimDirection = (targetUnit.GetWorldPosition() - unit.GetWorldPosition()).normalized;
                float rotationSpeed = 10f;
                transform.forward = Vector3.Lerp(transform.forward, aimDirection, Time.deltaTime * rotationSpeed);
                break;
            case State.Shooting:
                if (canShootBullet)
                {
                    Shoot();
                    canShootBullet = false;

                }
                break;
            case State.Cooloff:
                break;
        }

        if (stateTimer <= 0f)
        {
            NextState();
        }
```

```
    }

    private void NextState()
    {
        switch (state)
        {
            case State.Aiming:
                state = State.Shooting;
                float shootingStateTime = 0.1f;
                stateTimer = shootingStateTime;
                break;
            case State.Shooting:
                state = State.Cooloff;
                float cooloffStateTime = 0.5f;
                stateTimer = cooloffStateTime;
                break;
            case State.Cooloff:
                ActionComplete();
                break;
        }

        Debug.Log(state);
    }

    private void Shoot()
    {
        Debug.Log("Shoot");
        targetUnit.Damage();
    }

     public override int GetActionPointsCost()
    {
        return 1;
    }

    public override string GetActionName()
    {
        return "Shoot";
    }

    public override List<GridPosition> GetValidGridPositionList()
    {
        List<GridPosition> validGridPositionList = new();

        GridPosition unitGridPosition = unit.GetGridPosition();

        for (int x = - maxShootDistance; x <= maxShootDistance; x++)
        {
            for (int z = -maxShootDistance; z <= maxShootDistance; z++)
            {
                GridPosition offsetGridPosition = new(x, z);
```

```
            GridPosition testGridPosition = unitGridPosition + offsetGridPosition;

            // Check if the test grid position is within the valid range and not occupied by another unit
            if (!LevelGrid.Instance.IsValidGridPosition(testGridPosition)) continue;
            int testDistance = Mathf.Abs(x) + Mathf.Abs(z);
            if (testDistance > maxShootDistance) continue;

            // DoDo show shooting range even if there are no units to shoot at
            //validGridPositionList.Add(testGridPosition);


            if (!LevelGrid.Instance.HasAnyUnitOnGridPosition(testGridPosition)) continue;

            Unit targetUnit = LevelGrid.Instance.GetUnitAtGridPosition(testGridPosition);

            // Make sure we don't include friendly units. Continue the loop only if the unit is an enemy.
            if (targetUnit.IsEnemy() == unit.IsEnemy()) continue;

            validGridPositionList.Add(testGridPosition);
           // Debug.Log($"Testing grid position: {testGridPosition}");

        }

    }

    return validGridPositionList;
}


public override void TakeAction(GridPosition gridPosition, Action onActionComplete)
{
    ActionStart(onActionComplete);

    targetUnit = LevelGrid.Instance.GetUnitAtGridPosition(gridPosition);

    Debug.Log("Aiming");
    state = State.Aiming;
    float aimingStateTime = 1f;
    stateTimer = aimingStateTime;

    canShootBullet = true;
}

}
```

Assets/scripts/PlayerUnits/UnitActions/SpinAction.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;




/// <summary>
///      This class is responsible for spinning a unit around its Y-axis.
/// </summary>
/// remarks>
///      Change to turn towards the direction the mouse is pointing
/// </remarks>

public class SpinAction : BaseAction
{

    // public delegate void SpinCompleteDelegate();

   //  private Action onSpinComplete;
    private float totalSpinAmount = 0f;
    private void Update()
    {
        if(!isActive) return;

        float spinAddAmmount = 360f * Time.deltaTime;
        transform.eulerAngles += new Vector3(0, spinAddAmmount, 0);

        totalSpinAmount += spinAddAmmount;
        if (totalSpinAmount >= 360f)
        {
            ActionComplete();
        }

    }
    public override void TakeAction(GridPosition gridPosition , Action onActionComplete)
    {
        ActionStart(onActionComplete);

        totalSpinAmount = 0f;
    }

    public override string GetActionName()
    {
        return "Spin";
    }

    public override List<GridPosition> GetValidGridPositionList()
    {

        GridPosition unitGridPosition = unit.GetGridPosition();
```

```
        return new List<GridPosition>()
        {
            unitGridPosition
        };
    }

    public override int GetActionPointsCost()
    {
        return 2;
    }
}
```

