

RogueShooter – All Scripts

Generated: 2025-09-21 14.46 UTC

Files: 62

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/BulletProjectile.cs

```
using Mirror;
using UnityEngine;

public class BulletProjectile : NetworkBehaviour
{
    [SerializeField] private TrailRenderer trailRenderer;
    [SerializeField] private Transform bulletHitVfxPrefab;

    [SyncVar] private Vector3 targetPosition;

    public void Setup(Vector3 targetPosition)
    {
        this.targetPosition = targetPosition;
    }

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

        if (trailRenderer && !trailRenderer.emitting) trailRenderer.emitting = true;
    }

    private void Update()
    {
        Vector3 moveDirection = (targetPosition - transform.position).normalized;

        float distanceBeforeMoving = Vector3.Distance(transform.position, targetPosition);

        float moveSpeed = 200f; // Adjust the speed as needed
        transform.position += moveSpeed * Time.deltaTime * moveDirection;

        float distanceAfterMoving = Vector3.Distance(transform.position, targetPosition);

        // Check if we've reached or passed the target position
        if (distanceBeforeMoving < distanceAfterMoving)
        {
            transform.position = targetPosition;

            if (trailRenderer) trailRenderer.transform.parent = null;

            if (bulletHitVfxPrefab)
                Instantiate(bulletHitVfxPrefab, targetPosition, Quaternion.identity);

            // Network-aware destruction
            if (isServer) NetworkServer.Destroy(gameObject);
            else Destroy(gameObject);
        }
    }
}
```

```
}
```

RogueShooter – All Scripts

Assets/scripts/CameraManager.cs

```
using System;
using UnityEngine;

public class CameraManager : MonoBehaviour
{
    [SerializeField] private GameObject actionCameraGameObject;

    private void Start()
    {
        // BaseAction.OnAnyActionStarted += BaseAction_OnAnyActionStarted;
        // BaseAction.OnAnyActionCompleted += BaseAction_OnAnyActionCompleted;

        // HideActionCamera();
    }

    void OnEnable()
    {
        BaseAction.OnAnyActionStarted += BaseAction_OnAnyActionStarted;
        BaseAction.OnAnyActionCompleted += BaseAction_OnAnyActionCompleted;
        HideActionCamera();
    }

    void OnDisable()
    {
        BaseAction.OnAnyActionStarted -= BaseAction_OnAnyActionStarted;
        BaseAction.OnAnyActionCompleted -= BaseAction_OnAnyActionCompleted;
    }

    private void ShowActionCamera()
    {
        actionCameraGameObject.SetActive(true);
    }

    private void HideActionCamera()
    {
        actionCameraGameObject.SetActive(false);
    }

    private void BaseAction_OnAnyActionStarted(object sender, EventArgs e)
    {
        switch (sender)
        {
            case ShootAction shootAction:
                Unit shooterUnit = shootAction.GetUnit();
                Unit targetUnit = shootAction.GetTargetUnit();

                Vector3 cameraCharacterHeight = Vector3.up * 1.7f;
                Vector3 shootDir = (targetUnit.GetWorldPosition() - shooterUnit.GetWorldPosition()).normalized;
```

RogueShooter – All Scripts

```
        float shoulderOffsetAmount = 0.5f;
        Vector3 shoulderOffset = Quaternion.Euler(0, 90, 0) * shootDir * shoulderOffsetAmount;
        Vector3 actionCameraPosition =
            shooterUnit.GetWorldPosition() +
            cameraCharacterHeight +
            shoulderOffset +
            (shootDir * -1);

        actionCameraGameObject.transform.position = actionCameraPosition;
        actionCameraGameObject.transform.LookAt(targetUnit.GetWorldPosition() + cameraCharacterHeight);
        ShowActionCamera();
        break;
    }
}

private void BaseAction_OnAnyActionCompleted(object sender, EventArgs e)
{
    switch (sender)
    {
        case ShootAction shootAction:
            HideActionCamera();
            break;
    }
}
}
```

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

```
using System;
using System.Collections;
using UnityEngine;
using Utp;

/// <summary>
/// Control EnemyAI. Go through all possible actions what current enemy Unit can do and chose the best one.
/// Listen to TurnSystem and when turn OnTurnChanged, AI state switch WaitingForEnemyTurn to the TakingTurn state
/// and try to find best action to all enemy Units. All enemy Unit do this independently based on
/// action values.
/// </summary>
public class EnemyAI : MonoBehaviour
{
    public static EnemyAI Instance { get; private set; }

    private enum State
    {
        WaitingForEnemyTurn,
        TakingTurn,
        Busy,
    }

    private State state;
    private float timer;

    void Awake()
    {
        state = State.WaitingForEnemyTurn;

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

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

        if (GameNetworkManager.Instance != null &&
            GameNetworkManager.Instance.GetNetworkClientConnected() &&
            !GameNetworkManager.Instance.GetNetworkServerActive())
        {
            // Coop gamemode using IEnumerator RunEnemyTurnCoroutine() through the server. No local calls
            if (GameManager.SelectedMode == GameMode.CoOp)
                enabled = false;
        }
    }
}
```

RogueShooter – All Scripts

```
}

/*
void OnEnable()
{
    if (GameManager.SelectedMode == GameMode.SinglePlayer)
    {
        TurnSystem.Instance.OnTurnChanged += TurnSystem_OnTurnChanged;
    }
}
*/

void OnDisable()
{
    if (GameManager.SelectedMode == GameMode.SinglePlayer)
    {
        TurnSystem.Instance.OnTurnChanged -= TurnSystem_OnTurnChanged;
    }
}

private void Update()
{
    //NOTE! Only solo game!
    if (GameManager.SelectedMode != GameMode.SinglePlayer) return;
    if (TurnSystem.Instance.IsPlayerTurn()) return;

    //If game mode is SinglePlayer and is not PlayerTurn then runs Enemy AI.
    EnemyAITick(Time.deltaTime);
}

/// <summary>
/// Enemy start taking actions after small waiting time.
/// Update call this every frame.
/// </summary>
private bool EnemyAITick(float dt)
{
    switch (state)
    {
        // It is Player turn so keep waiting untill TurnSystem_OnTurnChanged switch state to TakingTurn.
        case State.WaitingForEnemyTurn:
            return false;

        case State.TakingTurn:
            timer -= dt;
            if (timer <= 0f)
            {
                //Return false when all Enemy Units have make they actions
                if (SelectEnemyUnitToTakeAction(SetStateTakingTurn))
                {
                    state = State.Busy;
                    return false;
                }
            }
    }
}
```

RogueShooter – All Scripts

```
        else
        {
            // If enemy cant make actions. Return turn back to player.
            // NOTE! In Coop mode CoopTurnCoordinator make this.
            if (GameManager.SelectedMode == GameMode.SinglePlayer)
            {
                TurnSystem.Instance.NextTurn();
            }

            // Enemy AI switch back to waiting.
            state = State.WaitingForEnemyTurn;
            return true;
        }
    }
    return false;

    case State.Busy:
        // When Enemy doing action just return.
        // Waiting c# Action call from base action and then call funktion SetStateTakingTurn()
        return false;
    }
    return false;
}

/// <summary>
/// c# Action callback. SelectEnemyUnitToTakeAction use this and when action is ready. This occurs
/// </summary>
private void SetStateTakingTurn()
{
    timer = 0.5f;
    state = State.TakingTurn;
}

/// <summary>
/// Go through all enemy Units on EnemyUnit List and try to take action.
/// </summary>
private bool SelectEnemyUnitToTakeAction(Action onEnemyAIActionComplete)
{
    foreach (Unit enemyUnit in UnitManager.Instance.GetEnemyUnitList())
    {
        if (enemyUnit == null)
        {
            Debug.LogWarning("[EnemyAI][UnitManager]EnemyUnit list is null:" + enemyUnit);
            continue;
        }
        if (TryTakeEnemyAIAction(enemyUnit, onEnemyAIActionComplete))
        {
            return true;
        }
    }
}
```

RogueShooter – All Scripts

```
        return false;
    }

    /// <summary>
    /// Selected Unit Go through all possible actions what Enemy Unit can do
    /// and choosing the best one based on them action value.
    /// Then make action if have enough action points.
    /// </summary>
    private bool TryTakeEnemyAIAction(Unit enemyUnit, Action onEnemyAIActionComplete)
    {
        // Contains Gridposition and action value (How good action is)
        EnemyAIAction bestEnemyAIAction = null;

        BaseAction bestBaseAction = null;

        // Choosing the best action, based on them action value.
        foreach (BaseAction baseAction in enemyUnit.GetBaseActionsArray())
        {
            if (!enemyUnit.CanSpendActionPointsToTakeAction(baseAction))
            {
                // Enemy cannot afford this action.
                continue;
            }

            if (bestEnemyAIAction == null)
            {
                bestEnemyAIAction = baseAction.GetBestEnemyAIAction();
                bestBaseAction = baseAction;
            }
            else
            {
                // Go trough all actions and take the best one.
                EnemyAIAction testEnemyAIAction = baseAction.GetBestEnemyAIAction();
                if (testEnemyAIAction != null && testEnemyAIAction.actionValue > bestEnemyAIAction.actionValue)
                {
                    bestEnemyAIAction = baseAction.GetBestEnemyAIAction();
                    bestBaseAction = baseAction;
                }
            }
        }

        // Try to take action
        if (bestEnemyAIAction != null && enemyUnit.TrySpendActionPointsToTakeAction(bestBaseAction))
        {
            bestBaseAction.TakeAction(bestEnemyAIAction.gridPosition, onEnemyAIActionComplete);
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

RogueShooter – All Scripts

```
/// <summary>
/// When turn changed. Switch state to taking turn and enemy turn start.
/// </summary>
private void TurnSystem_OnTurnChanged(object sender, EventArgs e)
{
    if (!TurnSystem.Instance.IsPlayerTurn())
    {
        state = State.TakingTurn;
        timer = 1f; // Small holding time before action.
    }
}

/// <summary>
/// When playing online: (Coop mode) Server handle All AI actions.
/// </summary>
[Mirror.Server]
public IEnumerator RunEnemyTurnCoroutine()
{
    SetStateTakingTurn();

    while (true)
    {
        if (TurnSystem.Instance.IsPlayerTurn())
        {
            Debug.LogWarning("[EnemyAI] Players get turn before AI has ended own turn! This sould not be possibble");
            yield break;
        }

        bool finished = EnemyAITick(Time.deltaTime);
        if (finished)
            yield break; // AI-Turn ready. CoopTurnCoordinator continue and give turn back to players.

        yield return null; // wait one frame.
    }
}
}
```

RogueShooter – All Scripts

Assets/scripts/Enemy/EnemyAIAction.cs

```
using UnityEngine;

[System.Serializable]
public class EnemyAIAction
{
    public GridPosition gridPosition;
    public int actionValue;
}
```

RogueShooter – All Scripts

Assets/scripts/Enemy/PathNode.cs

```
using Unity.VisualScripting;
using UnityEngine;

public class PathNode
{
    private GridPosition gridPosition;
    private int gCost;
    private int hCost;
    private int fCost;
    private PathNode cameFromPathNode;
    public PathNode(GridPosition gridPosition)
    {
        this.gridPosition = gridPosition;
    }

    public override string ToString()
    {
        return gridPosition.ToString();
    }

    public int GetGCost()
    {
        return gCost;
    }

    public int GetHCost()
    {
        return hCost;
    }

    public int GetFCost()
    {
        return fCost;
    }

    public void SetGCost(int gCost)
    {
        this.gCost = gCost;
    }

    public void SetHCost(int hCost)
    {
        this.hCost = hCost;
    }

    public void CalculateFCost()
    {
        fCost = gCost + hCost;
    }
}
```

RogueShooter – All Scripts

```
public void ResetCameFromPathNode()
{
    cameFromPathNode = null;
}

public void SetCameFromPathNode(PathNode pathNode)
{
    cameFromPathNode = pathNode;
}

public PathNode GetCameFromPathNode()
{
    return cameFromPathNode;
}

public GridPosition GetGridPosition()
{
    return gridPosition;
}
}
```


RogueShooter – All Scripts

Assets/scripts/FieldCleaner.cs

```
using System.Linq;
using UnityEngine;
using Utp;

public class FieldCleaner : MonoBehaviour
{
    public static void ClearAll()
    {
        // Varmista: älä yritä siivota puhtaalta clientiltä verkossa
        if (GameNetworkManager.Instance != null &&
            GameNetworkManager.Instance.GetNetWorkClientConnected() &&
            !GameNetworkManager.Instance.GetNetWorkServerActive())
        {
            Debug.LogWarning("[FieldCleaner] Don't clear field from a pure client.");
            return;
        }

        // 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);

        // Tyhjennä UnitManagerin listat (suojattu null-checkillä)
        UnitManager.Instance?.ClearAllUnitLists();

        // Nollaa myös ruudukon miehitys - sceneen jääneet objektit eivät jää kummittelemaan
        LevelGrid.Instance?.ClearAllOccupancy();
    }

    static void Despawn(GameObject go)
    {
        // if server is active, use Mirror's destroy; otherwise normal Unity Destroy
        if (GameNetworkManager.Instance.GetNetWorkServerActive())
        {
            GameNetworkManager.Instance.NetworkDestroy(go);
        }
        else
        {
            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;
    }

    private void Start()
    {
        // Varmista, että alkutila lähetetään kaikille UI:lle
        PlayerLocalTurnGate.Set(isPlayerTurn); // true = Player turn alussa
        OnTurnChanged?.Invoke(this, EventArgs.Empty); // jos haluat myös muut UI:t liikkeelle
    }

    public void NextTurn()
    {
        // Tarkista pelimoodi
        if (GameManager.SelectedMode == GameMode.SinglePlayer)
        {
            Debug.Log("SinglePlayer NextTurn");
            turnNumber++;
            isPlayerTurn = !isPlayerTurn;

            OnTurnChanged?.Invoke(this, EventArgs.Empty);

