Multiverse Survival

GROUP 8 PRE

Main parts

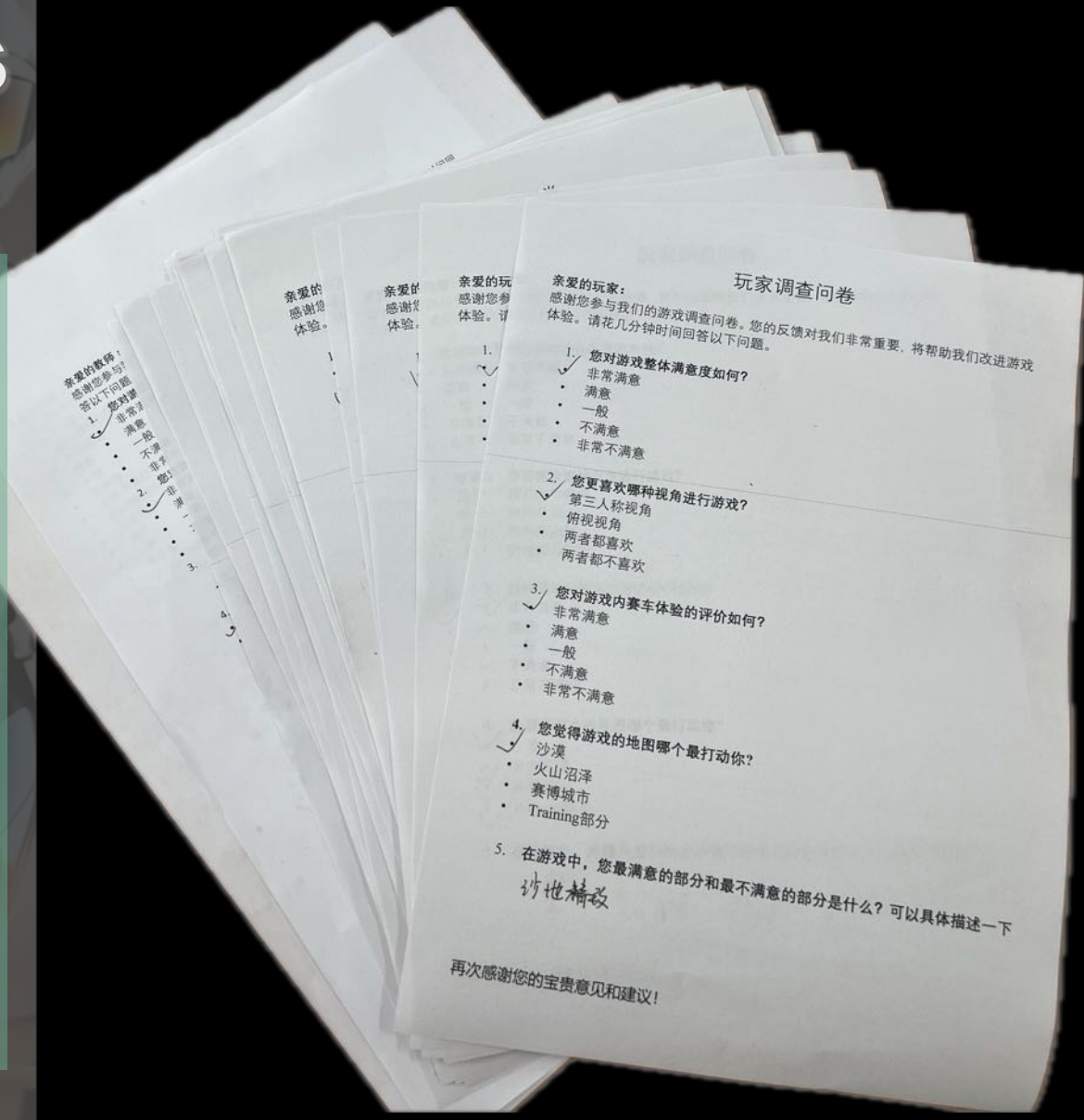
1. Evaluations and Introduction – 3min
2. Techniques in the characters – 3min
3. Weapons and Map - 1.5min
4. Enemies(Melee + range + boss) - 4min
5. Car and steering wheel – 3min
6. Online network – 3.5min
7. AI – 1min

Evaluation

- People love our game regardless of age
- Our game booths are one of the most popular ones

Evaluation - Results

- Hundreds of promotional cards
- Extremely satisfied first prize customers
- Thirty sets of prizes, hundreds of postcards
- Thirty-eight offline survey questionnaires were received
- Satisfying gaming experience and communication atmosphere



Evaluation

What have we done to become so welcomed?

Introduction – Game mechanism

Camera change between third-person view and top-down view

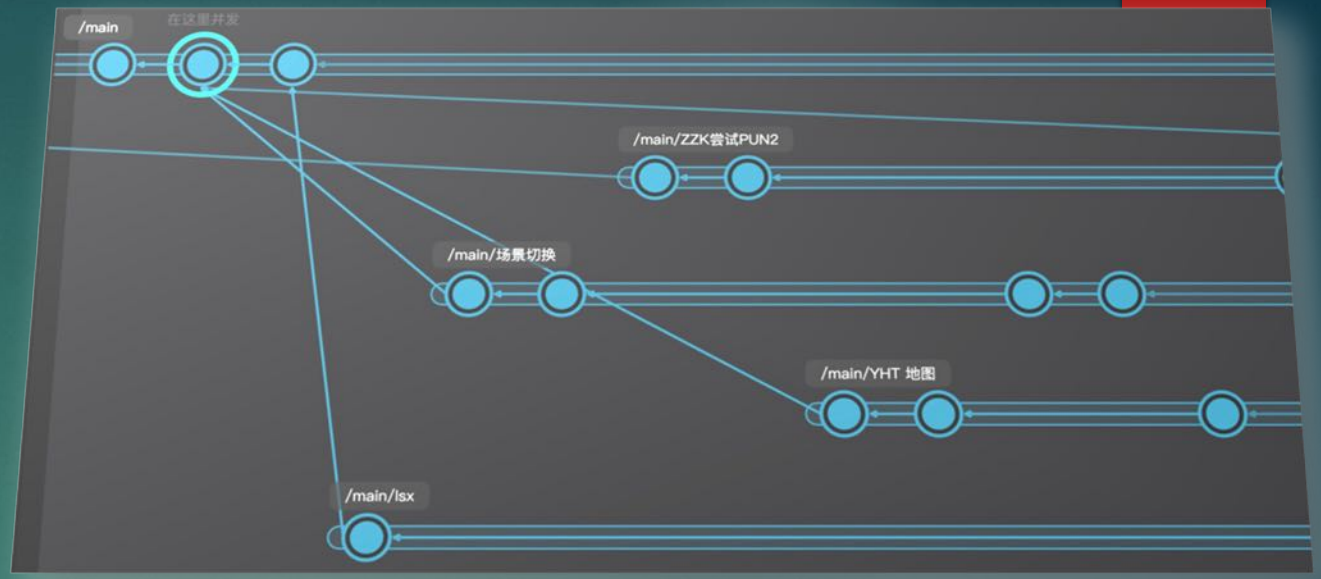
Full use of unity physical engine collider and rigid body

