Unity

Philippines Users Group

May 2013 Meet-up

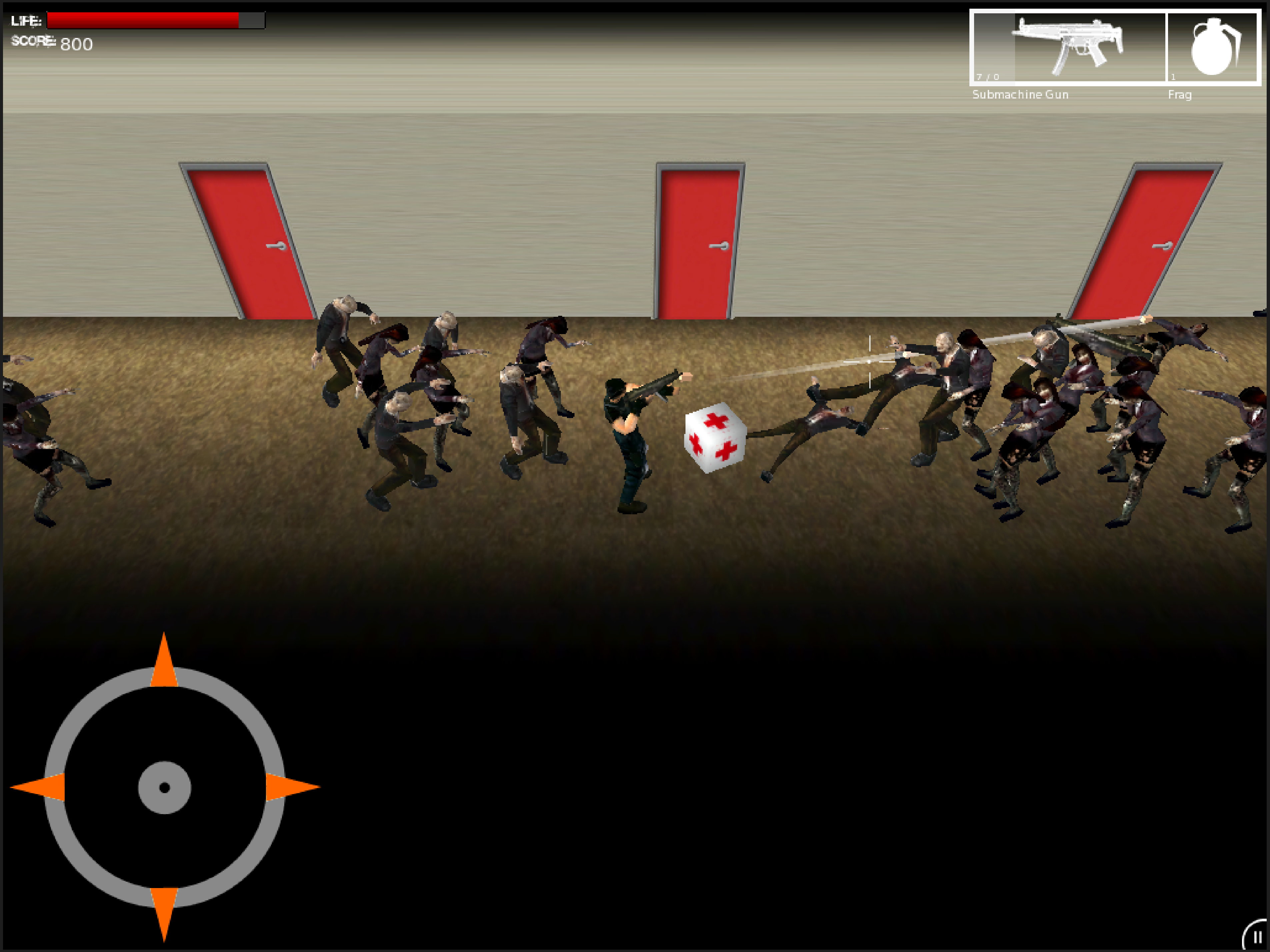
facebook.com/groups/unitypug

unity3d.org.ph

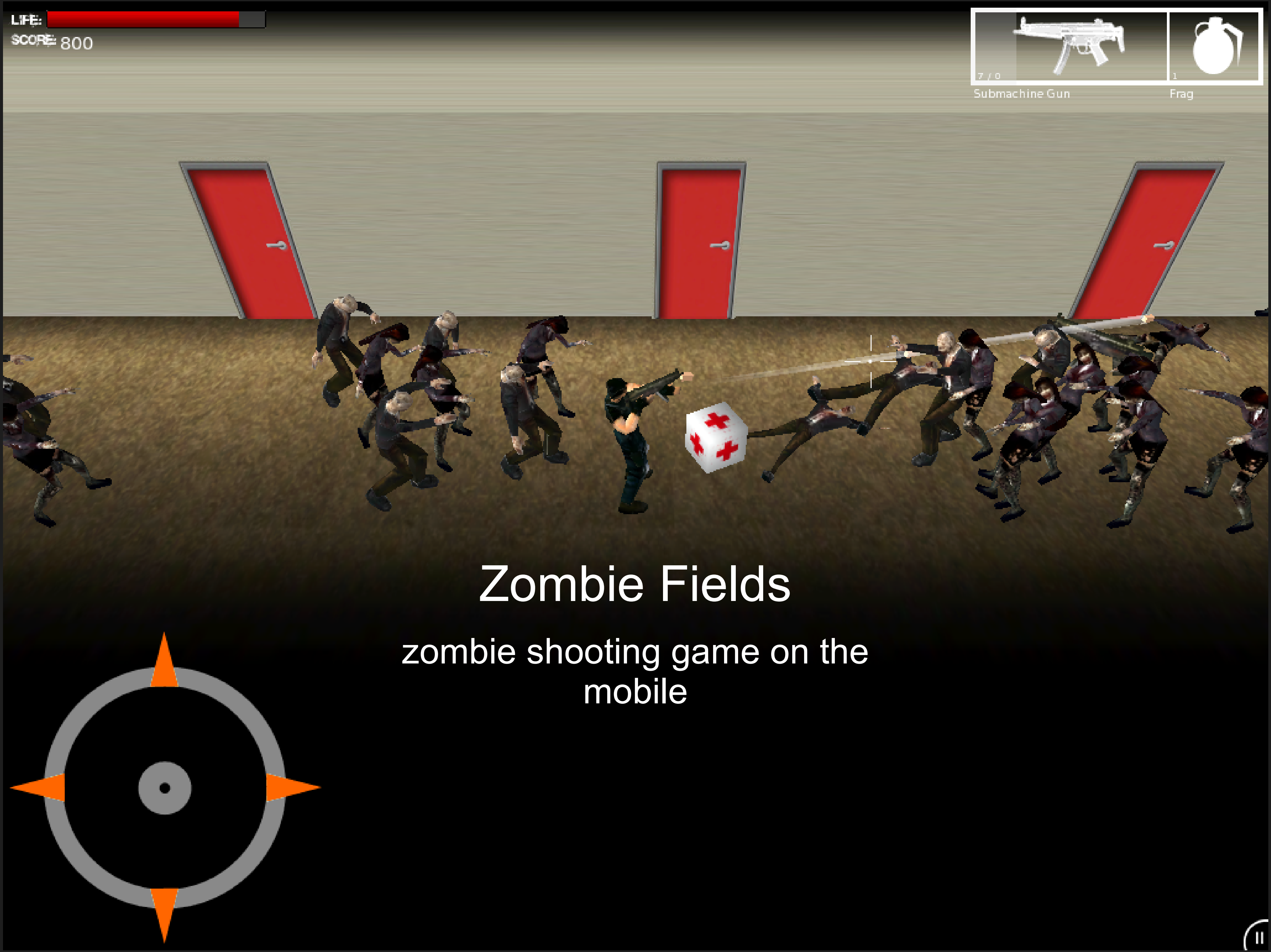
Software Architecture in Game Programming

Software Architecture in Game Programming

and why you should care about it

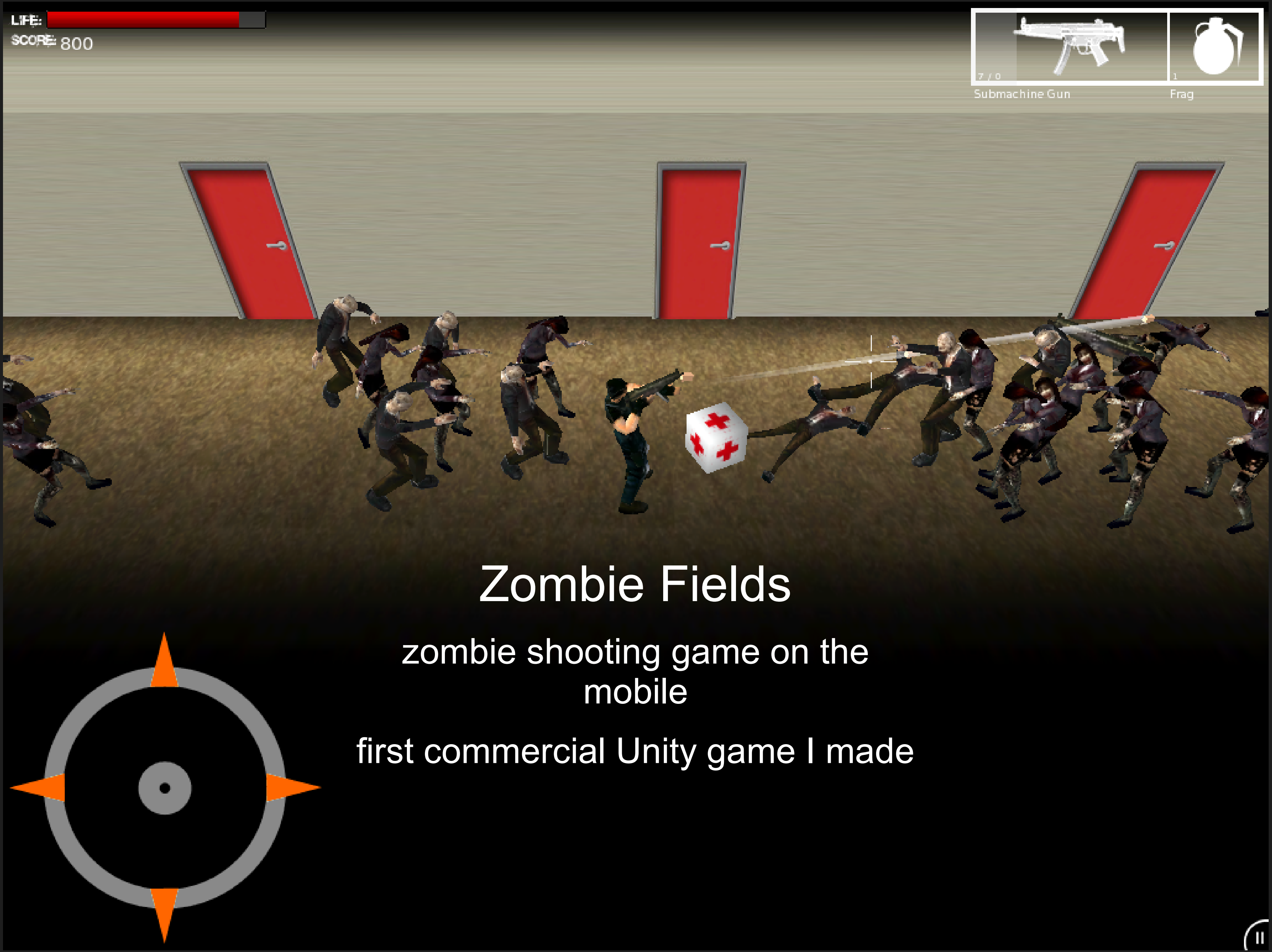

LIFE:
SCORE: 800
7 / 0
Submachine Gun
1
Frag

Zombie Fields



Zombie Fields

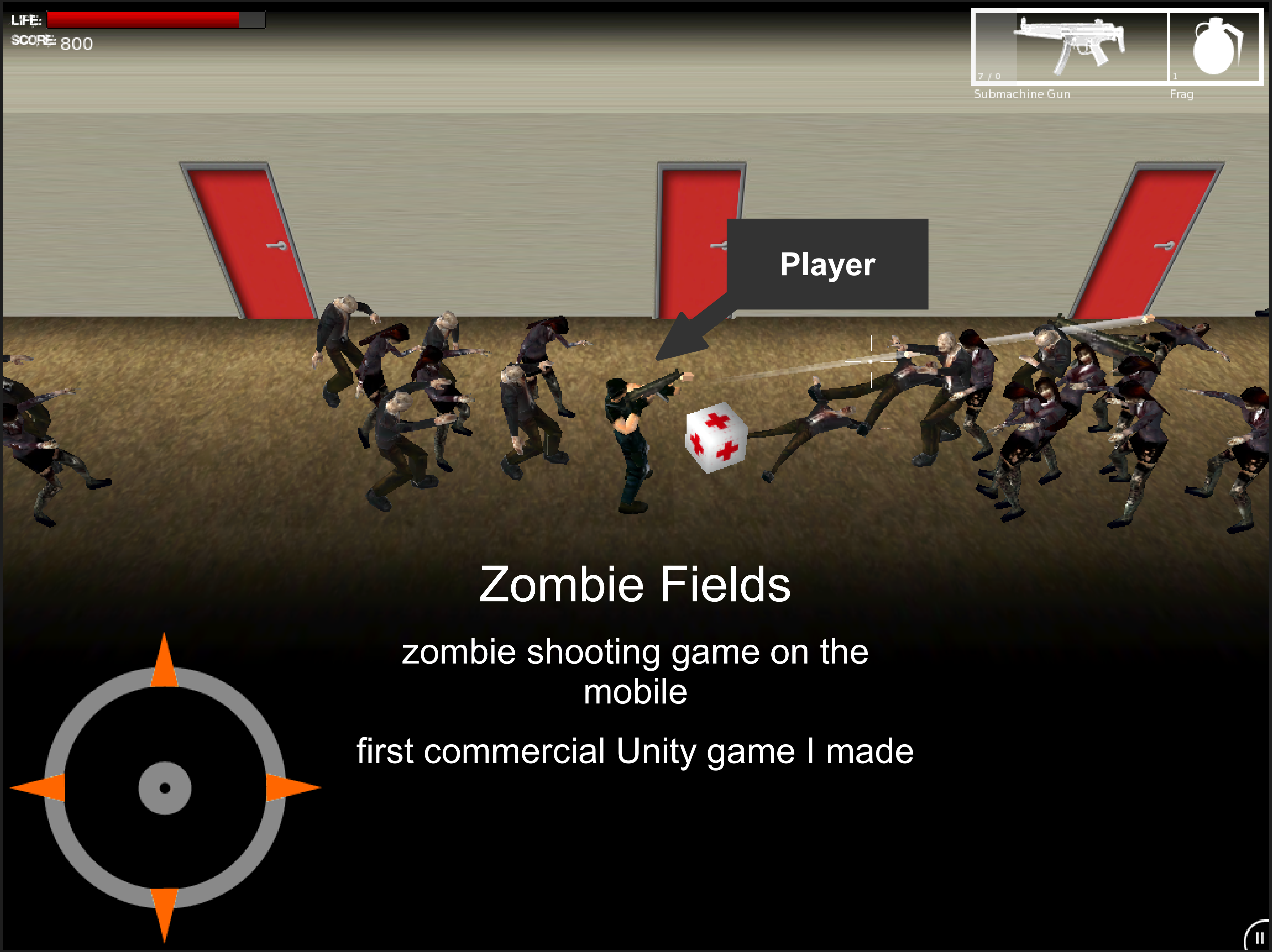
zombie shooting game on the mobile



Zombie Fields

zombie shooting game on the mobile

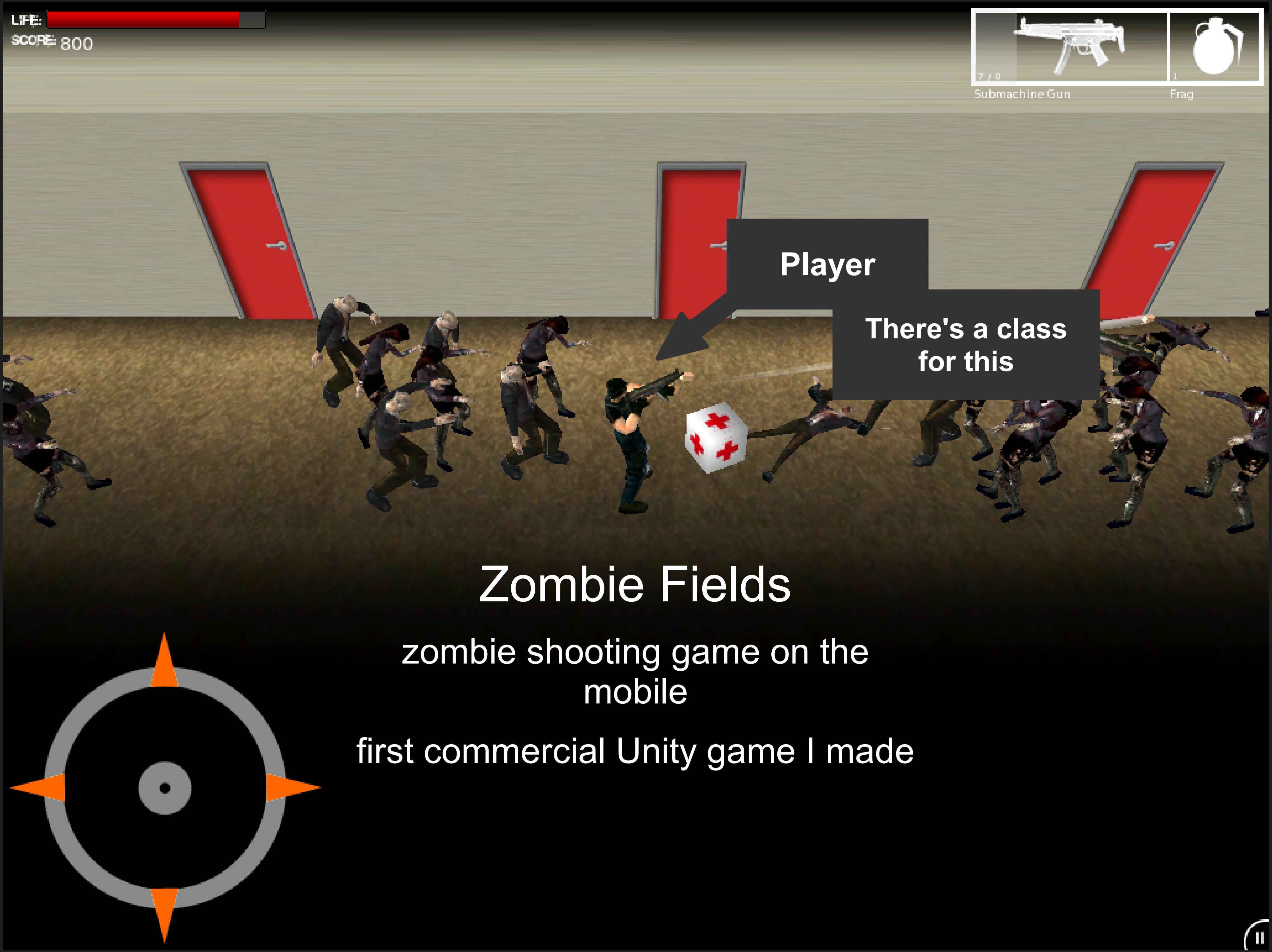
first commercial Unity game I made



Zombie Fields

zombie shooting game on the mobile

first commercial Unity game I made



Zombie Fields

zombie shooting game on the mobile

first commercial Unity game I made

Want to see my source code for the Player class?