            //Set Unit UI visibility
            PlayerLocalTurnGate.Set(isPlayerTurn);
        }
        else if (GameManager.SelectedMode == GameMode.CoOp)
        {
            Debug.Log("Co-Op mode: Proceeding to the next turn.");
            // Tee jotain erityistä CoOp-tilassa
        }
    }
}
```

RogueShooter – All Scripts

```
        else if (GameManager.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;
    }

    // ForcePhase on serverin kutsuma. Päivittää vuoron ja kutsuu OnTurnChanged
    public void ForcePhase(bool isPlayerTurn, bool incrementTurnNumber)
    {
        if (incrementTurnNumber) turnNumber++;
        this.isPlayerTurn = isPlayerTurn;
        OnTurnChanged?.Invoke(this, EventArgs.Empty);
    }

    // Päivitä HUD verkon kautta (co-op)
    public void SetHudFromNetwork(int newTurnNumber, bool isPlayersPhase)
    {
        turnNumber = newTurnNumber;
        isPlayerTurn = isPlayersPhase;
        OnTurnChanged?.Invoke(this, EventArgs.Empty); // <- päivittää HUDin kuten SP:ssä
    }
}
```

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))
        {
```

RogueShooter – All Scripts

```
{
    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

```
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;
    }
}
```

RogueShooter – All Scripts

Assets/scripts/GameModes/GameModeManager.cs

```
using UnityEngine;
using Utp;

/// <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;

    void Start()
    {
        // if game is offline, spawn singleplayer units
        if (!GameNetworkManager.Instance.IsNetworkActive())
        {
            SpawnUnits();
        }
        else
        {
            Debug.Log("Game is online, waiting for host/client to spawn units.");
        }
    }

    private void SpawnUnits()
    {
        if (SelectedMode == GameMode.SinglePlayer)
        {
            SpawnUnitsCoordinator.Instance.SpwanSinglePlayerUnits();
            return;
        }
    }
}
```

RogueShooter – All Scripts

Assets/scripts/GameModes/GameReset.cs

```
using UnityEngine.SceneManagement;

public static class GameReset
{
    public static void HardReloadSceneKeepMode()
    {
        // GameManager.SelectedMode säilyy, jos se on staattinen / DontDestroyOnLoad
        var scene = SceneManager.GetActiveScene().name;
        SceneManager.LoadScene(scene);
    }
}
```


RogueShooter – All Scripts

Assets/scripts/Grid/GridDebugObject.cs

```
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 object gridObject;
    public virtual void SetGridObject(object gridObject)
    {
        this.gridObject = gridObject;
    }
    protected virtual void Update()
    {
        textMeshPro.text = gridObject.ToString();
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Grid/GridObject.cs

```
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<GridObject> gridSystem;
    private GridPosition gridPosition;

    private List<Unit> unitList;

    public GridObject(GridSystem<GridObject> 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;
    }
}
```

RogueShooter – All Scripts

```
    }  
    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 GetHashCode.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);
    }
}
```

RogueShooter – All Scripts

```
    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);
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Grid/GridSystem.cs

```
using System;
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<TGridObject>
{
    private int width;
    private int height;
    private float cellSize;

    private TGridObject[,] gridObjectsArray;
    public GridSystem(int width, int height, float cellSize, Func<GridSystem<TGridObject>, GridPosition, TGridObject> createGridObject)
    {
        this.width = width;
        this.height = height;
        this.cellSize = cellSize;

        gridObjectsArray = new TGridObject[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] = createGridObject(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.
```

RogueShooter – All Scripts

```
/// 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 TGridObject 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;
}
}
```

RogueShooter – All Scripts

Assets/scripts/Grid/GridSystemVisual.cs

```
using System;
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; }

    [Serializable]
    public struct GridVisualTypeMaterial
    {
        public GridVisualType gridVisualType;
        public Material material;
    }
    public enum GridVisualType
    {
        white,
        Blue,
        Red,
        RedSoft,
        Yellow
    }

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

    /// 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;
    }
}
```


RogueShooter – All Scripts

```
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>();
        }
    }

    UnitActionSystem.Instance.OnSelectedActionChanged += UnitActionSystem_OnSelectedActionChanged;
    LevelGrid.Instance.onAnyUnitMoveGridPosition += LevelGrid_onAnyUnitMoveGridPosition;

    UpdateGridVisuals();
}

/*
void OnEnable()
{
    UnitActionSystem.Instance.OnSelectedActionChanged += UnitActionSystem_OnSelectedActionChanged;
    LevelGrid.Instance.onAnyUnitMoveGridPosition += LevelGrid_onAnyUnitMoveGridPosition;
}
*/

void OnDisable()
{
    UnitActionSystem.Instance.OnSelectedActionChanged -= UnitActionSystem_OnSelectedActionChanged;
    LevelGrid.Instance.onAnyUnitMoveGridPosition -= LevelGrid_onAnyUnitMoveGridPosition;
}

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();
        }
    }
}

private void ShowGridPositionRange(GridPosition gridPosition, int range, GridVisualType gridVisualType)
{
    List<GridPosition> gridPositionsList = new List<GridPosition>();
```

RogueShooter – All Scripts

```
for (int x = -range; x <= range; x++)
{
    for (int z = -range; z <= range; z++)
    {
        GridPosition testGridPosition = gridPosition + new GridPosition(x, z);

        if (!LevelGrid.Instance.IsValidGridPosition(testGridPosition))
        {
            continue;
        }

        int testDistance = Mathf.Abs(x) + Mathf.Abs(z);
        if (testDistance > range)
        {
            continue;
        }

        gridPositionsList.Add(testGridPosition);
    }
}

ShowGridPositionList(gridPositionsList, gridVisualType);
}

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

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

    BaseAction selectedAction = UnitActionSystem.Instance.GetSelectedAction();

    GridVisualType gridVisualType;

    switch (selectedAction)
    {
    {
        default:
        case MoveAction moveAction:
            gridVisualType = GridVisualType.white;
            break;
        case SpinAction spinAction:
            gridVisualType = GridVisualType.Blue;
            break;
    }
    }
```

RogueShooter – All Scripts

```
        case ShootAction shootAction:
            gridVisualType = GridVisualType.Red;

            ShowGridPositionRange(selectedUnit.GetGridPosition(), shootAction.GetMaxShootDistance(), GridVisualType.RedSoft);
            break;

    }

    ShowGridPositionList(
        selectedAction.GetValidGridPositionList(), gridVisualType);
}

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

private void LevelGrid_onAnyUnitMoveGridPosition(object sender, EventArgs e)
{
    UpdateGridVisuals();
}

private Material GetGridVisualTypeMaterial(GridVisualType gridVisualType)
{
    foreach (GridVisualTypeMaterial gridVisualTypeMaterial in gridVisualTypeMaterialList)
    {
        if (gridVisualTypeMaterial.gridVisualType == gridVisualType)
        {
            return gridVisualTypeMaterial.material;
        }
    }
    Debug.LogError("Cloud not find GridVisualTypeMaterial for GridVisualType" + gridVisualType);
    return null;
}
}
```

RogueShooter – All Scripts

Assets/scripts/Grid/GridSystemVisualSingle.cs

```
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(Material material)
    {
        meshRenderer.enabled = true;
        meshRenderer.material = material;
    }

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

RogueShooter – All Scripts

Assets/scripts/Grid/LevelGrid.cs

```
using System;
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; }

    public event EventHandler onAnyUnitMoveGridPosition;
    [SerializeField] private Transform debugPrefab;
    private GridSystem<GridObject> 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<GridObject>(10, 10, 2f,
            (GridSystem<GridObject> g, GridPosition gridPosition) => new GridObject(g, gridPosition));
        // 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)
```

RogueShooter – All Scripts

```
{
    GridObject gridObject = gridSystem.GetGridObject(gridPosition);
    gridObject.RemoveUnit(unit);
}

public void UnitMoveToGridPosition(GridPosition fromGridPosition, GridPosition toGridPosition, Unit unit)
{
    RemoveUnitAtGridPosition(fromGridPosition, unit);
    AddUnitAtGridPosition(toGridPosition, unit);
    onAnyUnitMoveGridPosition?.Invoke(this, EventArgs.Empty);
}

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++)
    {
```

RogueShooter – All Scripts

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

RogueShooter – All Scripts

Assets/scripts/Grid/PathFindingDebugGridObject.cs

```
using TMPPro;
using UnityEngine;

public class PathFindingDebugGridObject : GridDebugObject
{
    [SerializeField] private TextMeshPro gCostText;
    [SerializeField] private TextMeshPro hCostText;
    [SerializeField] private TextMeshPro fCostText;

    private PathNode pathNode;
    public override void SetGridObject(object gridObject)
    {
        base.SetGridObject(gridObject);
        pathNode = (PathNode)gridObject;
    }

    protected override void Update()
    {
        base.Update();
        gCostText.text = pathNode.GetGCost().ToString();
        hCostText.text = pathNode.GetHCost().ToString();
        fCostText.text = pathNode.GetFCost().ToString();
    }
}
```


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/Look At Camera.cs

```
using UnityEngine;

public class LookAtCamera : MonoBehaviour
{
    [SerializeField] private bool invert;

    private Transform cameraTransform;

    private void Awake()
    {
        cameraTransform = Camera.main.transform;
    }

    private void LateUpdate()
    {
        if (invert)
        {
            Vector3 dirToCamera = (cameraTransform.position - transform.position).normalized;
            transform.LookAt(transform.position + dirToCamera * -1);
        } else
        {
            transform.LookAt(cameraTransform);
        }
    }
}
```

RogueShooter – All Scripts

Assets/scripts/MenuUI/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; // Hidden 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);
    }
}
```

RogueShooter – All Scripts

Assets/scripts/MenuUI/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; // Hidden 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(OnClickCoOp);
        pvpButton.onClick.AddListener(OnClickPvP);
    }

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

    public void OnClickPvP()
    {
        GameManager.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;
    }
}
```

RogueShooter – All Scripts

```
        if (lg != null) lg.RebuildOccupancyFromScene();
    }

    // DODO: Resetoi käynnissä oleva peli takaisin aloitukseen.
    public void Reset()
    {
        // Pieni "siivous" ennen reloadia on ok, mutta ei pakollinen
        FieldCleaner.ClearAll();

        if (Mirror.NetworkServer.active)
        {
            ResetService.Instance.HardResetServerAuthoritative();
        }
        else if (Mirror.NetworkClient.active)
        {
            ResetService.Instance.CmdRequestHardReset();
        }
        else
        {
            // Yksinpeli
            GameReset.HardReloadSceneKeepMode();
        }
    }

    // DODO: Resetoi peli takaisin yksinpelimoodiin.
}
```

RogueShooter – All Scripts

Assets/scripts/Online/Authentication.cs

```
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/Online/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.IsNullOrEmpty(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);
    }
}
```

RogueShooter – All Scripts

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

    nm.JoinRelayServer();
}
}
```


RogueShooter – All Scripts

Assets/scripts/Online/CoopTurnCoordinator.cs

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

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

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

    [Server]
    public void TryAdvanceIfReady()
    {
        if (NetTurnManager.Instance.phase == TurnPhase.Players && NetTurnManager.Instance.endedPlayers.Count >= Mathf.Max(1, NetTurnManager.Instance.requiredCount))
        {
            StartCoroutine(ServerEnemyTurnThenNextPlayers());
        }
    }

    [Server]
    private IEnumerator ServerEnemyTurnThenNextPlayers()
    {
        // Asettaa vihollisen WordUI: (Action Points) näkyviin.
        UnitUIBroadcaster.Instance.BroadcastUnitWorldUIVisibility(true);

        // 1) Vihollisvuoro alkaa
        RpcTurnPhaseChanged(NetTurnManager.Instance.phase = TurnPhase.Enemy, NetTurnManager.Instance.turnNumber, false);

        // Silta unit/AP-logiikalle (sama kuin nyt)
        if (TurnSystem.Instance != null)
        {
            TurnSystem.Instance.ForcePhase(isPlayerTurn: false, incrementTurnNumber: false);
        }

        // Aja AI
        yield return RunEnemyAI();

        // 2) Paluu pelaajille + turn-numero + resetit
        NetTurnManager.Instance.turnNumber++;
        NetTurnManager.Instance.ResetTurnState();

        if (TurnSystem.Instance != null)
        {

```

RogueShooter – All Scripts

```
        TurnSystem.Instance.ForcePhase(isPlayerTurn: true, incrementTurnNumber: false);
    }

    // 3) Lähetä *kaikille* (host + clientit) HUD-päivitys SP-logiikan kautta
    RpcTurnPhaseChanged(NetTurnManager.Instance.phase = TurnPhase.Players, NetTurnManager.Instance.turnNumber, true);

    // Asettaa pelaajien WordUI: (Action Points) näkyviin.
    UnitUIBroadcaster.Instance.BroadcastUnitWorldUIVisibility(false);
}

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

// ---- Client-notifikaatiot UI:lle ----
[ClientRpc]
public void RpcTurnPhaseChanged(TurnPhase newPhase, int newTurnNumber, bool isPlayersPhase)
{
    // Päivitä paikallinen SP-UI-luuppi (ei Mirror-kutsuja)
    if (TurnSystem.Instance != null)
        TurnSystem.Instance.SetHudFromNetwork(newTurnNumber, isPlayersPhase);

    // Vaihe vaihtui → varmuuden vuoksi piilota mahdollinen "READY" -teksti
    var ui = FindFirstObjectByType<TurnSystemUI>();
    if (ui != null) ui.SetTeammateReady(false, null);
}

// Näyttää toiselle pelaajalle "Player X READY"
[ClientRpc]
public void RpcUpdateReadyStatus(int[] whoEndedIds, string[] whoEndedLabels)
{
    var ui = FindFirstObjectByType<TurnSystemUI>();
    if (ui == null) return;

    // Selvitä oma netId
    uint localId = 0;
    if (NetworkClient.connection != null && NetworkClient.connection.identity)
        localId = NetworkClient.connection.identity.netId;

    bool show = false;
    string label = null;

    // Jos joku muu kuin minä on valmis → näytä hänen labelinsa
    for (int i = 0; i < whoEndedIds.Length; i++)
    {
        if ((uint)whoEndedIds[i] != localId)
```