RogueShooter – All Scripts

Assets/scripts/PlayerUnits/UnitActions/UnitActionSystem.cs

```
using Mirror;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

/// <summary>
///      This script handles the unit action system in the game.
///      It allows the player to select units and perform actions on them, such as moving or shooting.
/// </summary>

public class UnitActionSystem : MonoBehaviour
{
    public static UnitActionSystem Instance { get; private set; }

    public event EventHandler OnSelectedUnitChanged;
    public event EventHandler OnSelectedActionChanged;
    public event EventHandler <bool> OnBusyChanged;
    public event EventHandler OnActionStarted;

    // This allows the script to only interact with objects on the specified layer
    [SerializeField] private LayerMask unitLayerMask;
    [SerializeField] private Unit selectedUnit;

    private BaseAction selectedAction;

    // Prevents the player from performing multiple actions at the same time
    private bool isBusy;

    private void Awake()
    {
        selectedUnit = null;
        // Ensure that there is only one instance in the scene
        if (Instance != null)
        {
            Debug.LogError("UnitActionSystem: More than one UnitActionSystem in the scene!" + transform + " " + Instance);
            Destroy(gameObject);
            return;
        }
        Instance = this;
    }

    private void Start()
    {

    }
    private void Update()
    {
        // Prevents the player from performing multiple actions at the same time
```

```
        if (isBusy) return;

        // if is not the player's turn, ignore input
        if(!TurnSystem.Instance.IsPlayerTurn()) return;

        // Ignore input if the mouse is over a UI element
        if(EventSystem.current.IsPointerOverGameObject()) return;

        // Check if the player is trying to select a unit or move the selected unit
        if (TryHandleUnitSelection()) return;

        HandleSelectedAction();
    }

    private void HandleSelectedAction()
    {
        if (Input.GetMouseButtonDown(0))
        {
            GridPosition mouseGridPosition = LevelGrid.Instance.GetGridPosition(MouseWorld.GetMouseWorldPosition());

            if (!selectedAction.IsValidGridPosition(mouseGridPosition)
            || !selectedUnit.TrySpendActionPointsToTakeAction(selectedAction))
            {
                return;
            }
            SetBusy();
            selectedAction.TakeAction(mouseGridPosition, ClearBusy);

            OnActionStarted?.Invoke(this, EventArgs.Empty);
        }
    }

    /// <summary>
    //      Prevents the player from performing multiple actions at the same time
    /// </summary>
    private void SetBusy()
    {
        isBusy = true;
        OnBusyChanged?.Invoke(this, isBusy);
    }

    /// <summary>
    ///     This method is called when the action is completed.
    /// </summary>
    private void ClearBusy()
    {
        isBusy = false;
        OnBusyChanged?.Invoke(this, isBusy);
    }

    /// <summary>
    ///     This method is called when the player clicks on a unit in the game world.
```

```
    ///     Check if the mouse is over a unit
    ///     If so, select the unit and return
    ///     If not, move the selected unit to the mouse position
    /// </summary>
    private bool TryHandleUnitSelection()
    {
        if(Input.GetMouseButtonDown(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out RaycastHit hit, float.MaxValue, unitLayerMask))
            {
                if (hit.transform.TryGetComponent<Unit>(out Unit unit))
                {
                    if(AuthorityHelper.HasLocalControl(unit) || unit == selectedUnit) return false;
                    SetSelectedUnit(unit);
                    return true;
                }
            }
        }

        return false;
    }

    /// <summary>
    ///     Sets the selected unit and triggers the OnSelectedUnitChanged event.
    ///     By defaults set the selected action to the unit's move action. The most common action.
    /// </summary>
    private void SetSelectedUnit(Unit unit)
    {
        if(unit.IsEnemy()) return;
        selectedUnit = unit;
        SetSelectedAction(unit.GetMoveAction());
        OnSelectedUnitChanged?.Invoke(this, EventArgs.Empty);
    }

    /// <summary>
    ///     Sets the selected action and triggers the OnSelectedActionChanged event.
    /// </summary>
    public void SetSelectedAction(BaseAction baseAction)
    {
        selectedAction = baseAction;
        OnSelectedActionChanged?.Invoke(this, EventArgs.Empty);
    }

    public Unit GetSelectedUnit()
    {
        return selectedUnit;
    }

    public BaseAction GetSelectedAction()
    {
        return selectedAction;
```

```
    }
}
```