Introduction - Game mechanism

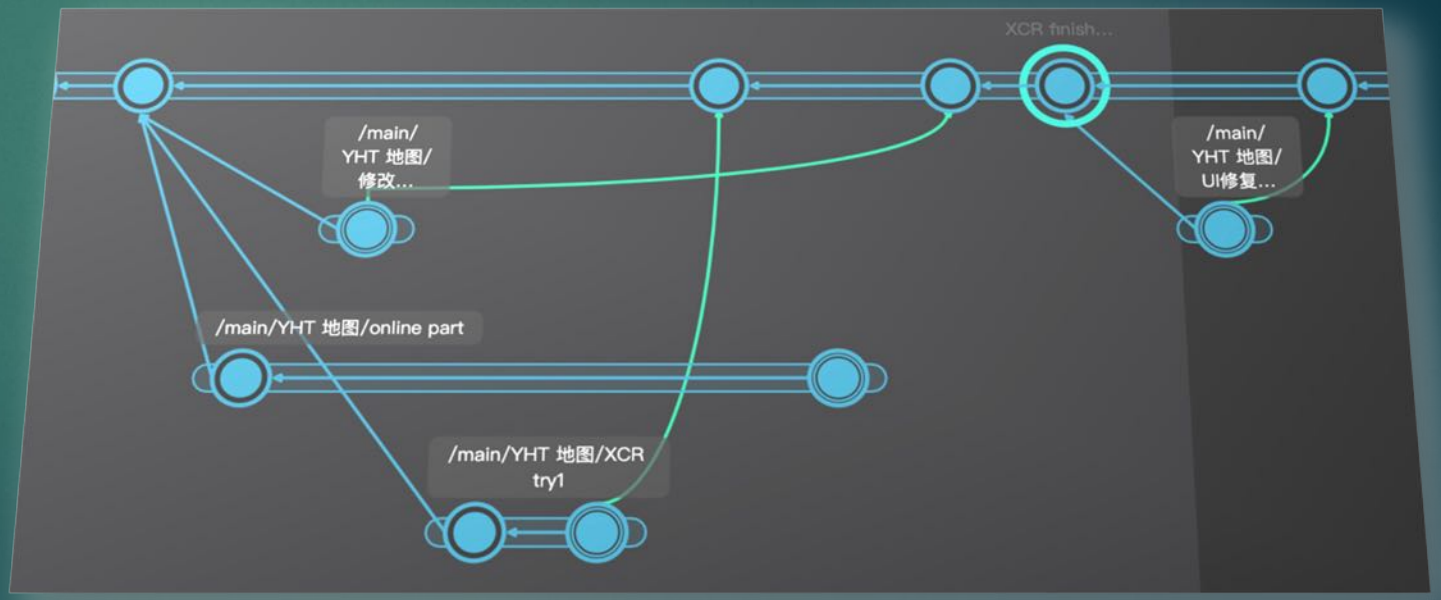
Realistic car control system

Random procedure level generation

Introduction – Version control



Great ideas are formed through trial and error



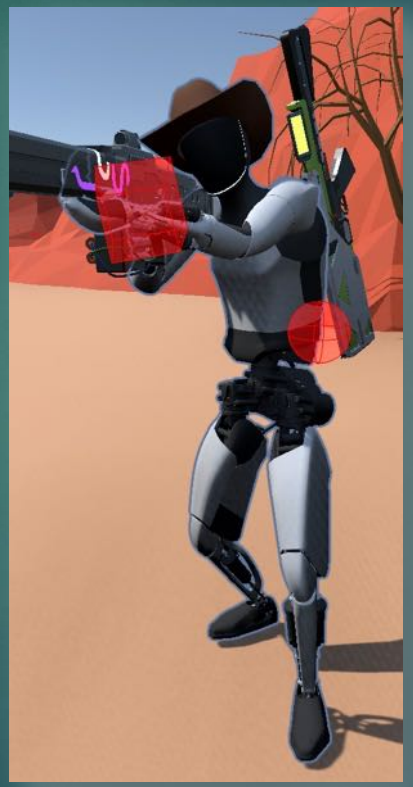
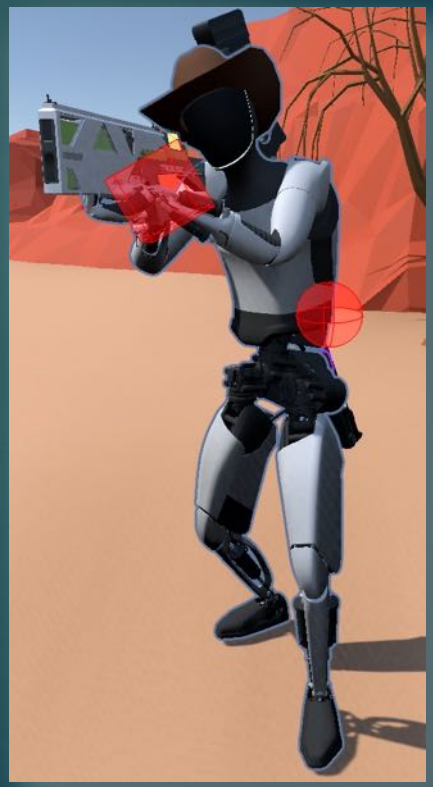

Techniques in the characters

Techniques in the characters - IK

Also known as Animation Rigging

Deeply customized Shooting action for every gun by manipulate IK component

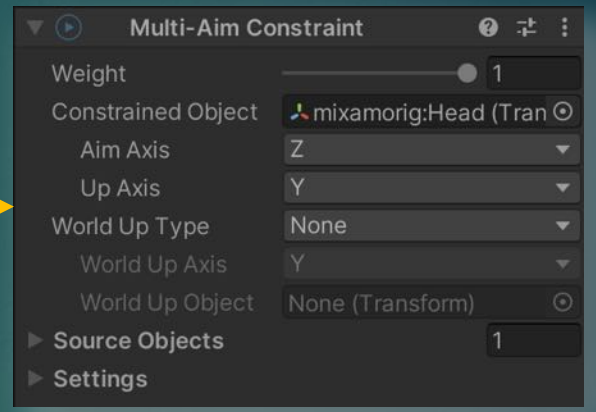


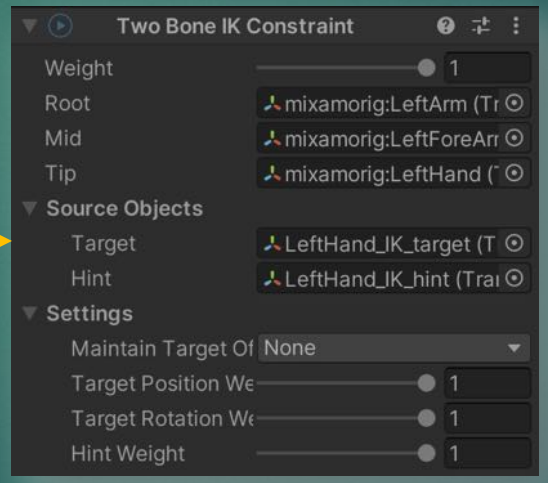




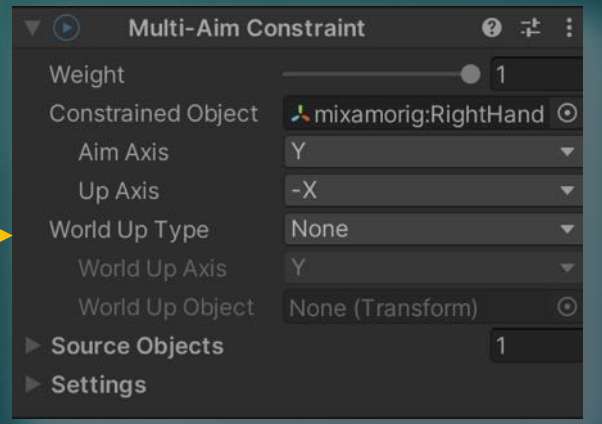
Techniques in the characters - IK

Also known as Animation Rigging

Deeply customized Shooting action for every gun by manipulate IK component








```
public void ReturnRig()
{
    visualController.MaximizeRigWeight();
    visualController.MaximizeLeftHandWeight();
}
```

Have a whole class to control Rig Weight

Techniques in the characters – Blend tree

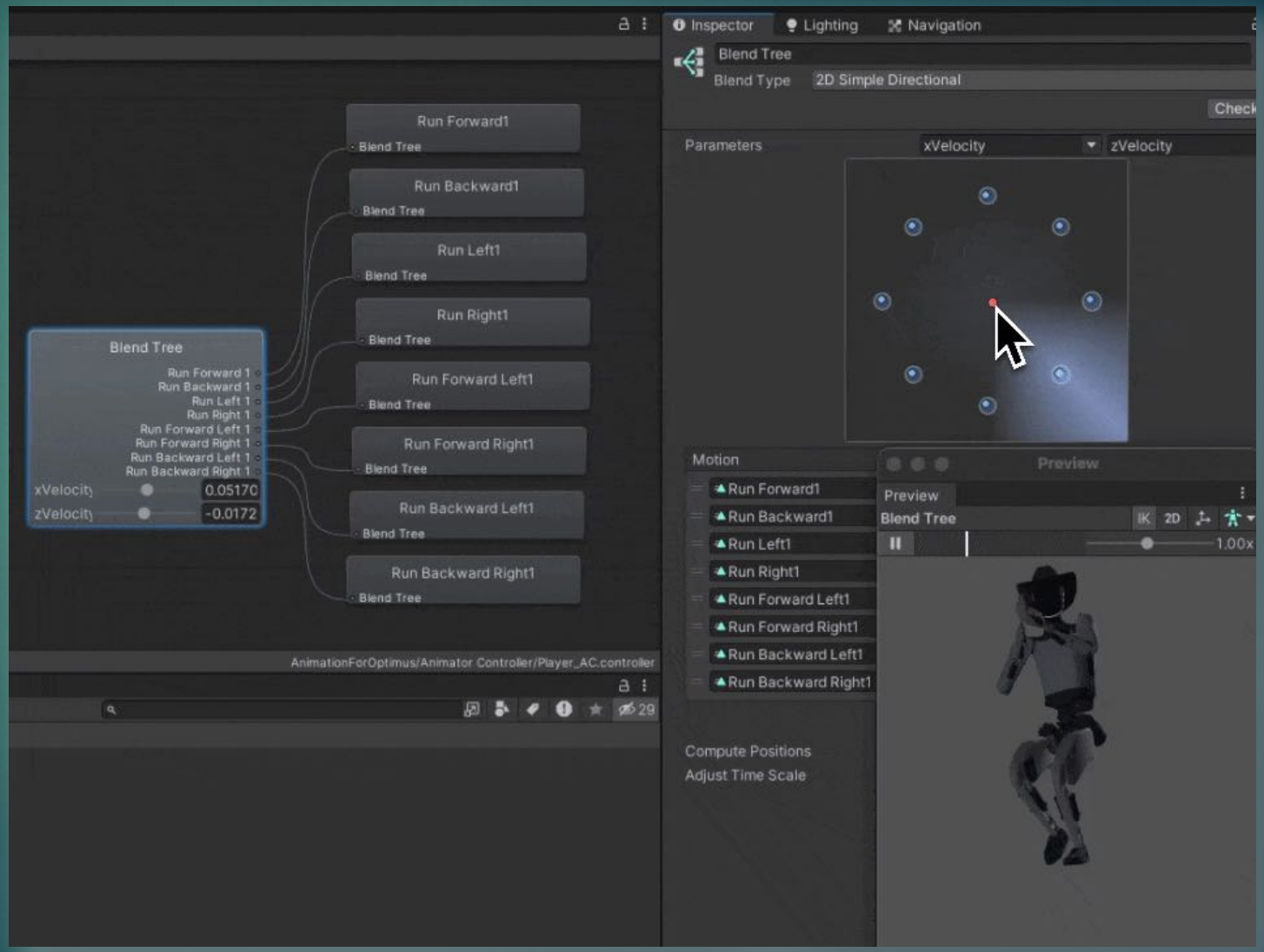
Using blend trees,

we can create actions

- of any intensity
- in any direction

through few key actions

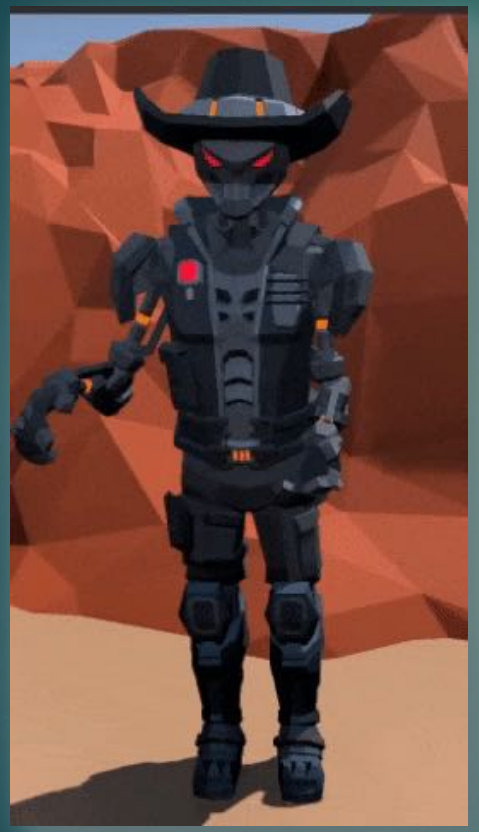
Special blend tree: generating single values(see this later)