H-here goes...

```
//#define PLAYER_DEBUG_AIMING

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

public class PlayerEquippedWeapon
{
        public int allAmmo
        {
                get{ return (sharedAmmo == WeaponType.None) ? PlayerData.Singleton.GetAmmo(type) : PlayerData.Singleton.GetAmmo(sharedAmmo); }
        }
        public int reserveAmmo
        {
                get{ return allAmmo - currentAmmo; }
        }
        public int currentAmmo = 0;
        public float timeSinceLastFire = 0.0f;
        public float shotSpread = 0.0f;

        public bool isReloading = false;
        public float timeReloadStarted = 0.0f;

        public int bulletsThatHit = 0;
        public int bulletsFired = 0;

        public WeaponType type = WeaponType.None;
        public WeaponType sharedAmmo = WeaponType.None;

        public bool HasAmmo()
        {
                return reserveAmmo > 0 || currentAmmo > 0;
        }

        public float GetAccuracy()
        {
                float accuracy = 0.0f;

                if (bulletsFired != 0)
                {
                        accuracy = ((float) bulletsThatHit/bulletsFired)*100.0f;
                        //accuracy = Mathf.Round(accuracy);
                        if (accuracy > 100.0f)
                        {
                                accuracy = 100.0f;
                        }
                }

                return accuracy;
        }

        public PlayerEquippedWeapon(WeaponType newType, WeaponType newSharedAmmo)
        {
                type = newType;
                sharedAmmo = newSharedAmmo;
        }

        public PlayerEquippedWeapon(WeaponType newType)
        {
                type = newType;
                sharedAmmo = WeaponType.None;
        }
}

public class Player : MonoBehaviour
{
        PlayerAnimation anim;
        PlayerMovement movement;
        CrossHair crosshair;

        public void OnPickUpItem()
        {
                currentResult.itemsPickedUp += 1;
                //Debug.Log("OnPickUpItem() " + currentResult.itemsPickedUp);
        }

        bool autoSwitchWeapon = false;

        float movementModifierDuration = 0.2f;
        float timeSinceLastMovementModifierApplied = 0.0f;

        //private LoadingScreen loadingScreen;
        SpawnSystem zombieSpawn;

        bool zBoyMode = false;

        public static Player GetPlayerScript(GameObject pGo)
        {
                if (pGo == null)
                {
                        return null;
                }
                while (GM.GetComponent<Player>(pGo) == null && pGo.transform.parent != null)
                {
                        pGo = pGo.transform.parent.gameObject;
                }
                Player p = GM.GetComponent<Player>(pGo);
                if (p == null)
                {
                        Debug.LogError("Player: Could not find gameobject's player script!");
                        return null;
                }
                return p;
        }

        ArmorData[] armorData;

        // Player Sounds
        //------------------------------------------------------------------------
        [SerializeField]
        AudioClip[] healSounds;

        [SerializeField]
        AudioClip[] hurtSounds;

        [SerializeField]
        AudioClip[] deathSounds;

        [SerializeField]
        string tauntFolder = "";

        [SerializeField]
        string[] startTaunts;

        [SerializeField]
        string[] shotgunTaunts;

        [SerializeField]
        string[] shotgunEquipTaunts;
```

```
        public void OnGrenadeExplode(GrenadeType t)
        {
                if (t == GrenadeType.Frag || t == GrenadeType.Landmine)
                {
                        PlayTaunt(explodeTaunts, 5);
                }
                else if (t == GrenadeType.Molotov)
                {
                        PlayTaunt(molotovTaunts, 5);
                }
        }

        void OnWin()
        {
                PlayTaunt(winTaunts, 5);

                showResults = true;
                crosshair.HideCrosshair();

                ComputeResultData();
                //StartCoroutine(StartFadeIn());

                ResultData resultsToShow = currentResult;
                if (Mode.currentGameMode == GameMode.Assault && (assault.AtFinalStage() || IsDead()))
                {
                        resultsToShow = totalResult;
                }

                _combatResultsScreen.Show(resultsToShow);

                OnEndGame();
        }

        void OnLose(GrenadeType grenade, Vector3 explodeForce)
        {
                Mode.OnPlayerLose();

                zombieSpawn.StopSpawning();

                anim.PlayDeathAnimation(explodeForce);

                // play death sound effect
                if (OptionsNGUIScreen.IsSoundsAllowed && deathSounds.Length > 0)
                {
                        AudioClip deathSound = deathSounds[Random.Range(0, deathSounds.Length)];
                        AudioSource.PlayClipAtPoint(deathSound, myTransform.position, OptionsNGUIScreen.SoundsVolume);
                }

                ComputeResultData();

                crosshair.HideCrosshair();

                //StartCoroutine(StartFadeIn());

                ResultData resultsToShow = currentResult;
                if (Mode.currentGameMode == GameMode.Assault && (assault.AtFinalStage() || IsDead()))
                {
                        resultsToShow = totalResult;
                }

                _combatResultsScreen.Show(resultsToShow);

                if (grenade != GrenadeType.None)
                {
                        PlayerStats.Singleton.OnDeath(grenade);
                }

                OnEndGame();
        }

        public void OnEndGame()
        {
                PlayerStats.Singleton.WriteStats();
                SaveUnusedWeapons();
        }

        void SaveUnusedWeapons()
        {
                PlayerData.Singleton.CommitChanges();
        }


        // Player Weapon Mount
        //------------------------------------------------------------------------
        [SerializeField]
        Transform weaponMount;

        // we need to store a handle to this so we can delete it later (when player switches weapons)
        Transform weaponMesh;

        void MountWeapon(WeaponData weapon)
        {
                if (weapon.dualWeild)
                {
                        muzzle = MountWeapon(weapon.mesh, ref weaponMesh, weaponMount);
                        muzzle2 = MountWeapon(weapon.mesh, ref weaponMesh2, weaponMount2);
                }
                else
                {
                        if (weaponMesh2 != null)
                        {
                                Destroy(weaponMesh2.gameObject);
                        }
                        muzzle = MountWeapon(weapon.mesh, ref weaponMesh, weaponMount);
                }
        }

        Transform MountWeapon(Transform weaponMeshPrefab, ref Transform weaponMeshHandle, Transform mountPoint)
        {
                if (weaponMeshHandle != null)
                {
                        Destroy(weaponMeshHandle.gameObject);
                }

                //Debug.Log("weapon " + obj.name + " mounted");
                weaponMeshHandle = Instantiate(weaponMeshPrefab, Vector3.zero, Quaternion.identity) as Transform;
                weaponMeshHandle.parent = mountPoint;
                weaponMeshHandle.localRotation = weaponMeshPrefab.rotation;
                weaponMeshHandle.localPosition = Vector3.zero;

                return weaponMeshHandle.Find("Armature/Muzzle");
        }

        void MountGrenade(Transform obj)
        {
                if (weaponMesh != null)
                {
                        Destroy(weaponMesh.gameObject);
                }
```

```
        AimCoords aim;

        void UpdateAim()
        {
                //Debug.Log("UpdateAim STA");

                Ray ray = Camera.main.ScreenPointToRay(aim.GetForScreenPointToRay());
                RaycastHit hit;

                bool foundZombie = false;
//#if (UNITY_IPHONE || UNITY_ANDROID)
                int targetableLayers = 1 << 12 | 1 << 10; // touchboxes or hitboxes
//#else
//              int targetableLayers = 1 << 10; // targetables (zombie hitboxes)
//#endif
                if (Physics.Raycast(ray, out hit, Mathf.Infinity, targetableLayers))
                {
                        if (hit.collider.gameObject.tag == "Zombie")
                        {
                                foundZombie = true;
                                aimTarget = hit.collider.bounds.center;

#if (PLAYER_DEBUG_AIMING)
                                //Debug.Log("aimed at zombie: " + aimTarget);
                                aimZombieTarget = aimTarget;
#endif

                        }
                }

                targetableLayers = 1 << 11; // aim plane
                if (Physics.Raycast(ray, out hit, Mathf.Infinity, targetableLayers))
                {
                        // face where crosshair is
                        //if (aim.fireLapse)
                        if (anim.InGrenadeThrowAnimation() == false)
                        {
                                myTransform.LookAt(hit.point);
                                Vector3 angle = myTransform.localEulerAngles;
                                angle.x = 0;
                                myTransform.localEulerAngles = angle;
                        }


                        if (!foundZombie)
                        {
                                aimTarget = hit.point;

                                // uncomment to lock muzzle to single plane
                                //Vector3 angle = muzzle.localEulerAngles;
                                //angle.x = 0;
                                //muzzle.localEulerAngles = angle;

#if (PLAYER_DEBUG_AIMING)
                                aimPlaneTarget = aimTarget;
                                //Debug.Log("aimed at aim plane: " + aimTarget);
#endif
                        }
                }
                else
                {
                        Debug.LogError("Aim Plane not found");
                }


                ray = Camera.main.ScreenPointToRay(aim.GetTouchForScreenPointToRay());
                targetableLayers = 1 << 12 | 1 << 10; // touchboxes or hitboxes
                foundZombie = false;
                if (Physics.Raycast(ray, out hit, Mathf.Infinity, targetableLayers))
                {
                        if (hit.collider.gameObject.tag == "Zombie")
                        {
                                foundZombie = true;
                                groundAimTarget = hit.collider.bounds.center;
                        }
                }

                if (!foundZombie)
                {
                        targetableLayers = 1 << 8; // ground
                        if (Physics.Raycast(ray, out hit, Mathf.Infinity, targetableLayers))
                        {
                                groundAimTarget = hit.point;
                        }
                        //else
                        //{
                        //      Debug.LogError("Ground not found");
                        //}
                }
                if ((aim.touchFire && inThrowGrenadeMode) || anim.InGrenadeThrowAnimation())
                {
                        // face where the grenade is thrown
                        myTransform.LookAt(groundAimTarget);
                        Vector3 angle = myTransform.localEulerAngles;
                        angle.x = 0;
                        myTransform.localEulerAngles = angle;
                }

                if (muzzle != null)
                {
                        Vector3 dir = aimTarget - muzzle.position;
                        dir.Normalize();
                        muzzleDirection.SetLookRotation(dir);
                }

                if (muzzle2 != null)
                {
                        Vector3 dir = aimTarget - muzzle2.position;
                        dir.Normalize();
                        muzzle2Direction.SetLookRotation(dir);
                }

#if (PLAYER_DEBUG_AIMING)
                //Debug.DrawRay(muzzle.position, muzzle.forward * 10, Color.blue);
                Debug.DrawRay(muzzle.position, dir * 5, Color.blue);

                Debug.DrawLine(muzzle.position, aimZombieTarget, Color.red);
                Debug.DrawLine(muzzle.position, aimPlaneTarget, Color.green);
                Debug.DrawLine(muzzle.position, groundAimTarget, Color.yellow);
#endif
                //Debug.Log("UpdateAim END\n");
        }


        // Second Weapon (for dual wield)
        //------------------------------------------------------------------------

        [SerializeField]
        Transform weaponMount2;
```

```
        void OnEquippedWeaponReloadComplete()
        {
                _weaponList.SetAmmo(equippedWeaponType, equippedWeapon.currentAmmo, eq
        }

        // Player Weapons
        //------------------------------------------------------------------------
        Dictionary<WeaponType, PlayerEquippedWeapon> weapon = new Dictionary<WeaponType, PlayerEqui

        WeaponData GetWeaponData(WeaponType t)
        {
                return ItemDatabase.GetWeaponData(t, PlayerData.Singleton.GetUpgradeLevel(t));
        }

        WeaponType equippedWeaponType = WeaponType.Handgun;
        PlayerEquippedWeapon equippedWeapon;
        WeaponData equippedWeaponInfo;

        List<WeaponType> cycleWeaponList = new List<WeaponType>();
        int cycleWeaponIdx = 0;

        bool autoSwitching = false;

        // for calculating accuracy at game end results
        int bulletsFired = 0;
        int bulletsThatHit = 0;

        public void OnHitZombie(WeaponType weaponType)
        {
                bulletsThatHit++;
                weapon[weaponType].bulletsThatHit++;
                PlayerStats.Singleton.OnHitZombie(weaponType);
        }

        public WeaponType GetEquippedWeaponType()
        {
                return equippedWeaponType;
        }

        public List<WeaponType> GetWeaponList()
        {
                return cycleWeaponList;
        }

        public PlayerEquippedWeapon GetEquippedWeaponInfo(int idx)
        {
                return weapon[cycleWeaponList[idx]];
        }

        public WeaponData GetWeaponInfo(int idx)
        {
                return GetWeaponData(cycleWeaponList[idx]);
        }

        public int GetReserveAmmo(WeaponType type)
        {
                if (weapon[type].sharedAmmo != WeaponType.None)
                {
                        return weapon[weapon[type].sharedAmmo].reserveAmmo;
                }
                else
                {
                        return weapon[type].reserveAmmo;
                }
        }

        void AddAmmo(WeaponType whichWeapon, int amount)
        {
                PlayerData.Singleton.AddAmmo(whichWeapon, amount);
        }

        // gain new ammo
        public void PickUpWeapon(WeaponType whichWeapon)
        {
                PlayerStats.Singleton.OnPickUp(whichWeapon);
                PlayerData.Singleton.OnPickUpWeapon(whichWeapon);

                WeaponData weaponPickedUpData = GetWeaponData(whichWeapon);

                bool isNewWeapon = false;
                if (weaponPickedUpData.usesSharedAmmo)
                {
                        isNewWeapon = PlayerData.Singleton.HasEverPickedUp(whic
                }
                else
                {
                        isNewWeapon = (weapon[whichWeapon].HasAmmo() == false
                }


                AddAmmo(weaponPickedUpData.ammoType, weaponPickedUpData.ammoAmount

                Debug.Log("weapon picked up: " + whichWeapon + " ammo type: " + weaponPicke

                if (isNewWeapon)
                {
                        // also reload this weapon since it's new
                        weapon[whichWeapon].currentAmmo += CalculateAmmoToLo
                        OnNewWeapon(whichWeapon);
                }
                else
                {
                        OnWeaponChangeAmmo(whichWeapon);
                }

                if (OptionsNGUIScreen.IsSoundsAllowed)
                {
                        AudioSource.PlayClipAtPoint(GetWeaponData(whichWeapon)
                }

                if (ShouldAutoSwitch(equippedWeaponType))
                {
                        autoSwitching = true;
                        EquipWeapon(whichWeapon);
                }
        }

        bool ShouldAutoSwitch(WeaponType weaponHeld)
        {
                return autoSwitchWeapon && (weaponHeld == WeaponType.Handgun || weaponHe
        }

        void SetEquippedWeapon(WeaponType whichWeapon)
        {
                if (equippedWeapon != null && !autoSwitching && equippedWeapon.isReloading &
                {
                        PlayerStats.Singleton.OnNewYorkReload();
                }

                equippedWeaponType = whichWeapon;
                equippedWeaponInfo = GetWeaponData(whichWeapon);
```

```
//#define PLAYER_DEBUG_AIMING

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

public class PlayerEquippedWeapon
{
    public int allAmmo
    {
        get{ return (sharedAmmo == WeaponType.None) ? PlayerData.Singleton.GetAmmo(type) : PlayerData.Singleton.GetAmmo(sharedAmmo); }
    }
    public int reserveAmmo
    {
        get{ return allAmmo - currentAmmo; }
    }
    public int currentAmmo = 0;
    public float timeSinceLastFire = 0.0f;
    public float shotSpread = 0.0f;

    public bool isReloading = false;
    public float timeReloadStarted = 0.0f;

    public int bulletsThatHit = 0;
    public int bulletsFired = 0;

    public WeaponType type = WeaponType.None;
    public WeaponType sharedAmmo = WeaponType.None;

    public bool HasAmmo()
    {
        return reserveAmmo > 0 || currentAmmo > 0;
    }

    public float GetAccuracy()
    {
        float accuracy = 0.0f;

        if (bulletsFired != 0)
        {
            accuracy = ((float) bulletsThatHit/bulletsFired)*100.0f;
            //accuracy = Mathf.Round(accuracy);
            if (accuracy > 100.0f)
            {
                accuracy = 100.0f;
            }
        }

        return accuracy;
    }

    public PlayerEquippedWeapon(WeaponType newType, WeaponType newSharedAmmo)
    {
        type = newType;
        sharedAmmo = newSharedAmmo;
    }

    public PlayerEquippedWeapon(WeaponType newType)
    {
        type = newType;
        sharedAmmo = WeaponType.None;
    }
}

public class Player : MonoBehaviour
{
    PlayerAnimation anim;
    PlayerMovement movement;
    CrossHair crosshair;

    public void OnPickUpItem()
    {
        currentResult.itemsPickedUp += 1;
        //Debug.Log("OnPickUpItem() " + currentResult.itemsPickedUp);
    }

    bool autoSwitchWeapon = false;

    float movementModifierDuration = 0.2f;
    float timeSinceLastMovementModifierApplied = 0.0f;

    //private LoadingScreen loadingScreen;
    SpawnSystem zombieSpawn;

    bool zBoyMode = false;

    public static Player GetPlayerScript(GameObject pGo)
    {
        if (pGo == null)
        {
            return null;
        }
        while (GM.GetComponent<Player>(pGo) == null && pGo.transform.parent != null)
        {
            pGo = pGo.transform.parent.gameObject;
        }
        Player p = GM.GetComponent<Player>(pGo);
        if (p == null)
        {
            Debug.LogError("Player: Could not find gameobject's player script!");
            return null;
        }
        return p;
    }

    ArmorData[] armorData;

    // Player Sounds
    //------------------------------------------------------------------------
    [SerializeField]
    AudioClip[] healSounds;

    [SerializeField]
    AudioClip[] hurtSounds;

    [SerializeField]
    AudioClip[] deathSounds;

    [SerializeField]
    string tauntFolder = "";

    [SerializeField]
    string[] startTaunts;

    [SerializeField]
    string[] shotgunTaunts;

    [SerializeField]
    string[] shotgunEquipTaunts;
```

```
    public void OnGrenadeExplode(GrenadeType t)
    {
        if (t == GrenadeType.Frag || t == GrenadeType.Landmine)
        {
            PlayTaunt(explodeTaunts, 5);
        }
        else if (t == GrenadeType.Molotov)
        {
            PlayTaunt(molotovTaunts, 5);
        }
    }

    void OnWin()
    {
        PlayTaunt(winTaunts, 5);

        showResults = true;
        crosshair.HideCrosshair();

        ComputeResultData();
        //StartCoroutine(StartFadeIn());

        ResultData resultsToShow = currentResult;
        if (Mode.currentGameMode == GameMode.Assault && (assault.AtFinalStage() || IsDead()))
        {
            resultsToShow = totalResult;
        }

        _combatResultsScreen.Show(resultsToShow);

        OnEndGame();
    }

    void OnLose(GrenadeType grenade, Vector3 explodeForce)
    {
        Mode.OnPlayerLose();

        zombieSpawn.StopSpawning();

        anim.PlayDeathAnimation(explodeForce);

        // play death sound effect
        if (OptionsNGUIScreen.IsSoundsAllowed && deathSounds.Length > 0)
        {
            AudioClip deathSound = deathSounds[Random.Range(0, deathSounds.Length)];
            AudioSource.PlayClipAtPoint(deathSound, myTransform.position, OptionsNGUIScreen.SoundsVolume);
        }

        ComputeResultData();

        crosshair.HideCrosshair();

        //StartCoroutine(StartFadeIn());

        ResultData resultsToShow = currentResult;
        if (Mode.currentGameMode == GameMode.Assault && (assault.AtFinalStage() || IsDead()))
        {
            resultsToShow = totalResult;
        }

        _combatResultsScreen.Show(resultsToShow);

        if (grenade != GrenadeType.None)
        {
            PlayerStats.Singleton.On[illegible](grenade);
        }

        OnEndGame();
    }

    public void OnEndGame()
    {
        PlayerStats.Singleton.WriteStats();
        SaveUnusedWeapons();
    }

    void SaveUnusedWeapons()
    {
        PlayerData.Singleton.CommitChanges();
    }


    // Player Weapon Mount
    //------------------------------------------------------------------------
    [SerializeField]
    Transform weaponMount;

    // we need to store a handle to this so we can delete it later (when player switches weapons)
    Transform weaponMesh;

    void MountWeapon(WeaponData weapon)
    {
        if (weapon.dualWeild)
        {
            muzzle = MountWeapon(weapon.mesh, ref weaponMesh, weaponMount);
            muzzle2 = MountWeapon(weapon.mesh, ref weaponMesh2, weaponMount2);
        }
        else
        {
            if (weaponMesh2 != null)
            {
                Destroy(weaponMesh2.gameObject);
            }
            muzzle = MountWeapon(weapon.mesh, ref weaponMesh, weaponMount);
        }
    }

    Transform MountWeapon(Transform weaponMeshPrefab, ref Transform weaponMeshHandle, Transform mountPoint)
    {
        if (weaponMeshHandle != null)
        {
            Destroy(weaponMeshHandle.gameObject);
        }

        //Debug.Log("weapon " + obj.name + " mounted");
        weaponMeshHandle = Instantiate(weaponMeshPrefab, Vector3.zero, Quaternion.identity) as Transform;
        weaponMeshHandle.parent = mountPoint;
        weaponMeshHandle.localRotation = weaponMeshPrefab.rotation;
        weaponMeshHandle.localPosition = Vector3.zero;

        return weaponMeshHandle.Find("Armature/Muzzle");
    }

    void MountGrenade(Transform obj)
    {
        if (weaponMesh != null)
        {
            Destroy(weaponMesh.gameObject);
        }
```

```
    AimCoords aim;

    void UpdateAim()
    {
        //Debug.Log("UpdateAim STA");

        Ray ray = Camera.main.ScreenPointToRay(aim.GetForScreenPointToRay());
        RaycastHit hit;

        bool foundZombie = false;
//#if (UNITY_IPHONE || UNITY_ANDROID)
        int targetableLayers = 1 << 12 | 1 << 10; // touchboxes or hitboxes
//#else
//      int targetableLayers = 1 << 10; // targetables (zombie hitboxes)
//#endif
        if (Physics.Raycast(ray, out hit, Mathf.Infinity, targetableLayers))
        {
            if (hit.collider.gameObject.tag == "Zombie")
            {
                foundZombie = true;
                aimTarget = hit.collider.bounds.center;
#if (PLAYER_DEBUG_AIMING)
                //Debug.Log("aimed at zombie: " + aimTarget);
                aimZombieTarget = aimTarget;
#endif
            }
        }

        targetableLayers = 1 << 11; // aim plane
        if (Physics.Raycast(ray, out hit, Mathf.Infinity, targetableLayers))
        {
            // face where crosshair is
            //if (aim.fireLapse)
            if (anim.InGrenadeThrowAnimation() == false)
            {
                myTransform.LookAt(hit.point);
                Vector3 angle = myTransform.localEulerAngles;
                angle.x = 0;
                myTransform.localEulerAngles = angle;
            }


            if (!foundZombie)
            {
                aimTarget = hit.point;

                // uncomment to lock muzzle to single plane
                //Vector3 angle = muzzle.localEulerAngles;
                //angle.x = 0;
                //muzzle.localEulerAngles = angle;

#if (PLAYER_DEBUG_AIMING)
                aimPlaneTarget = aimTarget;
                //Debug.Log("aimed at aim plane: " + aimTarget);
#endif
            }
        }
        else
        {
            Debug.LogError("Aim Plane not found");
        }

        ray = Camera.main.ScreenPointToRay(aim.GetTouchForScreenPointToRay());
        targetableLayers = 1 << 12 | 1 << 10; // touchboxes or hitboxes
        foundZombie = false;
        if (Physics.Raycast(ray, out hit, Mathf.Infinity, targetableLayers))
        {
            if (hit.collider.gameObject.tag == "Zombie")
            {
                foundZombie = true;
                groundAimTarget = hit.collider.bounds.center;
            }
        }

        if (!foundZombie)
        {
            targetableLayers = 1 << 8; // ground
            if (Physics.Raycast(ray, out hit, Mathf.Infinity, targetableLayers))
            {
                groundAimTarget = hit.point;
            }
            //else
            //{
            //      Debug.LogError("Ground not found");
            //}
        }
        if ((aim.touchFire && inThrowGrenadeMode) || anim.InGrenadeThrowAnimation())
        {
            // face where the grenade is thrown
            myTransform.LookAt(groundAimTarget);
            Vector3 angle = myTransform.localEulerAngles;
            angle.x = 0;
            myTransform.localEulerAngles = angle;
        }

        if (muzzle != null)
        {
            Vector3 dir = aimTarget - muzzle.position;
            dir.Normalize();
            muzzleDirection.SetLookRotation(dir);
        }

        if (muzzle2 != null)
        {
            Vector3 dir = aimTarget - muzzle2.position;
            dir.Normalize();
            muzzle2Direction.SetLookRotation(dir);
        }

#if (PLAYER_DEBUG_AIMING)
        //Debug.DrawRay(muzzle.position, muzzle.forward * 10, Color.blue);
        Debug.DrawRay(muzzle.position, dir * 5, Color.blue);

        Debug.DrawLine(muzzle.position, aimZombieTarget, Color.red);
        Debug.DrawLine(muzzle.position, aimPlaneTarget, Color.green);
        Debug.DrawLine(muzzle.position, groundAimTarget, Color.yellow);
#endif
        //Debug.Log("UpdateAim END\n");
    }

    // Second Weapon (for dual wield)
    //------------------------------------------------------------------------

    [SerializeField]
    Transform weaponMount2;
```

```
    void OnEquippedWeaponReloadComplete()
    {
        _weaponList.SetAmmo(equippedWeaponType, equippedWeapon.currentAmmo, eq
    }

    // Player Weapons
    //------------------------------------------------------------------------
    Dictionary<WeaponType, PlayerEquippedWeapon> weapon = new Dictionary<WeaponType, PlayerEqui

    WeaponData GetWeaponData(WeaponType t)
    {
        return ItemDatabase.GetWeaponData(t, PlayerData.Singleton.GetUpgradeLevel(t));
    }

    WeaponType equippedWeaponType = WeaponType.Handgun;
    PlayerEquippedWeapon equippedWeapon;
    WeaponData equippedWeaponInfo;

    List<WeaponType> cycleWeaponList = new List<WeaponType>();
    int cycleWeaponIdx = 0;

    bool autoSwitching = false;

    // for calculating accuracy at game end results
    int bulletsFired = 0;
    int bulletsThatHit = 0;

    public void OnHitZombie(WeaponType weaponType)
    {
        bulletsThatHit++;
        weapon[weaponType].bulletsThatHit++;
        PlayerStats.Singleton.OnHitZombie(weaponType);
    }

    public WeaponType GetEquippedWeaponType()
    {
        return equippedWeaponType;
    }

    public List<WeaponType> GetWeaponList()
    {
        return cycleWeaponList;
    }

    public PlayerEquippedWeapon GetEquippedWeaponInfo(int idx)
    {
        return weapon[cycleWeaponList[idx]];
    }

    public WeaponData GetWeaponInfo(int idx)
    {
        return GetWeaponData(cycleWeaponList[idx]);
    }

    public int GetReserveAmmo(WeaponType type)
    {
        if (weapon[type].sharedAmmo != WeaponType.None)
        {
            return weapon[weapon[type].sharedAmmo].reserveAmmo;
        }
        else
        {
            return weapon[type].reserveAmmo;
        }
    }

    void AddAmmo(WeaponType whichWeapon, int amount)
    {
        PlayerData.Singleton.AddAmmo(whichWeapon, amount);
    }

    // gain new ammo
    public void PickUpWeapon(WeaponType whichWeapon)
    {
        PlayerStats.Singleton.OnPickUp(whichWeapon);
        PlayerData.Singleton.OnPickUpWeapon(whichWeapon);

        WeaponData weaponPickedUpData = GetWeaponData(whichWeapon);

        bool isNewWeapon = false;
        if (weaponPickedUpData.usesSharedAmmo)
        {
            isNewWeapon = PlayerData.Singleton.HasEverPickedUp(whic
        }
        else
        {
            isNewWeapon = (weapon[whichWeapon].HasAmmo() == false
        }


        AddAmmo(weaponPickedUpData.ammoType, weaponPickedUpData.ammoAmount

        Debug.Log("weapon picked up: " + whichWeapon + " ammo type: " + weaponPicke

        if (isNewWeapon)
        {
            // also reload this weapon since it's new
            weapon[whichWeapon].currentAmmo += CalculateAmmoToLo
            OnNewWeapon(whichWeapon);
        }
        else
        {
            OnWeaponChangeAmmo(whichWeapon);
        }

        if (OptionsNGUIScreen.IsSoundsAllowed)
        {
            AudioSource.PlayClipAtPoint(GetWeaponData(whichWeapon)
        }

        if (ShouldAutoSwitch(equippedWeaponType))
        {
            autoSwitching = true;
            EquipWeapon(whichWeapon);
        }
    }

    bool ShouldAutoSwitch(WeaponType weaponHeld)
    {
        return autoSwitchWeapon && (weaponHeld == WeaponType.Handgun || weaponHe
    }

    void SetEquippedWeapon(WeaponType whichWeapon)
    {
        if (equippedWeapon != null && !autoSwitching && equippedWeapon.isReloading &
        {
            PlayerStats.Singleton.OnNewYorkReload();
        }

        equippedWeaponType = whichWeapon;
        equippedWeaponInfo = GetWeaponData(whichWeapon);
```

See it all move at

http://youtu.be/GfhAZg_T2Gg

2,544

lines of code!