RogueShooter – All Scripts

```
        {
            show = true;
            label = (i < whoEndedLabels.Length) ? whoEndedLabels[i] : "Teammate";
            break;
        }
    }

    ui.SetTeammateReady(show, label);
}

// ---- Server-apurit ----
[Server] string GetLabelByNetId(uint id)
{
    foreach (var kvp in NetworkServer.connections)
    {
        var conn = kvp.Value;
        if (conn != null && conn.identity && conn.identity.netId == id)
            return conn.connectionId == 0 ? "Player 1" : "Player 2";
    }
    return "Teammate";
}

[Server]
public string[] BuildEndedLabels()
{
    // HashSetin järjestys ei ole merkityksellinen, näytetään mikä tahansa toinen
    return NetTurnManager.Instance.endedPlayers.Select(id => GetLabelByNetId(id)).ToArray();
}
}
```

RogueShooter – All Scripts

Assets/scripts/Online/GameNetworkManager.cs

```
using System;
using System.Collections.Generic;
using Mirror;
using UnityEngine;
using Unity.Services.Relay.Models;

namespace Utp
{
    [RequireComponent(typeof(UtpTransport))]
    public class GameNetworkManager : NetworkManager
    {
        public static GameNetworkManager Instance { get; private set; }
        private UtpTransport utpTransport;

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

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

            base.Awake();

            utpTransport = GetComponent<UtpTransport>();

            string[] args = 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");
                        }
                    }
                }
            }
        }
    }
}
```

RogueShooter – All Scripts

```
    }  
  }  
}  
  
public override void OnStartServer()  
{  
    base.OnStartServer();  
    SpawnUnitsCoordinator.Instance.SetEnemiesSpawned(false);  
  
    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()  
{  
    utpTransport.useRelay = false;  
    StartHost();  
}
```

RogueShooter – All Scripts

```
/// <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)
{
    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.");
    });
}
```

RogueShooter – All Scripts

```
});  
}  
  
public override void OnValidate()  
{  
    base.OnValidate();  
}  
  
/// <summary>  
/// Tämä metodi spawnaa jokaiselle clientille oman Unitin ja tekee siitä heidän ohjattavan yksikkönsä.  
/// </summary>  
public override void OnServerAddPlayer(NetworkConnectionToClient conn)  
{  
  
    if (playerPrefab == null)  
    {  
        Debug.LogError("[NM] Player Prefab (EmptySquad) puuttuu!");  
        return;  
    }  
    base.OnServerAddPlayer(conn);  
  
    // 2) päättää host vs client  
    bool isHost = conn.connectionId == 0;  
  
    // 3) spawnaa pelaajan yksiköt ja anna authority niihin  
    var units = SpawnUnitsCoordinator.Instance.SpawnPlayersForNetwork(conn, isHost);  
    foreach (var unit in units)  
    {  
        NetworkServer.Spawn(unit, conn); // authority tälle pelaajalle  
    }  
  
    // päivitä pelaajamäärä koordinaattorille  
    var coord = NetTurnManager.Instance;  
    //var coord = CoopTurnCoordinator.Instance;  
    if (coord != null)  
        coord.ServerUpdateRequiredCount(NetworkServer.connections.Count);  
  
    // --- VERSUS (PvP) – host aloittaa ---  
    if (GameManager.SelectedMode == GameMode.Versus)  
    {  
        var pc = conn.identity != null ? conn.identity.GetComponent<PlayerController>() : null;  
        if (pc != null && PvPTurnCoordinator.Instance != null)  
        {  
            // Rekisteröi pelaaja PvP-vuoroon (host saa aloitusvuoron PvPTurnCoordinatorissa)  
            PvPTurnCoordinator.Instance.ServerRegisterPlayer(pc);  
        }  
        else  
        {  
            Debug.LogWarning("[NM] PvP rekisteröinti epäonnistui: PlayerController tai PvPTurnCoordinator puuttuu.");  
        }  
    }  
}
```

RogueShooter – All Scripts

```
}

[Server]
public void ServerSpawnEnemies()
{
    // Pyydä SpawnUnitsCoordinatoria luomaan viholliset
    var enemies = SpawnUnitsCoordinator.Instance.SpawnEnemies();

    // Synkronoi viholliset verkkoon Mirrorin avulla
    foreach (var enemy in enemies)
    {
        if (enemy != null)
        {
            NetworkServer.Spawn(enemy);
        }
    }
}

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

public bool IsNetworkActive()
{
    return GetNetWorkServerActive() || GetNetWorkClientConnected();
}

public bool GetNetWorkServerActive()
{
    return NetworkServer.active;
}

public bool GetNetWorkClientConnected()
{
    return NetworkClient.isConnected;
}

public NetworkConnection NetWorkClientConnection()
{
    return NetworkClient.connection;
}

public void NetworkDestroy(GameObject go)
{
    NetworkServer.Destroy(go);
}
```


RogueShooter – All Scripts

```
}  
  
public void SetEnemies()  
{  
    SpawnUnitsCoordinator.Instance.SetEnemiesSpawned(false);  
  
    if (GameManager.SelectedMode == GameMode.CoOp)  
    {  
        ServerSpawnEnemies();  
    }  
}  
  
}  
}
```

RogueShooter – All Scripts

Assets/scripts/Online/NetTurnManager.cs

```
using UnityEngine;
using Mirror;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
///<summary>
/// NetTurnManager coordinates turn phases in a networked multiplayer game.
/// It tracks which players have ended their turns and advances the game phase accordingly.
///</summary>
public enum TurnPhase { Players, Enemy }
public class NetTurnManager : NetworkBehaviour
{
    public static NetTurnManager Instance { get; private set; }
    [SyncVar] public TurnPhase phase = TurnPhase.Players;
    [SyncVar] public int turnNumber = 1;

    // Seurannat (server)
    [SyncVar] public int endedCount = 0;
    [SyncVar] public int requiredCount = 0; // päivitetään kun pelaajia liittyy/lähtee

    public readonly HashSet<uint> endedPlayers = new();

    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 (GameManager.SelectedMode == GameMode.CoOp) requiredCount = 2;
    }

    [Server]
    public void 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>();
```

RogueShooter – All Scripts

```
        if (pc) pc.ServerSetHasEnded(false); // <<< TÄRKEIN RIVI
    }

    // Tyhjennä "Player X READY" -teksti kaikilta. Käytössä vain Co-opissa
    if (GameManager.SelectedMode == GameMode.CoOp)
    {
        CoopTurnCoordinator.Instance.RpcUpdateReadyStatus(System.Array.Empty<int>(), System.Array.Empty<string>());
    }
}

[Server]
public void ServerPlayerEndedTurn(uint playerNetId)
{
    // PvP: siirrä vuoro heti vastustajalle
    if (GameManager.SelectedMode == GameMode.Versus)
    {
        if (PvPTurnCoordinator.Instance)
            PvPTurnCoordinator.Instance.ServerHandlePlayerEndedTurn(playerNetId);
        return;
    }

    if (phase != TurnPhase.Players) return; // ei lasketa jos ei pelaajavuoro
    if (!endedPlayers.Add(playerNetId)) return; // älä laske tuplia

    endedCount = endedPlayers.Count;

    // Ilmoita kaikille, KUKA on valmis → UI näyttää "Player X READY" toisella pelaajalla. Käytössä vain Co-opissa
    if (GameManager.SelectedMode == GameMode.CoOp)
    {
        // Asettaa yksiköiden UI Näkyvyydet
        UnitUIBroadcaster.Instance.BroadcastUnitWorldUIVisibility(false);

        CoopTurnCoordinator.Instance.
            RpcUpdateReadyStatus(
                endedPlayers.Select(id => (int)id).ToArray(),
                CoopTurnCoordinator.Instance.BuildEndedLabels()
            );

        CoopTurnCoordinator.Instance.TryAdvanceIfReady();
    }
}

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

    if (GameManager.SelectedMode == GameMode.CoOp)
    {
        CoopTurnCoordinator.Instance.TryAdvanceIfReady();
    }
}
```

```
}  
}
```

RogueShooter – All Scripts

Assets/scripts/Online/PvpClientState.cs

```
using UnityEngine;
using System;
public class PvpClientState : MonoBehaviour
{
    public static bool IsMyTurn { get; set; }
}

public static class PvpClientEvents
{
    public static event Action<uint, int> OnTurnChanged;

    public static void RaiseTurnChanged(uint turnOwnerNetId, int turnNo)
        => OnTurnChanged?.Invoke(turnOwnerNetId, turnNo);
}
```

RogueShooter – All Scripts

Assets/scripts/Online/PvpPerception.cs

```
using System.Reflection;
using Mirror;
using UnityEngine;

public class PvpPerception : MonoBehaviour
{
    // Kutsu tätä aina kun vuoro vaihtuu (ja bootstrapissa)
    public static void ApplyEnemyFlagsLocally(bool isMyTurn)
    {
        var units = FindObjectsByType<Unit>(FindObjectsSortMode.None);

        foreach (var u in units)
        {
            var ni = u.GetComponent<NetworkIdentity>();
            if (!ni) continue;

            // Onko tämä yksikkö minun (tässä clientissä)?
            bool unitIsMine = ni.isOwned || ni.isLocalPlayer;

            // Vuorologiikka:
            // - Jos on MINUN vuoro: vastustajan yksiköt ovat enemy
            // - Jos EI ole minun vuoro: MINUN omat yksiköt ovat enemy
            bool enemy = isMyTurn ? !unitIsMine : unitIsMine;

            SetUnitEnemyFlag(u, enemy);
        }
    }

    static void SetUnitEnemyFlag(Unit u, bool enemy)
    {
        // Unitissa on [SerializeField] private bool isEnemy; -> käytä BindingFlagsia! :contentReference[oaicite:1]{index=1}
        var field = typeof(Unit).GetField("isEnemy",
            BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public);
        if (field != null) { field.SetValue(u, enemy); return; }

        // Varalle, jos joskus lisääs setterin
        var m = typeof(Unit).GetMethod("SetEnemy",
            BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic,
            null, new[] { typeof(bool) }, null);
        if (m != null) { m.Invoke(u, new object[] { enemy }); return; }

        Debug.LogWarning("[PvP] Unitilta puuttuu isEnemy/SetEnemy(bool). Lisää jompikumpi.");
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Online/PvPTurnCoordinator.cs

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

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

    [SyncVar] private uint currentOwnerNetId; // kumman pelaajan vuoro on

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

    // Kutsutaan, kun pelaaja liittyy. Hostista tehdään aloitusvuoron omistaja.
    [Server]
    public void ServerRegisterPlayer(PlayerController pc)
    {
        // Host (connectionId == 0) asettaa aloitusvuoron, jos ei vielä asetettu
        if (currentOwnerNetId == 0 && pc.connectionToClient != null && pc.connectionToClient.connectionId == 0)
        {
            currentOwnerNetId = pc.netId;
            pc.ServerSetHasEnded(false); // host saa toimia
            foreach (var other in GetAllPlayers().Where(p => p != pc))
                other.ServerSetHasEnded(true); // muut lukkoon varmuudeksi

            RpcTurnChanged(GetTurnNumber(), currentOwnerNetId);
        }
        else
        {
            // Myöhemmin liittynyt (client) - lukitaan kunnes hänen vuoronsa alkaa
            pc.ServerSetHasEnded(true);
            RpcTurnChanged(GetTurnNumber(), currentOwnerNetId);
        }
    }

    // Kutsutaan, kun joku painaa End Turn
    [Server]
    public void ServerHandlePlayerEndedTurn(uint whoEndedNetId)
    {
        var players = GetAllPlayers().ToList();
        var ended = players.FirstOrDefault(p => p.netId == whoEndedNetId);
        var next = players.FirstOrDefault(p => p.netId != whoEndedNetId);
        if (next == null) return; // ei vastustajaa vielä

        // Nosta vuorolaskuria (kierrätetään olemassaolevaa turnNumberia)
        if (NetTurnManager.Instance) NetTurnManager.Instance.turnNumber++;
    }
}
```

RogueShooter – All Scripts

```
currentOwnerNetId = next.netId;

// Anna seuraavalle vuoro
next.ServerSetHasEnded(false); // avaa syötteen ja nappulan
// ended pysyy lukossa (hasEndedThisTurn = true)
RpcTurnChanged(GetTurnNumber(), currentOwnerNetId);
}

int GetTurnNumber() => NetTurnManager.Instance ? NetTurnManager.Instance.turnNumber : 1;

[ClientRpc]
void RpcTurnChanged(int newTurnNumber, uint ownerNetId)
{
    // Päivitä paikallinen HUD "player/enemy turn" -logiikalla
    bool isMyTurn = false;
    if (NetworkClient.connection != null && NetworkClient.connection.identity != null)
        isMyTurn = NetworkClient.connection.identity.netId == ownerNetId;

    PvpPerception.ApplyEnemyFlagsLocally(isMyTurn);

    if (TurnSystem.Instance != null)
        TurnSystem.Instance.SetHudFromNetwork(newTurnNumber, isMyTurn);
}

[Server]
IEnumerable<PlayerController> GetAllPlayers()
{
    foreach (var kvp in NetworkServer.connections)
    {
        var id = kvp.Value.identity;
        if (!id) continue;
        var pc = id.GetComponent<PlayerController>();
        if (pc) yield return pc;
    }
}
}
```


RogueShooter – All Scripts

Assets/scripts/Online/ResetService.cs

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

public class ResetService : NetworkBehaviour
{
    public static ResetService Instance;

    // LIPPU: ajetaan post-reset -alustus, kun uusi scene on valmis
    public static bool PendingHardReset;

    void Awake() => Instance = this;