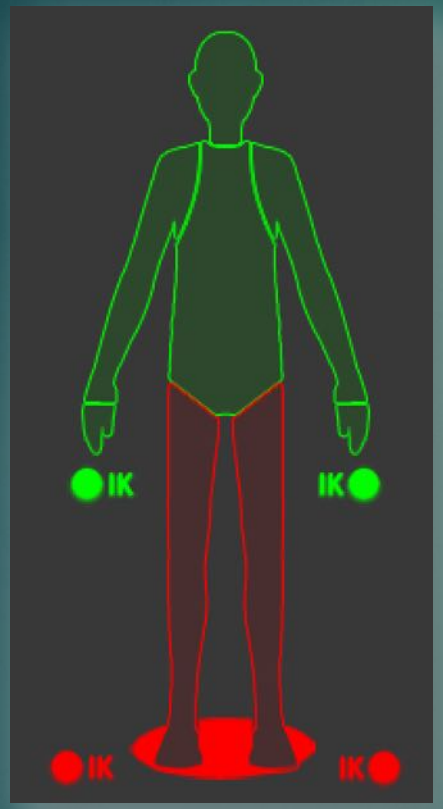
Techniques in the characters – motion capture

- Unparalleled gestures
- Special Customization

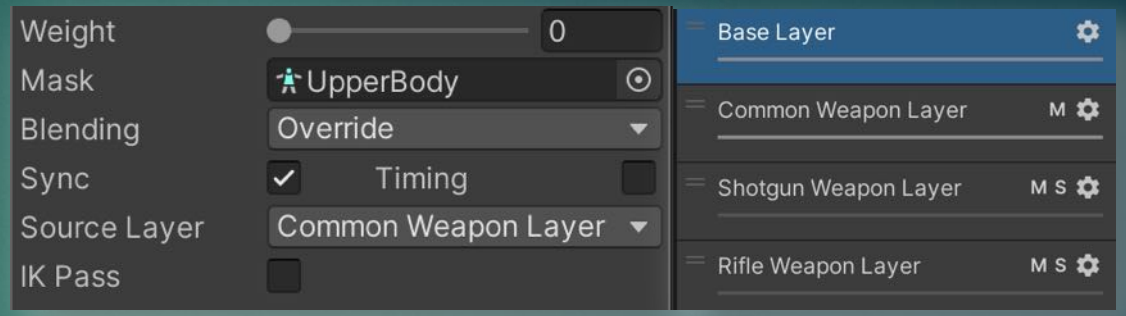
PERCEPTION NEURON 3 PRO



Techniques in the characters – other details



Avatar mask to control separately



Animation Sync to improve efficiency

```
private void AssignInputEvents()
{
    controls = player.controls;
    controls.Character.Movement.performed += context => moveInput = context.ReadValue<Vector2>();
    controls.Character.Movement.canceled += context =>
    {
        StopFootstepsSFX();
        moveInput = Vector2.zero;
    };
    controls.Character.Run.performed += context =>
    {
        speed = runSpeed;
        isRunning = true;
    };
    controls.Character.Run.canceled += context =>
    {
        speed = walkSpeed;
        isRunning = false;
    };
}
```

As usual, we handle everything in our code


Weapon and Map

Weapon system – Type

Pistol

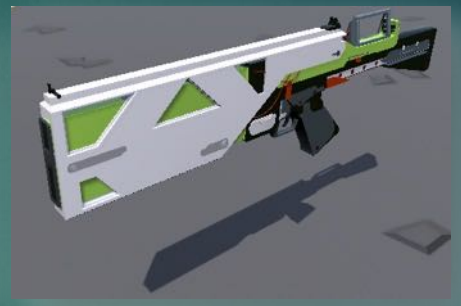
Quick to draw,
Quick to use,
Quick to kill

Revolver

Suppose to
one shot enemies

Shotgun

It’s like a rifle,
But an opposite



Auto-Rifle

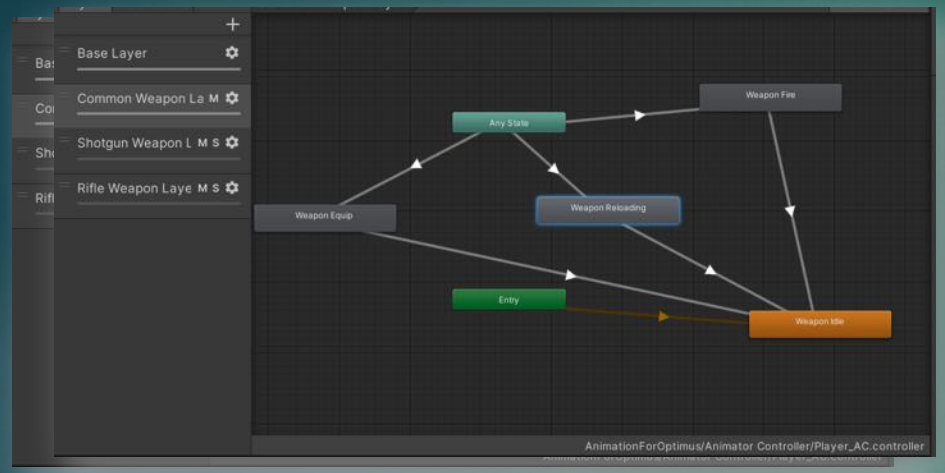
Good shredder
for enemies

Rifle

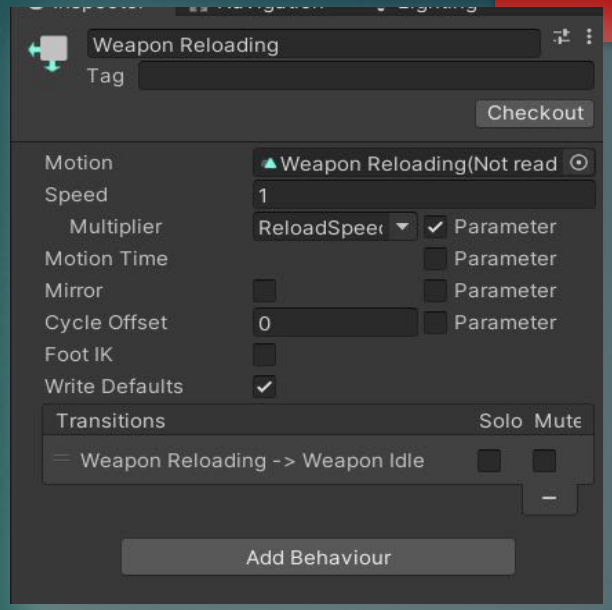
Good on a big
distance

Weapon system - Animation

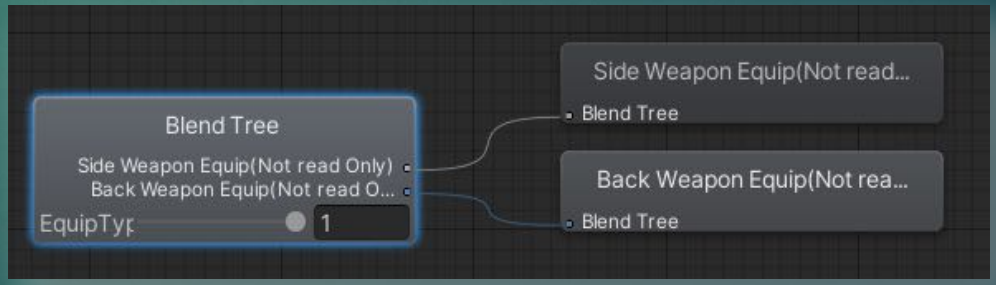
Animator of weapon system



Reload animation



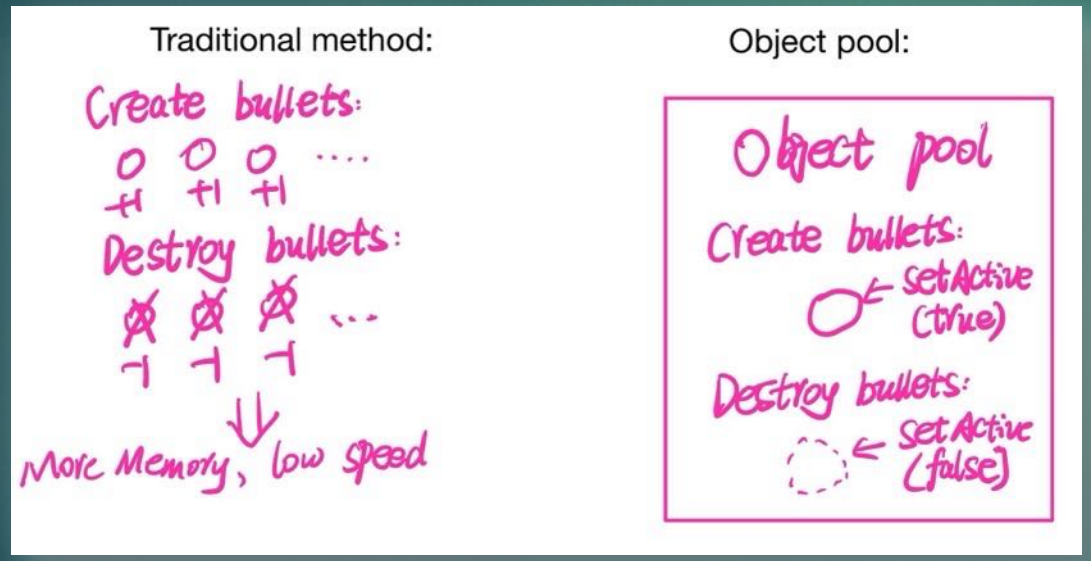
Equip animation



Weapon system - Optimization

A new optimization technique -- Object pool

- Object pools can be used in various scenarios that require frequent instantiation and destruction of game objects.
- Mainly used here for the creation and destruction of weapon bullets
- The main goal is to reduce the frequency of frequent memory allocation and garbage collection, improve the efficiency of game operation.




```
public GameObject GetObject(GameObject prefab, Transform target)
{
    if (poolDictionary.ContainsKey(prefab) == false)
    {
        InitializeNewPool(prefab);
    }

    if (poolDictionary[prefab].Count == 0)
    {
        CreateNewObject(prefab);
    }

    GameObject objectToGet = poolDictionary[prefab].Dequeue();

    objectToGet.transform.position = target.position;
    objectToGet.transform.parent = null;

    // Traditional method: Instantiate(object)
    objectToGet.SetActive(true);

    return objectToGet;
}
```

```
private void ReturnToPool(GameObject objectToReturn)
{
    GameObject originalPrefab = objectToReturn.GetComponent<PooledObject>().originalPrefab;

    // Traditional method: Destroy(object)
    objectToReturn.SetActive(false);
    objectToReturn.transform.parent = transform;

    poolDictionary[originalPrefab].Enqueue(objectToReturn);
}
```