2,544

lines of code!

That's not something to be proud of.

2,544

lines of code!

That's not something to be proud of.

That is madness!

2,544

lines of code!

That's not something to be proud of.

That is madness!

2,544

lines of code!

That's not something to be proud of.

That is madness!

No teammate will want to read that amount of code in one file.

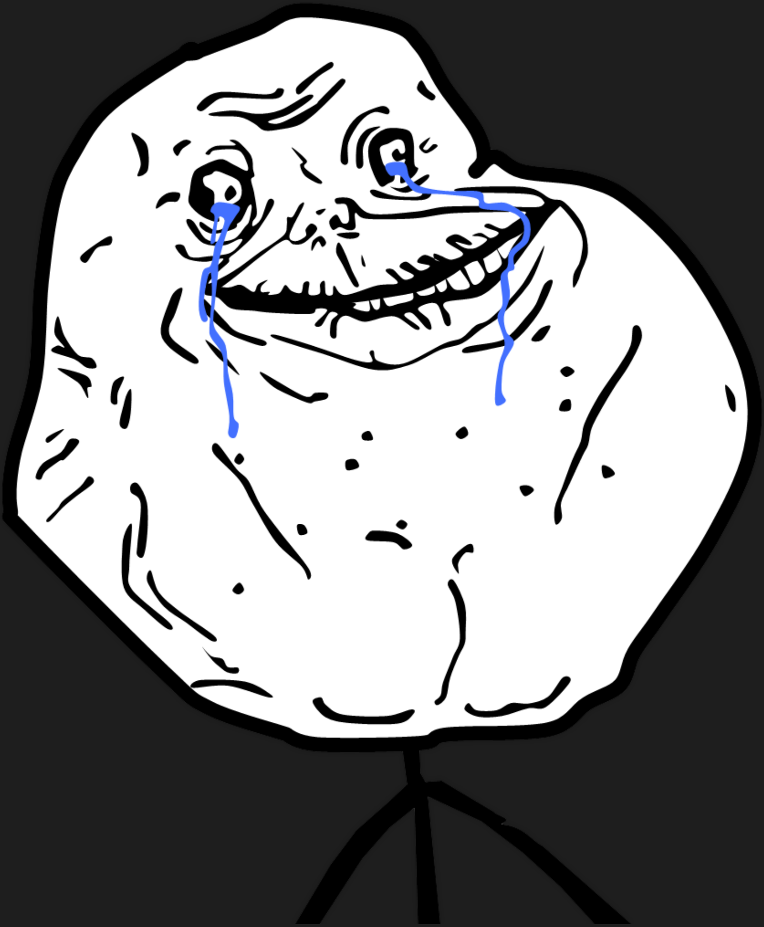
What if you work alone?

What if you work alone?

What if you don't care about readability?

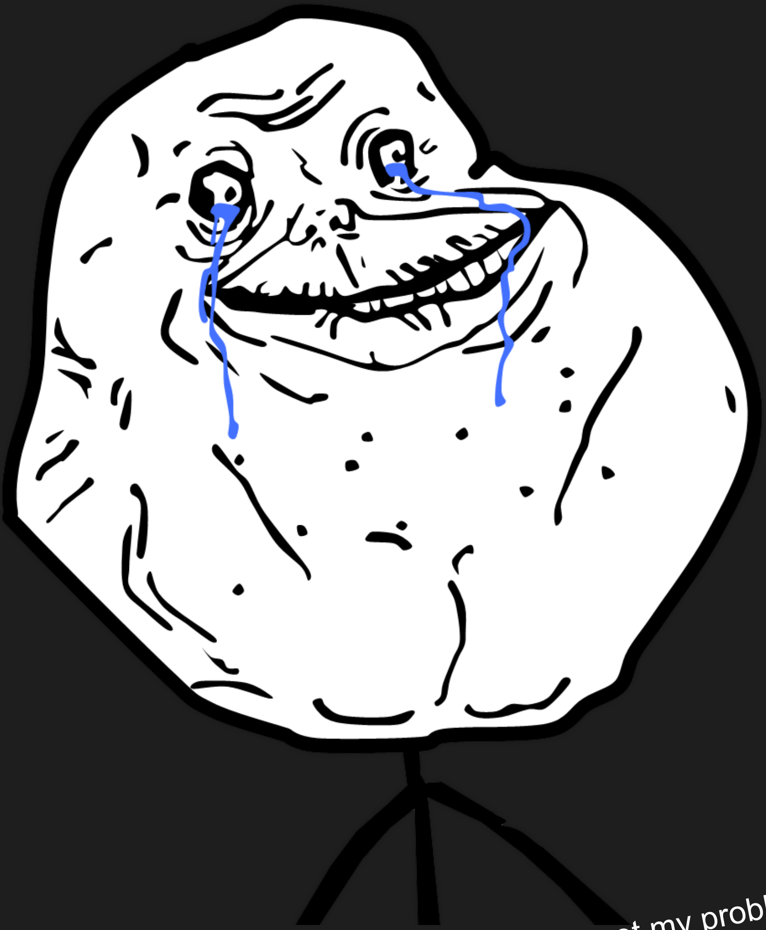
What if you work alone? (forever?)

What if you don't care about readability?



What if you work alone? (forever?)

What if you don't care about readability?



(Disclaimer: This one's not my problem.)

Better organized code
is still important.

Better organized code
is still important.

Even if you work
(forever) alone.

Better organized code
is still important.

Even if you work
(forever) alone.

Why?

Let's say I want to upgrade my Zombie Fields game to Unity 4

Let's say I want to upgrade my Zombie Fields game to Unity 4

Let's say I want to upgrade my Zombie Fields game to Unity 4

New Particle Effects system!

Let's say I want to upgrade my Zombie Fields game to Unity 4

New Particle Effects system!

New Animation system!

Let's say I want to upgrade my Zombie Fields game to Unity 4

New Particle Effects system!

New Animation system!

(Mecanim)

Let's say I want to upgrade my Zombie Fields game to Unity 4

New Particle Effects system!

New Animation system!

(Mecanim)

Cool, let's switch to Mecanim!

Let's say I want to upgrade my Zombie Fields game to Unity 4

New Particle Effects system!

New Animation system!

(Mecanim)

Cool, let's switch to Mecanim!

Wait...

Let's say I want to upgrade my Zombie Fields game to Unity 4

New Particle Effects system!

New Animation system!

(Mecanim)

Cool, let's switch to Mecanim!

Wait...

That means editing my Player class to remove old animation system and replace with Mecanim code.

I would need to revisit my 2,544 lines of code…

I would need to revisit my 2,544 lines of code...

And look for places where I use the old animation system, and replace them with code that uses Mecanim.

I would need to revisit my 2,544 lines of code...

And look for places where I use the old animation system, and replace them with code that uses Mecanim.

In fact, **every time** I want to replace something in my humongous Player class, I need to search through those 2,544 lines...

I would need to revisit my 2,544 lines of code...

And look for places where I use the old animation system, and replace them with code that uses Mecanim.

In fact, **every time** I want to replace something in my humongous Player class, I need to search through those 2,544 lines...

I would need to revisit my 2,544 lines of code...

And look for places where I use the old animation system, and replace them with code that uses Mecanim.

In fact, **every time** I want to replace something in my humongous Player class, I need to search through those 2,544 lines...

and this is just a mobile game, folks

How did it end up like that?

How did it end up like that?

Why am I making things harder for myself?

As you can see, it's a problem

As you can see, it's a problem

about COMPLEXITY

As you can see, it's a problem

about COMPLEXITY

about INFORMATION OVERLOAD

But turns out problems like these can be solved!

But turns out problems like these can be solved!

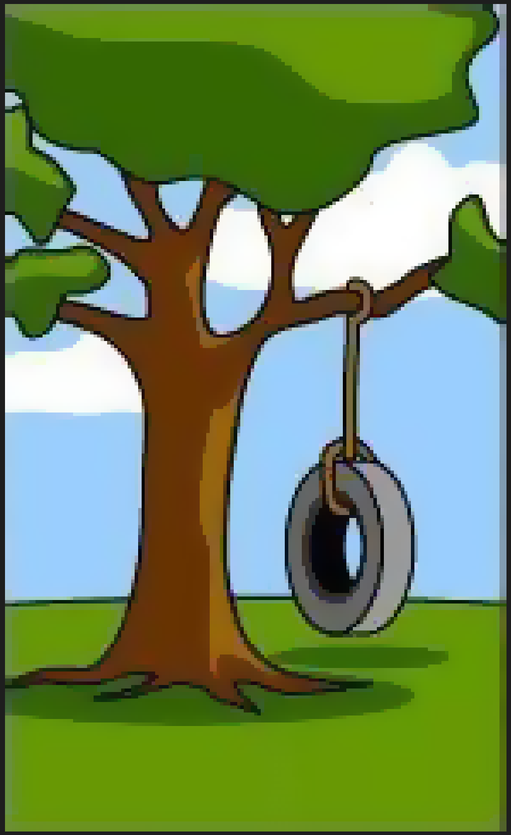
reducing COMPLEXITY!

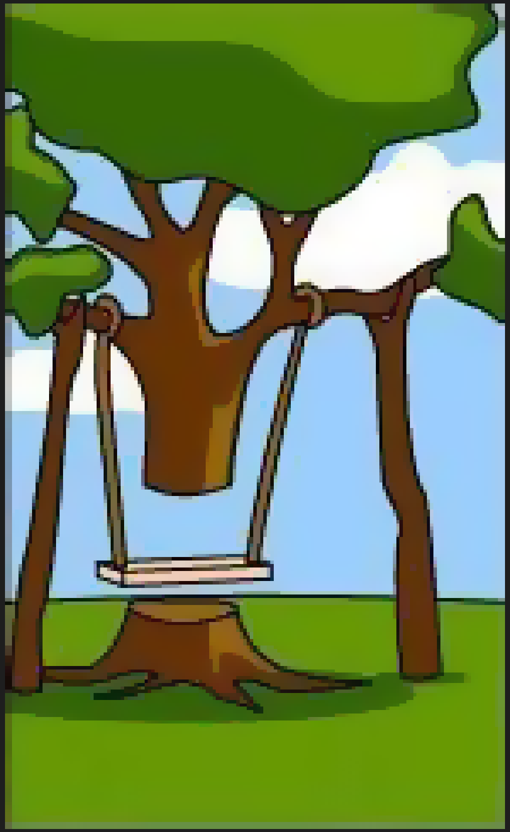
But turns out problems like these can be solved!

reducing COMPLEXITY!

managing INFORMATION!

So this doesn't have to happen!


What the game designer wanted


What the game programmer made

It's called

It's called

Software

Architecture

It's called

Software Architecture

(or “code architecture”)

It's called

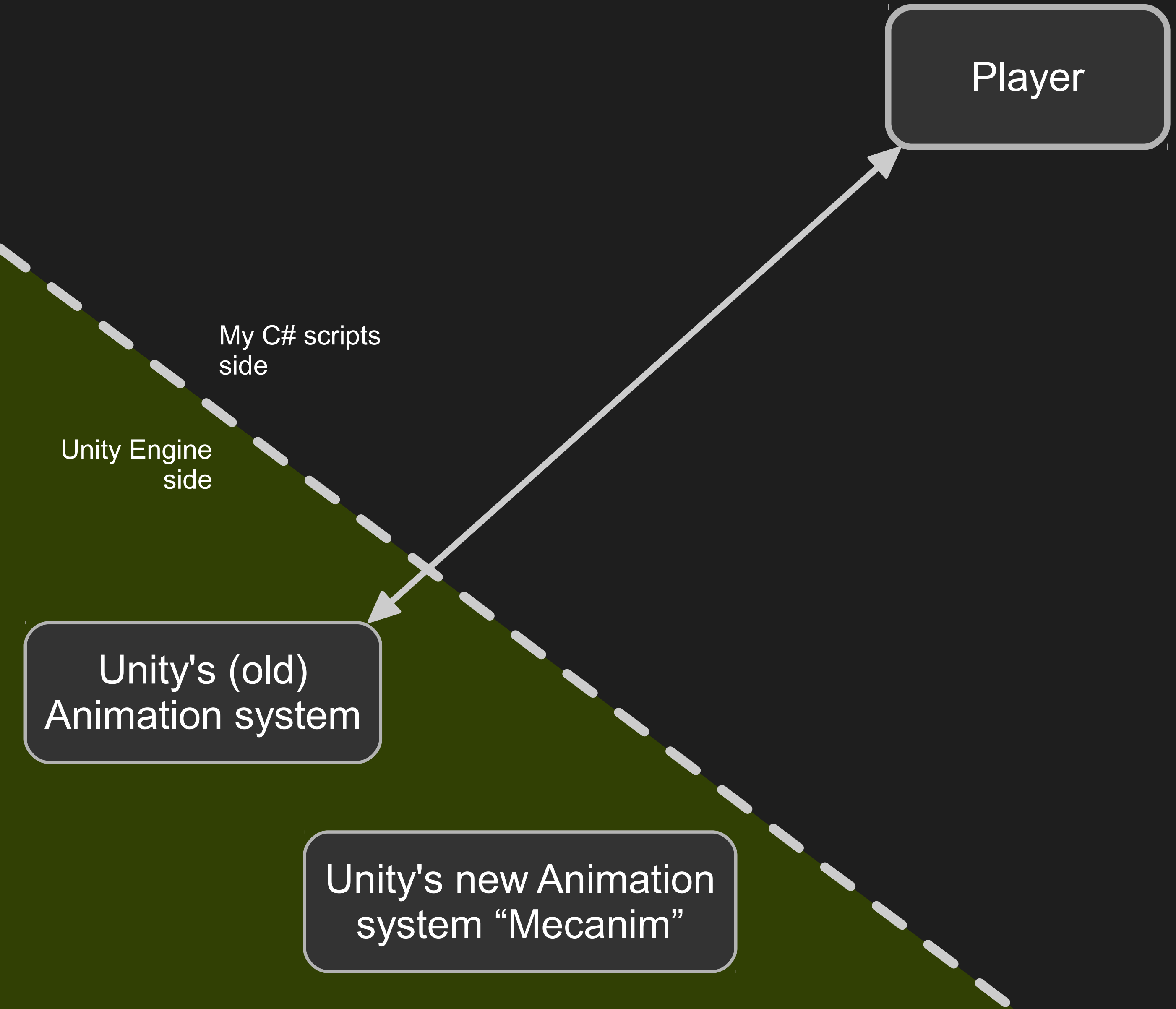
Software Architecture

(or “code architecture”)

(same thing)

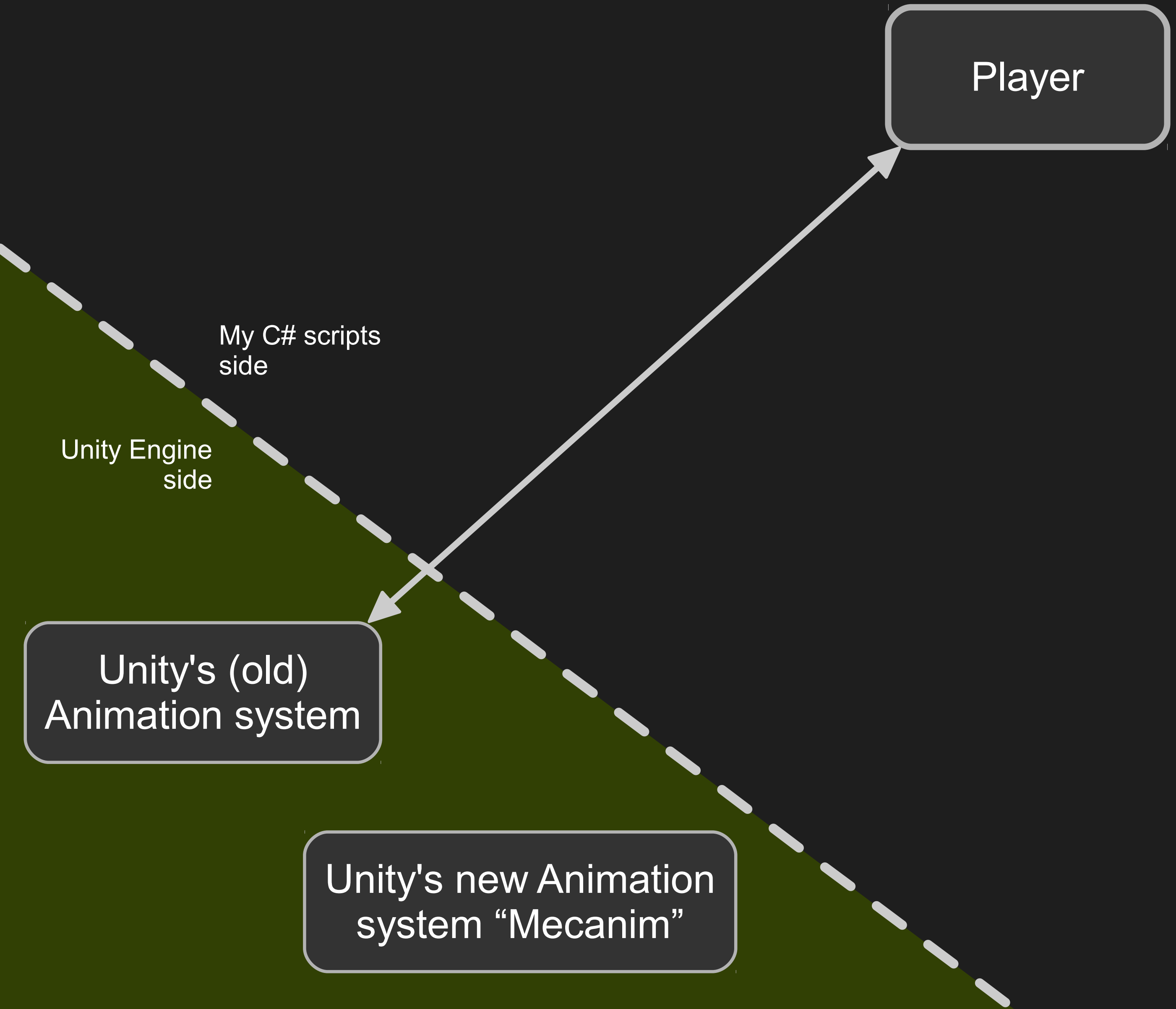
For example, in the earlier situation...

For example, in the earlier situation...

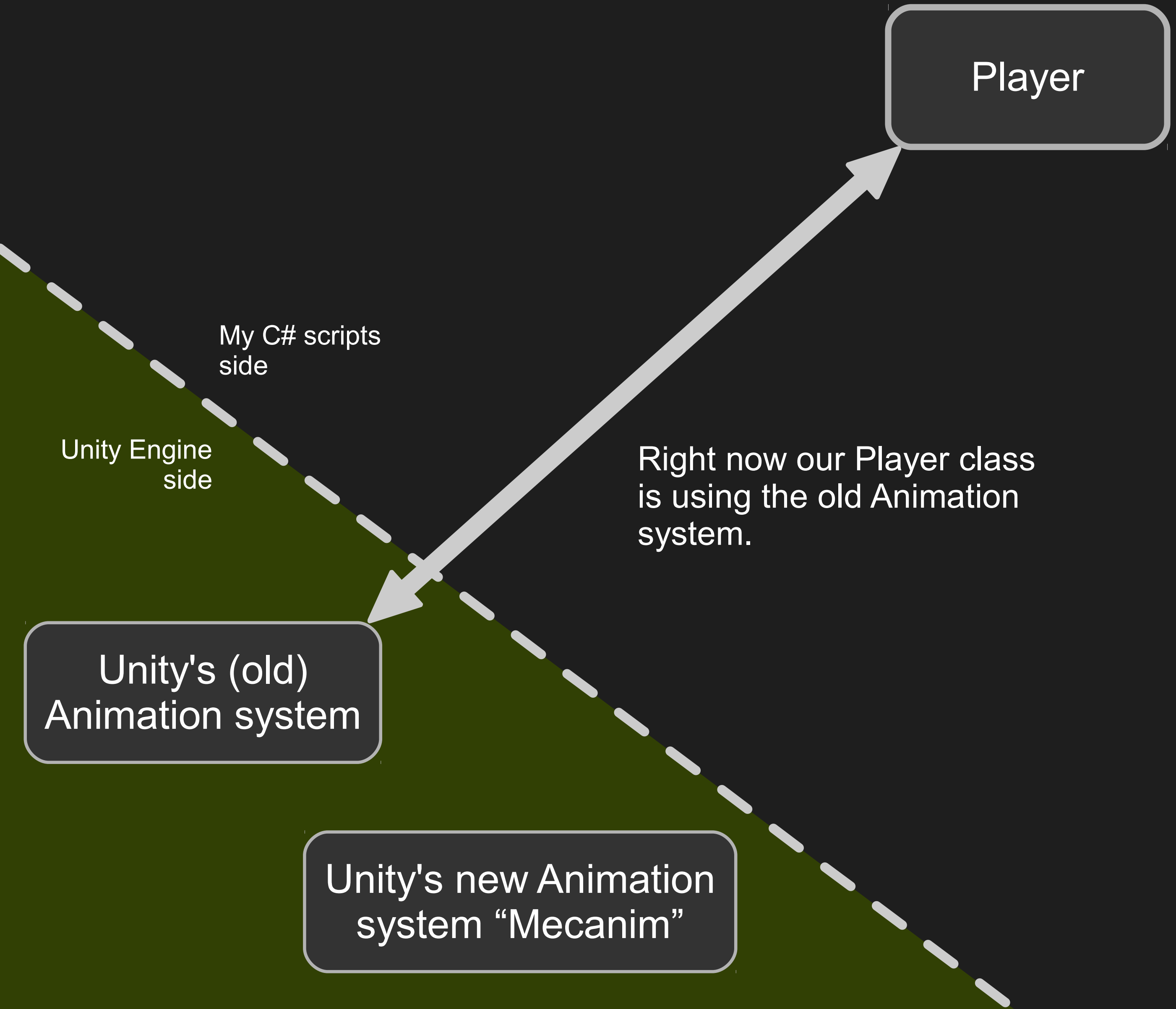

Player
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

For example, in the earlier situation...