    [Command(requiresAuthority = false)]
    public void CmdRequestHardReset()
    {
        if (!NetworkServer.active) return;
        HardResetServerAuthoritative();
    }

    [Server]
    public void HardResetServerAuthoritative()
    {
        PendingHardReset = true; // <-- vain lippu päälle
        var nm = (NetworkManager)NetworkManager.singleton;
        var scene = SceneManager.GetActiveScene().name;
        nm.ServerChangeScene(scene);
        // ÄLÄ tee mitään tähän enää
    }

    [ClientRpc]
    public void RpcPostResetClientInit(int turnNumber)
    {
        // odota 1 frame että UI-komponentit ovat ehtineet OnEnable/subscribe
        StartCoroutine(_ClientInitCo(turnNumber));
    }

    private IEnumerator _ClientInitCo(int turnNumber)
    {
        yield return null;

        // 1) Avaa paikallinen "saa toimia" -portti (triggaa LocalPlayerTurnChanged)
        PlayerLocalTurnGate.SetCanAct(true);

        // 2) Päivitä HUD (näyttää "Players turn", aktivoi End Turn -napin logiikkaasi vasten)
        TurnSystem.Instance?.SetHudFromNetwork(turnNumber, true);
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Online/ServerPostResetBootstrap.cs

```
// ServerPostResetBootstrap.cs
using System.Collections;
using Mirror;
using UnityEngine;
using Utp;

public class ServerPostResetBootstrap : NetworkBehaviour
{
    private IEnumerator Start()
    {
        // Ajetaan vain serverillä ja vain resetin jälkeen
        if (!isServer) yield break;
        if (!ResetService.PendingHardReset) yield break;
        ResetService.PendingHardReset = false;

        // Odota 1 frame että kaikki singletons/OnEnable ehtivät alustua
        yield return null;
        yield return null;

        // 1) Nollaa vuorologiikka ja pysäytä vanhat AI-koroutiinat varmuuden vuoksi
        EnemyAI.Instance?.StopAllCoroutines();
        NetTurnManager.Instance.ResetTurnState();

        // 2) Spawnaa viholliset serveriltä
        GameNetworkManager.Instance.SetEnemies();

        // 3) Rakenna occupancy
        LevelGrid.Instance?.RebuildOccupancyFromScene();

        // 4) Pakota UI Player-vuoroon ja synkkaa kaikille
        NetTurnManager.Instance.turnNumber = 1;
        NetTurnManager.Instance.phase = TurnPhase.Players;

        // SP/UI-puolelle
        TurnSystem.Instance?.ForcePhase(isPlayerTurn: true, incrementTurnNumber: false);

        // Co-op: tyhjennä ready/ended ja kerro vaiheesta
        var ended = System.Array.Empty<int>();
        CoopTurnCoordinator.Instance?.RpcUpdateReadyStatus(
            ended,
            CoopTurnCoordinator.Instance.BuildEndedLabels()
        );

        CoopTurnCoordinator.Instance?.RpcTurnPhaseChanged(
            NetTurnManager.Instance.phase,
            NetTurnManager.Instance.turnNumber,
            false // ei enemy
        );

        // Vihollisten world-UI piiloon
```

RogueShooter – All Scripts

```
UnitUIBroadcaster.Instance?.BroadcastUnitWorldUIVisibility(false);  
// Kerro kaikille klienteille, että nyt saa toimia + päivitä HUD ***  
ResetService.Instance.RpcPostResetClientInit(NetTurnManager.Instance.turnNumber);  
}  
}
```

RogueShooter – All Scripts

Assets/scripts/Online/Sync/NetworkSync.cs

```
using Mirror;
using Mirror.Examples.CharacterSelection;
using UnityEngine;

/// <summary>
/// NetworkSync is a static helper class that centralizes all network-related actions.
///
/// Responsibilities:
/// - Provides a single entry point for spawning and synchronizing networked effects and objects.
/// - Decides whether the game is running in server/host mode, client mode, or offline mode.
/// - In online play:
///   - If running on the server/host, spawns objects directly with NetworkServer.Spawn.
///   - If running on a client, forwards the request to the local NetworkSyncAgent, which relays it to the server via Command.
/// - In offline/singleplayer mode, simply instantiates objects locally with Instantiate.
///
/// Usage:
/// Call the static methods from gameplay code (e.g. UnitAnimator, Actions) instead of
/// directly instantiating or spawning prefabs. This ensures consistent behavior in all game modes.
///
/// Example:
/// NetworkSync.SpawnBullet(bulletPrefab, shootPoint.position, targetPosition);
/// </summary>
public static class NetworkSync
{
    /// <summary>
    /// Spawns a bullet projectile in the game world.
    /// Handles both offline (local Instantiate) and online (NetworkServer.Spawn) scenarios.
    ///
    /// In server/host:
    ///   - Instantiates and spawns the bullet directly with NetworkServer.Spawn.
    /// In client:
    ///   - Forwards the request to NetworkSyncAgent.Local, which executes a Command.
    /// In offline:
    ///   - Instantiates the bullet locally.
    /// </summary>
    /// <param name="bulletPrefab">The bullet prefab to spawn (must have NetworkIdentity if used online).</param>
    /// <param name="spawnPos">The starting position of the bullet (usually weapon muzzle).</param>
    /// <param name="targetPos">The target world position the bullet should travel towards.</param>
    public static void SpawnBullet(GameObject bulletPrefab, Vector3 spawnPos, Vector3 targetPos)
    {
        if (NetworkServer.active) // Online: server or host
        {
            var go = Object.Instantiate(bulletPrefab, spawnPos, Quaternion.identity);
            if (go.TryGetComponent<BulletProjectile>(out var bp))
                bp.Setup(new Vector3(targetPos.x, spawnPos.y, targetPos.z));
            NetworkServer.Spawn(go);
            return;
        }
    }
}
```

RogueShooter – All Scripts

```
if (NetworkClient.active) // Online: client
{
    if (NetworkSyncAgent.Local != null)
    {
        NetworkSyncAgent.Local.CmdSpawnBullet(spawnPos, targetPos);
    }
    else
    {
        // fallback if no local agent found (shouldn't happen in a correct setup)
        Debug.LogWarning("[NetworkSync] No Local NetworkSyncAgent found, falling back to local Instantiate.");
        var go = Object.Instantiate(bulletPrefab, spawnPos, Quaternion.identity);
        if (go.TryGetComponent<BulletProjectile>(out var bp))
            bp.Setup(new Vector3(targetPos.x, spawnPos.y, targetPos.z));
    }
}
else
{
    // Offline / Singleplayer: just instantiate locally
    var go = Object.Instantiate(bulletPrefab, spawnPos, Quaternion.identity);
    if (go.TryGetComponent<BulletProjectile>(out var bp))
        bp.Setup(new Vector3(targetPos.x, spawnPos.y, targetPos.z));
}
}

/// <summary>
/// Apply damage to a Unit in SP/Host/Client modes.
/// - Server/Host: call HealthSystem.Damage directly (authoritative).
/// - Client: send a Command via NetworkSyncAgent to run on server.
/// - Offline: call locally.
/// </summary>
public static void ApplyDamage(Unit target, int amount)
{
    if (target == null) return;

    if (NetworkServer.active) // Online: server or host
    {
        var healthSystem = target.GetComponent<HealthSystem>();
        if (healthSystem == null) return;

        healthSystem.Damage(amount);
        UpdateHealthBarUI(healthSystem, target);
        return;
    }

    if (NetworkClient.active) // Online: client
    {
        var ni = target.GetComponent<NetworkIdentity>();
        if (ni && NetworkSyncAgent.Local != null)
        {
            NetworkSyncAgent.Local.CmdApplyDamage(ni.netId, amount);
            return;
        }
    }
}
```

RogueShooter – All Scripts

```
    }

    // Offline fallback
    target.GetComponent<HealthSystem>()?.Damage(amount);
}

private static void UpdateHealthBarUI(HealthSystem healthSystem, Unit target)
{
    // → ilmoita kaikille clientele, jotta UnitWorldUI saa eventin
    if (NetworkSyncAgent.Local == null)
    {
        // haetaan mikä tahansa agentti serveriltä (voi olla erillinen manageri)
        var agent = Object.FindFirstObjectByType<NetworkSyncAgent>();
        if (agent != null)
            agent.ServerBroadcastHp(target, healthSystem.GetHealth(), healthSystem.GetHealthMax());
    }
    else
    {
        NetworkSyncAgent.Local.ServerBroadcastHp(target, healthSystem.GetHealth(), healthSystem.GetHealthMax());
    }
}

public static void BroadcastActionPoints(Unit unit, int apValue)
{
    if (unit == null) return;

    if (NetworkServer.active)
    {
        var agent = Object.FindFirstObjectByType<NetworkSyncAgent>();
        if (agent != null)
            agent.ServerBroadcastAp(unit, apValue);
        return;
    }

    // CLIENT-haara: lähetä peilauspyyntö serverille
    if (NetworkClient.active && NetworkSyncAgent.Local != null)
    {
        var ni = unit.GetComponent<NetworkIdentity>();
        if (ni) NetworkSyncAgent.Local.CmdMirrorAp(ni.netId, apValue);
    }
}

/// <summary>
/// Server: Control when Pleyers can see own and others Unit stats,
/// Like only active player AP(Action Points) are visible.
/// When is Enemy turn only Enemy Units Action points are visible.
/// Solo and Versus mode handle this locally because there is no need synchronisation.
/// </summary>

public static void SpawnRagdoll(GameObject prefab, Vector3 pos, Quaternion rot, uint sourceUnitNetId, Transform originalRootBone)
{
    if (NetworkServer.active)
```

RogueShooter – All Scripts

```
{
    var go = Object.Instantiate(prefab, pos, rot);

    if (go.TryGetComponent<RagdollPoseBinder>(out var ragdollPoseBinder))
    {
        ragdollPoseBinder.sourceUnitNetId = sourceUnitNetId;
    }
    else
    {
        Debug.LogWarning("[Ragdoll] Ragdoll prefab lacks RagdollPoseBinder component.");
    }

    NetworkServer.Spawn(go);
    return;
}

// offline fallback
var off = Object.Instantiate(prefab, pos, rot);
if (off.TryGetComponent<UnitRagdoll>(out var unitRagdoll))
    unitRagdoll.Setup(originalRootBone);
}
```

RogueShooter – All Scripts

Assets/scripts/Online/Sync/NetworkSyncAgent.cs

```
using Mirror;
using UnityEngine;
/// <summary>
/// NetworkSyncAgent is a helper NetworkBehaviour to relay Commands from clients to the server.
/// Each client should have exactly one instance of this script in the scene, usually attached to the PlayerController GameObject.
///
/// Responsibilities:
/// - Receives local calls from NetworkSync (static helper).
/// - Sends Commands to the server when the local player performs an action (e.g. shooting).
/// - On the server, instantiates and spawns networked objects (like projectiles).
/// </summary>
public class NetworkSyncAgent : NetworkBehaviour
{
    public static NetworkSyncAgent Local; // Easy access for NetworkSync static helper
    [SerializeField] private GameObject bulletPrefab; // Prefab for the bullet projectile

    public override void OnStartLocalPlayer()
    {
        base.OnStartLocalPlayer();
        Local = this;
    }

    /// <summary>
    /// Command from client → server.
    /// The client requests the server to spawn a bullet at the given position.
    /// The server instantiates the prefab, sets it up, and spawns it to all connected clients.
    /// </summary>
    /// <param name="spawnPos">World position where the bullet starts (usually weapon muzzle).</param>
    /// <param name="targetPos">World position the bullet is travelling towards.</param>
    [Command(requiresAuthority = true)]
    public void CmdSpawnBullet(Vector3 spawnPos, Vector3 targetPos)
    {
        if (bulletPrefab == null) { Debug.LogWarning("[NetSync] bulletPrefab missing"); return; }

        // Instantiate on the server
        var go = Instantiate(bulletPrefab, spawnPos, Quaternion.identity);

        // Setup target on the projectile
        if (go.TryGetComponent<BulletProjectile>(out var bp))
        {
            bp.Setup(new Vector3(targetPos.x, spawnPos.y, targetPos.z));
        }

        // Spawn across the network
        NetworkServer.Spawn(go);
    }

    /// <summary>
    /// Client → Server: resolve target by netId and apply damage on server.
    /// then broadcast the new HP to all clients for UI.
    </summary>
}
```


RogueShooter – All Scripts

```
/// </summary>
[Command(requiresAuthority = true)]
public void CmdApplyDamage(uint targetNetId, int amount)
{
    if (!NetworkServer.spawned.TryGetValue(targetNetId, out var targetNi) || targetNi == null)
        return;

    var unit = targetNi.GetComponent<Unit>();
    var hs = targetNi.GetComponent<HealthSystem>();
    if (unit == null || hs == null)
        return;

    // 1) Server tekee damagen (kuten ennenkin)
    hs.Damage(amount);

    // 2) Heti perään broadcast → kaikki clientit päivittävät oman UI:nsa
    //     (ServerBroadcastHp kutsuu RpcNotifyHpChanged → hs.ApplyNetworkHealth(..) clientillä)
    ServerBroadcastHp(unit, hs.GetHealth(), hs.GetHealthMax());
}

// ---- SERVER-puolen helperit: kutsu näitä palvelimelta
[Server]
public void ServerBroadcastHp(Unit unit, int current, int max)
{
    var ni = unit.GetComponent<NetworkIdentity>();
    if (ni) RpcNotifyHpChanged(ni.netId, current, max);
}

[Server]
public void ServerBroadcastAp(Unit unit, int ap)
{
    var ni = unit.GetComponent<NetworkIdentity>();
    if (ni) RpcNotifyApChanged(ni.netId, ap);
}

// ---- SERVER → ALL CLIENTS: HP-muutos ilmoitus
[ClientRpc]
void RpcNotifyHpChanged(uint unitNetId, int current, int max)
{
    if (!NetworkClient.spawned.TryGetValue(unitNetId, out var id) || id == null) return;

    var hs = id.GetComponent<HealthSystem>();
    if (hs == null) return;

    hs.ApplyNetworkHealth(current, max);
}

// ---- SERVER → ALL CLIENTS: AP-muutos ilmoitus
[ClientRpc]
void RpcNotifyApChanged(uint unitNetId, int ap)
{
    ApplyApClient(unitNetId, ap);
}
```