Assets/scripts/PlayerUnits/UnitController/PlayerController.cs

```csharp
using System;
using Mirror;
using UnityEngine;

public class PlayerController : NetworkBehaviour
{
    [SyncVar] public bool hasEndedThisTurn;

    // UI-nappi kutsuu tätä (vain local player)
    public void ClickEndTurn()
    {
        if (!isLocalPlayer) return;
        if (hasEndedThisTurn) return;             // ei tuplia
        if (CoopTurnCoordinator.Instance && CoopTurnCoordinator.Instance.phase != TurnPhase.Players) return;
        Debug.Log("[PC] ClickEndTurn → CmdEndTurn()");
        CmdEndTurn();
    }

    [Command(requiresAuthority = true)]

    void CmdEndTurn()
    {
        Debug.Log($"[PC][SERVER] CmdEndTurn called by player {netId}");
        if (hasEndedThisTurn) return;
        hasEndedThisTurn = true;
        Debug.Log("[PC][SERVER] CmdEndTurn received");
        // Varmista myös että koordinaattori löytyy serveripuolelta:
        if (CoopTurnCoordinator.Instance == null)
        {
            Debug.LogWarning("[PC][SERVER] CoopTurnCoordinator.Instance is NULL on server!");
            return;
        }
        CoopTurnCoordinator.Instance.ServerPlayerEndedTurn(netIdentity.netId);
    }

    // Server kutsuu tämän kierroksen alussa nollatakseen tilan
    [Server]
    public void ServerSetHasEnded(bool v)
    {
        hasEndedThisTurn = v;
        Debug.Log($"[PC][SERVER] ServerSetHasEnded({v}) for player {netId}");
        TargetNotifyCanAct(connectionToClient, !v);
    }

    [TargetRpc]
    void TargetNotifyCanAct(NetworkConnectionToClient __, bool canAct)
    {
        Debug.Log($"[PC][CLIENT] TargetNotifyCanAct({canAct})");
        // UIEndTurnButton.interactable = canAct;
        var ui = FindFirstObjectByType<TurnSystemUI>();
```

```
        if (ui != null)
        ui.SetCanAct(canAct);
    }
}
```

Assets/scripts/PlayerUnits/UnitsControlUI/TurnSystemUI.cs

```
using System;
using Mirror;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class TurnSystemUI : MonoBehaviour
{
    [SerializeField] private Button endTurnButton;
    [SerializeField] private TextMeshProUGUI turnNumberText;              // (valinnainen, käytä SP:ssä)
    [SerializeField] private GameObject enemyTurnVisualGameObject;        // (valinnainen, käytä SP:ssä)

    bool isCoop;
    private PlayerController localPlayerController;

    void Start()
    {
        isCoop = GameModeManager.SelectedMode == GameMode.CoOp;

        // kiinnitä handler tasan kerran
        if (endTurnButton != null)
        {
            endTurnButton.onClick.RemoveAllListeners();
            endTurnButton.onClick.AddListener(OnEndTurnClicked);
        }

        if (isCoop)
        {
            // Co-opissa nappi on DISABLED kunnes serveri kertoo että saa toimia
            SetCanAct(false);
        }
        else
        {
            // Singleplayerissa kuuntele vuoron vaihtumista
            if (TurnSystem.Instance != null)
            {
                TurnSystem.Instance.OnTurnChanged += TurnSystem_OnTurnChanged;
                UpdateForSingleplayer();
            }
        }
    }

    void OnDisable()
    {
        if (!isCoop && TurnSystem.Instance != null)
            TurnSystem.Instance.OnTurnChanged -= TurnSystem_OnTurnChanged;
    }

    // ====== julkinen kutsu PlayerController.TargetNotifyCanAct:ista ======
    public void SetCanAct(bool canAct)
```

```
{
    if (endTurnButton == null) return;

    endTurnButton.onClick.RemoveListener(OnEndTurnClicked);
    if (canAct) endTurnButton.onClick.AddListener(OnEndTurnClicked);

    endTurnButton.gameObject.SetActive(canAct);   // jos haluat pitää aina näkyvissä, vaihda SetActive(true)
    endTurnButton.interactable = canAct;
}

// ====== nappi ======
private void OnEndTurnClicked()
{
    // Päättele co-op -tila tilannekohtaisesti (ei SelectedMode)
    bool isCoopNow =
        CoopTurnCoordinator.Instance != null &&
        (NetworkServer.active || NetworkClient.isConnected);

    if (!isCoopNow)
    {
        Debug.Log("[UI] EndTurn clicked (SP)");
        if (TurnSystem.Instance != null)
        {
            TurnSystem.Instance.NextTurn();
        }
        else
        {
            Debug.LogWarning("[UI] TurnSystem.Instance is null");
        }
        return;
    }

    Debug.Log("[UI] EndTurn clicked (Co-op)");

    CacheLocalPlayerController();
    if (localPlayerController == null)
    {
        Debug.LogWarning("[UI] Local PlayerController not found");
        return;
    }

    // Estä tuplaklikki
    SetCanAct(false);

    // Lähetä serverille
    localPlayerController.ClickEndTurn();
}

private void CacheLocalPlayerController()
{
    if (localPlayerController == null)
    {
```

```
            var conn = NetworkClient.connection;
            if (conn != null && conn.identity != null)
            {
                localPlayerController = conn.identity.GetComponent<PlayerController>();
            }
        }
    }

    // ====== singleplayer UI (valinnainen) ======
    private void TurnSystem_OnTurnChanged(object s, EventArgs e) => UpdateForSingleplayer();

    private void UpdateForSingleplayer()
    {
        if (turnNumberText != null)
            turnNumberText.text = "Turn: " + TurnSystem.Instance.GetTurnNumber();

        if (enemyTurnVisualGameObject != null)
            enemyTurnVisualGameObject.SetActive(!TurnSystem.Instance.IsPlayerTurn());

        if (endTurnButton != null)
            endTurnButton.gameObject.SetActive(TurnSystem.Instance.IsPlayerTurn());
    }
}
```