So here's our Player class:

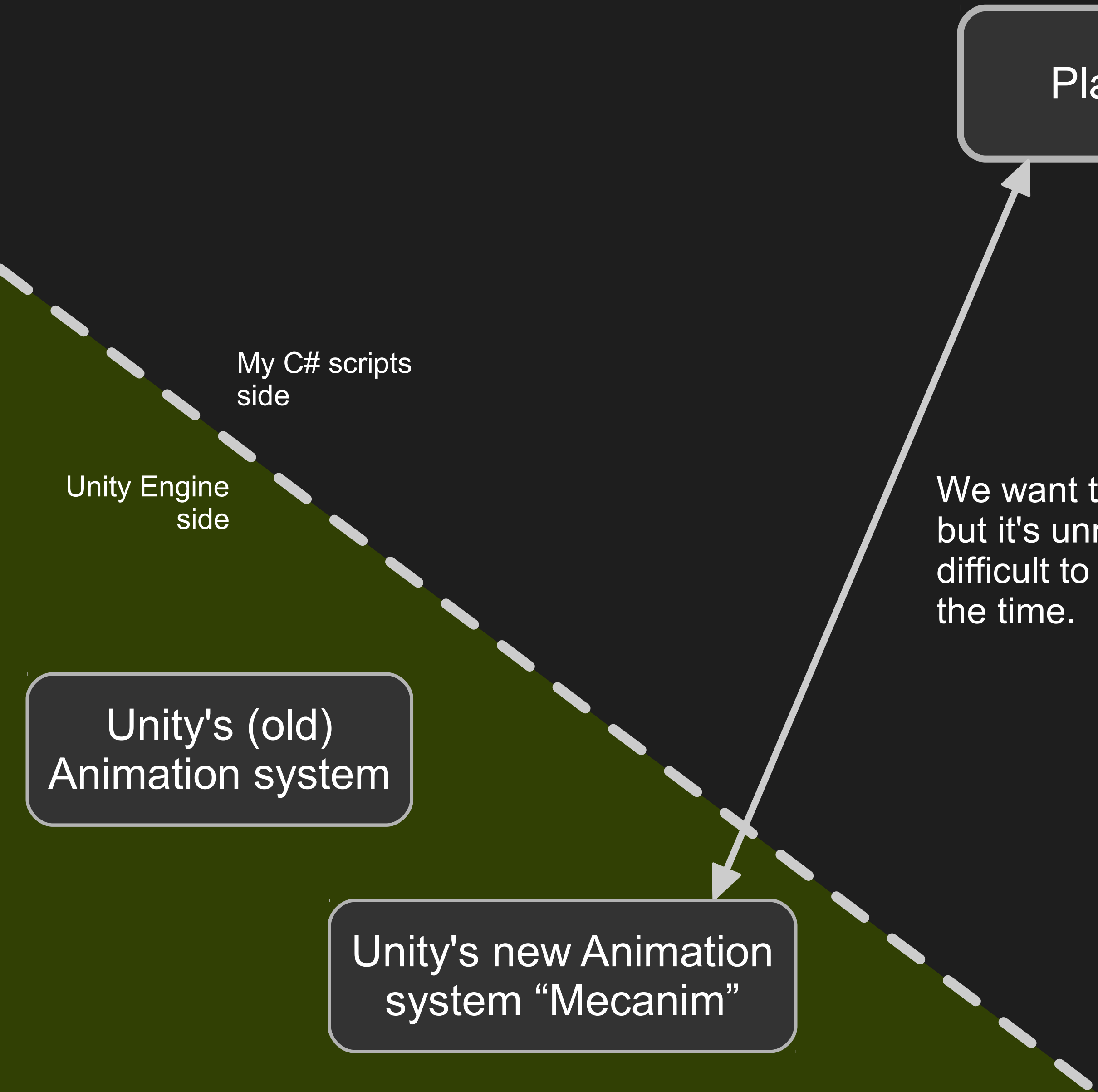

Player
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system "Mecanim"

For example, in the earlier situation...


Player
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

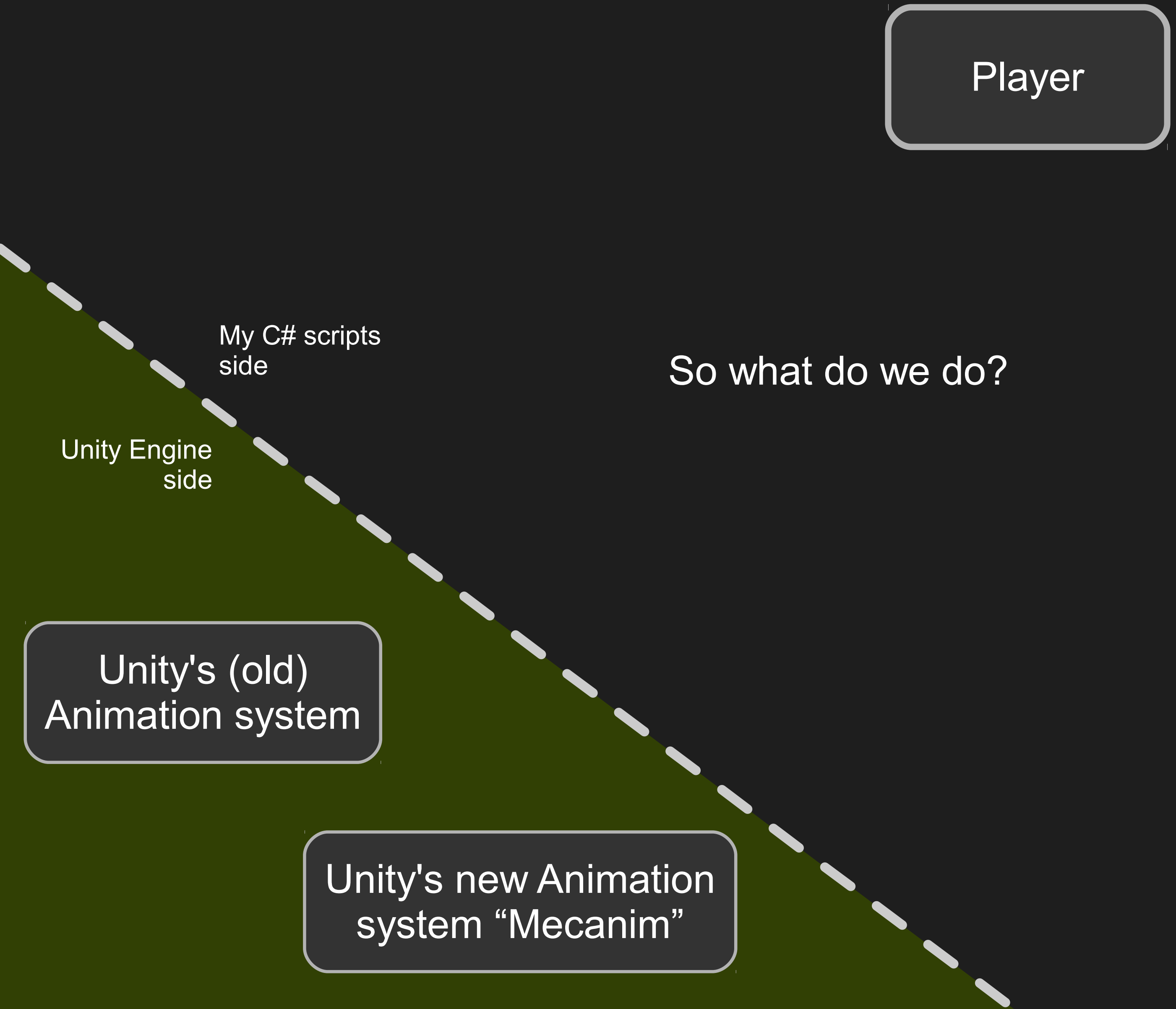
Right now our Player class is using the old Animation system.

For example, in the earlier situation...


Player
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system "Mecanim"

We want this to happen, but it's unreasonably difficult to do it like this all the time.

For example, in the earlier situation...

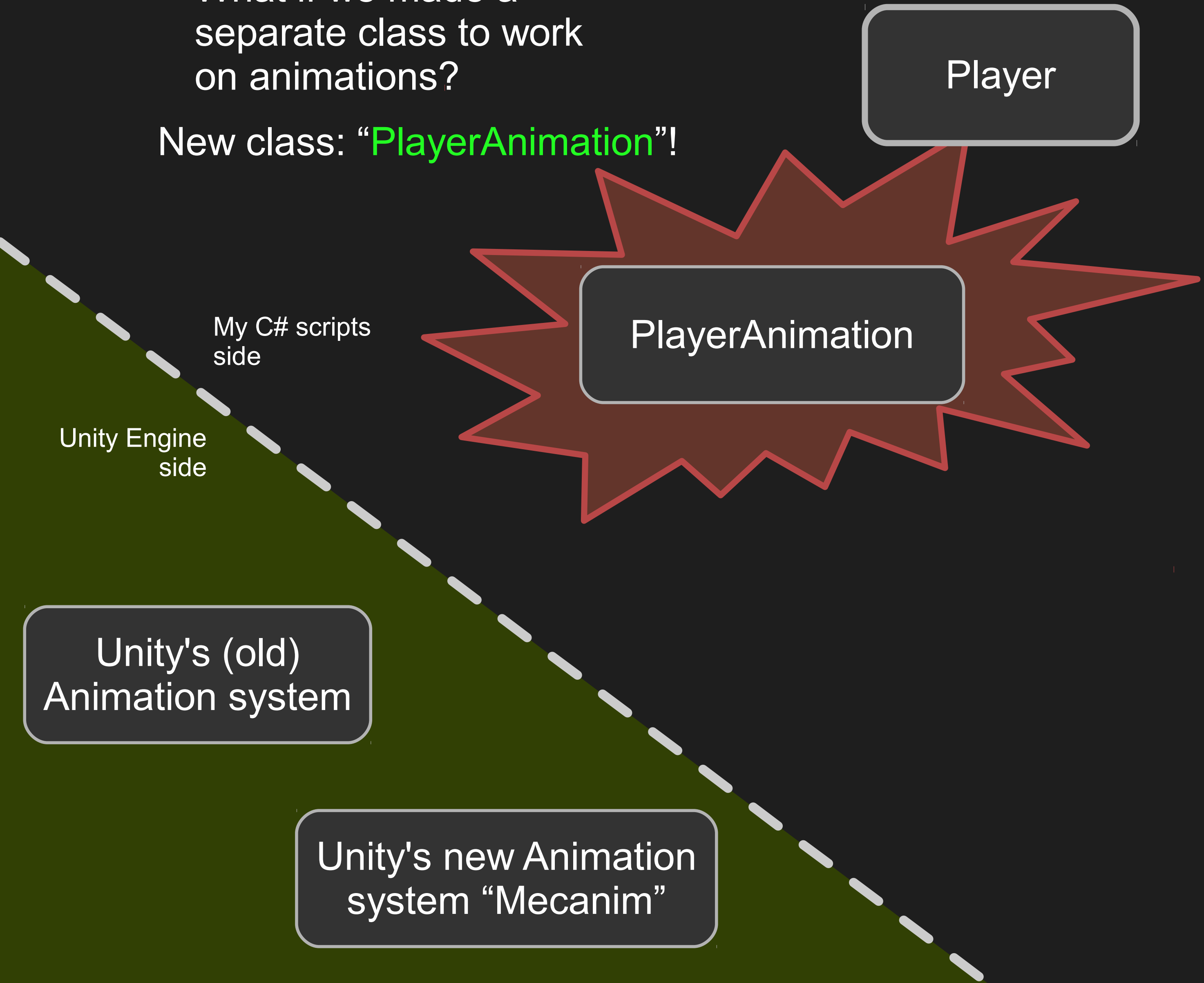

Player
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

So what do we do?

For example, in the earlier situation...

What if we made a separate class to work on animations?

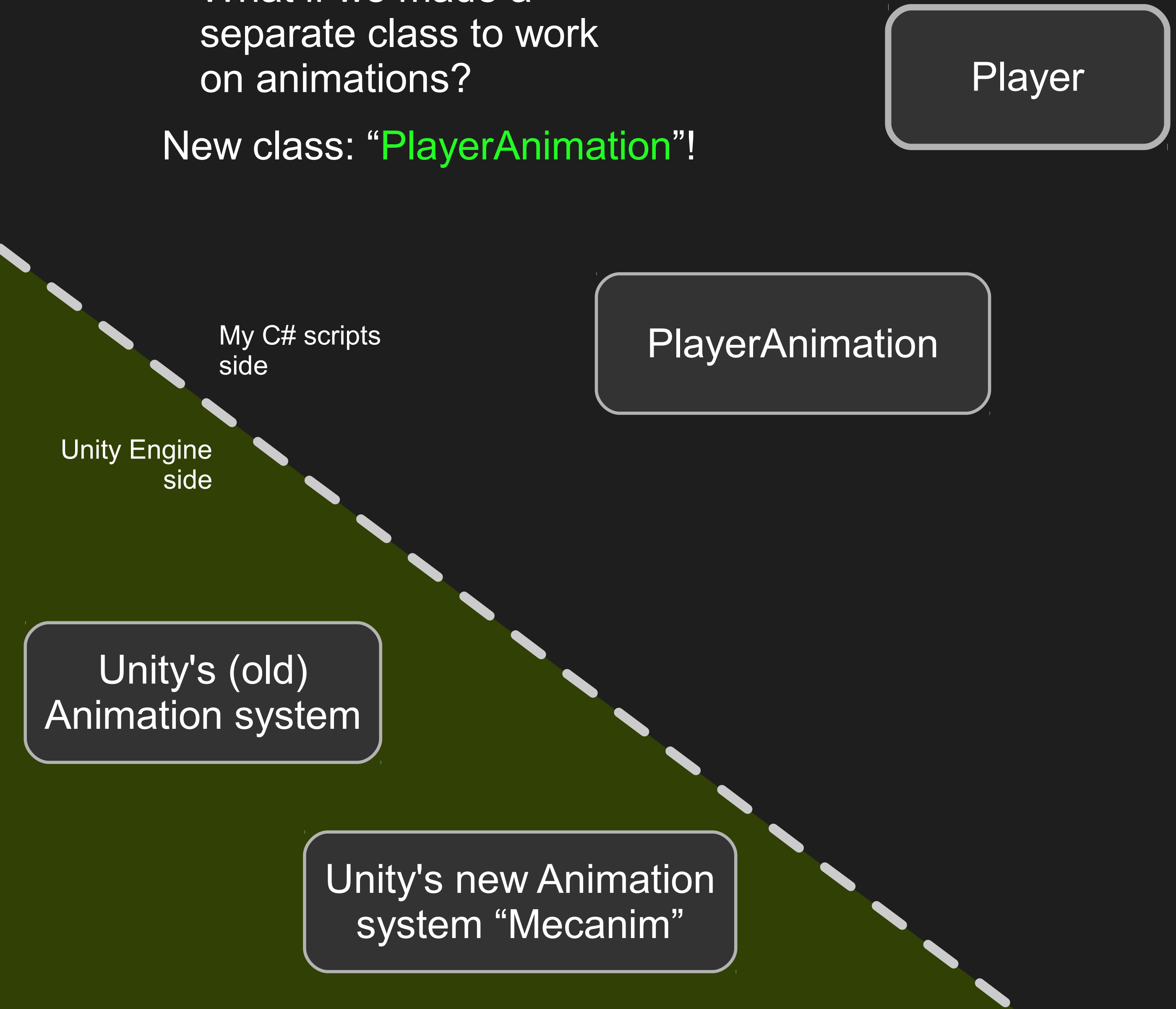
New class: “PlayerAnimation”!


Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

For example, in the earlier situation...

What if we made a separate class to work on animations?

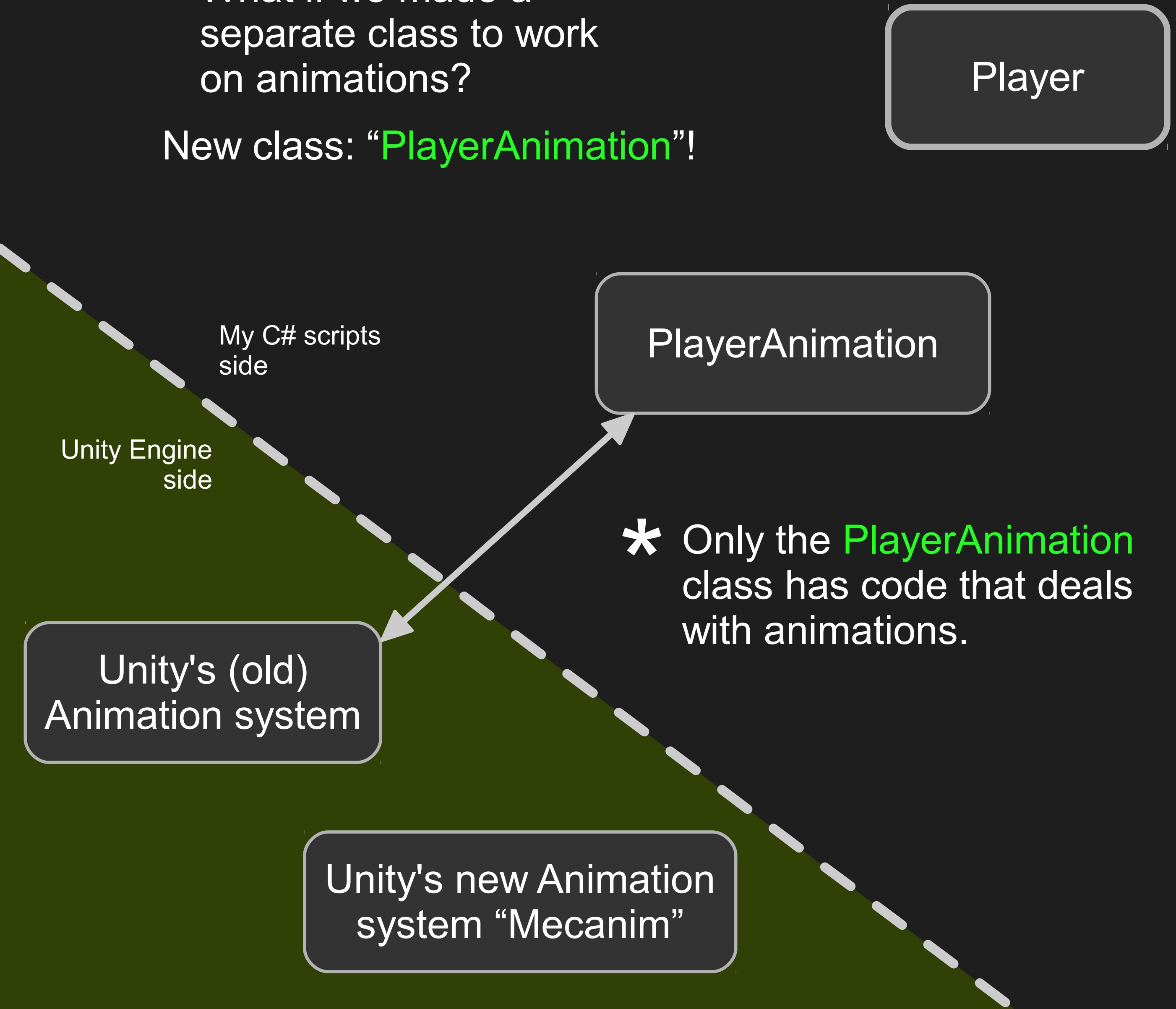
New class: “PlayerAnimation”!


Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

For example, in the earlier situation...

What if we made a separate class to work on animations?

New class: “PlayerAnimation”!

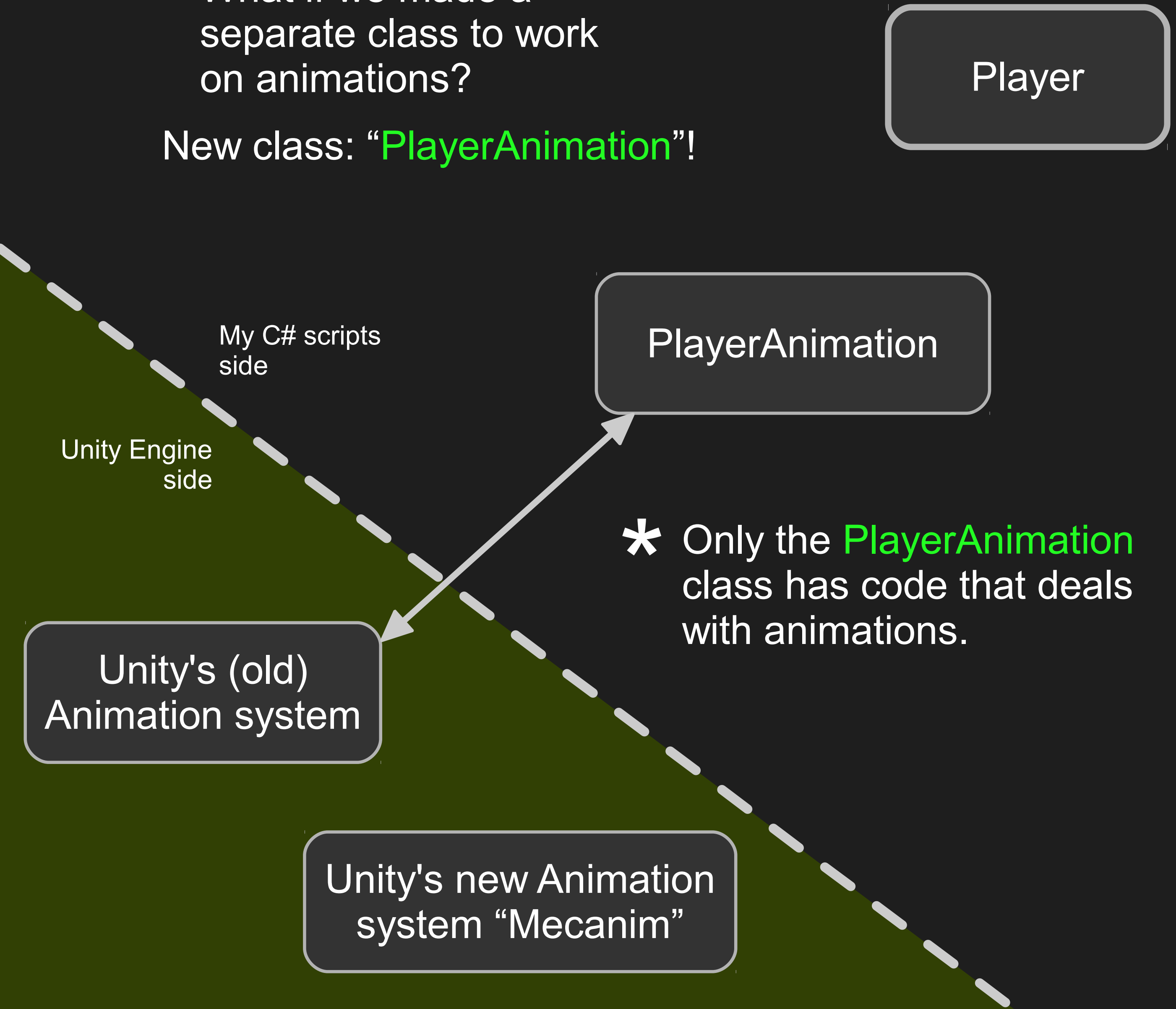

Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

* Only the PlayerAnimation class has code that deals with animations.

For example, in the earlier situation...

What if we made a separate class to work on animations?

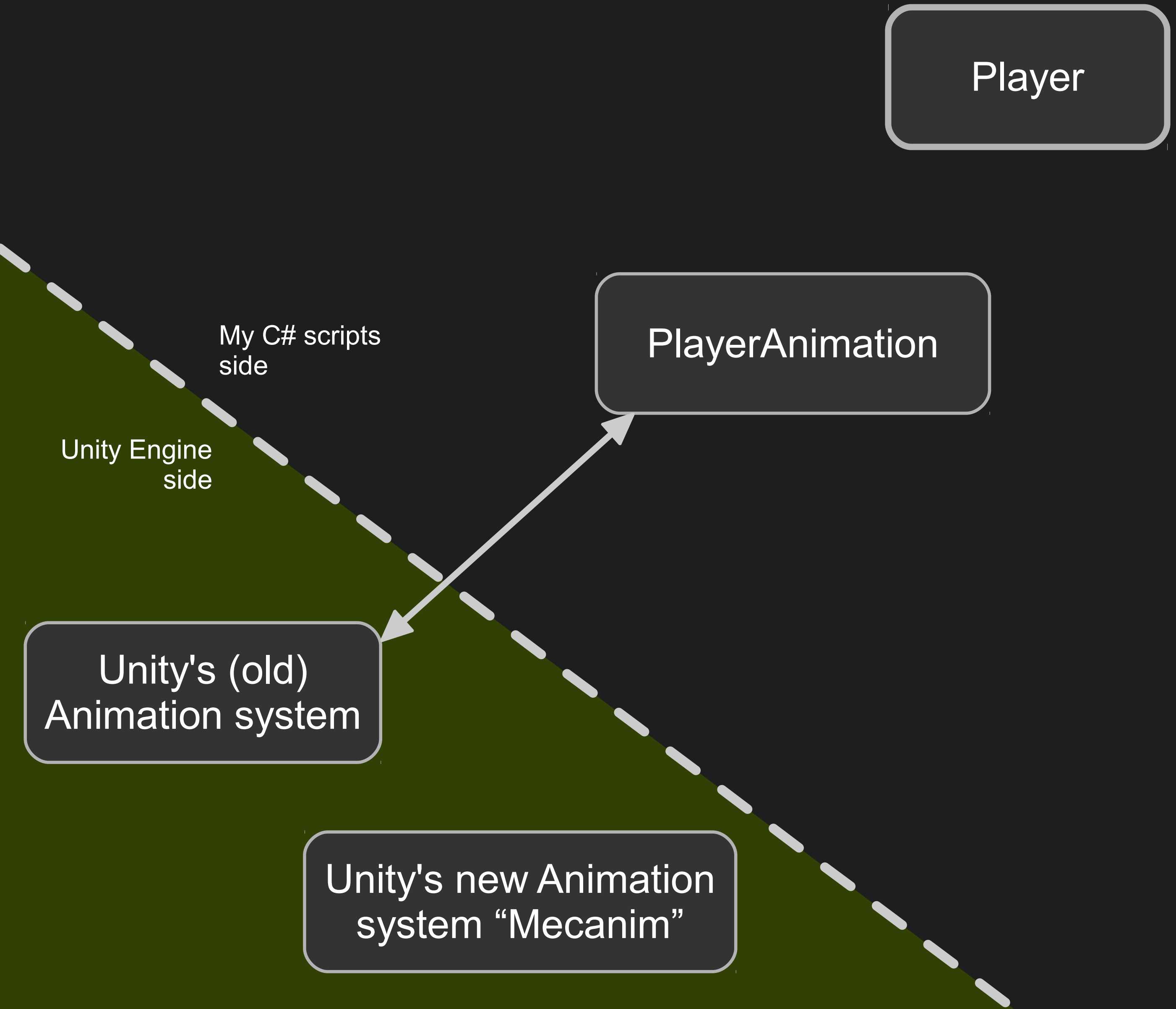
New class: “PlayerAnimation”!


Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

* All animation code in **Player** is removed.

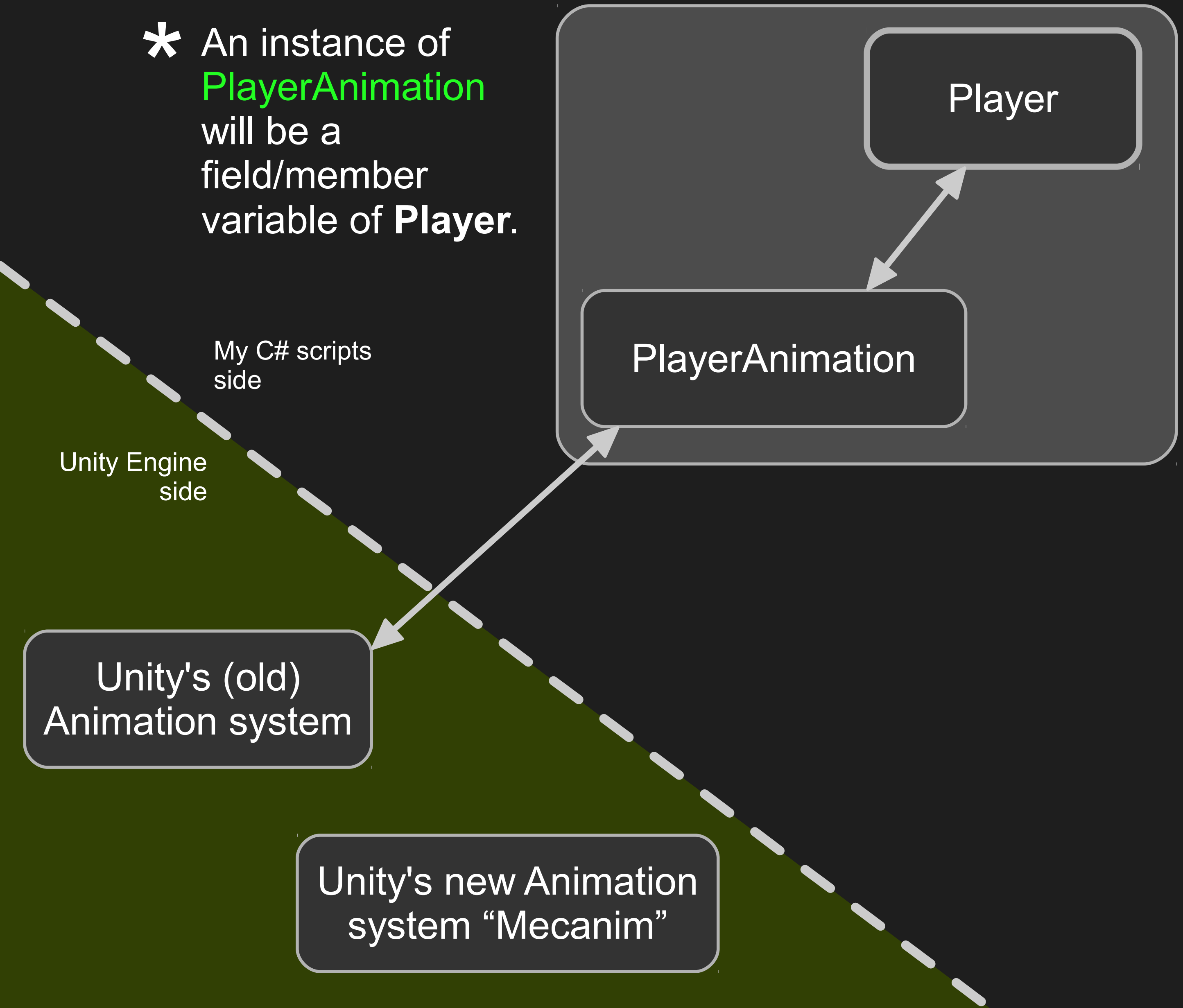
* Only the PlayerAnimation class has code that deals with animations.

For example, in the earlier situation...

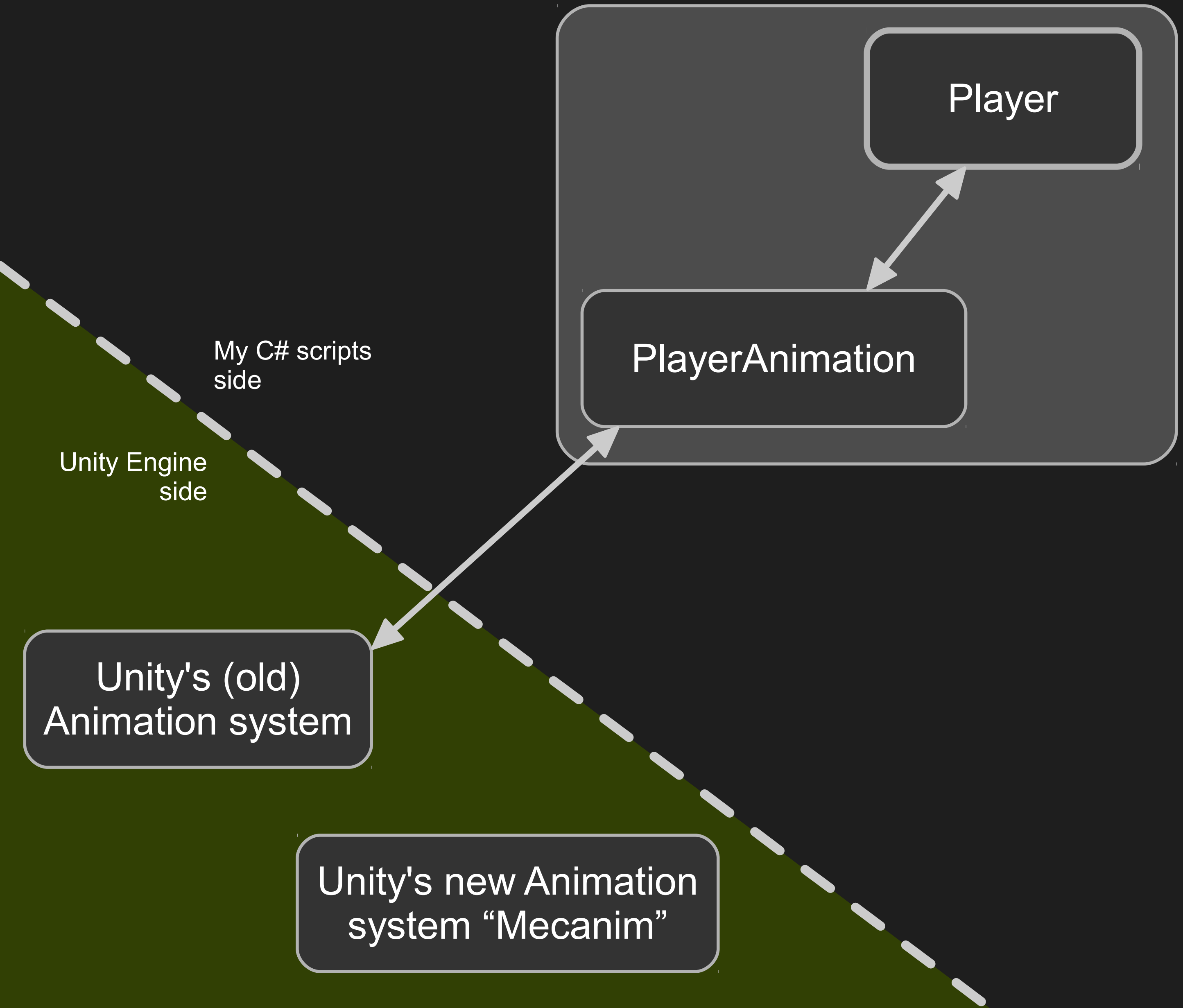

Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system "Mecanim"

For example, in the earlier situation...

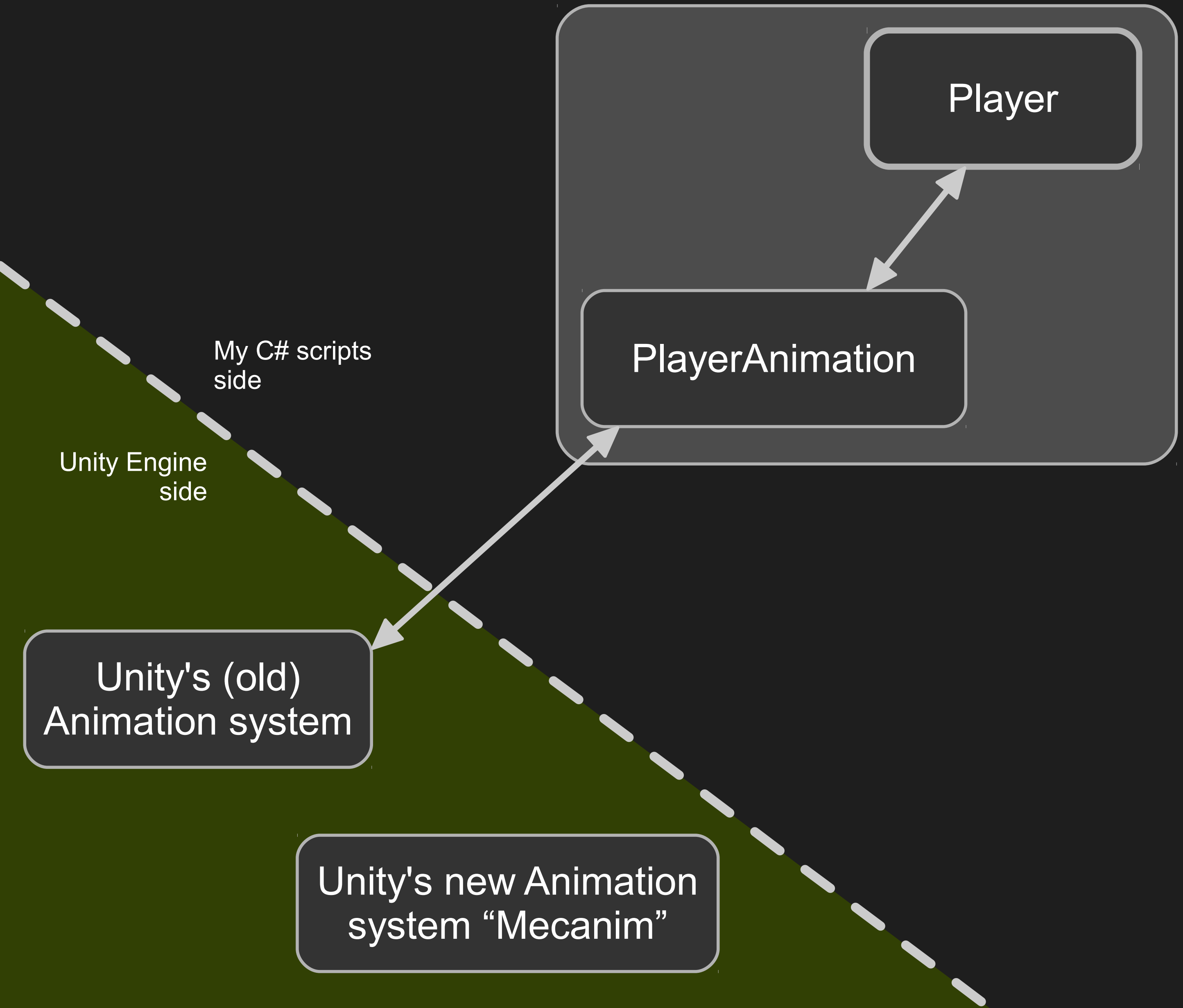
* An instance of PlayerAnimation will be a field/member variable of **Player**.


Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system "Mecanim"

For example, in the earlier situation...


Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system "Mecanim"

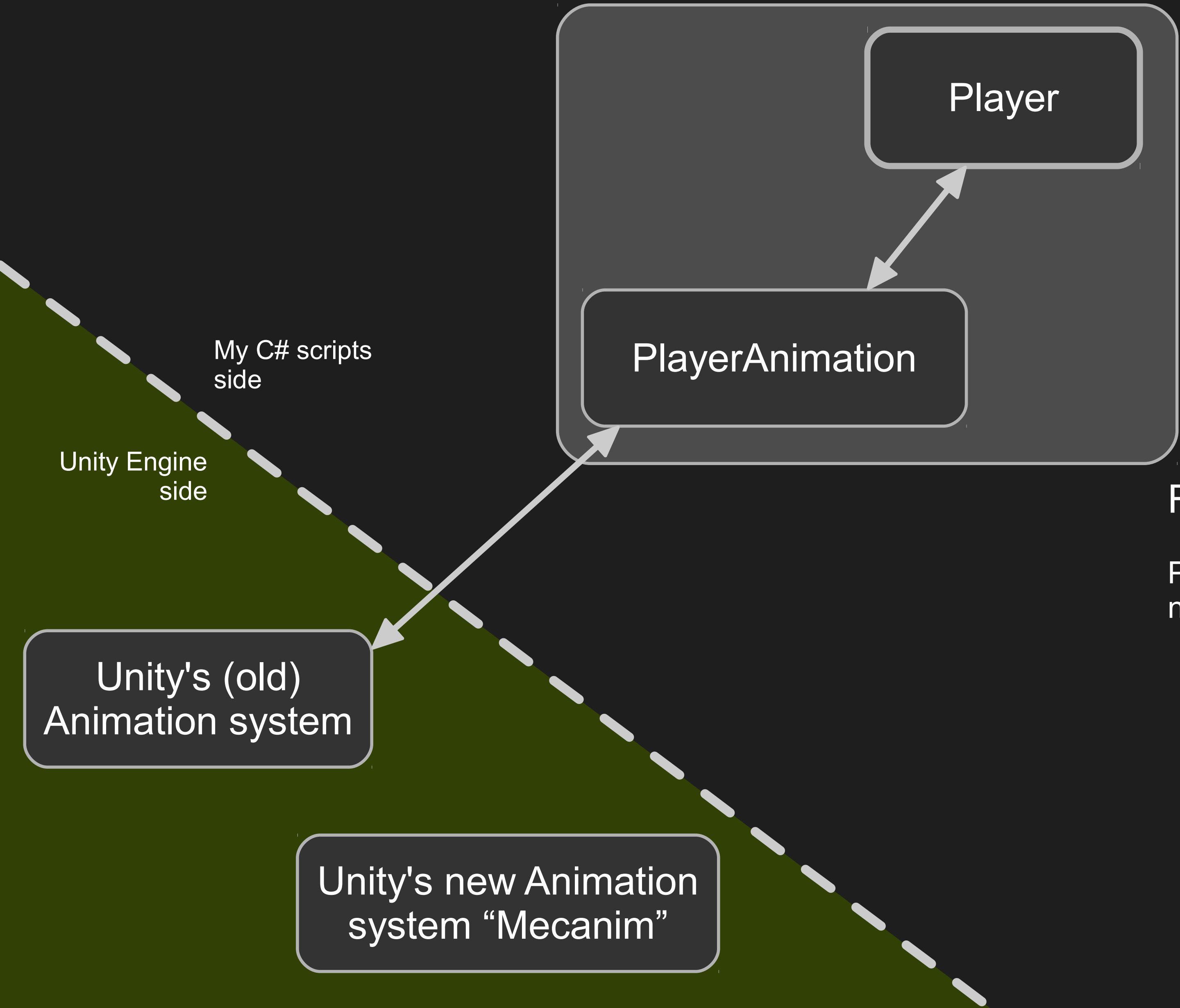
For example, in the earlier situation...


Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

* **Player** class will **not** handle animations directly.

Instead, it will pass requests to the PlayerAnimation class.

For example, in the earlier situation...


Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

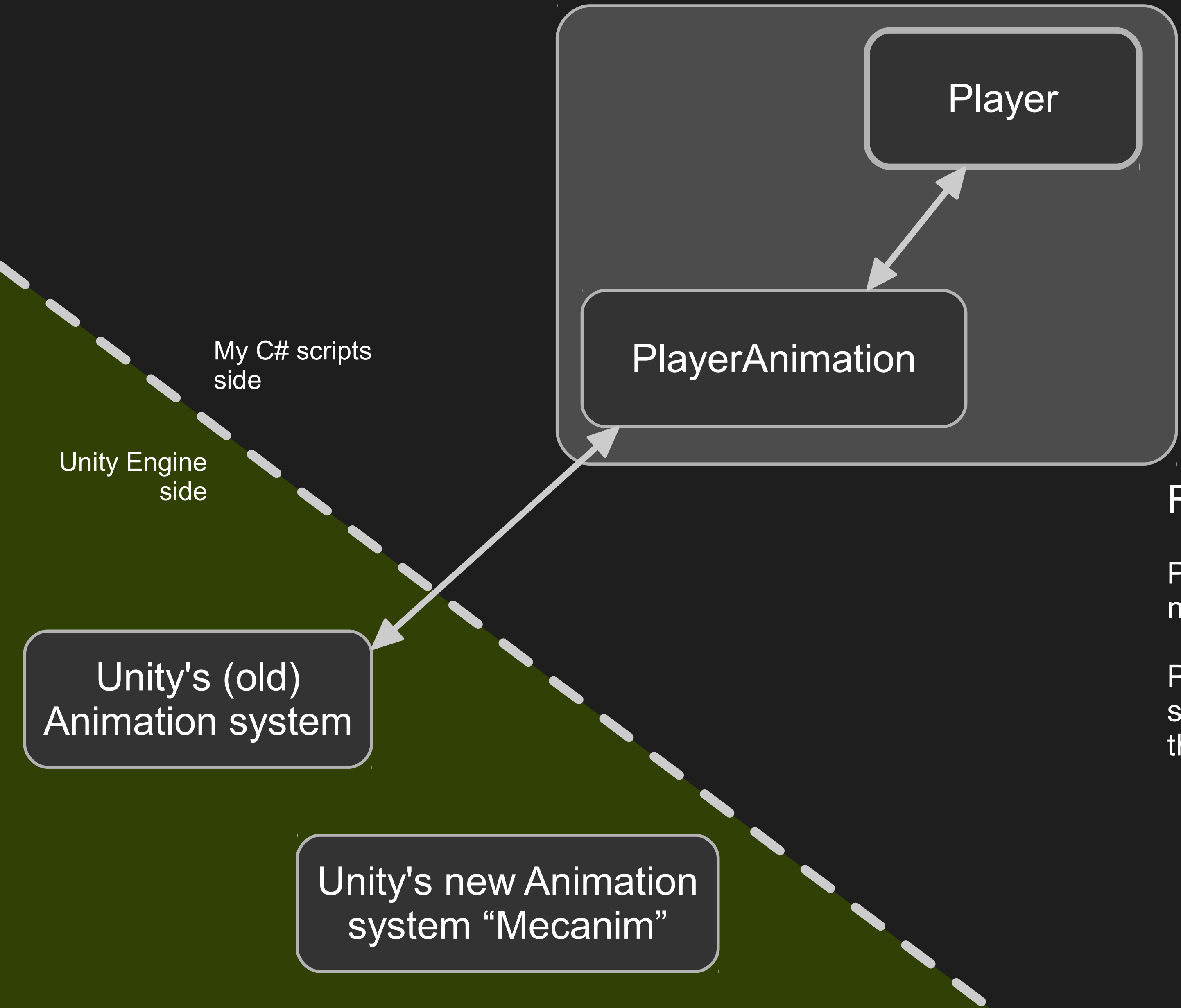
* **Player** class will **not** handle animations directly.

Instead, it will pass requests to the PlayerAnimation class.

Requests such as:

Player: “Hey, PlayerAnimation, you need to make the 3d model walk!”

For example, in the earlier situation...


Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

* **Player** class will **not** handle animations directly.

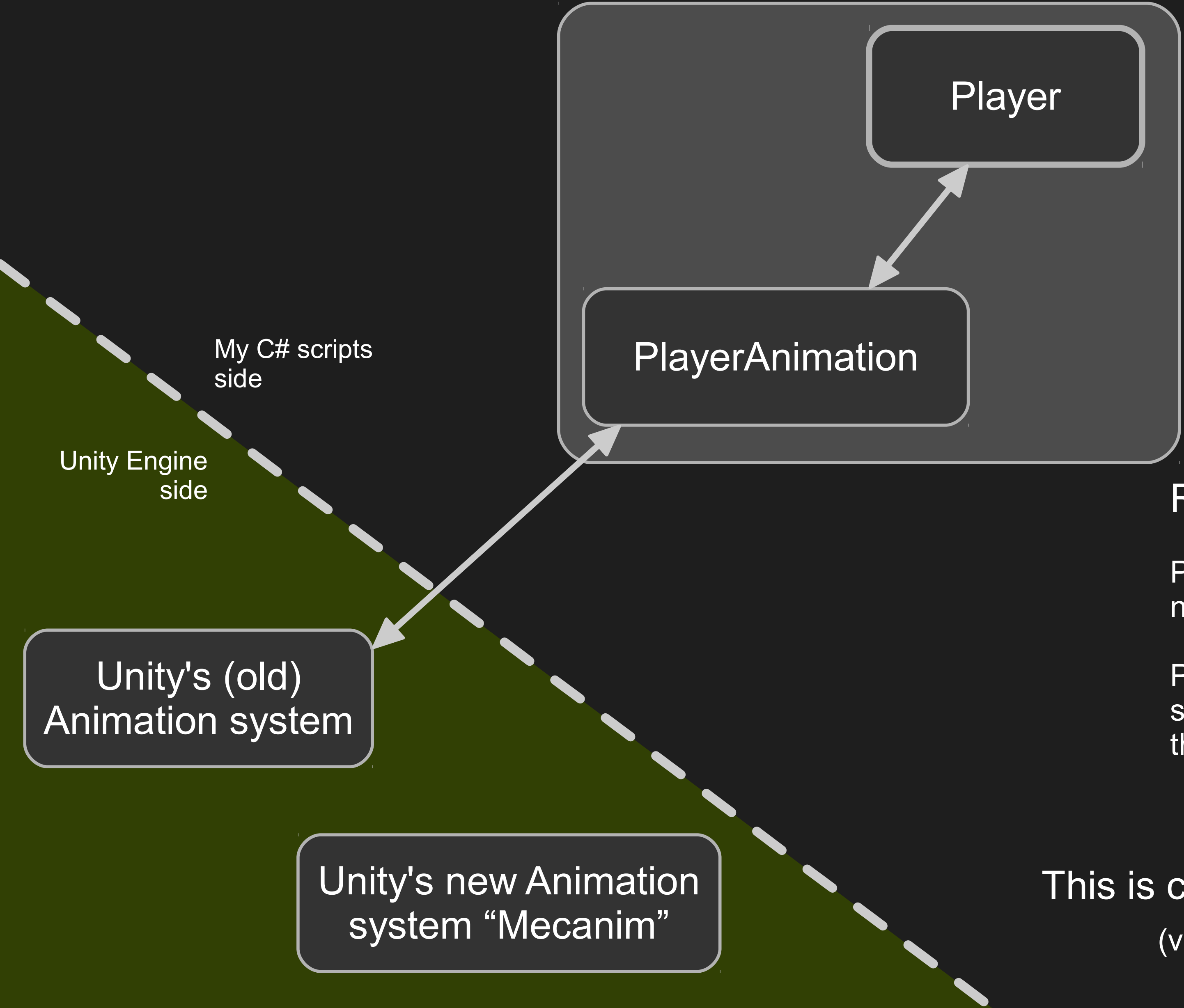
Instead, it will pass requests to the PlayerAnimation class.