RogueShooter – All Scripts

```
}

[Command]
public void CmdMirrorAp(uint unitNetId, int ap)
{
    RpcNotifyApChanged(unitNetId, ap);
}

void ApplyApClient(uint unitNetId, int ap)
{
    if (!NetworkClient.spawned.TryGetValue(unitNetId, out var id) || id == null) return;
    var unit = id.GetComponent<Unit>();
    if (!unit) return;

    unit.ApplyNetworkActionPoints(ap); // päivittää arvon + triggaa eventin
}
}
```

RogueShooter – All Scripts

Assets/scripts/Player/PlayerController.cs

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

///<summary>
/// PlayerController handles per-player state in a networked game.
/// Each connected player has one PlayerController instance attached to PlayerController GameObject prefab
/// It tracks whether the player has ended their turn and communicates with the UI.
///</summary>
public class PlayerController : NetworkBehaviour
{
    [SyncVar] public bool hasEndedThisTurn;

    public static PlayerController Local; // helppo viittaus UI:lle

    public override void OnStartLocalPlayer()
    {
        base.OnStartLocalPlayer();
        Local = this;
    }

    // UI-nappi kutsuu tätä (vain local player)
    public void ClickEndTurn()
    {
        if (!isLocalPlayer) return;
        if (hasEndedThisTurn) return;
        if (NetTurnManager.Instance && NetTurnManager.Instance.phase != TurnPhase.Players) return;
        CmdEndTurn();
    }

    [Command(requiresAuthority = true)]
    void CmdEndTurn()
    {
        if (hasEndedThisTurn) return;
        hasEndedThisTurn = true;

        // Estä kaikki toiminnot clientillä
        TargetNotifyCanAct(connectionToClient, false);

        // Varmista myös että koordinaattori löytyy serveripuolelta:
        if (NetTurnManager.Instance == null)
        {
            Debug.LogWarning("[PC][SERVER] NetTurnManager.Instance is NULL on server!");
            return;
        }

        NetTurnManager.Instance.ServerPlayerEndedTurn(netIdentity.netId);
    }
}
```

RogueShooter – All Scripts

```
// Server kutsuu tämän kierroksen alussa nollatakseen tilan
[Server]
public void ServerSetHasEnded(bool v)
{
    hasEndedThisTurn = v;
    TargetNotifyCanAct(connectionToClient, !v);
}

[TargetRpc]
void TargetNotifyCanAct(NetworkConnectionToClient __, bool canAct)
{
    // Update End Turn Button
    var ui = FindFirstObjectByType<TurnSystemUI>();
    if (ui != null)
        ui.SetCanAct(canAct);
    if (!canAct) ui.SetTeammateReady(false, null);

    // Lock/Unlock UnityActionSystem input
    if (UnityActionSystem.Instance != null)
    {
        if (canAct) UnityActionSystem.Instance.UnlockInput();
        else UnityActionSystem.Instance.LockInput();
    }

    // Set AP visibility in versus game
    PlayerLocalTurnGate.Set(canAct);
}
}
```

RogueShooter – All Scripts

Assets/scripts/Player/PlayerLocalTurnGate.cs

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

/// <summary>
/// Static gate that tracks whether the local player turn is. (e.g., enabling/disabling UI).
/// Other systems can subscribe to the <see cref="LocalPlayerTurnChanged"/> event to update their state
/// </summary>
///
public static class PlayerLocalTurnGate
{
    // public static int PlayerReady { get; private set; }

    // public static event Action<int> OnPlayerReadyChanged;
    /// <summary>
    /// Gets whether the local player can currently act.
    /// </summary>
    public static bool LocalPlayerTurn { get; private set; }

    /// <summary>
    /// Event fired whenever the <see cref="LocalPlayerTurn"/> state changes.
    /// The bool argument indicates the new state.
    /// </summary>
    public static event Action<bool> LocalPlayerTurnChanged;

    /// <summary>
    /// Updates the <see cref="LocalPlayerTurn"/> state.
    /// If the value changes, invokes <see cref="LocalPlayerTurnChanged"/> to notify listeners.
    /// </summary>
    /// <param name="canAct">True if the player may act; false otherwise.</param>
    public static void Set(bool canAct)
    {
        if (LocalPlayerTurn == canAct) return;
        LocalPlayerTurn = canAct;
        LocalPlayerTurnChanged?.Invoke(LocalPlayerTurn);
    }

    public static void SetCanAct(bool canAct)
    {
        LocalPlayerTurn = canAct;
        LocalPlayerTurnChanged?.Invoke(LocalPlayerTurn);
    }
}
```

RogueShooter – All Scripts

Assets/scripts/SpawnUnitsCoordinator.cs

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

public class SpawnUnitsCoordinator : MonoBehaviour
{
    public static SpawnUnitsCoordinator Instance { get; private set; }
    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),
    };

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

    public GameObject[] SpawnPlayersForNetwork(NetworkConnectionToClient conn, bool isHost)
    {
        GameObject unitPrefab = GetUnitPrefabForPlayer(isHost);
        Vector3[] spawnPoints = GetSpawnPositionsForPlayer(isHost);

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

```

RogueShooter – All Scripts

```
        Debug.LogError($"[NM] {(isHost ? "hostSpawnPositions" : "clientSpawnPositions")} ei ole asetettu!");
        return null;
    }

    var spawnedPlayersUnit = new GameObject[spawnPoints.Length];
    for (int i = 0; i < spawnPoints.Length; i++)
    {
        var playerUnit = Instantiate(unitPrefab, spawnPoints[i], Quaternion.identity);
        if (playerUnit.TryGetComponent<Unit>(out var u) && conn.identity != null)
            u.OwnerId = conn.identity.netId;
        spawnedPlayersUnit[i] = playerUnit;
    }

    return spawnedPlayersUnit;
}

public GameObject GetUnitPrefabForPlayer(bool isHost)
{
    if (unitHostPrefab == null || unitClientPrefab == null)
    {
        Debug.LogError("Unit prefab references not set in SpawnUnitsCoordinator!");
        return null;
    }

    return isHost ? unitHostPrefab : unitClientPrefab;
}

public Vector3[] GetSpawnPositionsForPlayer(bool isHost)
{
    if (hostSpawnPositions.Length == 0 || clientSpawnPositions.Length == 0)
    {
        Debug.LogError("Spawn position arrays not set in SpawnUnitsCoordinator!");
        return new Vector3[0];
    }

    return isHost ? hostSpawnPositions : clientSpawnPositions;
}

public GameObject[] SpawnEnemies()
{
    var spawnedEnemies = new GameObject[enemySpawnPositions.Length];

    for (int i = 0; i < enemySpawnPositions.Length; i++)
    {
        var enemy = Instantiate(GetEnemyPrefab(), enemySpawnPositions[i], Quaternion.identity);
        spawnedEnemies[i] = enemy;
    }

    SetEnemiesSpawned(true);
    return spawnedEnemies;
}
```

RogueShooter – All Scripts

```
public Vector3[] GetEnemySpawnPositions()
{
    if (enemySpawnPositions.Length == 0)
    {
        Debug.LogError("Enemy spawn position array not set in SpawnUnitsCoordinator!");
        return new Vector3[0];
    }

    return enemySpawnPositions;
}

public void SetEnemiesSpawned(bool value)
{
    enemiesSpawned = value;
}
public bool AreEnemiesSpawned()
{
    return enemiesSpawned;
}

public GameObject GetEnemyPrefab()
{
    if (enemyPrefab == null)
    {
        Debug.LogError("Enemy prefab reference not set in SpawnUnitsCoordinator!");
        return null;
    }
    return enemyPrefab;
}

public void SpwanSinglePlayerUnits()
{
    SpawnPlayer1UnitsOffline();
    SpawnEnemyUnitsOffline();
}

// Singleplayer Gamemode Spawn units. hardcoded for now.
// Later we can make it more generic with arrays and prefabs like in Co-op.
private void SpawnPlayer1UnitsOffline()
{
    Instantiate(unitHostPrefab, hostSpawnPositions[0], Quaternion.identity);
    Instantiate(unitHostPrefab, hostSpawnPositions[1], Quaternion.identity);
}
private void SpawnEnemyUnitsOffline()
{
    Instantiate(enemyPrefab, enemySpawnPositions[0], Quaternion.identity);
    Instantiate(enemyPrefab, enemySpawnPositions[1], Quaternion.identity);
}
}
```


RogueShooter – All Scripts

Assets/scripts/Testing.cs

```
using System.Collections.Generic;
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))
        {
            GridPosition mouseGridPosition = LevelGrid.Instance.GetGridPosition(MouseWorld.GetMouseWorldPosition());
            GridPosition startGridPosition = new GridPosition(0, 0);

            List<GridPosition> gridPositionList = PathFinding.Instance.FindPath(startGridPosition, mouseGridPosition);

            for (int i = 0; i < gridPositionList.Count - 1; i++)
            {
                Debug.DrawLine(
                    LevelGrid.Instance.GetWorldPosition(gridPositionList[i]),
                    LevelGrid.Instance.GetWorldPosition(gridPositionList[i + 1]),
                    Color.white,
                    10f
                );
            }
        }

        //Resetoi pelin alkamaan alusta.
        if (Input.GetKeyDown(KeyCode.R))
        {
            if (Mirror.NetworkServer.active) {
                ResetService.Instance.HardResetServerAuthoritative();
            } else if (Mirror.NetworkClient.active) {
                // käskyt serveriä
                ResetService.Instance.CmdRequestHardReset();
            } else {
                GameReset.HardReloadSceneKeepMode();
            }
        }
    }
}
```

```
}

```

RogueShooter – All Scripts

Assets/scripts/Units/EmptySquad.cs

```
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
{

}
```

RogueShooter – All Scripts

Assets/scripts/Units/HealthSystem.cs

```
using System;
using UnityEngine;

public class HealthSystem : MonoBehaviour
{
    public event EventHandler OnDead;
    public event EventHandler OnDamaged;

    [SerializeField] private int health = 100;
    private int healthMax;

    void Awake()
    {
        healthMax = health;
    }

    public void Damage(int damageAmount)
    {
        health -= damageAmount;
        if (health <= 0)
        {
            health = 0;
            Die();
        }

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

    private void Die()
    {
        OnDead?.Invoke(this, EventArgs.Empty);
    }

    public float GetHealthNormalized()
    {
        return (float)health / healthMax;
    }

    public int GetHealth()
    {
        return health;
    }

    public int GetHealthMax()
    {
        return healthMax;
    }

    public void ApplyNetworkHealth(int current, int max)
```

RogueShooter – All Scripts

```
{  
    healthMax = Mathf.Max(1, max);  
    health    = Mathf.Clamp(current, 0, healthMax);  
    OnDamaged?.Invoke(this, EventArgs.Empty);  
}  
}
```

RogueShooter – All Scripts

Assets/scripts/Units/PathFinding.cs

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

/// <summary>
/// Finds a shortest path on a grid between two grid cells using the A* algorithm
/// with 8-directional movement (N, NE, E, SE, S, SW, W, NW).
/// </summary>
public class PathFinding : MonoBehaviour
{
    public static PathFinding Instance { get; private set; }

    /// <summary>
    /// Movement cost for a straight (orthogonal) step.
    /// </summary>
    private const int MOVE_STRAIGHT_COST = 10;

    /// <summary>
    /// Movement cost for a diagonal step.
    /// </summary>
    private const int MOVE_DIAGONAL_COST = 14;

    /// <summary>
    /// (Optional) Prefab used to draw debug visuals for the grid.
    /// </summary>
    [SerializeField] private Transform gridDebugPrefab;
    private int width;
    private int height;
    private float cellSize;

    /// <summary>
    /// Logical grid holding <see cref="PathNode"/> objects used by A*.
    /// </summary>
    private GridSystem<PathNode> gridSystem;

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

        gridSystem = new GridSystem<PathNode>(10, 10, 2f,
            (GridSystem<PathNode> g, GridPosition gridPosition) => new PathNode(gridPosition));
        gridSystem.CreateDebugObjects(gridDebugPrefab);
    }
}
```

RogueShooter – All Scripts

```
}

/// <summary>
/// Computes the shortest path from <paramref name="startGridPosition"/> to <paramref name="endGridPosition"/>
/// using A* search. Allows both orthogonal and diagonal moves.
/// </summary>
/// <param name="startGridPosition">Start cell in grid coordinates.</param>
/// <param name="endGridPosition">Target cell in grid coordinates.</param>
/// <returns>
/// Ordered list of grid positions from start to end (inclusive) if a path exists;
/// otherwise <c>null</c>.
/// </returns>
public List<GridPosition> FindPath(GridPosition startGridPosition, GridPosition endGridPosition)
{
    List<PathNode> openList = new();
    List<PathNode> closedList = new();

    PathNode startNode = gridSystem.GetGridObject(startGridPosition);
    PathNode endNode = gridSystem.GetGridObject(endGridPosition);

    openList.Add(startNode);

    // Initialize all nodes with "infinite" g-cost and clear path data.
    for (int x = 0; x < gridSystem.GetWidth(); x++)
    {
        for (int z = 0; z < gridSystem.GetHeight(); z++)
        {
            GridPosition gridPosition = new GridPosition(x, z);
            PathNode pathNode = gridSystem.GetGridObject(gridPosition);

            pathNode.SetGCost(int.MaxValue);
            pathNode.SetHCost(0);
            pathNode.CalculateFCost();
            pathNode.ResetCameFromPathNode();
        }
    }

    // Seed start node.
    startNode.SetGCost(0);
    startNode.SetHCost(CalculateDistance(startGridPosition, endGridPosition));
    startNode.CalculateFCost();

    // A* loop.
    while (openList.Count > 0)
    {
        PathNode currentNode = GetLowestFCostPathNode(openList);

        // Goal reached: reconstruct and return path.
        if (currentNode == endNode)
        {
            return CalculatePath(endNode);
        }
    }
}
```