Mini Map

Drag

```
public void OnBeginDrag(PointerEventData eventData)
{
    dragOffset = map.position - Input.mousePosition;
}

public void OnDrag(PointerEventData eventData)
{
    PosionArea p = GetPositionArea();
    Vector3 tmp = Input.mousePosition + dragOffset;
    float x = Mathf.Clamp(tmp.x, p.minX, p.maxX);
    float y = Mathf.Clamp(tmp.y, p.minY, p.maxY);
    map.position = new Vector3(x, y, 0);

    UpdateIconsPosition();
}

public void OnEndDrag(PointerEventData eventData)
{
    UpdatePivot(GetCurPivot());
}
```

Zoom

```
float scale = 1;
public void OnZoom(float detalSize)
{
    scale += detalSize;
    scale = Mathf.Clamp(scale, 1, 5);
    map.localScale = Vector3.one * scale;

    PosionArea p = GetPositionArea();
    float x = Mathf.Clamp(map.position.x, p.minX, p.maxX);
    float y = Mathf.Clamp(map.position.y, p.minY, p.maxY);
    map.position = new Vector3(x, y, 0);

    UpdateIconsPosition();
}
```

Synchronization

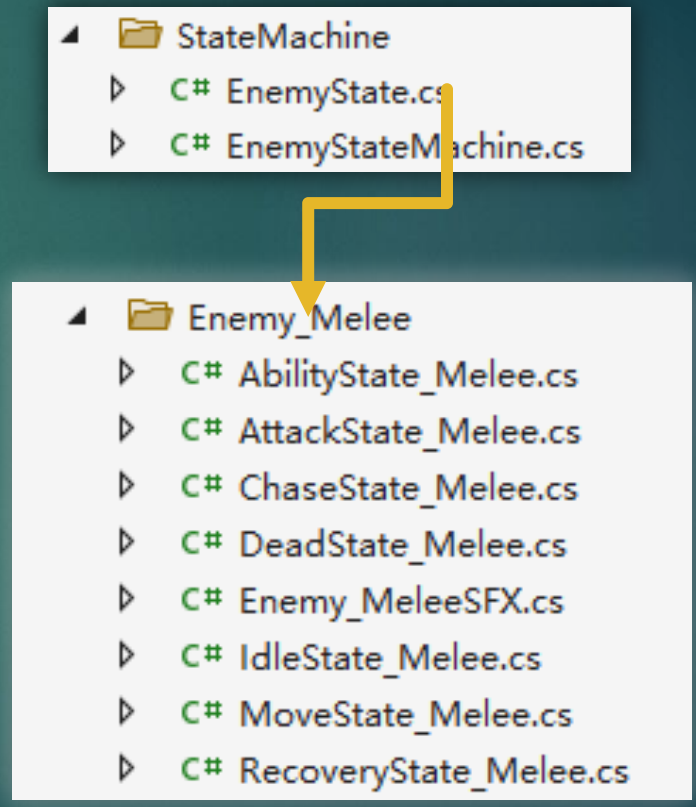
```
int playerIconID = 0;
public void SetPlayerIcon()
{
    Vector3 angle = new Vector3(0, 0, 180 - player.eulerAngles.y);

    if(playerIconID == 0)
    {
        playerIconID = mapWin.AddIconOnMap(playerUV.x, playerUV.y, Color.green);
    }
    mapWin.UpdateIconPosition(playerIconID, playerUV.x, playerUV.y,angle);
}
```


Enemy Melee

Enemy Melee – State Machine

1. A state machine is a system that manages the behavior of enemies. It comprises a series of states, where each state represents the enemy's behavior in a specific situation. The state machine can **switch between these states** in response to various events and conditions in the game.

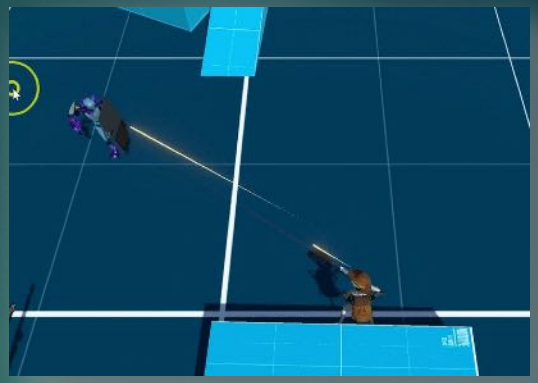
2. Meanwhile, by **encapsulating** the behavior of each state in its own class, the code structure becomes clearer, easier to understand, and maintain.



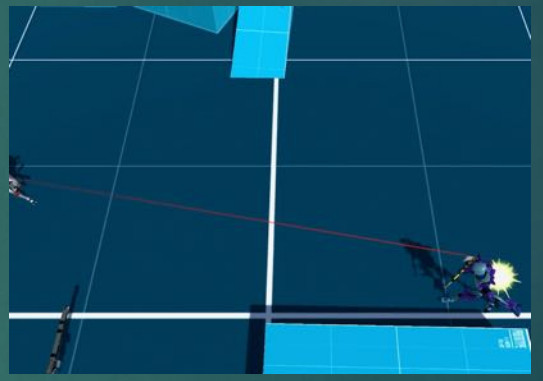
Enemy Melee – Four Variants

- Enemy_Melee - Axe throw Variant
- Enemy_Melee - Dodge Variant
- Enemy_Melee - Regular Variant
- Enemy_Melee - Shield Variant

Enemy Settings
Melee Type: Axe Throw
Weapon Tpye: Throw

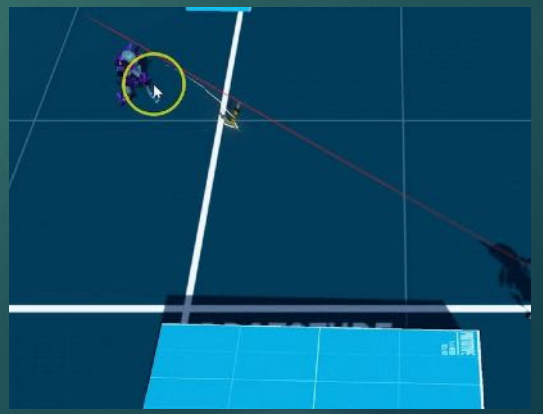


Enemy Settings
Melee Type: Dodge
Weapon Tpye: Unarmed



Enemy Settings
Melee Type: Shield
Weapon Tpye: One Hand

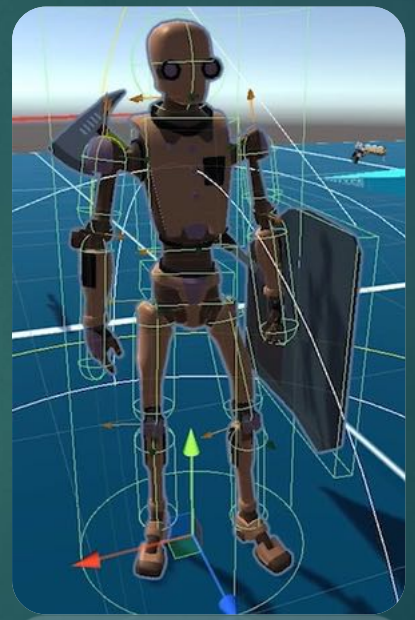
Enemy Settings
Melee Type: Regular
Weapon Tpye: One Hand