Requests such as:

Player: “Hey, PlayerAnimation, you need to make the 3d model walk!”

Player: “I don't need to worry about stuff like which animations to blend, that's your job. Just make him walk!”

For example, in the earlier situation...


Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system "Mecanim"

* **Player** class will **not** handle animations directly.

Instead, it will pass requests to the PlayerAnimation class.

Requests such as:

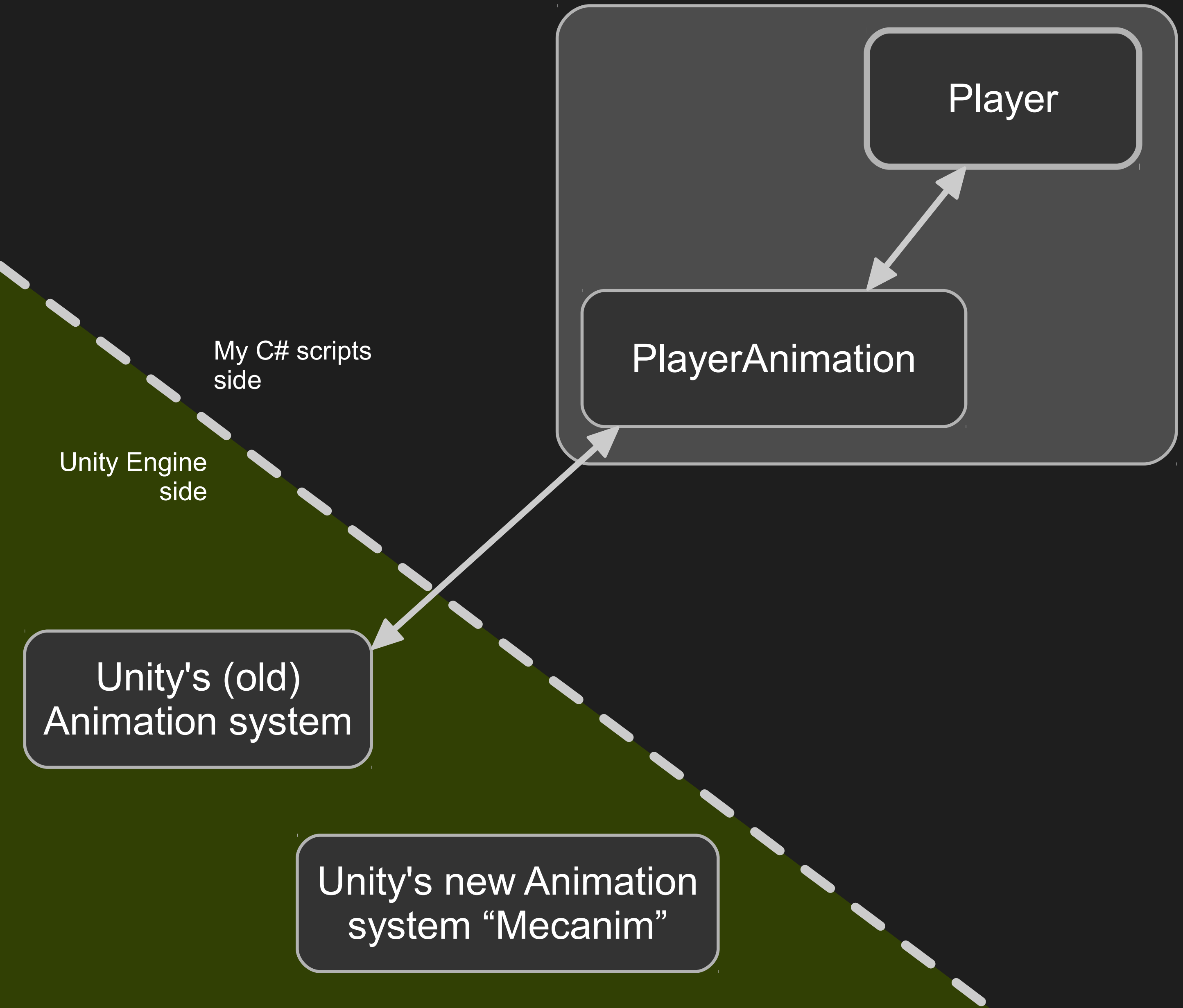
Player: "Hey, PlayerAnimation, you need to make the 3d model walk!"

Player: "I don't need to worry about stuff like which animations to blend, that's your job. Just make him walk!"

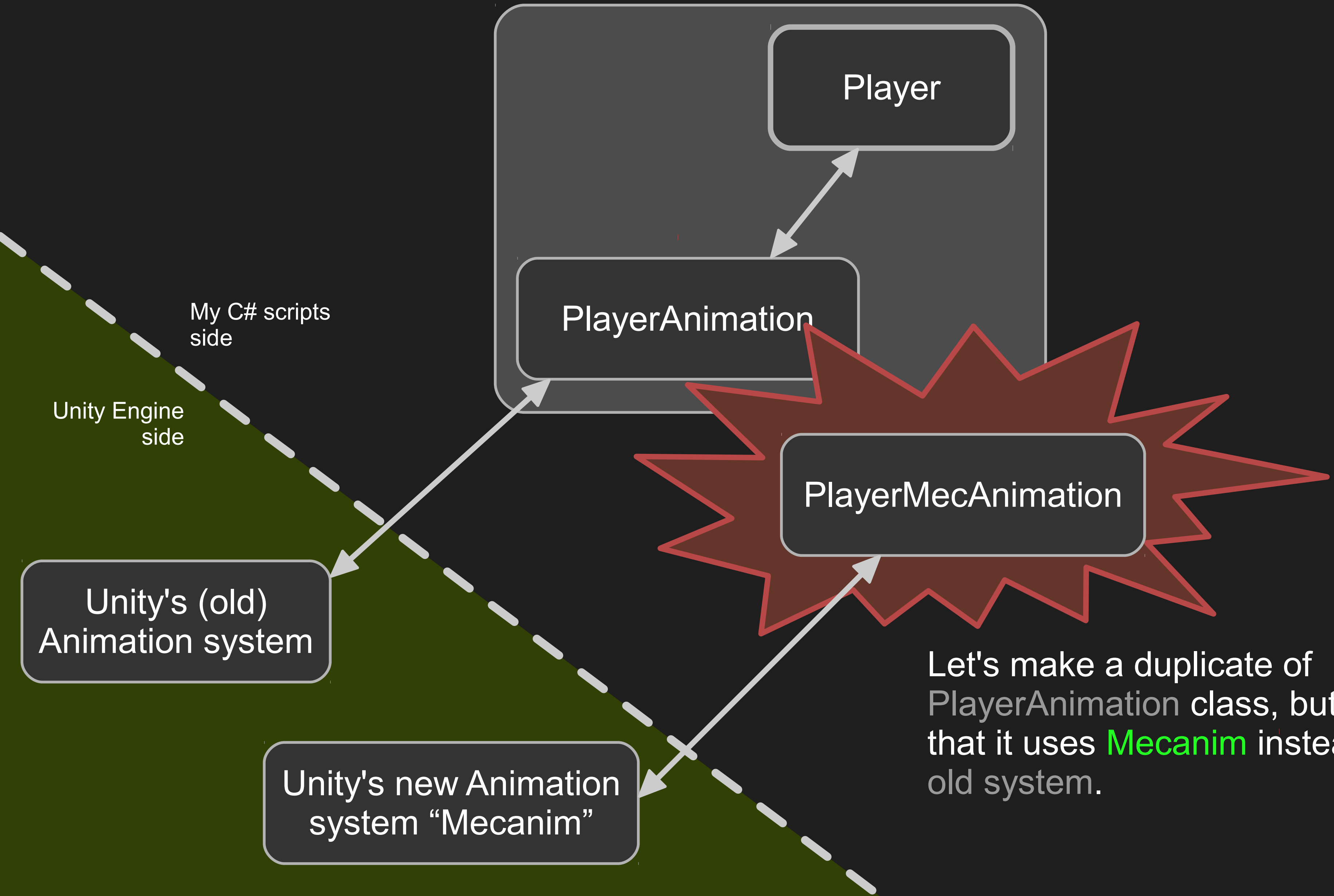
This is called "abstraction".

(very important)

Now, to replace it with Mecanim...

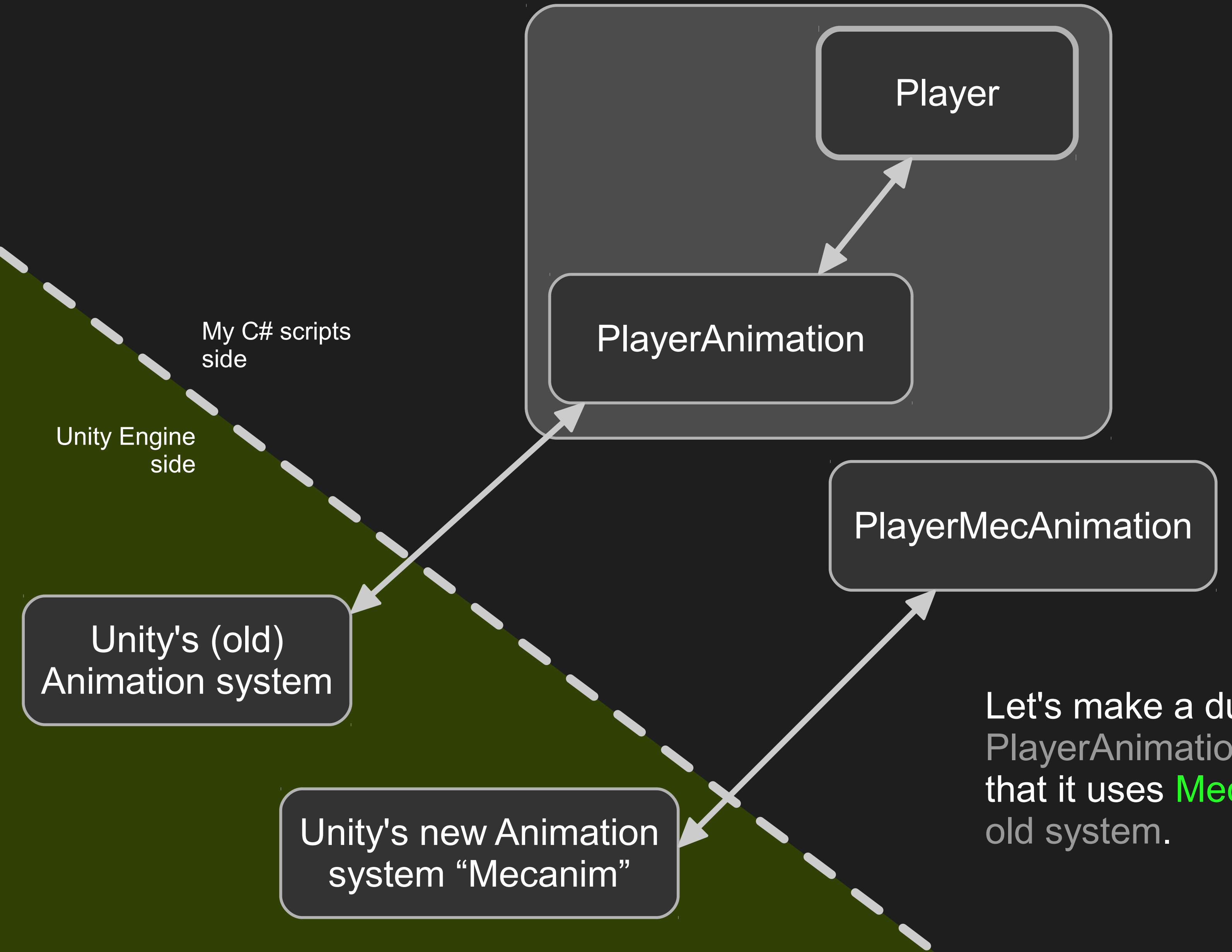

Player
PlayerAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system "Mecanim"

Now, to replace it with Mecanim...



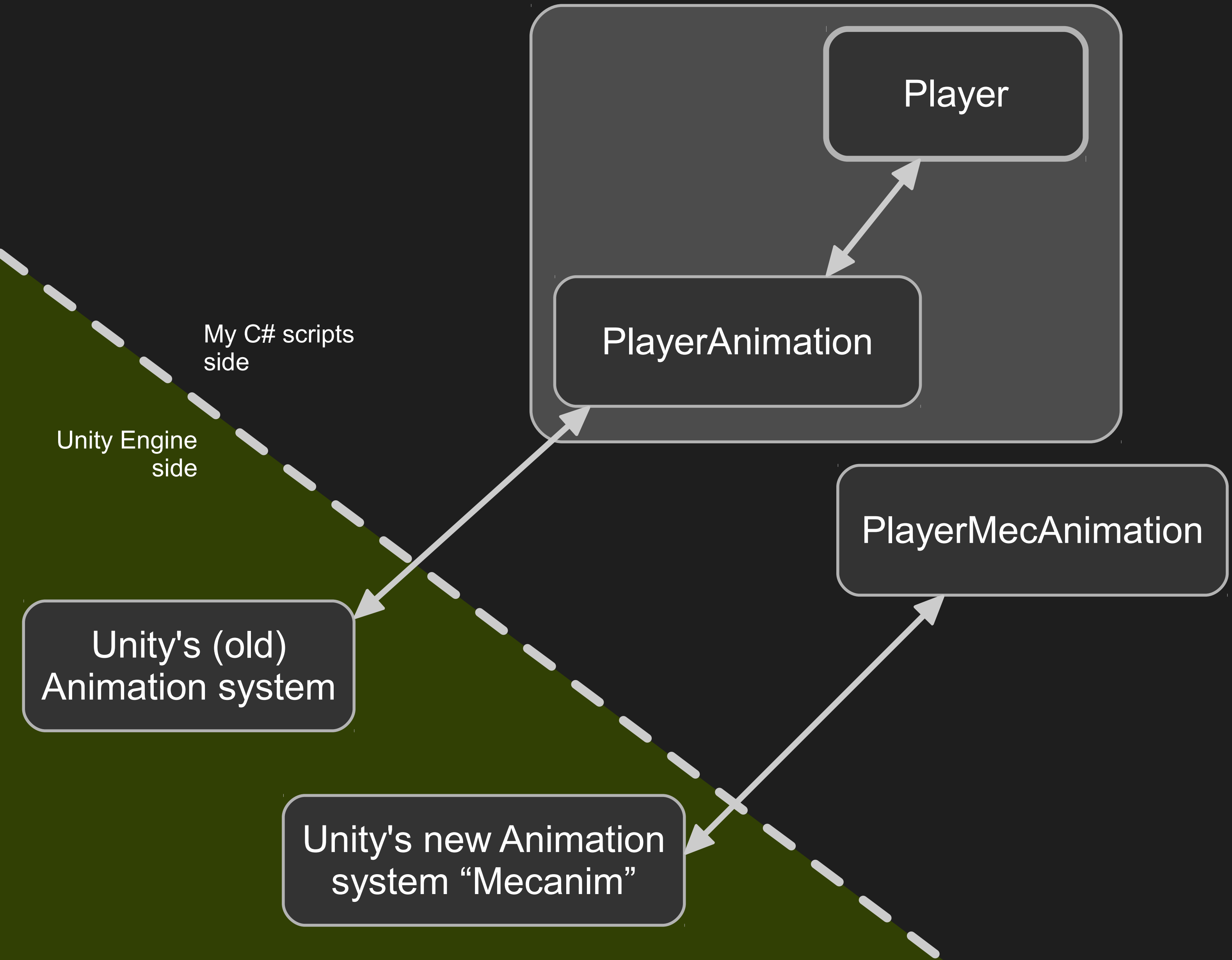
Let's make a duplicate of PlayerAnimation class, but edit it, so that it uses Mecanim instead of the old system.

Now, to replace it with Mecanim...



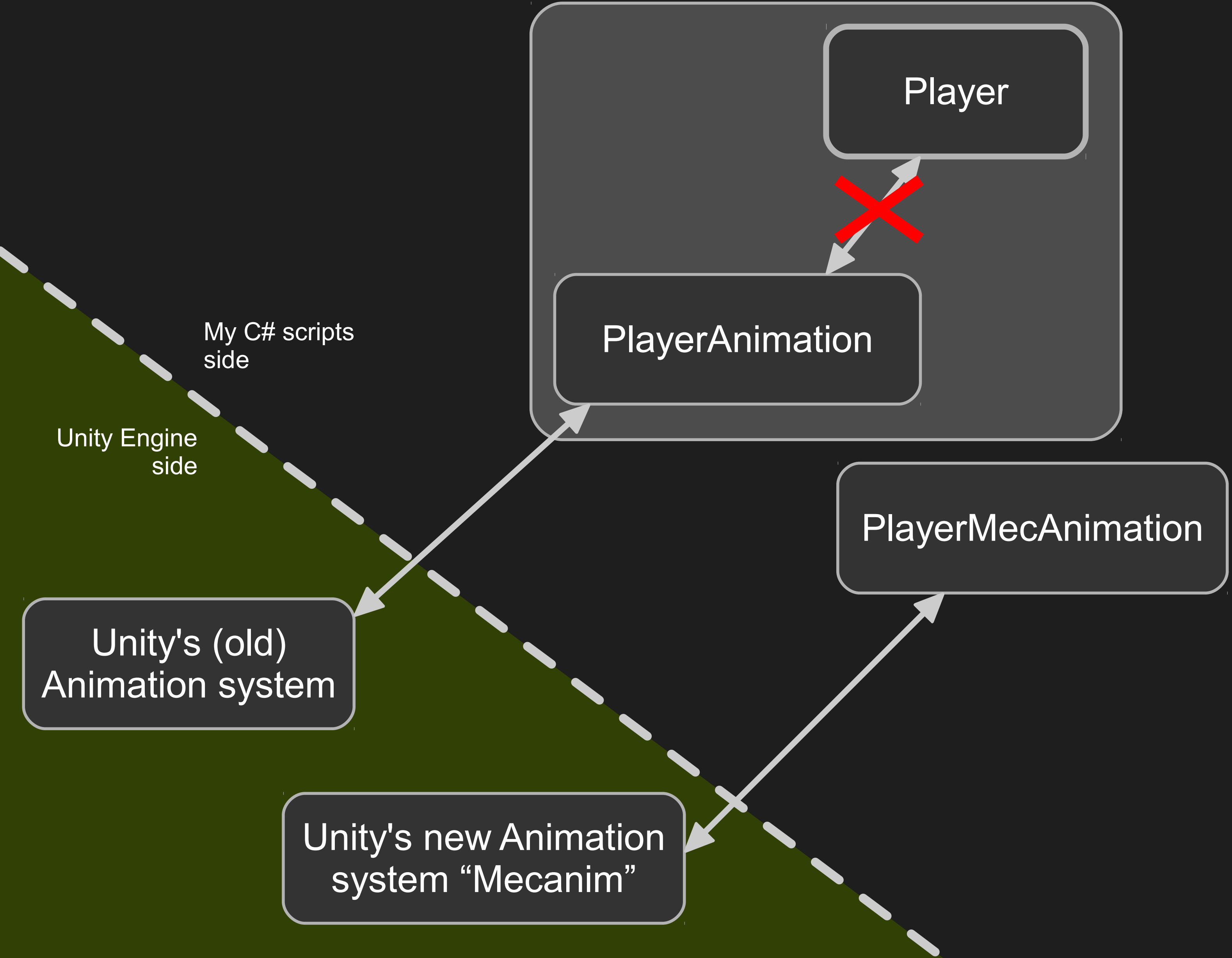
Let's make a duplicate of PlayerAnimation class, but edit it, so that it uses Mecanim instead of the old system.

Now, to replace it with Mecanim...



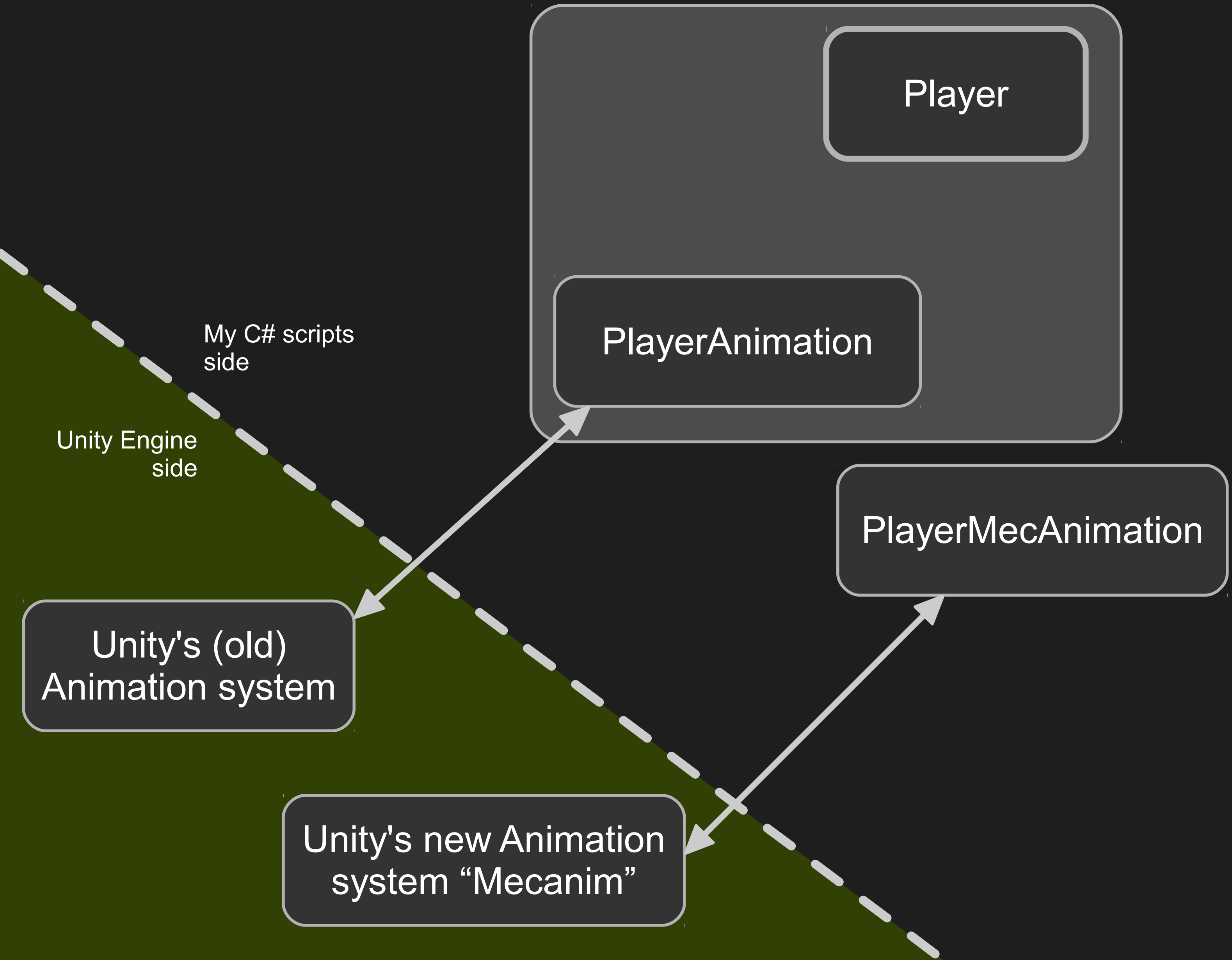
Now, all we need to do is switch from the old PlayerAnimation class to the new PlayerMecAnimation class.