RogueShooter – All Scripts

```
    }

    openList.Remove(currentNode);
    closedList.Add(currentNode);

    foreach (PathNode neighbourNode in GetNeighbourList(currentNode))
    {
        if (closedList.Contains(neighbourNode))
        {
            continue;
        }

        int tentativeGCost =
            currentNode.GetGCost() + CalculateDistance(currentNode.GetGridPosition(), neighbourNode.GetGridPosition());

        // Found a cheaper path to neighbour: update its scores and parent.
        if (tentativeGCost < neighbourNode.GetGCost())
        {
            neighbourNode.SetCameFromPathNode(currentNode);
            neighbourNode.SetGCost(tentativeGCost);
            neighbourNode.SetHCost(CalculateDistance(neighbourNode.GetGridPosition(), endGridPosition));
            neighbourNode.CalculateFCost();

            if (!openList.Contains(neighbourNode))
            {
                openList.Add(neighbourNode);
            }
        }
    }

    // No Path found
    return null;
}

/// <summary>
/// Heuristic + step cost between two grid positions assuming 8-directional movement:
/// uses the standard "octile" distance with straight and diagonal step costs.
/// </summary>
/// <param name="gridPositionA">First grid position.</param>
/// <param name="gridPositionB">Second grid position.</param>
/// <returns>Estimated movement cost from A to B.</returns>
public int CalculateDistance(GridPosition gridPositionA, GridPosition gridPositionB)
{
    GridPosition gridPositionDistance = gridPositionA - gridPositionB;
    int xDistance = Mathf.Abs(gridPositionDistance.x);
    int zDistance = Mathf.Abs(gridPositionDistance.z);
    int remaining = Math.Abs(xDistance - zDistance);
    return MOVE_DIAGONAL_COST * Mathf.Min(xDistance, zDistance) + MOVE_STRAIGHT_COST * remaining;
}

/// <summary>
```


RogueShooter – All Scripts

```
/// Returns the node with the lowest f-cost from the given list.
/// </summary>
/// <param name="pathNodeList">Candidate nodes (typically the open list).</param>
/// <returns>Node with the smallest f-cost.</returns>
private PathNode GetLowestFCostPathNode(List<PathNode> pathNodeList)
{
    PathNode lowestFCostPathNode = pathNodeList[0];
    for (int i = 0; i < pathNodeList.Count; i++)
    {
        if (pathNodeList[i].GetFCost() < lowestFCostPathNode.GetFCost())
        {
            lowestFCostPathNode = pathNodeList[i];
        }
    }
    return lowestFCostPathNode;
}

/// <summary>
/// Gets the path node at grid coordinates (x, z).
/// </summary>
private PathNode GetNode(int x, int z)
{
    return gridSystem.GetGridObject(new GridPosition(x, z));
}

/// <summary>
/// Returns all valid 8-directional neighbours of the given node (clamped to grid bounds).
/// Order: left (and diagonals), right (and diagonals), then vertical up/down.
/// </summary>
/// <param name="currentNode">Node whose neighbours are requested.</param>
/// <returns>List of neighbouring <see cref="PathNode"/> objects.</returns>
private List<PathNode> GetNeighbourList(PathNode currentNode)
{
    List<PathNode> neighbourList = new();

    GridPosition gridPosition = currentNode.GetGridPosition();

    if (gridPosition.x - 1 >= 0)
    {
        // Left
        neighbourList.Add(GetNode(gridPosition.x - 1, gridPosition.z + 0));

        if (gridPosition.z - 1 >= 0)
        {
            // Left Down
            neighbourList.Add(GetNode(gridPosition.x - 1, gridPosition.z - 1));
        }

        if (gridPosition.z + 1 < gridSystem.GetHeight())
        {
            // left Up
            neighbourList.Add(GetNode(gridPosition.x - 1, gridPosition.z + 1));
        }
    }
}
```

RogueShooter – All Scripts

```
    }
}

if (gridPosition.x + 1 < gridSystem.GetWidth())
{
    // Right
    neighbourList.Add(GetNode(gridPosition.x + 1, gridPosition.z + 0));

    if (gridPosition.z - 1 >= 0)
    {
        // Right Down
        neighbourList.Add(GetNode(gridPosition.x + 1, gridPosition.z - 1));
    }

    if (gridPosition.z + 1 < gridSystem.GetHeight())
    {
        // Right Up
        neighbourList.Add(GetNode(gridPosition.x + 1, gridPosition.z + 1));
    }
}

if (gridPosition.z - 1 >= 0)
{
    // Down
    neighbourList.Add(GetNode(gridPosition.x - 0, gridPosition.z - 1));
}
if (gridPosition.z + 1 < gridSystem.GetHeight())
{
    // Up
    neighbourList.Add(GetNode(gridPosition.x + 0, gridPosition.z + 1));
}

return neighbourList;
}

/// <summary>
/// Reconstructs the path by walking back from <paramref name="endNode"/> via CameFrom pointers,
/// then converts it into a list of <see cref="GridPosition"/>s from start to end.
/// </summary>
/// <param name="endNode">Goal node reached by A*.</param>
/// <returns>Ordered list of grid positions representing the path.</returns>
private List<GridPosition> CalculatePath(PathNode endNode)
{
    List<PathNode> pathNodeList = new List<PathNode>();
    pathNodeList.Add(endNode);
    PathNode currentNode = endNode;
    while (currentNode.GetCameFromPathNode() != null)
    {
        pathNodeList.Add(currentNode.GetCameFromPathNode());
        currentNode = currentNode.GetCameFromPathNode();
    }
}
```

RogueShooter – All Scripts

```
        pathNodeList.Reverse();

        List<GridPosition> gridPositionList = new();
        foreach (PathNode pathNode in pathNodeList)
        {
            gridPositionList.Add(pathNode.GetGridPosition());
        }

        return gridPositionList;
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Units/Unit.cs

```
using Mirror;
using System;
using System.Collections;
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>
[RequireComponent(typeof(HealthSystem))]
[RequireComponent(typeof(MoveAction))]
[RequireComponent(typeof(SpinAction))]
public class Unit : NetworkBehaviour
{
    private const int ACTION_POINTS_MAX = 2;

    [SyncVar] public uint OwnerId;

    public static event EventHandler OnAnyActionPointsChanged;
    public static event EventHandler OnAnyUnitSpawned;
    public static event EventHandler OnAnyUnitDead;

    [SerializeField] public bool isEnemy;

    private GridPosition gridPosition;
    private HealthSystem healthSystem;

    private BaseAction[] baseActionsArray;

    private int actionPoints = ACTION_POINTS_MAX;

    private void Awake()
    {
        healthSystem = GetComponent<HealthSystem>();
        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;
    }
}
```

RogueShooter – All Scripts

```
healthSystem.OnDead += HealthSystem_OnDead;

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

private void Update()
{
    GridPosition newGridPosition = LevelGrid.Instance.GetGridPosition(transform.position);
    if (newGridPosition != gridPosition)
    {
        GridPosition oldGridposition = gridPosition;
        gridPosition = newGridPosition;
        LevelGrid.Instance.UnitMoveToGridPosition(oldGridposition, newGridPosition, this);
    }
}

/// <summary>
///     When unit get destroyed, this clears grid system under destroyed unit.
/// </summary>
void OnDestroy()
{
    if (LevelGrid.Instance != null)
    {
        gridPosition = LevelGrid.Instance.GetGridPosition(transform.position);
        LevelGrid.Instance.RemoveUnitAtGridPosition(gridPosition, this);
    }
}

public T GetAction<T>() where T : BaseAction
{
    foreach (BaseAction baseAction in baseActionsArray)
    {
        if (baseAction is T t)
        {
            return t;
        }
    }
    return null;
}

public GridPosition GetGridPosition()
{
    return gridPosition;
}

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

RogueShooter – All Scripts

```
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())
    {
        return true;
    }
    return false;
}

private void SpendActionPoints(int amount)
{
    actionPoints -= amount;

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

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)
{
    actionPoints = ACTION_POINTS_MAX;
    OnAnyActionPointsChanged?.Invoke(this, EventArgs.Empty);
}

/// <summary>
///     Online: Updating ActionPoints usage to otherplayers.
/// </summary>
public void ApplyNetworkActionPoints(int ap)
{
    if (actionPoints == ap) return;
```

RogueShooter – All Scripts

```
        actionPoints = ap;
        OnAnyActionPointsChanged?.Invoke(this, EventArgs.Empty);
    }

    public bool IsEnemy()
    {
        return isEnemy;
    }

    private void HealthSystem_OnDead(object sender, System.EventArgs e)
    {
        OnAnyUnitDead?.Invoke(this, EventArgs.Empty);

        if (!NetworkServer.active)
        {
            Destroy(gameObject);
            return;
        }

        // Online: Hide Unit before destroy it, so that client have time to create own ragdoll from original Unit pose.
        // After some time hidden Unit get destroyed.

        SetSoftHiddenLocal(true);
        RpcSetSoftHidden(true);
        StartCoroutine(DestroyAfter(0.30f));
    }

    private IEnumerator DestroyAfter(float seconds)
    {
        yield return new WaitForSeconds(seconds);
        NetworkServer.Destroy(gameObject);
    }

    [ClientRpc]
    private void RpcSetSoftHidden(bool hidden)
    {
        SetSoftHiddenLocal(hidden);
    }

    private void SetSoftHiddenLocal(bool hidden)
    {
        foreach (var r in GetComponentsInChildren<Renderer>(true))
            r.enabled = !hidden;

        foreach (var c in GetComponentsInChildren<Collider>(true))
            c.enabled = !hidden;

        if (TryGetComponent<Animator>(out var anim))
            anim.enabled = !hidden;
    }
}
```

RogueShooter - All Scripts

```
public float GetHealthNormalized()
{
    return healthSystem.GetHealthNormalized();
}

}
```


RogueShooter – All Scripts

Assets/scripts/Units/UnitActions/Actions/BaseAction.cs

```
using UnityEngine;
using Mirror;
using System;
using System.Collections.Generic;
using Mirror.BouncyCastle.Asn1.Esf;

/// <summary>
/// Base class for all unit actions in the game.
/// This class inherits from NetworkBehaviour and provides common functionality for unit actions.
/// </summary>
[RequireComponent(typeof(Unit))]
public abstract class BaseAction : NetworkBehaviour
{
    public static event EventHandler OnAnyActionStarted;
    public static event EventHandler OnAnyActionCompleted;

    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;

        OnAnyActionStarted?.Invoke(this, EventArgs.Empty);
    }
}
```

RogueShooter – All Scripts

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

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

public Unit GetUnit()
{
    return unit;
}

// ----- ENEMY AI ACTIONS -----

/// <summary>
/// ENEMY AI:
/// Empty ENEMY AI ACTIONS abstract class.
/// Every Unit action like MoveAction.cs, ShootAction.cs and so on defines this differently
/// Contains gridposition and action value
/// </summary>
public abstract EnemyAIAction GetEnemyAIAction(GridPosition gridPosition);

/// <summary>
/// ENEMY AI:
/// Making a list all possible actions an enemy Unit can take, and shorting them
/// based on highest action value.(Gives the enemy the best outcome)
/// The best Action is in the enemyAIActionList[0]
/// </summary>
public EnemyAIAction GetBestEnemyAIAction()
{
    List<EnemyAIAction> enemyAIActionList = new();

    List<GridPosition> validActionGridPositionList = GetValidGridPositionList();

    foreach (GridPosition gridPosition in validActionGridPositionList)
    {
        // All actions have own EnemyAIAction to set griposition and action value.
        EnemyAIAction enemyAIAction = GetEnemyAIAction(gridPosition);
        enemyAIActionList.Add(enemyAIAction);
    }

    if (enemyAIActionList.Count > 0)
    {
        enemyAIActionList.Sort((a, b) => b.actionValue - a.actionValue);
        return enemyAIActionList[0];
    }
    else
    {
        // No possible Enemy AI Actions
    }
}
```

RogueShooter - All Scripts

```
        return null;  
    }  
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitActions/Actions/MoveAction.cs

```
using System;
using System.Collections.Generic;
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
{
    public event EventHandler OnStartMoving;
    public event EventHandler OnStopMoving;
    [SerializeField] private int maxMoveDistance = 4;
    private Vector3 targetPosition;

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

    private void Update()
    {
        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
            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);
        }
        else
        {
            OnStopMoving?.Invoke(this, EventArgs.Empty);
            ActionComplete();
        }
    }

    public override void TakeAction(GridPosition gridPosition, Action onActionComplete)
    {
        targetPosition = LevelGrid.Instance.GetWorldPosition(gridPosition);
    }
}
```

RogueShooter – All Scripts

```
        OnStartMoving?.Invoke(this, EventArgs.Empty);
        ActionStart(onActionComplete);
    }

    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);
            }
        }
        return validGridPositionList;
    }

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

    /// <summary>
    /// ENEMY AI:
    /// Move toward to Player unit to make shoot action.
    /// </summary>
    public override EnemyAIAction GetEnemyAIAction(GridPosition gridPosition)
    {
        int targetCountAtGridPosition = unit.GetAction<ShootAction>().GetTargetCountAtPosition(gridPosition);

        return new EnemyAIAction
        {
            gridPosition = gridPosition,
            actionValue = targetCountAtGridPosition * 10,
        };
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitActions/Actions/ShootAction.cs

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

public class ShootAction : BaseAction
{
    public event EventHandler<OnShootEventArgs> OnShoot;

    public class OnShootEventArgs : EventArgs
    {
        public Unit targetUnit;
        public Unit shootingUnit;
    }

    private enum State
    {
        Aiming,
        Shooting,
        Cooloff
    }

    private State state;
    private int maxShootDistance = 5;

    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;
                }
        }
    }
}
```

