JANNE SILLANPÄÄ

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


public class MasterAIController : MonoBehaviour
{
    public GameObject player;

    private List<EnemyAI> allEnemies = new List<EnemyAI>();

    private List<string> attackerList;
    private List<string> flankerList;
    private List<string> supportList;


    // Updating enemy roles dynamically
    private void UpdateRoles()
    {
        int enemyCount = allEnemies.Count;
        EnemyAI closestEnemyToPlayer = null;

        List<string> enemyRoles = new List<string>();

        List<EnemyAI> availableEnemiesToAssign = new List<EnemyAI>(allEnemies);

        // Get the role of every enemy
        foreach (EnemyAI enemy in allEnemies)
        {
            enemyRoles.Add(enemy.GetRole());
            closestEnemyToPlayer = enemy;
        }

        // Assign the enemies into lists based on their role
        attackerList = enemyRoles.FindAll(s => s.Equals("Attacker"));
        flankerList = enemyRoles.FindAll(s => s.Equals("Flanker"));
        supportList = enemyRoles.FindAll(s => s.Equals("Support"));


        // If there are no attackers assign the closest enemy to player into an attacker
        if (attackerList.Count < 1)
        {
            float distance = 0;

            foreach(EnemyAI enemy in availableEnemiesToAssign)
            {
                if (closestEnemyToPlayer != null)
                {
                    float currentDistance = Vector3.Distance(enemy.gameObject.transform.position,
player.transform.position);

                    if (distance > currentDistance || distance == 0)
                    {
                        closestEnemyToPlayer = enemy;
                        distance = currentDistance;
                    }
                }
            }

            RemoveFromRoleList(closestEnemyToPlayer);
```

```
            availableEnemiesToAssign.Remove(closestEnemyToPlayer);
            closestEnemyToPlayer.SetRole("Attacker");
            attackerList.Add("Attacker");

        }

        // Remove all attackers from the available enemies to assign. This means attackers cannot
change roles dynamically
        availableEnemiesToAssign.RemoveAll(enemy => enemy.GetRole() == "Attacker");


         // Unassign every flanker and support in order to dynamically assign them
        if (flankerList.Count >= 1)
        {
            EnemyAI flanker = availableEnemiesToAssign.Find(enemy => enemy.GetRole() == "Flanker");
            availableEnemiesToAssign.Remove(flanker);
        }
        if (supportList.Count >= 1)
        {
            EnemyAI support = availableEnemiesToAssign.Find(enemy => enemy.GetRole() == "Support");
            availableEnemiesToAssign.Remove(support);
        }

        // If we have no flankers and we have available enemies to assign then create one flanker
        if (flankerList.Count < 1 && availableEnemiesToAssign.Count > 0)
        {
            foreach (EnemyAI enemy in availableEnemiesToAssign)
            {
                RemoveFromRoleList(enemy);
                availableEnemiesToAssign.Remove(enemy);
                enemy.SetRole("Flanker");
                flankerList.Add("Flanker");
                break;
            }
        }

        // If we have no supports and we have available enemies to assign then create one support
        if (supportList.Count < 1 && availableEnemiesToAssign.Count > 0)
        {
            foreach (EnemyAI enemy in availableEnemiesToAssign)
            {
                RemoveFromRoleList(enemy);

                availableEnemiesToAssign.Remove(enemy);
                enemy.SetRole("Support");
                supportList.Add("Support");
                break;
            }
        }


        // Assign rest of the enemies in a 3-2-1 ratio  Attacker-Flanker-Support
        if (availableEnemiesToAssign.Count > 0)
        {
            int startingCount = availableEnemiesToAssign.Count;

            // 3-2-1 RATIO
            int supportCount = Mathf.RoundToInt(startingCount * 0.16666666666f);
            int flankerCount = supportCount * 2;
            int attackerCount = supportCount * 3;

            int addedCount = supportCount + flankerCount + attackerCount;

            // Make sure the the amount of enemies is correct
```

```
    if (addedCount != startingCount)
    {
        int difference = addedCount - startingCount;

        attackerCount = attackerCount - difference;

        addedCount = supportCount + flankerCount + attackerCount;
    }

    // First assign the supports
    foreach (EnemyAI enemy in availableEnemiesToAssign)
    {
        RemoveFromRoleList(enemy);
        enemy.SetRole("Support");
        supportList.Add("Support");
        --supportCount;

        if (supportCount <= 0)
        {
            break;
        }
    }

    availableEnemiesToAssign.RemoveAll(enemy => enemy.GetRole() == "Support");

    // Assign the flankers
    foreach (EnemyAI enemy in availableEnemiesToAssign)
    {
        RemoveFromRoleList(enemy);
        enemy.SetRole("Flanker");
        flankerList.Add("Flanker");
        --flankerCount;

        if (flankerCount <= 0)
        {
            break;
        }
    }

    availableEnemiesToAssign.RemoveAll(enemy => enemy.GetRole() == "Flanker");


    // Assign the attackers
    foreach (EnemyAI enemy in availableEnemiesToAssign)
    {
        RemoveFromRoleList(enemy);
        enemy.SetRole("Attacker");
        attackerList.Add("Attacker");
        --attackerCount;

        if (attackerCount <= 0)
        {
            break;
        }
    }

}

// DONE

// Print the counts
//Debug.Log("Attackers: " + attackerList.Count);
//Debug.Log("Flankers: " + flankerList.Count);
//Debug.Log("Supports: " + supportList.Count);
```

```
    }

    // Helper function for removing roles from enemies
    private void RemoveFromRoleList(EnemyAI thisAI)
    {
        if (thisAI.GetRole() == "Flanker")
        {
            flankerList.RemoveAt(flankerList.Count - 1);
        }
        else if (thisAI.GetRole() == "Support")
        {
            supportList.RemoveAt(supportList.Count - 1);
        }
    }

    // Every time an enemy is added or removed from the game, update the roles
    public void AddEnemy(EnemyAI newEnemy)
    {
        allEnemies.Add(newEnemy);
        UpdateRoles();
    }

    public void RemoveEnemy(EnemyAI newEnemy)
    {
        allEnemies.Remove(newEnemy);
        UpdateRoles();
    }



    public List<EnemyAI> GetEnemies(){ return allEnemies; }

}
```