Enemy Melee – Random

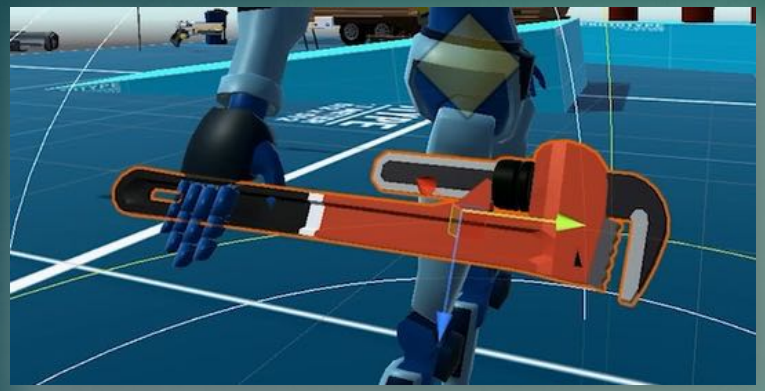
1. Random Look

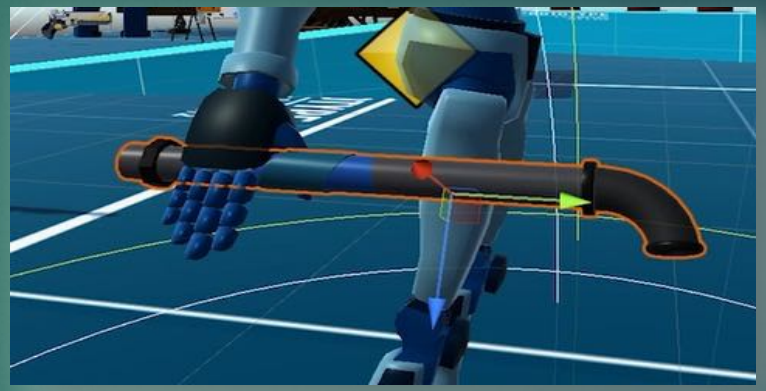
By randomly adding different texture

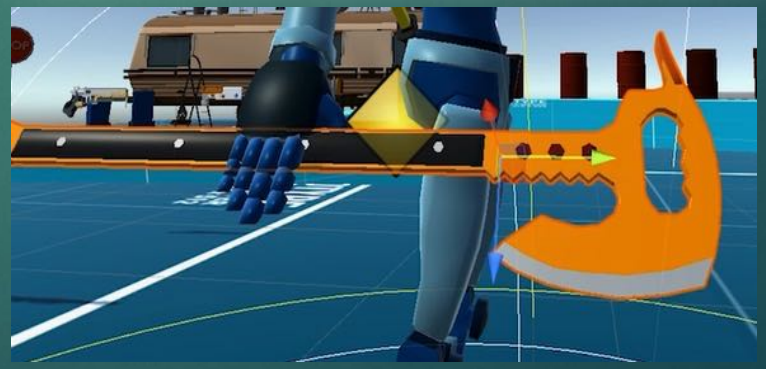


Enemy Melee – Random

2. Random Weapon







Enemy Melee – Random

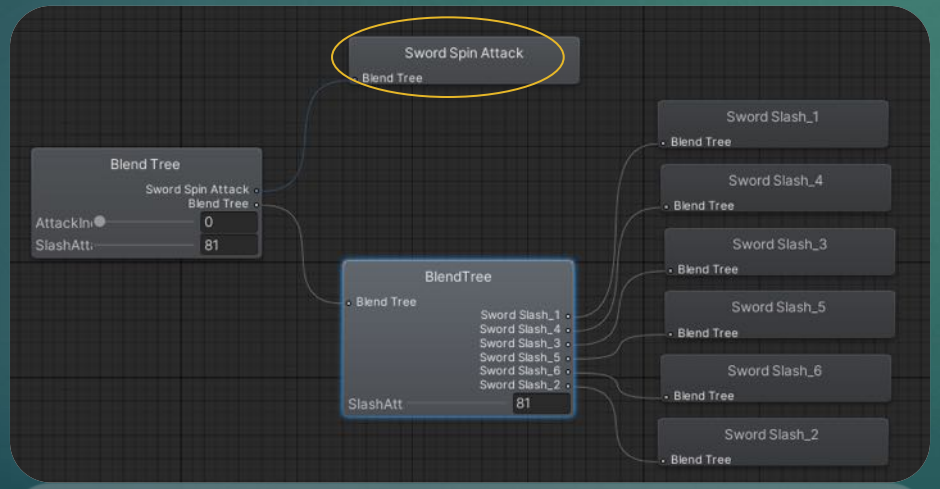
3. Random Corruption

Special features - Attack

```
enemy.anim.SetFloat("SlashAttackIndex", Random.Range(0, 6)); //six attacks with index 0~5
```

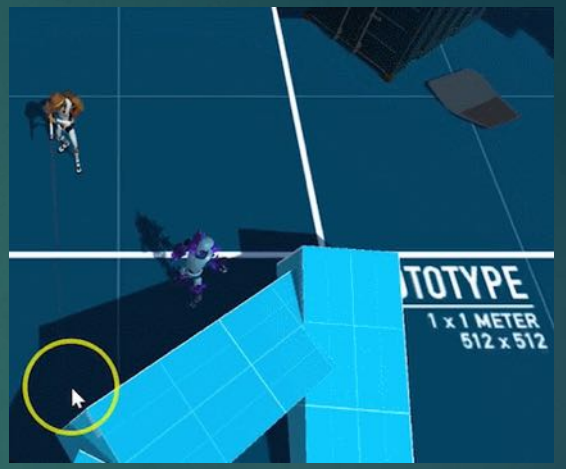
In Unity, we used the Blend Tree feature to achieve smooth and natural transitions between animations. The Blend Tree makes these transitions more smooth.

For melee enemies, there are six different melee attack animations and a special attack like "Sword Spin Attack". A randomly generated index is used to select which animation to play.

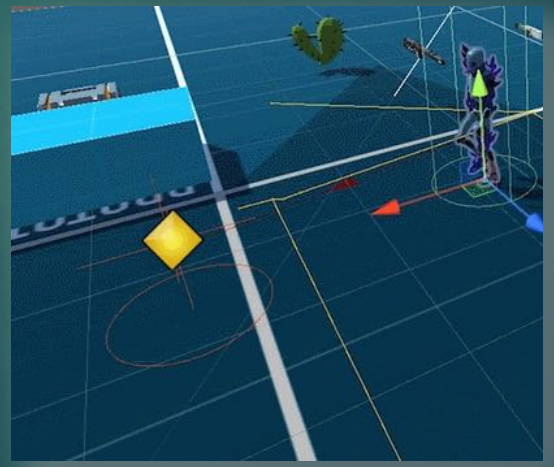


Special features – Manual Rotation (Optimization)

To make the enemy move more naturally, we first calculate the direction from the enemy's position to the target position. Then, we create a “Quaternion” to represent the rotation of this direction. Finally, we use the “Quaternion.Slerp” method to smoothly rotate the enemy's orientation to this rotation.




```
15 个引用 | Changed by 35277992400@qq.com on 2024年5月14日
public void FaceTarget(Vector3 target, float turnSpeed1 =0)
{
    Quaternion targetRotation = Quaternion.LookRotation(target - transform.position);

    Vector3 currentEulerAngles = transform.rotation.eulerAngles;

    if(turnSpeed1 ==0)
        turnSpeed1 = this.turnSpeed;

    float yRotation =
        Mathf.LerpAngle(currentEulerAngles.y, targetRotation.eulerAngles.y, turnSpeed1 * Time.deltaTime);

    transform.rotation = Quaternion.Euler(currentEulerAngles.x, yRotation, currentEulerAngles.z);

}
```



Special features – Animator override (Optimization)

Original	Override
Dodge Roll	None (Animation Clip)
Dwarf Walk	None (Animation Clip)
Fast Run	None (Animation Clip)
Happy Idle	None (Animation Clip)
Melee_Axe Throw	None (Animation Clip)
Melee_Recovery_01	Threatening
Melee_Recovery_02	Boxing_Recovery
Run with shield	None (Animation Clip)
Sword Slash_1	Boxing_attack_1
Sword Slash_2	Boxing_attack_2
Sword Slash_3	Boxing_attack_3
Sword Slash_4	Boxing_attack_4
Sword Slash_5	Boxing_attack_5
Sword Slash_6	Boxing_attack_6
Sword Spin Attack	Flying Kick

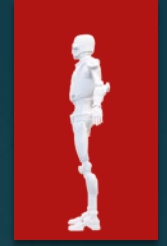
The Animator Override Controller allows to

- replace the original animation clips in an Animator Controller with specific animation clips,
- enabling customization of animations for a given character or object.

Benefits include

- optimized performance,
- ease of management and maintenance,
- flexibility,
- and reduced development time and costs.

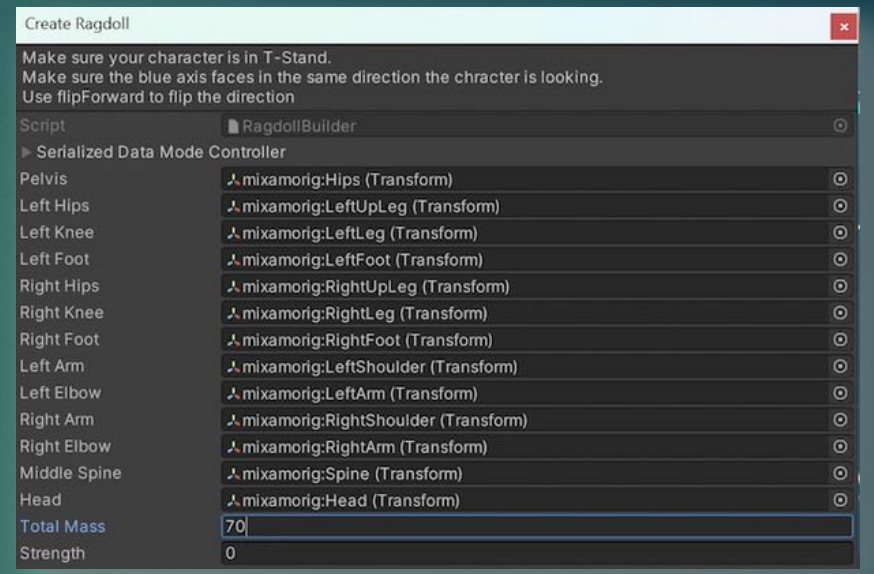
Special features – Rogdoll

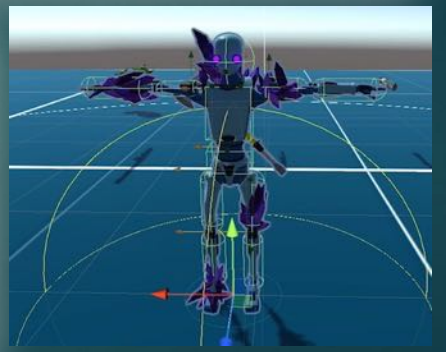


In the game, we didn't use pre-set animations, but instead used the Ragdoll effect to simulate the natural reaction of a character after death.

First, we created Ragdolls for each part of the character's body, and then activated the ragdoll effect when the character dies. allowing the character's body parts to be independently influenced by physical forces.

```
6 个引用 | Changed by lishaoxu1234@gmail.com on 2024年5月7日
public void RagdollActive(bool active)
{
    foreach (Rigidbody rb in ragdollRigidbodies)
    {
        rb.isKinematic = !active;
    }
}
```





Allowing the character's body parts to be independently influenced by physical forces.