RogueShooter – All Scripts

Assets/scripts/PlayerUnits/UnitsControlUI/UnitActionBusyUI.cs

```csharp
using UnityEngine;

/// <summary>
///     This class is responsible for displaying the busy UI when the unit action system is busy
/// </summary>
public class UnitActionBusyUI : MonoBehaviour
{
    private void Start()
    {
        UnitActionSystem.Instance.OnBusyChanged += UnitActionSystem_OnBusyChanged;

        Hide();

    }
    private void Show()
    {
        gameObject.SetActive(true);
    }
    private void Hide()
    {
        gameObject.SetActive(false);
    }
    /// <summary>
    ///     This method is called when the unit action system is busy or not busy
    /// </summary>
    private void UnitActionSystem_OnBusyChanged(object sender, bool isBusy)
    {
        if (isBusy)
        {
            Show();
        }
        else
        {
            Hide();
        }
    }
}
```

RogueShooter – All Scripts

Assets/scripts/PlayerUnits/UnitsControlUI/UnitActionButtonUI.cs

```csharp
using UnityEngine;
using UnityEngine.UI;
using TMPro;

/// <summary>
///     This class is responsible for displaying the action button TXT in the UI
/// </summary>

public class UnitActionButtonUI : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI textMeshPro;
    [SerializeField] private Button actionButton;
    [SerializeField] private GameObject actionButtonSelectedVisual;

    private BaseAction baseAction;

    public void SetBaseAction(BaseAction baseAction)
    {
        this.baseAction = baseAction;
        textMeshPro.text = baseAction.GetActionName().ToUpper();

        actionButton.onClick.AddListener(() =>
        {
            UnitActionSystem.Instance.SetSelectedAction(baseAction);
        } );

    }

    public void UpdateSelectedVisual()
    {
        BaseAction selectedbaseAction = UnitActionSystem.Instance.GetSelectedAction();
        actionButtonSelectedVisual.SetActive(selectedbaseAction == baseAction);
    }

}
```