RogueShooter – All Scripts

```
        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;
    }
}

private void Shoot()
{
    OnShoot?.Invoke(this, new OnShootEventArgs
    {
        targetUnit = targetUnit,
        shootingUnit = unit
    });

    NetworkSync.ApplyDamage(targetUnit, 30);
}

public override int GetActionPointsCost()
{
    return 1;
}

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

RogueShooter – All Scripts

```
public List<GridPosition> GetValidActionGridPositionList(GridPosition unitGridPosition)
{
    List<GridPosition> validGridPositionList = new();

    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;
            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);
        }
    }

    return validGridPositionList;
}

public override void TakeAction(GridPosition gridPosition, Action onActionComplete)
{
    targetUnit = LevelGrid.Instance.GetUnitAtGridPosition(gridPosition);

    state = State.Aiming;
    float aimingStateTime = 1f;
    stateTimer = aimingStateTime;

    canShootBullet = true;

    ActionStart(onActionComplete);
}

public Unit GetTargetUnit()
{
    return targetUnit;
}
/*
public void ClearTarget()
```


RogueShooter – All Scripts

```
{
    targetUnit = null;
}
*/

public int GetMaxShootDistance()
{
    return maxShootDistance;
}

/// ----- AI -----
/// <summary>
/// ENEMY AI: Make a list about Player Units what Enemy Unit can shoot.
/// </summary>
public override List<GridPosition> GetValidGridPositionList()
{
    GridPosition unitGridPosition = unit.GetGridPosition();
    return GetValidActionGridPositionList(unitGridPosition);
}

/// <summary>
/// ENEMY AI: How "good" target is. Target who have a lowest health, gets a higher actionvalue
/// </summary>
public override EnemyAIAction GetEnemyAIAction(GridPosition gridPosition)
{
    Unit targetUnit = LevelGrid.Instance.GetUnitAtGridPosition(gridPosition);

    return new EnemyAIAction
    {
        gridPosition = gridPosition,
        actionValue = 100 + Mathf.RoundToInt((1 - targetUnit.GetHealthNormalized()) * 100f), //Take at target who have a lowest health.
    };
}

public int GetTargetCountAtPosition(GridPosition gridPosition)
{
    return GetValidActionGridPositionList(gridPosition).Count;
}
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitActions/Actions/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
{
    private float totalSpinAmount = 0f;
    private void Update()
    {
        if (!isActive) return;

        // Aja paikallisesti vain SinglePlayerissa tai jos tämä instanssi on serveri (host)
        bool driveHere = GameManager.SelectedMode == GameMode.SinglePlayer || isServer;
        if (!driveHere) return;

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

        totalSpinAmount += spinAddAmount;
        if (totalSpinAmount >= 360f)
        {
            ActionComplete();
        }
    }
    public override void TakeAction(GridPosition gridPosition, Action onActionComplete)
    {
        totalSpinAmount = 0f;
        ActionStart(onActionComplete);
    }

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

    public override List<GridPosition> GetValidGridPositionList()
    {
        GridPosition unitGridPosition = unit.GetGridPosition();
```

RogueShooter – All Scripts

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

    public override int GetActionPointsCost()
    {
        return 1;
    }

    /// <summary>
    /// ENEMY AI:
    /// Currently this action has no value. Just testing!
    /// </summary>
    public override EnemyAIAction GetEnemyAIAction(GridPosition gridPosition)
    {
        return new EnemyAIAction
        {
            gridPosition = gridPosition,
            actionValue = 0,
        };
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitActions/UnitActionSystem.cs

```
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;
    }
}
```

RogueShooter – All Scripts

```
// 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 (selectedUnit == null || selectedAction == null) return;
        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
```

RogueShooter – All Scripts

```
/// 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());
    SetSelectedAction(unit.GetAction<MoveAction>());
    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;
}
```

RogueShooter – All Scripts

```
    }  
  
    // Lock/Unlock input methods for PlayerController when playing online  
    public void LockInput() { if (!isBusy) SetBusy(); }  
    public void UnlockInput() { if (isBusy) ClearBusy(); }  
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitAnimator.cs

```
using UnityEngine;
using System;

[RequireComponent(typeof(MoveAction))]
public class UnitAnimator : MonoBehaviour
{
    [SerializeField] private Animator animator;
    [SerializeField] private GameObject bulletProjectilePrefab;
    [SerializeField] private Transform shootPointTransform;

    private void Awake()
    {
        if (TryGetComponent<MoveAction>(out MoveAction moveAction))
        {
            moveAction.OnStartMoving += MoveAction_OnStartMoving;
            moveAction.OnStopMoving += MoveAction_OnStopMoving;
        }

        if (TryGetComponent<ShootAction>(out ShootAction shootAction))
        {
            shootAction.OnShoot += ShootAction_OnShoot;
        }
    }

    /*
    void OnEnable()
    {
        if (TryGetComponent<MoveAction>(out MoveAction moveAction))
        {
            moveAction.OnStartMoving += MoveAction_OnStartMoving;
            moveAction.OnStopMoving += MoveAction_OnStopMoving;
        }

        if (TryGetComponent<ShootAction>(out ShootAction shootAction))
        {
            shootAction.OnShoot += ShootAction_OnShoot;
        }
    }
    */

    void OnDisable()
    {
        if (TryGetComponent<MoveAction>(out MoveAction moveAction))
        {
            moveAction.OnStartMoving -= MoveAction_OnStartMoving;
            moveAction.OnStopMoving -= MoveAction_OnStopMoving;
        }

        if (TryGetComponent<ShootAction>(out ShootAction shootAction))
```


RogueShooter – All Scripts

```
    {
        shootAction.OnShoot -= ShootAction_OnShoot;
    }
}

private void MoveAction_OnStartMoving(object sender, EventArgs e)
{
    animator.SetBool("IsRunning", true);
}

private void MoveAction_OnStopMoving(object sender, EventArgs e)
{
    animator.SetBool("IsRunning", false);
}

private void ShootAction_OnShoot(object sender, ShootAction.OnShootEventArgs e)
{
    animator.SetTrigger("Shoot");
    Vector3 target = e.targetUnit.GetWorldPosition();
    target.y = shootPointTransform.position.y;
    NetworkSync.SpawnBullet(bulletProjectilePrefab, shootPointTransform.position, target);
}
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitManager.cs

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

public class UnitManager : MonoBehaviour
{
    public static UnitManager Instance { get; private set; }
    private List<Unit> unitList;
    private List<Unit> friendlyUnitList;
    private List<Unit> enemyUnitList;

    private void Awake()
    {
        if (Instance != null)
        {
            Debug.LogError("There's more than one UnitManager! " + transform + " - " + Instance);
            Destroy(gameObject);
            return;
        }
        Instance = this;

        unitList = new List<Unit>();
        friendlyUnitList = new List<Unit>();
        enemyUnitList = new List<Unit>();
    }

    private void Start()
    {
        Unit.OnAnyUnitSpawned += Unit_OnAnyUnitSpawned;
        Unit.OnAnyUnitDead += Unit_OnAnyUnitDead;
    }

    void OnEnable()
    {
        Unit.OnAnyUnitSpawned += Unit_OnAnyUnitSpawned;
        Unit.OnAnyUnitDead += Unit_OnAnyUnitDead;
    }

    void OnDisable()
    {
        Unit.OnAnyUnitSpawned -= Unit_OnAnyUnitSpawned;
        Unit.OnAnyUnitDead -= Unit_OnAnyUnitDead;
    }

    private void Unit_OnAnyUnitSpawned(object sender, EventArgs e)
    {
        Unit unit = sender as Unit;
        Debug.Log(unit + "Spawn");
    }
}
```

RogueShooter – All Scripts

```
        unitList.Add(unit);

        if (unit.IsEnemy())
        {
            enemyUnitList.Add(unit);
        }
        else
        {
            friendlyUnitList.Add(unit);
        }
    }

    private void Unit_OnAnyUnitDead(object sender, EventArgs e)
    {
        Unit unit = sender as Unit;
        Debug.Log(unit + "Dead");
        unitList.Remove(unit);

        if (unit.IsEnemy())
        {
            enemyUnitList.Remove(unit);
        }
        else
        {
            friendlyUnitList.Remove(unit);
        }
    }

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

    public List<Unit> GetFriendlyUnitList()
    {
        return friendlyUnitList;
    }

    public List<Unit> GetEnemyUnitList()
    {
        return enemyUnitList;
    }

    public void ClearAllUnitLists()
    {
        unitList.Clear();
        friendlyUnitList.Clear();
        enemyUnitList.Clear();
    }
}
```

--

RogueShooter – All Scripts

Assets/scripts/Units/UnitRagdoll/RagdollPoseBinder.cs

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

/// <summary>
/// Online: Client need this to get destroyed unit rootbone to create ragdoll form it.
/// </summary>
public class RagdollPoseBinder : NetworkBehaviour
{
    [SyncVar] public uint sourceUnitNetId;

    [ClientCallback]
    private void Start()
    {
        StartCoroutine(ApplyPoseWhenReady());
    }

    private IEnumerator ApplyPoseWhenReady()
    {
        var (root, why) = TryFindOriginalRootBone(sourceUnitNetId);
        if (root != null)
        {
            if (TryGetComponent<UnitRagdoll>(out var unitRagdoll))
                unitRagdoll.Setup(root);
            yield break;
        }

        Debug.Log($"[Ragdoll] waiting root for netId {sourceUnitNetId} ({why})");

        yield return new WaitForEndOfFrame();
        Debug.LogWarning($"[RagdollPoseBinder] Source root not found for netId {sourceUnitNetId}");
    }

    private static (Transform root, string why) TryFindOriginalRootBone(uint netId)
    {
        if (netId == 0) return (null, "netId==0");
        if (!Mirror.NetworkClient.spawned.TryGetValue(netId, out var id) || id == null)
            return (null, "identity not in NetworkClient.spawned");

        // Löydä UnitRagdollSpawn myös hierarkiasta
        var spawner = id.GetComponent<UnitRagdollSpawn>()
            ?? id.GetComponentInChildren<UnitRagdollSpawn>(true)
            ?? id.GetComponentInParent<UnitRagdollSpawn>();
        if (spawner == null) return (null, "UnitRagdollSpawn missing under identity");

        if (spawner.OriginalRagdollRootBone == null) return (null, "OriginalRagdollRootBone null");
        return (spawner.OriginalRagdollRootBone, null);
    }
}
```

--

RogueShooter – All Scripts

Assets/scripts/Units/UnitRagdoll/UnitRagdoll.cs

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

public class UnitRagdoll : MonoBehaviour
{
    [SerializeField] private Transform ragdollRootBone;

    public Transform Root => ragdollRootBone;

    public void Setup(Transform originalRootBone)
    {
        MatchAllChildTransforms(originalRootBone, ragdollRootBone);
        ApplyPushForceToRagdoll(ragdollRootBone, 100f, transform.position, 10f);
    }

    /// <summary>
    /// Sets all ragdoll bones to match dying unit bones rotation and position
    /// </summary>
    private static void MatchAllChildTransforms(Transform sourceRoot, Transform targetRoot)
    {
        var stack = new Stack<(Transform sourceBone, Transform targetBone)>();
        stack.Push((sourceRoot, targetRoot));

        while (stack.Count > 0)
        {
            var (currentSourceBone, currentTargetBone) = stack.Pop();

            currentTargetBone.SetPositionAndRotation(currentSourceBone.position, currentSourceBone.rotation);

            if (currentSourceBone.childCount == currentTargetBone.childCount)
            {
                for (int i = 0; i < currentSourceBone.childCount; i++)
                {
                    stack.Push((currentSourceBone.GetChild(i), currentTargetBone.GetChild(i)));
                }
            }
        }
    }

    private void ApplyPushForceToRagdoll(Transform root, float pushForce, Vector3 pushPosition, float PushRange)
    {
        foreach (Transform child in root)
        {
            if (child.TryGetComponent<Rigidbody>(out Rigidbody childRigidbody))
            {
                childRigidbody.AddExplosionForce(pushForce, pushPosition, PushRange);
            }
        }
    }
}
```

RogueShooter - All Scripts

```
        ApplyPushForceToRagdoll(child, pushForce, pushPosition, PushRange);  
    }  
}
```


RogueShooter – All Scripts

Assets/scripts/Units/UnitRagdoll/UnitRagdollSpawn.cs

```
using System;
using UnityEngine;

[RequireComponent(typeof(HealthSystem))]
public class UnitRagdollSpawn : MonoBehaviour
{
    [SerializeField] private Transform ragdollPrefab;
    [SerializeField] private Transform originalRagdollRootBone;
    public Transform OriginalRagdollRootBone => originalRagdollRootBone;

    private HealthSystem HealthSystem;

    private void Awake()
    {
        HealthSystem = GetComponent<HealthSystem>();
        HealthSystem.OnDead += HealthSystem_OnDied;
    }

    private void HealthSystem_OnDied(object sender, EventArgs e)
    {
        var ni = GetComponentInParent<Mirror.NetworkIdentity>();
        uint id = ni ? ni.netId : 0;

        NetworkSync.SpawnRagdoll(
            ragdollPrefab.gameObject,
            transform.position,
            transform.rotation,
            id,
            originalRagdollRootBone);
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitsControlUI/TurnSystemUI.cs

```
using System;
using UnityEngine;
using UnityEngine.UI;
using TMPPro;
using Utp;