Enemy Range

Enemy_Range - Five Type

- Pistol
- Revolver
- auto rifle
- Rifle
- shotgun





Enemy_Range – Mechanism

Weapon Hold Type

Common

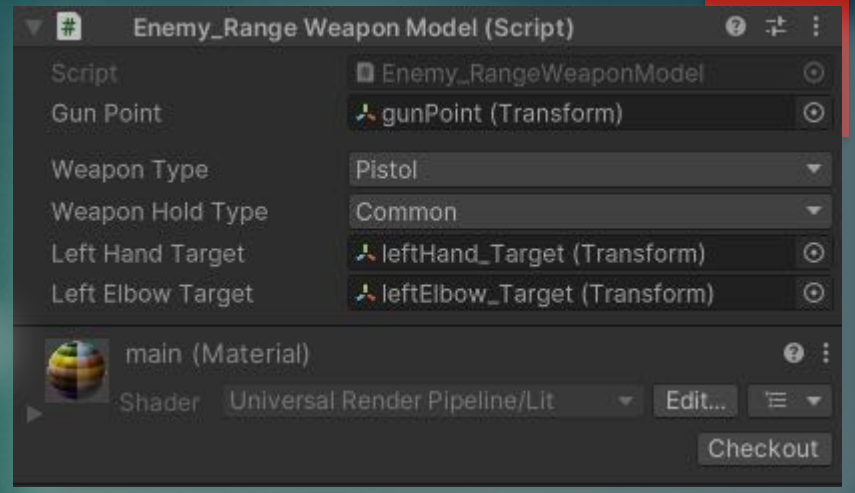
High hold

Low hold

Enemy_Range - Mechanism

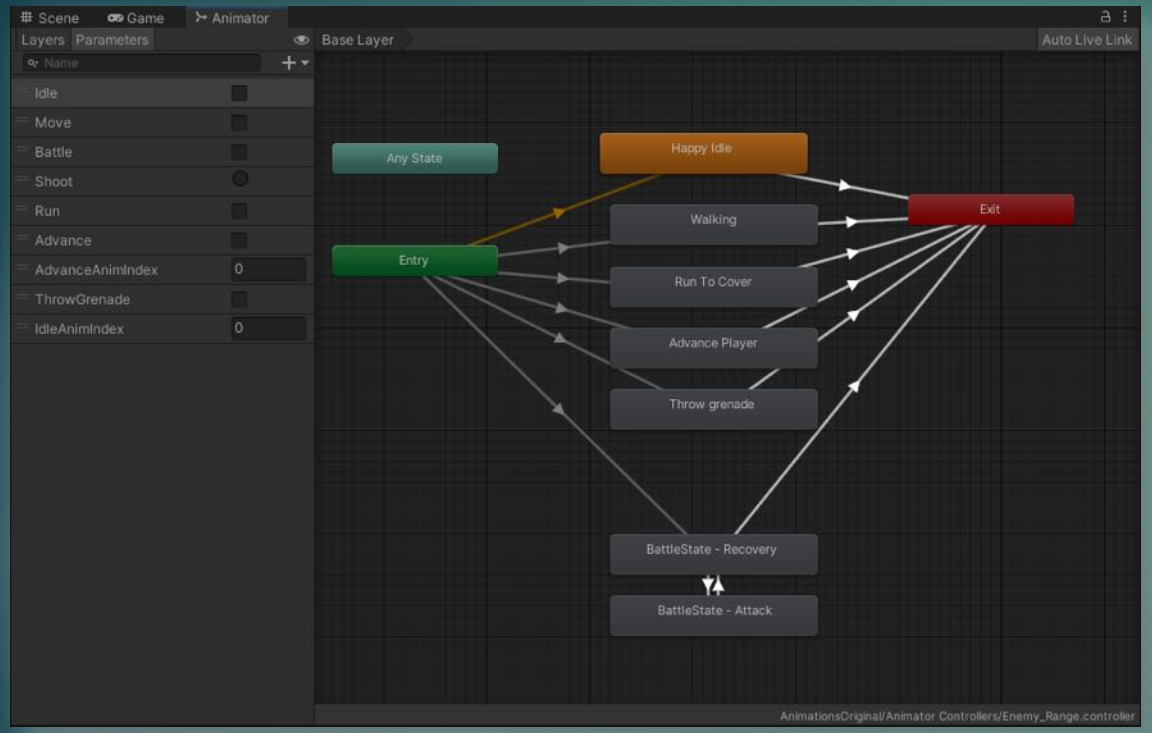
Weapon Hold Type(Same as player)

```
public enum Enemy_RangeWeaponType { Pistol, Revolver, Shotgun, AutoRifle, Rifle }
public enum Enemy_RangeWeaponHoldType { Common, LowHold, HighHold};
```

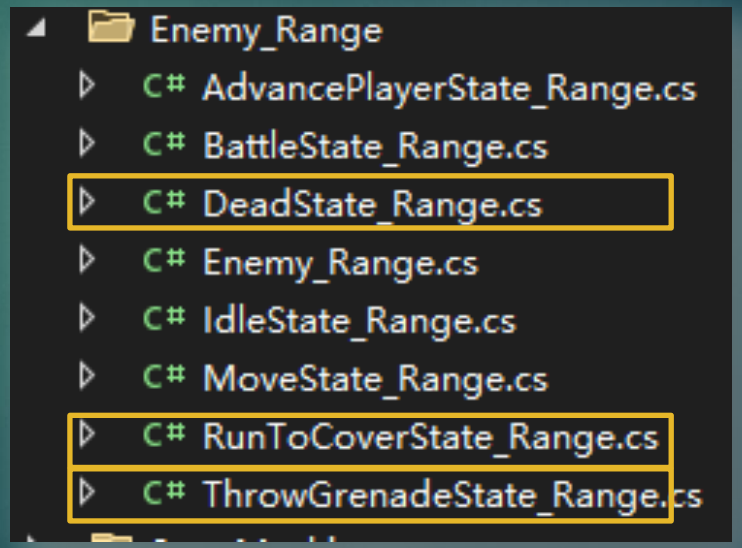


- Setup weapon
- IK constraint
- Take transform value
- Initialize

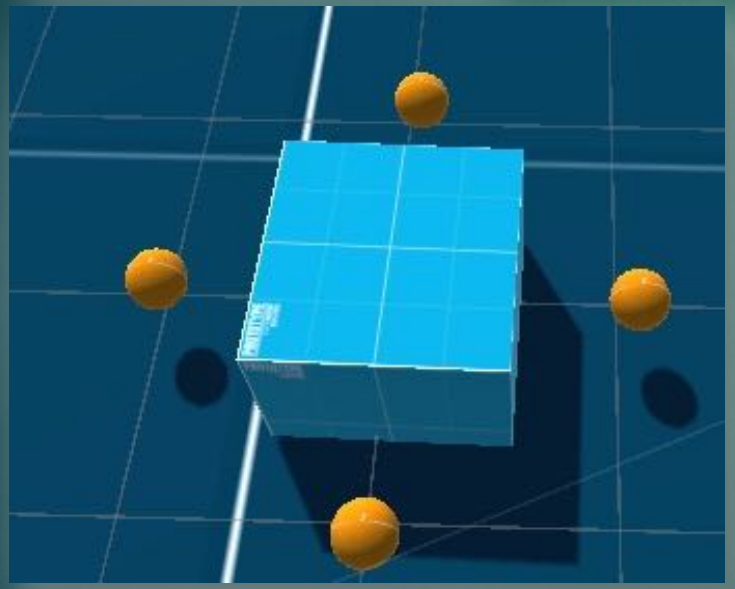
Enemy_Range – Mechanism: StateMachine

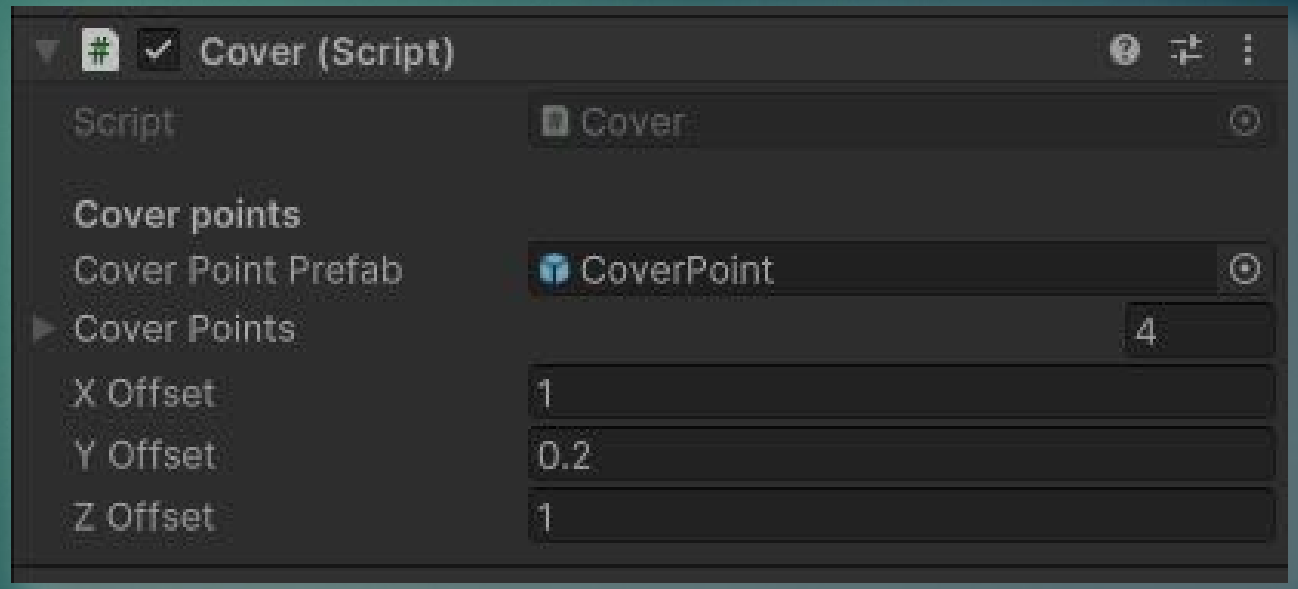



```
protected override void Awake()
{
    base.Awake();

    idleState = new IdleState_Range(this, stateMachine, "Idle");
    moveState = new MoveState_Range(this, stateMachine, "Move");
    battleState = new BattleState_Range(this, stateMachine, "Battle");
    runToCoverState = new RunToCoverState_Range(this, stateMachine, "Run");
    advancePlayerState = new AdvancePlayerState_Range(this, stateMachine, "Advance");
    throwGrenadeState = new ThrowGrenadeState_Range(this, stateMachine, "ThrowGrenade");
    deadState = new DeadState_Range(this, stateMachine, "Idle");// idle is a place holder,we using ragdoll
}
```