Now, to replace it with Mecanim...



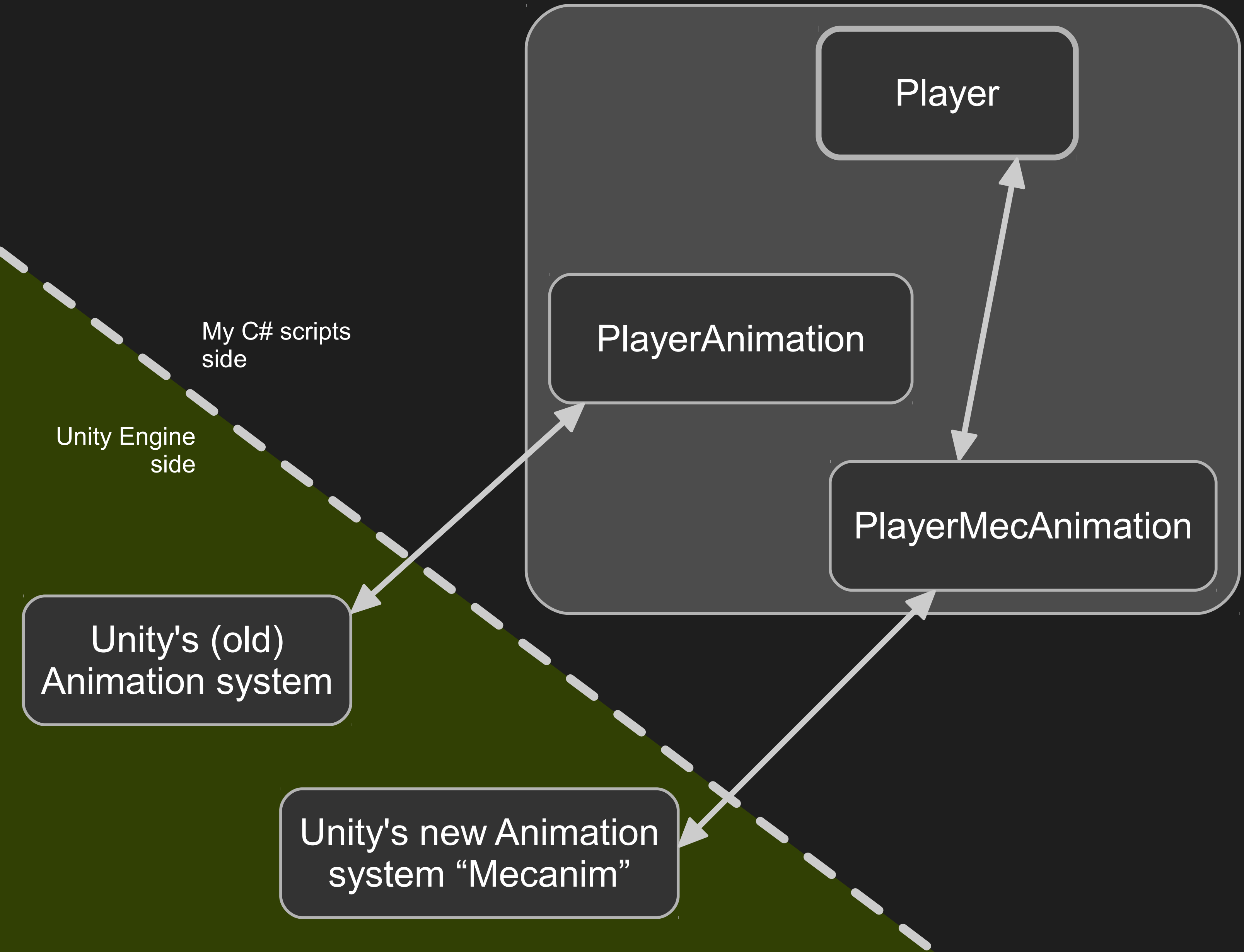
Now, all we need to do is switch from the old PlayerAnimation class to the new PlayerMecAnimation class.

Now, to replace it with Mecanim...



Now, all we need to do is switch from the old PlayerAnimation class to the new PlayerMecAnimation class.

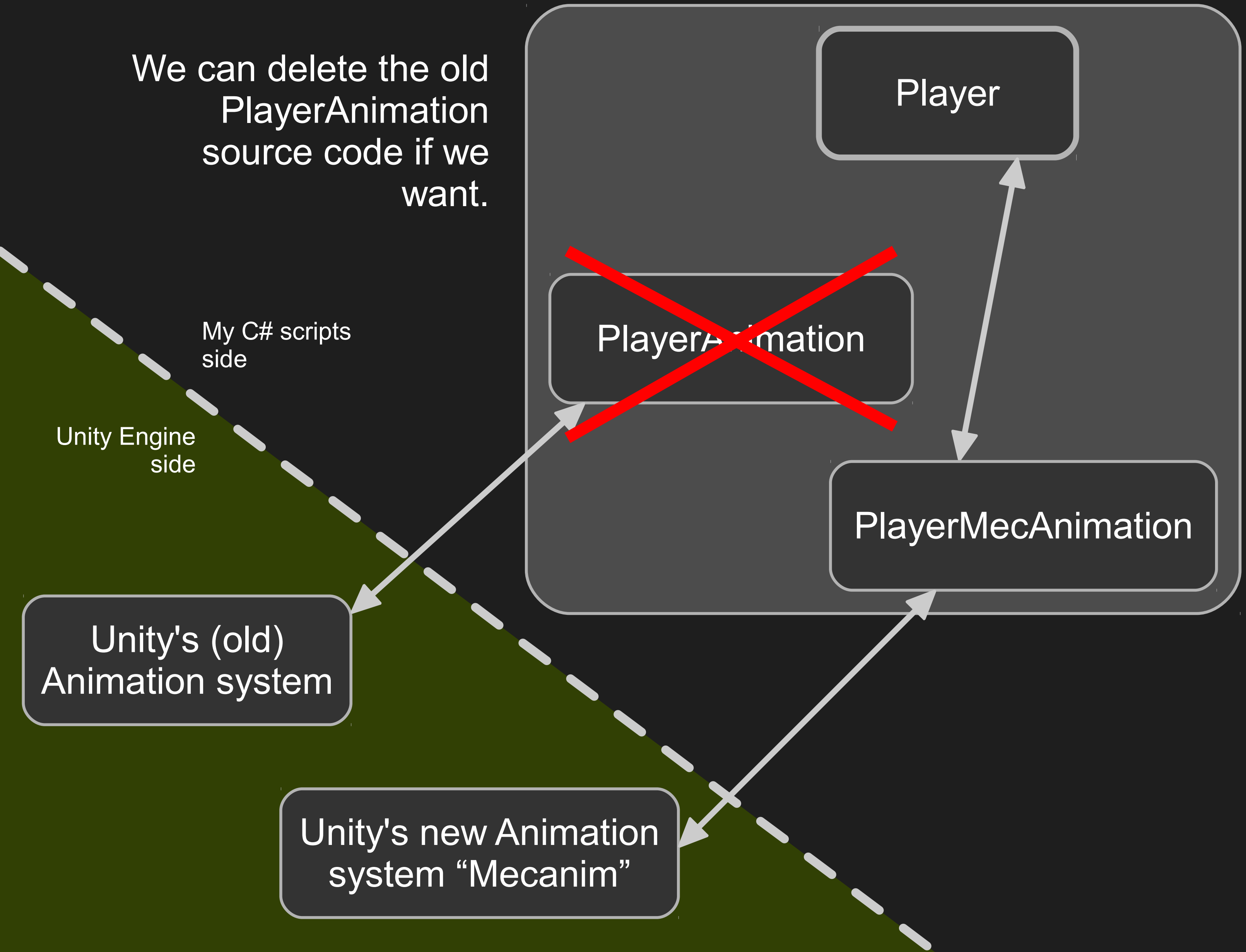
Now, to replace it with Mecanim...



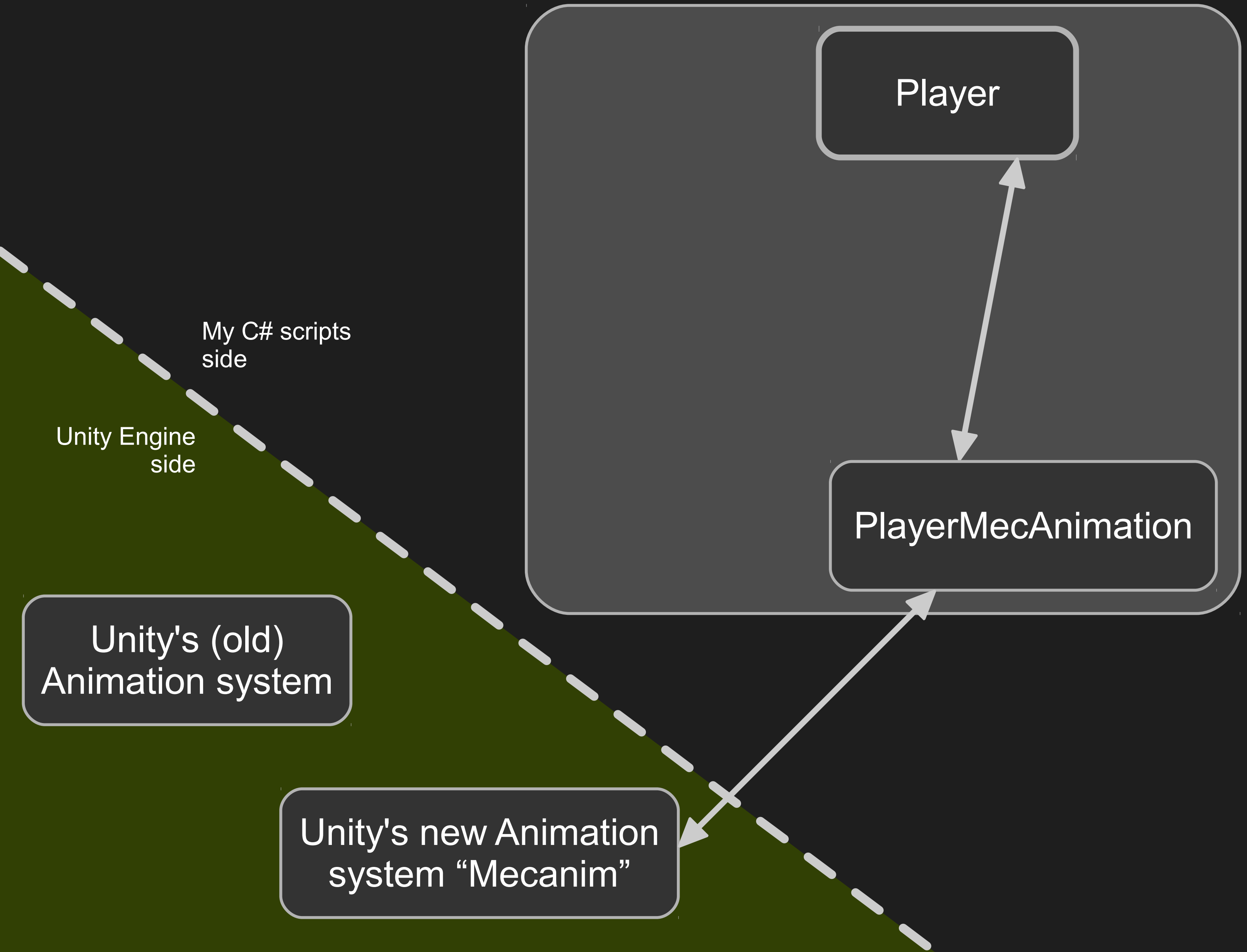
Now, all we need to do is switch from the old PlayerAnimation class to the new PlayerMecAnimation class.

Now, to replace it with Mecanim...

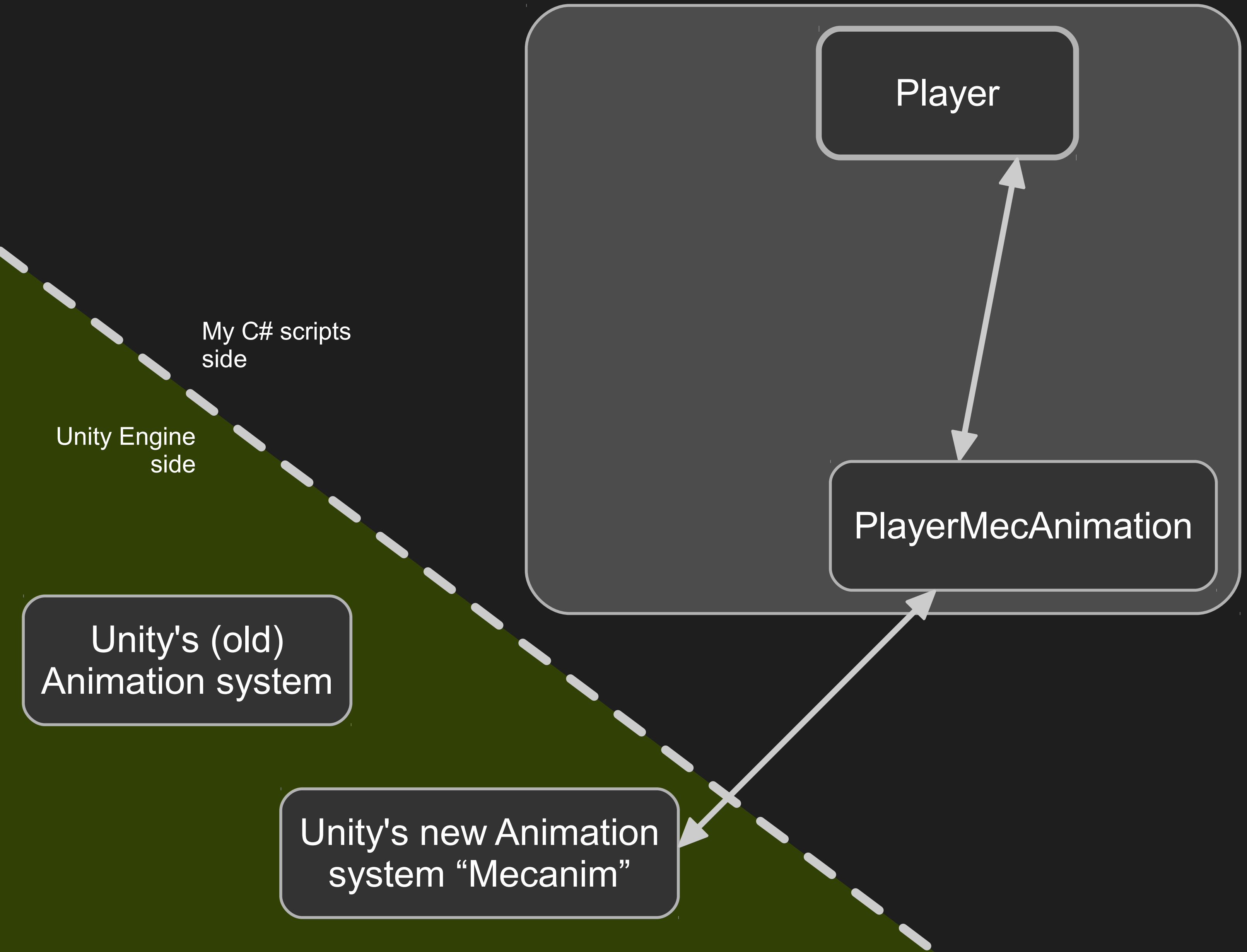
We can delete the old PlayerAnimation source code if we want.



Now, to replace it with Mecanim...

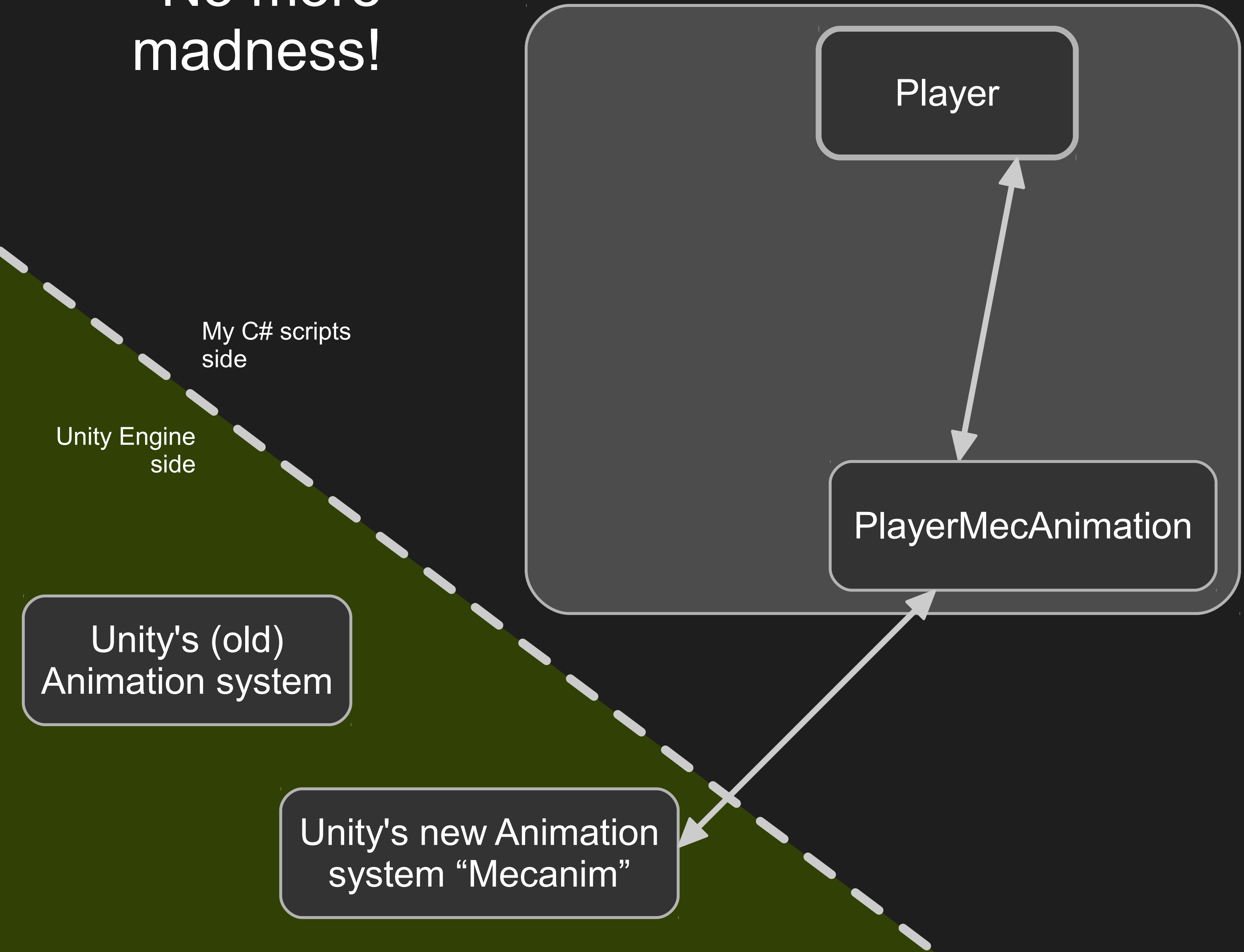

Player
PlayerMecAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system "Mecanim"

We're done!


Player
PlayerMecAnimation
My C# scripts side
Unity Engine side
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

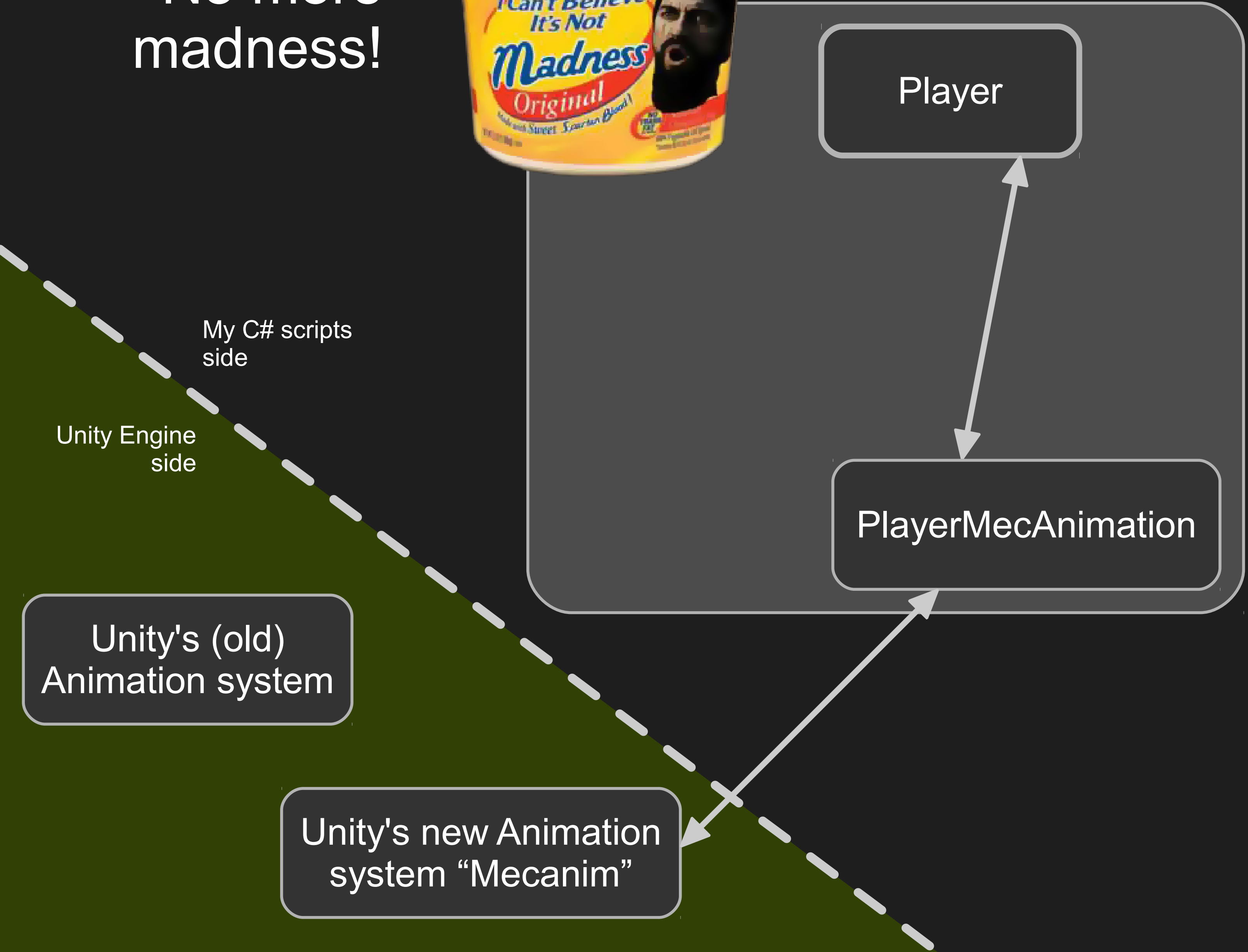
We're done!

No more madness!