///<summary>
/// TurnSystemUI manages the turn system user interface.
/// It handles both singleplayer and multiplayer modes.
/// In multiplayer, it interacts with PlayerController to manage turn ending.
/// It also updates UI elements based on the current turn state.
///</summary>
public class TurnSystemUI : MonoBehaviour
{
    [SerializeField] private Button endTurnButton;
    [SerializeField] private TextMeshProUGUI turnNumberText;           // (valinnainen, käytä SP:ssä)
    [SerializeField] private GameObject enemyTurnVisualGameObject;    // (valinnainen, käytä SP:ssä)
    [SerializeField] private TextMeshProUGUI playerReadyText;         // (Online)

    bool isCoop;
    private PlayerController localPlayerController;

    void Start()
    {
        isCoop = GameManager.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
            TurnSystem.Instance.OnTurnChanged += TurnSystem_OnTurnChanged;
            SetCanAct(false);
        }
        else
        {
            // Singleplayerissa kuuntele vuoron vaihtumista
            if (TurnSystem.Instance != null)
            {
                TurnSystem.Instance.OnTurnChanged += TurnSystem_OnTurnChanged;
                UpdateForSingleplayer();
            }
        }

        if (playerReadyText) playerReadyText.gameObject.SetActive(false);
    }
}
```

RogueShooter – All Scripts

```
}

void OnDisable()
{
    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 isOnline =
        NetTurnManager.Instance != null &&
        (GameNetworkManager.Instance.GetNetworkServerActive() || GameNetworkManager.Instance.GetNetworkClientConnected());
    if (!isOnline)
    {
        if (TurnSystem.Instance != null)
        {
            TurnSystem.Instance.NextTurn();
        }
        else
        {
            Debug.LogWarning("[UI] TurnSystem.Instance is null");
        }
        return;
    }
}

CacheLocalPlayerController();
if (localPlayerController == null)
{
    Debug.LogWarning("[UI] Local PlayerController not found");
    return;
}
// Instantly lock input
if (UnitActionSystem.Instance != null)
{
    UnitActionSystem.Instance.LockInput();
}
// Prevent double clicks
SetCanAct(false);
```

RogueShooter – All Scripts

```
// Lähetä serverille
localPlayerController.ClickEndTurn();

//Päivitä player ready hud
}

private void CacheLocalPlayerController()
{
    if (localPlayerController != null) return;

    // 1) Varmista helpoimman kautta
    if (PlayerController.Local != null)
    {
        localPlayerController = PlayerController.Local;
        return;
    }

    // 2) Fallback: Mirrorin client-yhteyden identity
    var conn = GameNetworkManager.Instance != null
        ? GameNetworkManager.Instance.NetworkClientConnection()
        : null;
    if (conn != null && conn.identity != null)
    {
        localPlayerController = conn.identity.GetComponent<PlayerController>();
        if (localPlayerController != null) return;
    }

    // 3) Viimeinen oljenkorsi: etsi skenestä local-pelaaja
    var pcs = FindObjectsByType<PlayerController>(FindObjectsSortMode.InstanceID);
    foreach (var pc in pcs)
    {
        if (pc.isLocalPlayer) { localPlayerController = pc; break; }
    }
}

// ===== 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

```
// Kutsutaan verkosta
public void SetTeammateReady(bool visible, string whoLabel = null)
{
    if (!playerReadyText) return;
    if (visible)
    {
        playerReadyText.text = $"{whoLabel} READY";
        playerReadyText.gameObject.SetActive(true);
    }
    else
    {
        playerReadyText.gameObject.SetActive(false);
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitsControlUI/UnitActionBusyUI.cs

```
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();
    }

    void OnEnable()
    {
        UnitActionSystem.Instance.OnBusyChanged += UnitActionSystem_OnBusyChanged;
    }

    void OnDisable()
    {
        UnitActionSystem.Instance.OnBusyChanged -= UnitActionSystem_OnBusyChanged;
    }

    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/Units/UnitsControlUI/UnitActionButtonUI.cs

```
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);
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Units/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()
    {
        if (UnitActionSystem.Instance != null)
        {
            UnitActionSystem.Instance.OnSelectedUnitChanged += UnitActionSystem_OnSelectedUnitChanged;
            UnitActionSystem.Instance.OnSelectedActionChanged += UnitActionSystem_OnSelectedActionChanged;
            UnitActionSystem.Instance.OnActionStarted += UnitActionSystem_OnActionStarted;
        }
        else
        {
            Debug.Log("UnitActionSystem instance found.");
        }
        if (TurnSystem.Instance != null)
        {
            TurnSystem.Instance.OnTurnChanged += TurnSystem_OnTurnChanged;
        }
        else
        {
            Debug.Log("TurnSystem instance not found.");
        }

        Unit.OnAnyActionPointsChanged += Unit_OnAnyActionPointsChanged;
    }

    /*
```


RogueShooter – All Scripts

```
void OnEnable()
{
    if (UnitActionSystem.Instance != null)
    {
        UnitActionSystem.Instance.OnSelectedUnitChanged += UnitActionSystem_OnSelectedUnitChanged;
        UnitActionSystem.Instance.OnSelectedActionChanged += UnitActionSystem_OnSelectedActionChanged;
        UnitActionSystem.Instance.OnActionStarted += UnitActionSystem_OnActionStarted;
    } else
    {
        Debug.Log("UnitActionSystem instance found.");
    }
    if (TurnSystem.Instance != null)
    {
        TurnSystem.Instance.OnTurnChanged += TurnSystem_OnTurnChanged;
    } else
    {
        Debug.Log("TurnSystem instance not found.");
    }

    Unit.OnAnyActionPointsChanged += Unit_OnAnyActionPointsChanged;
}
*/
void OnDisable()
{
    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 CreateUnitActionButton()
{
    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);
    }
}
```

RogueShooter – All Scripts

```
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()
{
    // Jos tekstiä ei ole kytketty Inspectorissa, poistu siististi
    if (actionPointsText == null) return;

    // Jos järjestelmä ei ole vielä valmis, näytä viiva
    if (UnitActionSystem.Instance == null)
    {
        actionPointsText.text = "Action Points: -";
        return;
    }
    Unit selectedUnit = UnitActionSystem.Instance.GetSelectedUnit();
    if (selectedUnit == null)
    {
        actionPointsText.text = "Action Points: -";
        return;
    }
}
```

RogueShooter – All Scripts

```
        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();
    }
}
```

RogueShooter – All Scripts

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

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

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

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

    /*
    private void OnDestroy()
    {

```

RogueShooter – All Scripts

```
        if (UnitActionSystem.Instance != null)
            UnitActionSystem.Instance.OnSelectedUnitChanged -= UnitActionSystem_OnSelectedUnitChanged;
    }
    */
    private void UnitActionSystem_OnSelectedUnitChanged(object sender, EventArgs empty)
    {
        UpdateVisual();
    }

    private void UpdateVisual()
    {
        if (!this || meshRenderer == null || UnitActionSystem.Instance == null) return;
        var selected = UnitActionSystem.Instance.GetSelectedUnit();
        meshRenderer.enabled = unit != null && selected == unit;
    }
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitStatsUI/UnitUIBroadcaster.cs

```
using Mirror;

public class UnitUIBroadcaster : NetworkBehaviour
{
    public static UnitUIBroadcaster Instance { get; private set; }
    void Awake() { if (Instance == null) Instance = this; }

    // Tätä saa kutsua vain serveri (hostin serveripuoli)
    [Server]
    public void BroadcastUnitWorldUIVisibility(bool allready)
    {
        if (!NetworkServer.active) return;

        // käy kaikki serverillä tunnetut unitit läpi
        foreach (var kvp in NetworkServer.spawned)
        {
            var unit = kvp.Value.GetComponent<Unit>();
            if (!unit) continue;

            // serveri voi laskea logiikan: pitääkö tämän unitin AP näkyä
            bool visible = ShouldBeVisible(unit, allready);

            // lähetä client-puolelle että tämän unitin UI asetetaan
            RpcSetUnitUIVisibility(unit.netId, visible);
        }
    }

    // Tätä kutsuu serveri, suoritetaan kaikilla clienteilla
    [ClientRpc]
    private void RpcSetUnitUIVisibility(uint unitId, bool visible)
    {
        if (NetworkClient.spawned.TryGetValue(unitId, out var ni) && ni != null)
        {
            var ui = ni.GetComponentInChildren<UnitWorldUI>();
            if (ui != null) ui.SetVisible(visible);
        }
    }

    // serverilogiikka omistajan perusteella
    [Server]
    private bool ShouldBeVisible(Unit unit, bool allready)
    {
        // Kaikki pelaajat ovat valmiina joten näytetään vain vihollisen AP pisteet.
        if (allready)
        {
            return unit.IsEnemy();
        }

        // Co-Op
        bool playersPhase = TurnSystem.Instance.IsPlayerTurn();
    }
}
```

RogueShooter – All Scripts

```
bool ownerEnded = false;
if (unit.OwnerId != 0 &&
    NetworkServer.spawned.TryGetValue(unit.OwnerId, out var ownerId) &&
    ownerId != null)
{
    var pc = ownerId.GetComponent<PlayerController>();
    if (pc != null) ownerEnded = pc.hasEndedThisTurn;
}

// 2) Päättä näkyvyys
if (playersPhase)
{
    // Pelaajavaihe: näytä kaikki ei-viholliset, joiden omistaja EI ole lopettanut
    return !unit.IsEnemy() && !ownerEnded;
}
else
{
    // Vihollisvaihe: näytä vain viholliset
    return unit.IsEnemy();
}
}
```

RogueShooter – All Scripts

Assets/scripts/Units/UnitStatsUI/UnitWorldUI.cs

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

/// <summary>
/// Displays world-space UI for a single unit, including action points and health bar.
/// Reacts to turn events and ownership rules to show or hide UI visibility
/// </summary>
public class UnitWorldUI : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI actionPointsText;
    [SerializeField] private Unit unit;
    [SerializeField] private Image healthBarImage;
    [SerializeField] private HealthSystem healthSystem;

    /// <summary>
    /// Reference to the unit this UI belongs to.
    /// Which object's visibility do we want to change?
    /// </summary>
    [Header("Visibility")]
    [SerializeField] private GameObject actionPointsRoot;

    /// <summary>
    /// Cached network identity for ownership.
    /// </summary>
    private NetworkIdentity unitIdentity;

    // --- NEW: tiny static registry for ready owners (co-op only) ---
    // private static readonly HashSet<uint> s_readyOwners = new();
    // public static bool HasOwnerEnded(uint ownerId) => s_readyOwners.Contains(ownerId);

    private void Awake()
    {
        unitIdentity = unit ? unit.GetComponent<NetworkIdentity>() : GetComponentInParent<NetworkIdentity>();
    }

    private void Start()
    {
        // unitIdentity = unit ? unit.GetComponent<NetworkIdentity>() : GetComponentInParent<NetworkIdentity>();

        Unit.OnAnyActionPointsChanged += Unit_OnAnyActionPointsChanged;
        healthSystem.OnDamaged += HealthSystem_OnDamaged;
        UpdateActionPointsText();
        UpdateHealthBarUI();
    }
}
```


RogueShooter – All Scripts

```
// Co-opissa. Ei paikallista seuranta. Ainoastaan alku asettelu
if (GameManager.SelectedMode == GameMode.CoOp)
{
    if (unit.IsEnemy())
    {
        actionPointsRoot.SetActive(false);
    }

    return;
}

PlayerLocalTurnGate.LocalPlayerTurnChanged += PlayerLocalTurnGate_LocalPlayerTurnChanged;
PlayerLocalTurnGate_LocalPlayerTurnChanged(PlayerLocalTurnGate.LocalPlayerTurn);
}

/*
private void OnEnable()
{
    Unit.OnAnyActionPointsChanged += Unit_OnAnyActionPointsChanged;
    healthSystem.OnDamaged += HealthSystem_OnDamaged;
    PlayerLocalTurnGate.LocalPlayerTurnChanged += PlayerLocalTurnGate_LocalPlayerTurnChanged;
}
*/

private void OnDisable()
{
    Unit.OnAnyActionPointsChanged -= Unit_OnAnyActionPointsChanged;
    healthSystem.OnDamaged -= HealthSystem_OnDamaged;
    PlayerLocalTurnGate.LocalPlayerTurnChanged -= PlayerLocalTurnGate_LocalPlayerTurnChanged;
}

private void OnDestroy()
{
    Unit.OnAnyActionPointsChanged -= Unit_OnAnyActionPointsChanged;
    healthSystem.OnDamaged -= HealthSystem_OnDamaged;
    PlayerLocalTurnGate.LocalPlayerTurnChanged -= PlayerLocalTurnGate_LocalPlayerTurnChanged;
}

private void UpdateActionPointsText()
{
    actionPointsText.text = unit.GetActionPoints().ToString();
}

private void Unit_OnAnyActionPointsChanged(object sender, EventArgs e)
{
    UpdateActionPointsText();
}

private void UpdateHealthBarUI()
{

```

RogueShooter – All Scripts

```
        healthBarImage.fillAmount = healthSystem.GetHealthNormalized();
    }

    /// <summary>
    /// Event handler: refreshes the health bar UI when this unit takes damage.
    /// </summary>
    private void HealthSystem_OnDamaged(object sender, EventArgs e)
    {
        UpdateHealthBarUI();
    }

    /// <summary>
    /// SinglePlayer/Versus: paikallinen turn-gate. Co-opissa ei käytetä.
    /// </summary>
    private void PlayerLocalTurnGate_LocalPlayerTurnChanged(bool canAct)
    {
        if (GameManager.SelectedMode == GameMode.CoOp) return; // Co-op: näkyvyys tulee RPC:stä
        if (!this || !gameObject) return;

        bool showAp;
        if (GameManager.SelectedMode == GameMode.SinglePlayer)
        {
            showAp = canAct ? !unit.IsEnemy() : unit.IsEnemy();
        }
        else // Versus
        {
            bool unitIsMine = unitIdentity && unitIdentity.isOwned;
            showAp = (canAct && unitIsMine) || (!canAct && !unitIsMine);
        }

        actionPointsRoot.SetActive(showAp);
    }

    public void SetVisible(bool visible)
    {
        actionPointsRoot.SetActive(visible);
    }
}
```