Enemy_Range – Mechanism: Cover





Enemy_Range - Mechanism

Cover

Find Valid Cover : Distance check

- Not occupied
- Far from Player
- Not behind Player
- Not too Close to Last Cover

```
public List<CoverPoint> GetValidCoverPoints(Transform enemy)
{
    List<CoverPoint> validCoverPoints = new List<CoverPoint>();

    foreach (CoverPoint coverPoint in coverPoints)
    {
        if (IsValidCoverPoint(coverPoint, enemy))
            validCoverPoints.Add(coverPoint);
    }

    return validCoverPoints;
}
```

```
private bool IsValidCoverPoint(CoverPoint coverPoint, Transform enemy)
{
    if (coverPoint.occupied)
        return false;

    if (IsFutherestFromPlayer(coverPoint) == false)
        return false;

    if (IsCoverCloseToPlayer(coverPoint))
        return false;

    if (IsCoverBehindPlayer(coverPoint, enemy))
        return false;

    if (IsCoverCloseToLastCover(coverPoint, enemy))
        return false;

    return true;
}
```

Enemy_Range – Mechanism: Throw Grenade


```
private Vector3 CalculateLaunchVelocity(Vector3 target, float timeToTarget)
{
    Vector3 direction = target - transform.position;
    Vector3 directionXZ = new Vector3(direction.x, 0 , direction.z);

    Vector3 velocityXZ = directionXZ / timeToTarget;

    float velocityY =
        (direction.y - (Physics.gravity.y * Mathf.Pow(timeToTarget, 2)) / 2) / timeToTarget;

    Vector3 launchVelocity = velocityXZ + Vector3.up * velocityY;

    return launchVelocity;
}
```

Y= v*t +1/2(g * t^2) g is negative

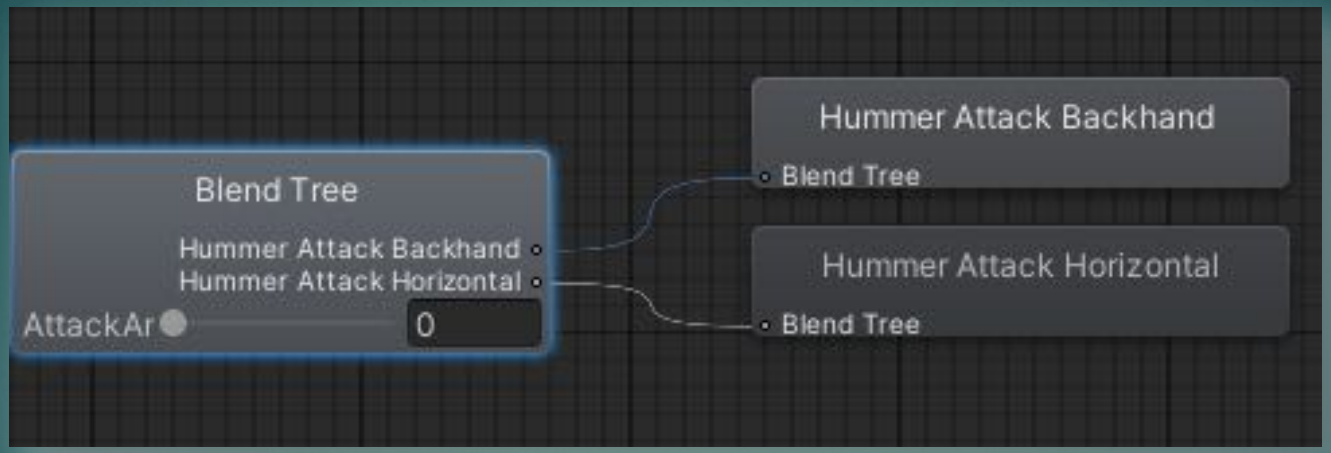
Enemy_Boss:

Flame Thrower

Hammer

Enemy_Boss - Mechanism - hammer

Two types of attack method , based on probability blend tree

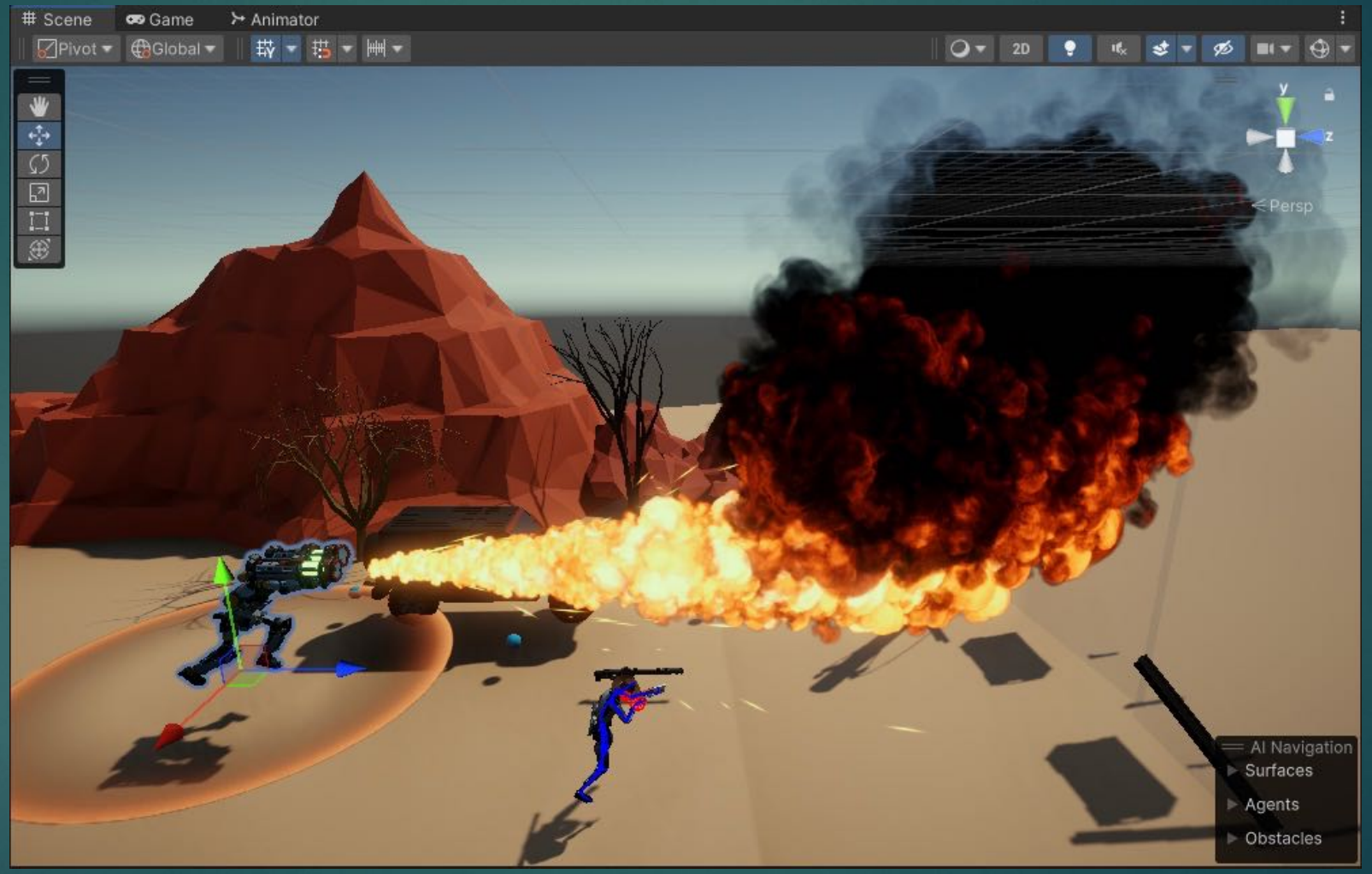



```
24 个引用
public override void Enter()
{
    base.Enter();

    enemy.bossVisuals.EnableWeaponTrail(true);


    enemy.anim.SetFloat("AttackAnimIndex", Random.Range(0, 2)); // have two attacks with index 0&1
    enemy.agent.isStopped = true;

    stateTimer = 1f;
}
```