Assets/scripts/PlayerUnits/UnitsControlUI/UnitActionSystemUI.cs

```
using System;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

/// <summary>
///      This class is responsible for displaying the action buttons for the selected unit in the UI.
///      It creates and destroys action buttons based on the selected unit's actions.
/// </summary>

public class UnitActionSystemUI : MonoBehaviour
{

    [SerializeField] private Transform actionButtonPrefab;
    [SerializeField] private Transform actionButtonContainerTransform;
    [SerializeField] private TextMeshProUGUI actionPointsText;

    private List<UnitActionButtonUI> actionButtonUIList;

    private void Awake()
    {
        actionButtonUIList = new List<UnitActionButtonUI>();
    }

    private void Start()
    {
        UnitActionSystem.Instance.OnSelectedUnitChanged += UnitActionSystem_OnSelectedUnitChanged;
        UnitActionSystem.Instance.OnSelectedActionChanged += UnitActionSystem_OnSelectedActionChanged;
        UnitActionSystem.Instance.OnActionStarted += UnitActionSystem_OnActionStarted;
        TurnSystem.Instance.OnTurnChanged += TurnSystem_OnTurnChanged;
        Unit.OnAnyActionPointsChanged += Unit_OnAnyActionPointsChanged;
    }

    private void CreateUnitActionButtons()
    {

        Unit selectedUnit = UnitActionSystem.Instance.GetSelectedUnit();
        if (selectedUnit == null)
        {
            Debug.Log("No selected unit found.");
            return;
        }
        actionButtonUIList.Clear();

        foreach (BaseAction baseAction in selectedUnit.GetBaseActionsArray())
        {
            Transform actionButtonTransform = Instantiate( actionButtonPrefab, actionButtonContainerTransform);
            UnitActionButtonUI actionButtonUI = actionButtonTransform.GetComponent<UnitActionButtonUI>();
```

```
            actionButtonUI.SetBaseAction(baseAction);
            actionButtonUIList.Add(actionButtonUI);

        }
    }

    private void DestroyActionButtons()
    {
        foreach (Transform child in actionButtonContainerTransform)
        {
            Destroy(child.gameObject);
        }
    }

    private void UnitActionSystem_OnSelectedUnitChanged(object sender, EventArgs e)
    {
        DestroyActionButtons();
        CreateUnitActionButtons();
        UpdateSelectedVisual();
        UpdateActionPointsVisual();
    }

    private void UnitActionSystem_OnSelectedActionChanged(object sender, EventArgs e)
    {
        UpdateSelectedVisual();
    }

    private void UnitActionSystem_OnActionStarted(object sender, EventArgs e)
    {
        UpdateActionPointsVisual();
    }

    private void UpdateSelectedVisual()
    {
        foreach (UnitActionButtonUI actionButtonUI in actionButtonUIList)
        {
            actionButtonUI.UpdateSelectedVisual();
        }
    }

    private void UpdateActionPointsVisual()
    {
        Unit selectedUnit = UnitActionSystem.Instance.GetSelectedUnit();
        actionPointsText.text = "Action Points: " + selectedUnit.GetActionPoints();
    }

    /// <summary>
    ///     This method is called when the turn changes. It resets the action points UI to the maximum value.
    /// </summary>
    private void TurnSystem_OnTurnChanged(object sender, EventArgs e)
    {
        UpdateActionPointsVisual();
```

```
    }

    /// <summary>
    ///     This method is called when the action points of any unit change. It updates the action points UI.
    /// </summary>
    private void Unit_OnAnyActionPointsChanged(object sender, EventArgs e)
    {
        UpdateActionPointsVisual();
    }

}
```

Assets/scripts/PlayerUnits/UnitSelectedVisual.cs

```
using System;
using UnityEngine;

/// <summary>
/// This class is responsible for displaying a visual indicator when a unit is selected in the game.
/// It uses a MeshRenderer component to show or hide the visual representation of the selected unit.
/// </summary>
public class UnitSelectedVisual : MonoBehaviour
{
    [SerializeField] private Unit unit;
    [SerializeField] private MeshRenderer meshRenderer;

    //private MeshRenderer meshRenderer;

    private void Awake()
    {
        // meshRenderer = GetComponent<MeshRenderer>();
        // meshRenderer.enabled = false;
        if (!meshRenderer) meshRenderer = GetComponentInChildren<MeshRenderer>(true);
        if (meshRenderer) meshRenderer.enabled = false;
    }

    private void Start()
    {
        if (UnitActionSystem.Instance != null)
        {
            UnitActionSystem.Instance.OnSelectedUnitChanged += UnitActionSystem_OnSelectedUnitChanged;
            UpdateVisual();
        }
    }

    private void OnDestroy()
    {
        if (UnitActionSystem.Instance != null)
            UnitActionSystem.Instance.OnSelectedUnitChanged -= UnitActionSystem_OnSelectedUnitChanged;
    }

    private void UnitActionSystem_OnSelectedUnitChanged(object sender, EventArgs empty)
    {
        UpdateVisual();
    }

    private void UpdateVisual()
    {
        /*
        if (unit == UnitActionSystem.Instance.GetSelectedUnit())
        {
            meshRenderer.enabled = true;
        }
        else
```

```
        {
            meshRenderer.enabled = false;
        }
        */
        if (!this || meshRenderer == null || UnitActionSystem.Instance == null) return;
        var selected = UnitActionSystem.Instance.GetSelectedUnit();
        meshRenderer.enabled = (unit != null && selected == unit);

    }
}
```

RogueShooter – All Scripts

Assets/scripts/Testing.cs

```
using UnityEngine;

/// <summary>
/// This class is responsible for testing the grid system and unit actions in the game.
/// It provides functionality to visualize the grid positions and interact with unit actions.
/// </summary>
public class Testing : MonoBehaviour
{

    [SerializeField] private Unit unit;
    private void Start()
    {

    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.T))
        {
            GridSystemVisual.Instance.HideAllGridPositions();
            GridSystemVisual.Instance.ShowGridPositionList(
                unit.GetMoveAction().
                GetValidGridPositionList());
        }
    }
}
```