My C# scripts side
Unity Engine side
Player
PlayerMecAnimation
Unity's (old) Animation system
Unity's new Animation system “Mecanim”

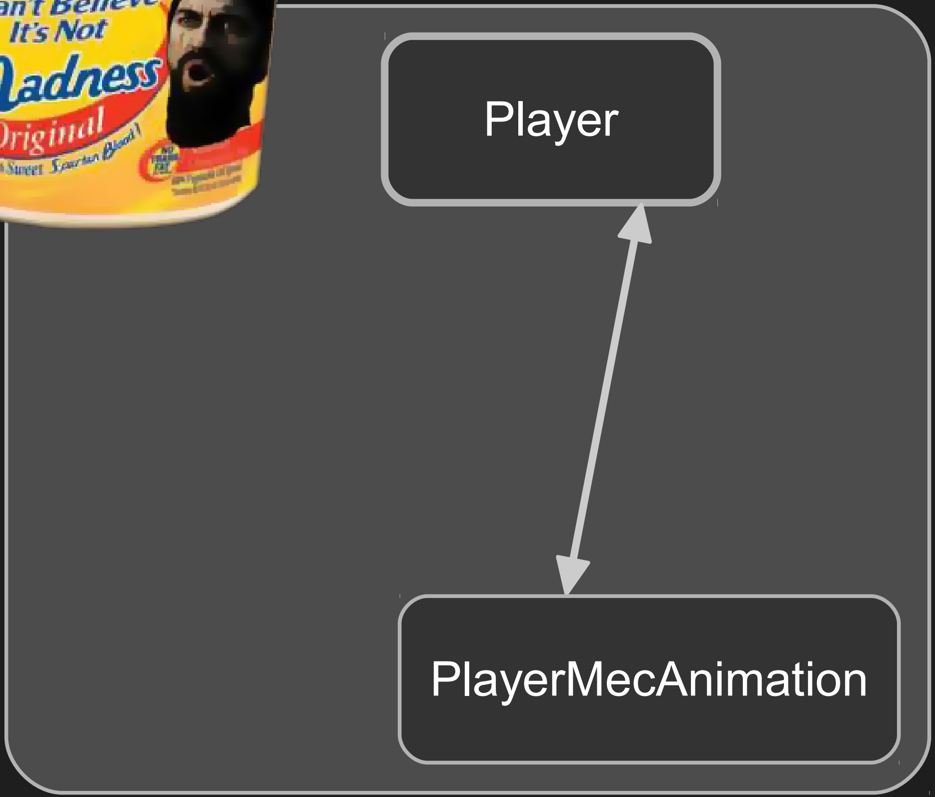
We're done!

No more madness!


My C# scripts side
Unity Engine side
Player
PlayerMecAnimation
Unity's (old) Animation system
Unity's new Animation system "Mecanim"

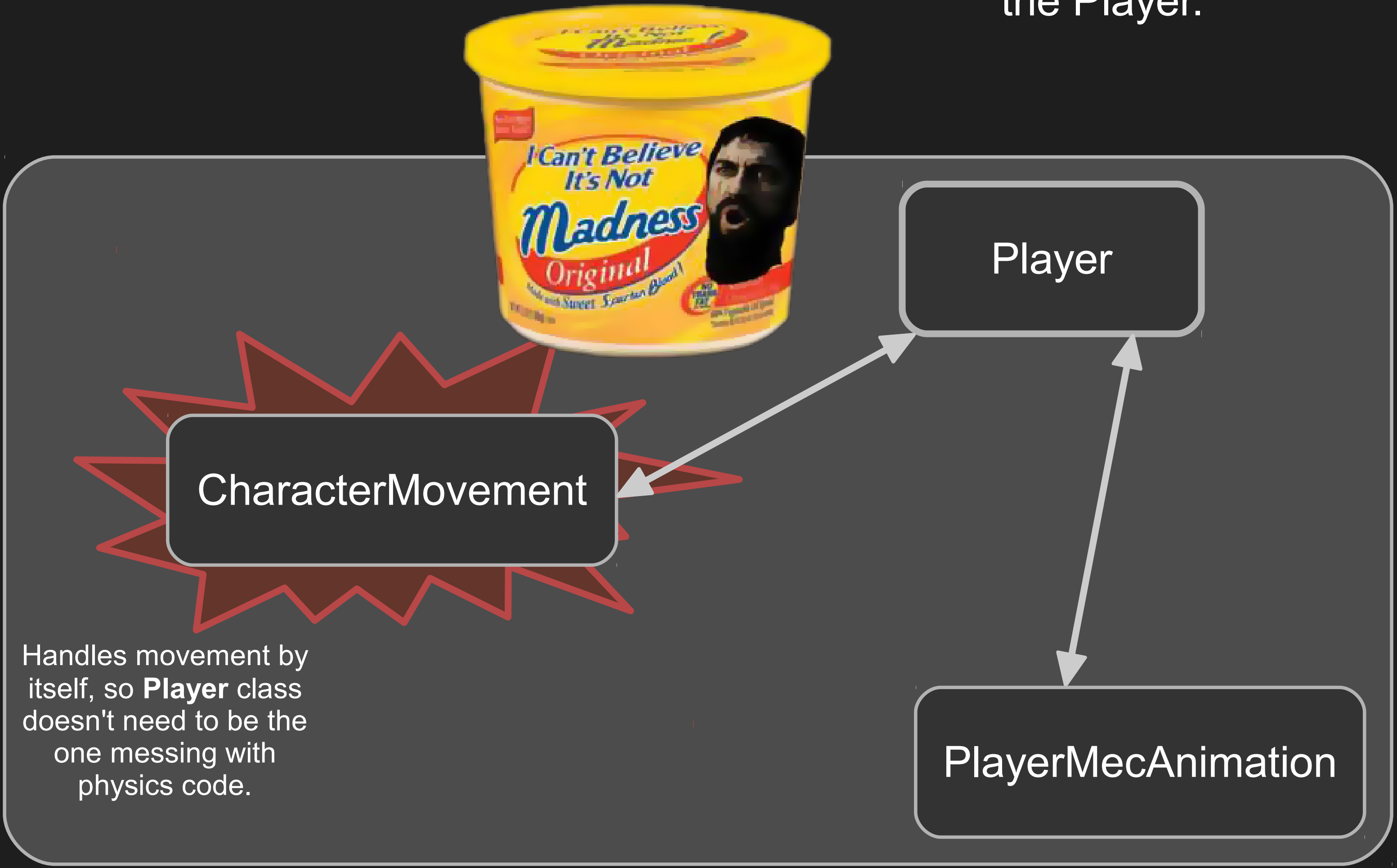
We're done!

In fact, it makes sense to create a separate class for every major part of the Player.



We're done!

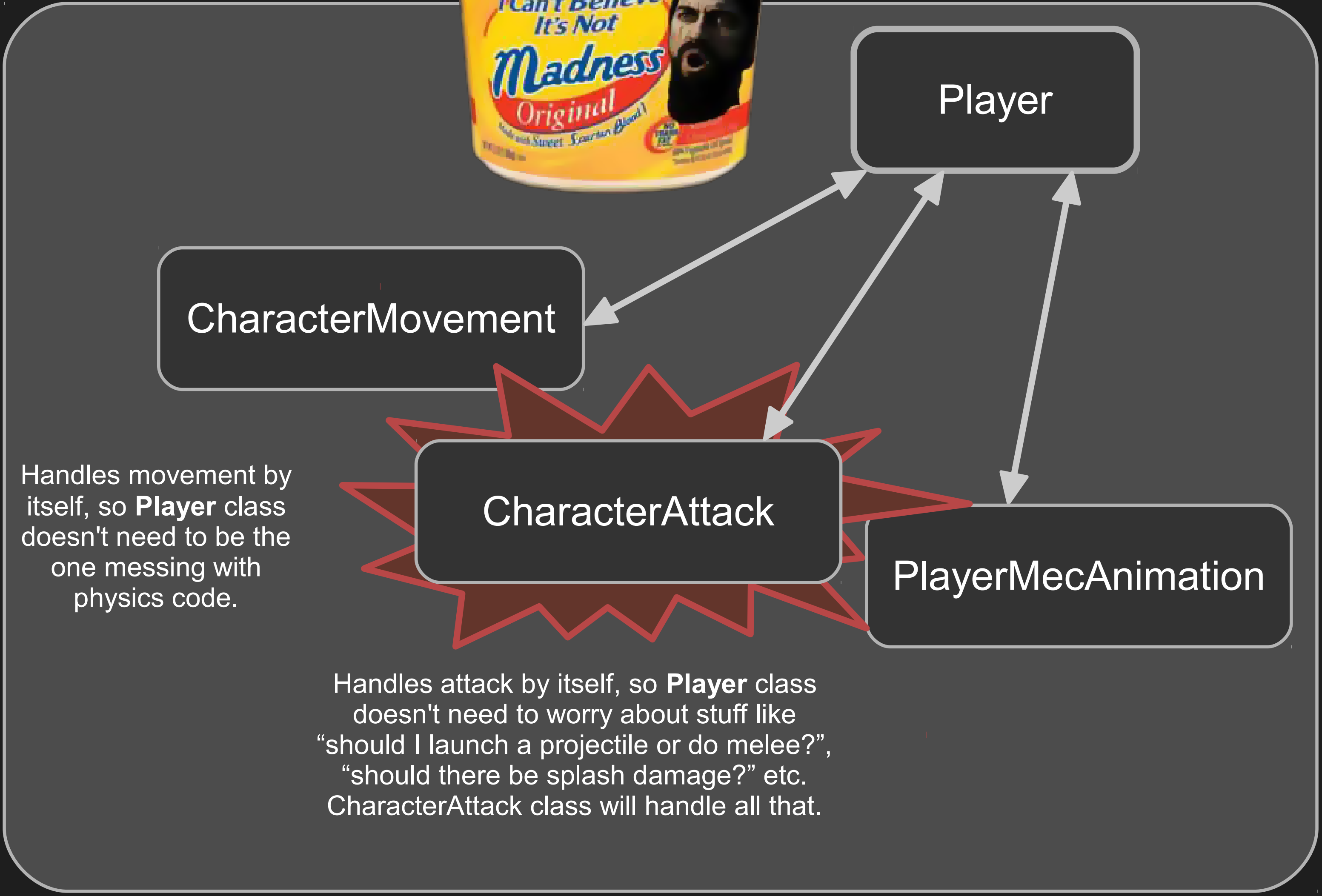
In fact, it makes sense to create a separate class for every major part of the Player.



Handles movement by itself, so **Player** class doesn't need to be the one messing with physics code.

We're done!

In fact, it makes sense to create a separate class for every major part of the Player.

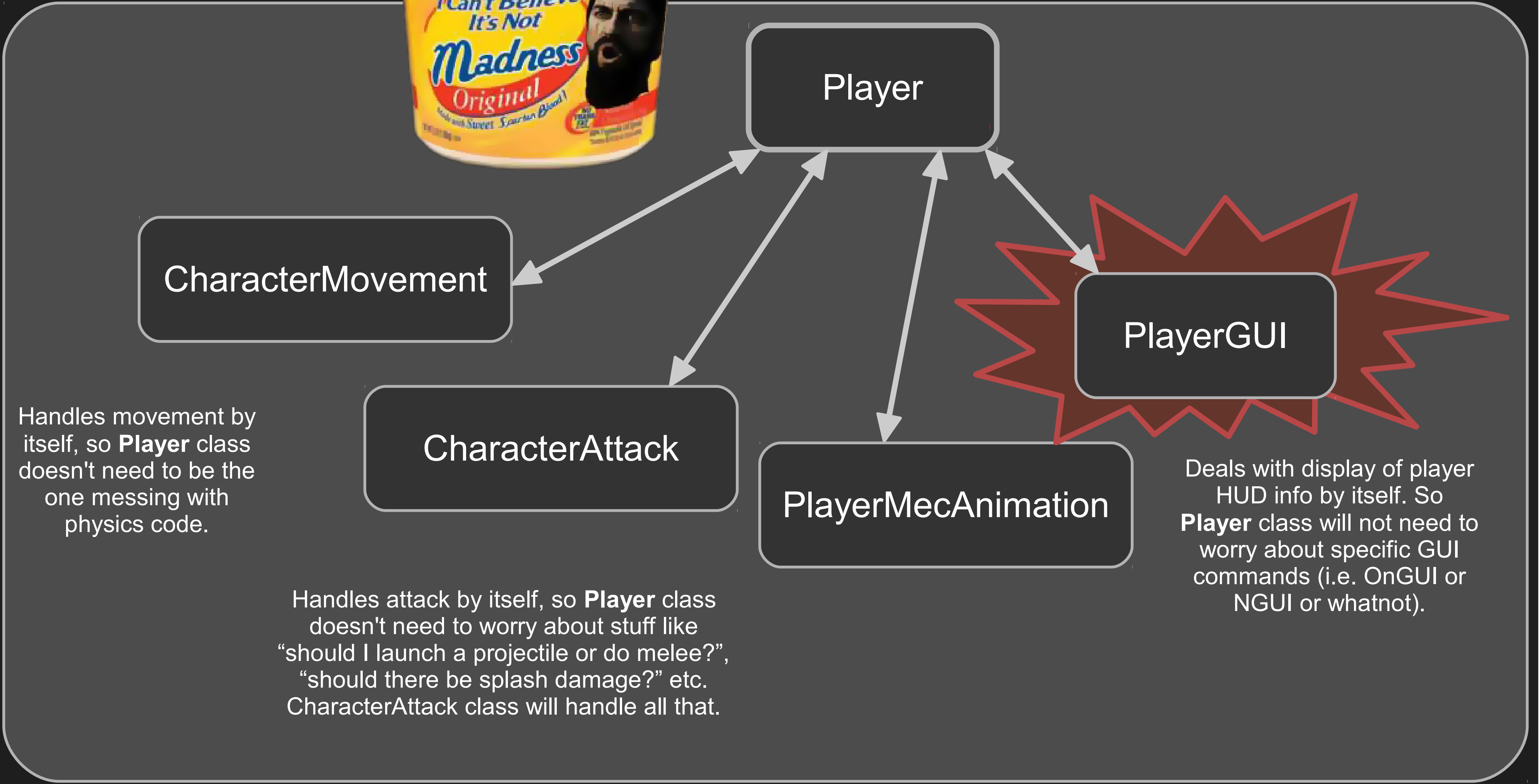


Handles movement by itself, so **Player** class doesn't need to be the one messing with physics code.

Handles attack by itself, so **Player** class doesn't need to worry about stuff like “should I launch a projectile or do melee?”, “should there be splash damage?” etc. CharacterAttack class will handle all that.

We're done!

In fact, it makes sense to create a separate class for every major part of the Player.



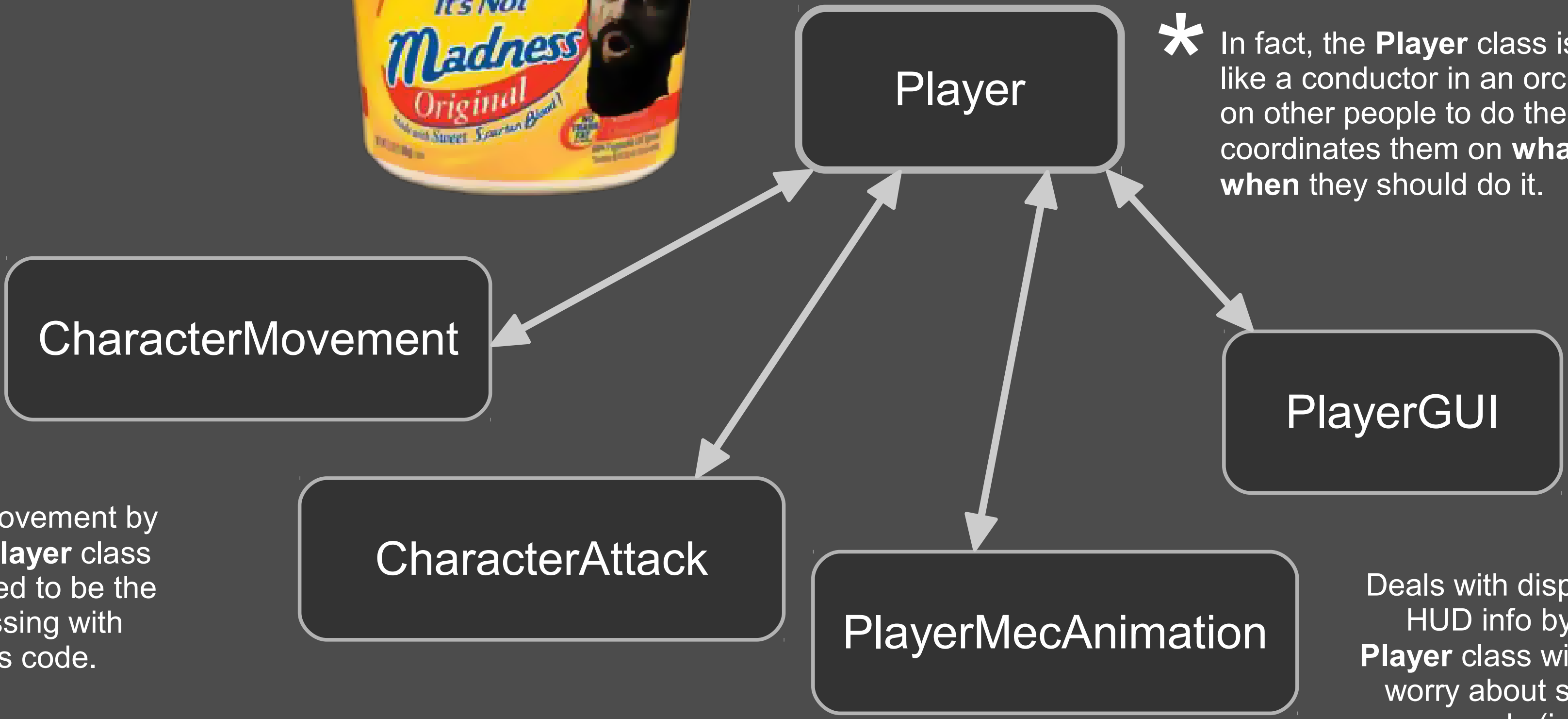
Handles movement by itself, so **Player** class doesn't need to be the one messing with physics code.

Handles attack by itself, so **Player** class doesn't need to worry about stuff like “should I launch a projectile or do melee?”, “should there be splash damage?” etc. CharacterAttack class will handle all that.

Deals with display of player HUD info by itself. So **Player** class will not need to worry about specific GUI commands (i.e. OnGUI or NGUI or whatnot).

We're done!

In fact, it makes sense to create a separate class for every major part of the Player.



* In fact, the **Player** class is now more like a conductor in an orchestra, relying on other people to do the work, but he coordinates them on **what** to do and **when** they should do it.

Handles movement by itself, so **Player** class doesn't need to be the one messing with physics code.

Handles attack by itself, so **Player** class doesn't need to worry about stuff like "should I launch a projectile or do melee?", "should there be splash damage?" etc. CharacterAttack class will handle all that.

Deals with display of player HUD info by itself. So **Player** class will not need to worry about specific GUI commands (i.e. OnGUI or NGUI or whatnot).

Disclaimer!

Disclaimer!

That “technique” I presented is not a silver bullet.

Disclaimer!

Note: we call them Design Patterns

That “technique” I presented is not a silver bullet.

Disclaimer!

Note: we call them Design Patterns

That "technique" I presented is not a silver bullet.

It worked for me, but that doesn't mean it will work for every problem I (or you) will encounter.

Disclaimer!

Note: we call them Design Patterns

That “technique” I presented is not a silver bullet.

It worked for me, but that doesn't mean it will work for every problem I (or you) will encounter.

There are many “techniques”.

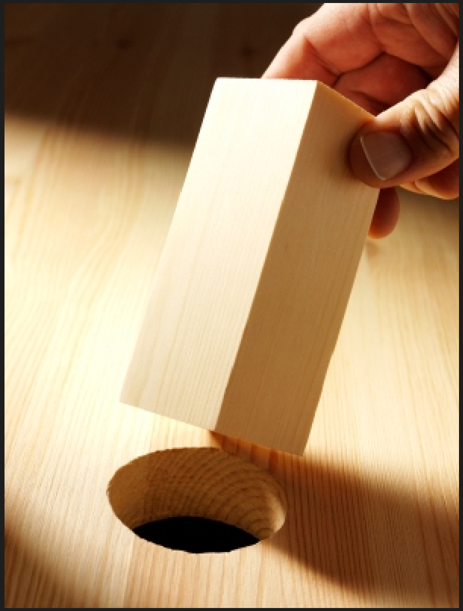
Disclaimer!

Note: we call them Design Patterns

That “technique” I presented is not a silver bullet.

It worked for me, but that doesn't mean it will work for every problem I (or you) will encounter.

There are many “techniques”. So choose the right tool for the right job.



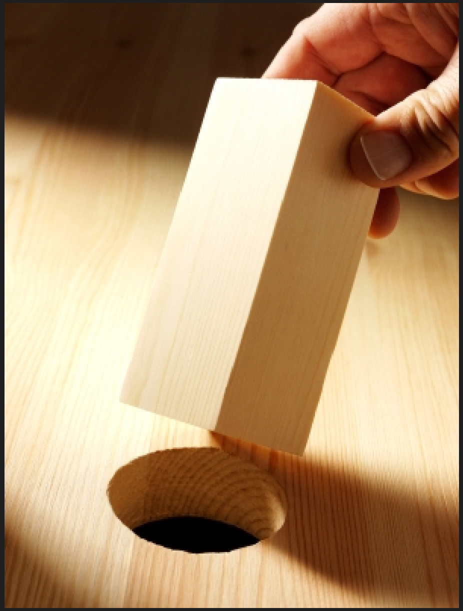
Disclaimer!

Note: we call them Design Patterns

That “technique” I presented is not a silver bullet.

It worked for me, but that doesn't mean it will work for every problem I (or you) will encounter.

There are many “techniques”. So choose the right tool for the right job.


That's not gonna fit, bro

So want to learn those other “techniques”?

So want to learn those other “techniques”?

Want to be a good game programmer?

So want to learn those other “techniques”?

Want to be a good game programmer?

Learn Software Architecture!

So want to learn those other "techniques"?

Want to be a good game programmer?

Learn Software Architecture!

"Software architecture should be a basic skill every game programmer must have."

So want to learn those other “techniques”?

Want to be a good game programmer?

Learn Software Architecture!

“Software architecture should be a basic skill every game programmer must have.”

-Me

So want to learn those other “techniques”?

Want to be a good game programmer?

Learn Software Architecture!

“Software architecture should be a basic skill every game programmer must have.”

-Me

(You can very damn well quote me on that)

A few books to start with...

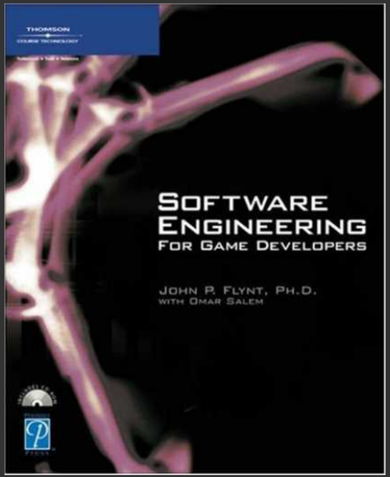
Take note:

> You better have a fairly good grasp of your programming language of choice (an object-oriented one) before you start learning this.

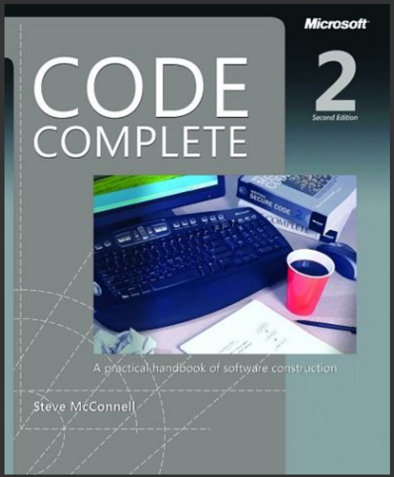
(click on the book cover for a link where to buy it)

A Brain-Friendly Guide
Head First Design Patterns
Eric Freeman & Elisabeth Freeman with Kathy Sierra & Bert Bates
O'REILLY

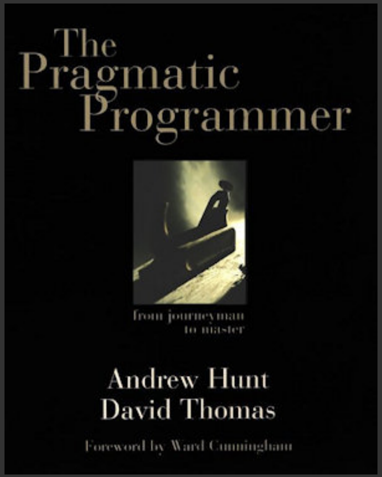
Head First Design Patterns


Software Engineering for Game Developers
John P. Flynt, Ph.D. with Omar Salem

Software Engineering for Game Developers



Code Complete



The Pragmatic Programmer

Game Coding Complete

Question me!

Ferdinand Joseph Fernandez

Chief Technological Officer, Dreamlords Digital, Inc.
Admin & Co-founder, Unity Philippines Users Group

@AnomalusUndrdog
victisgame.wordpress.com
anomalousunderdog.blogspot.com