Enemy_Boss

Mechanism

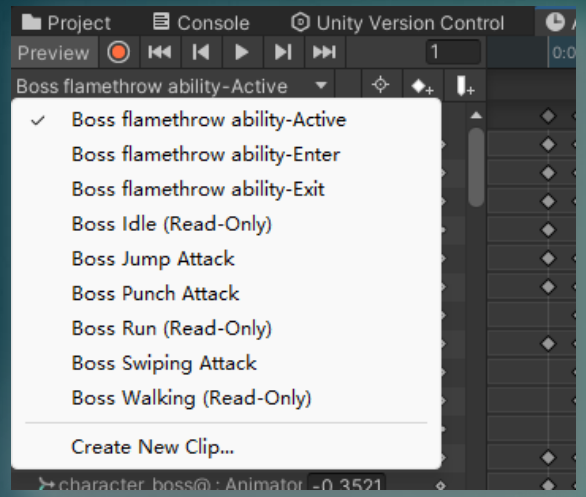
- Flame thrower



Enemy_Boss

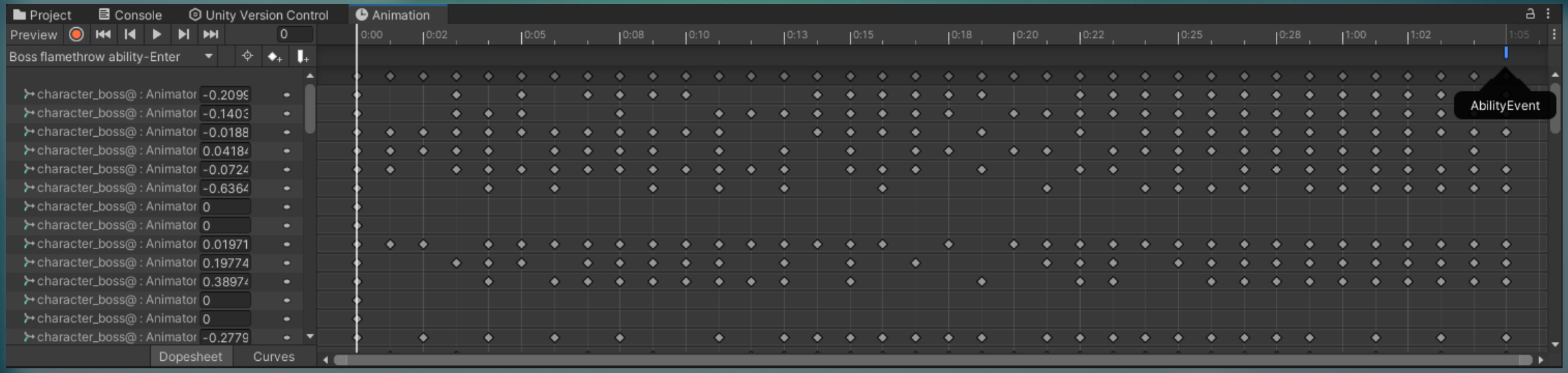
Mechanism

- Flame thrower



- Enter Ability state
- Play animation “Enter”
- Play animation “Active”
- show flame effect

- hide flame effect
- Play animation “Exit”
- Stop Ability state

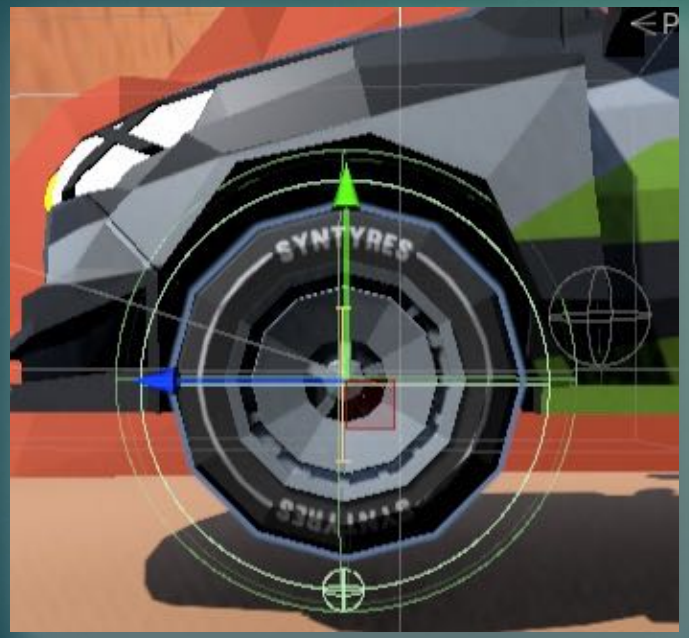


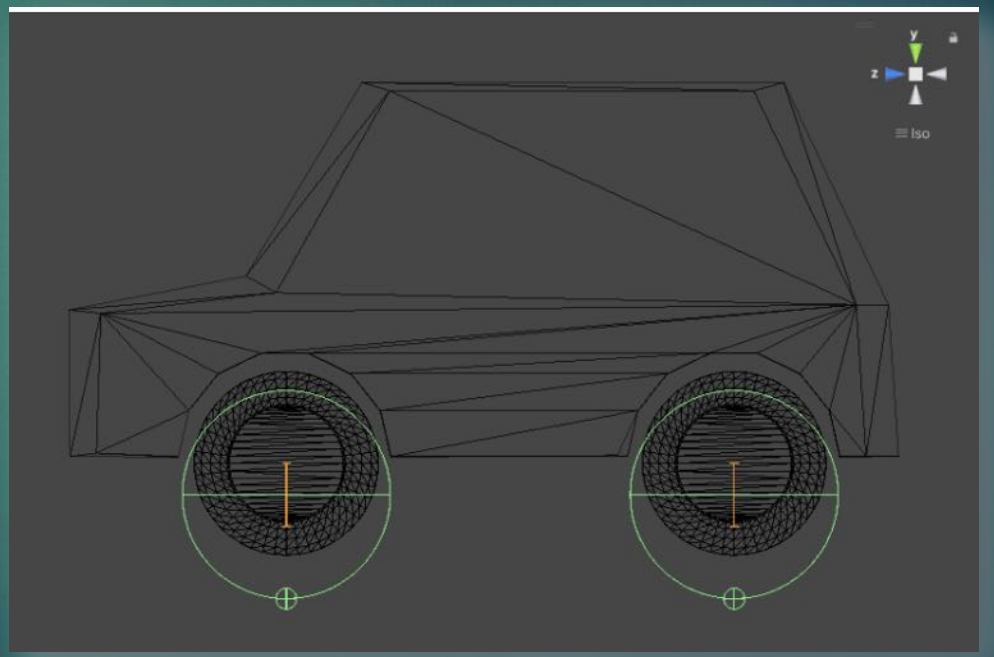

Techniques in Cars

Techniques in Cars - wheelColider

The Wheel collider

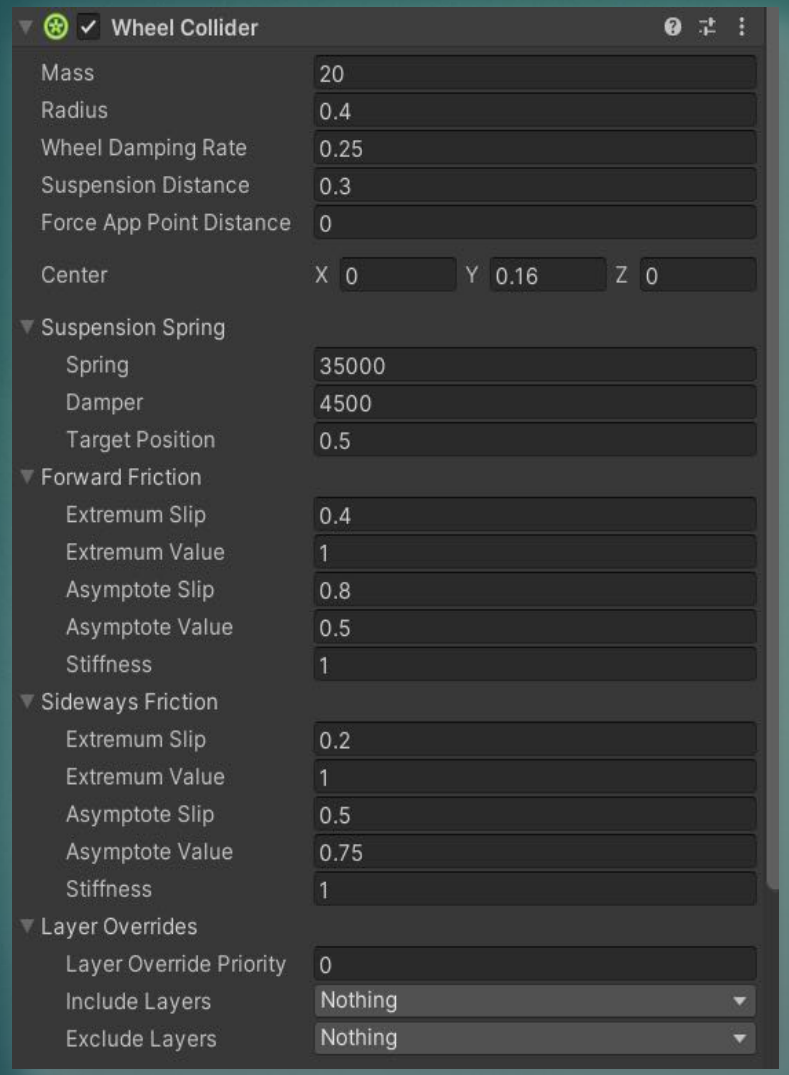
- is a collider for ground vehicles.
- has built-in collision detection, wheel physics, and a slip-based tire friction model.
- Through the collider, different effects can be set to realize the physical simulation of the wheel.

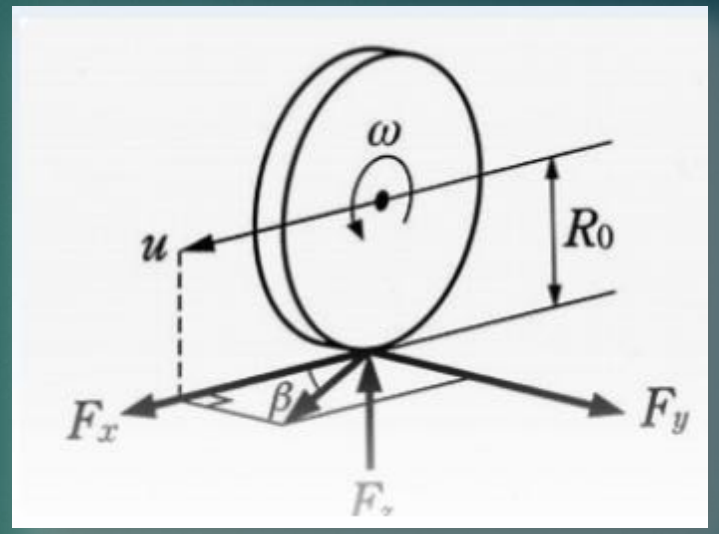




Techniques in Cars - wheelColider

Property
Mass
Radius
Wheel Damping Rate
Suspension Distance
Force App Point Distance
Center

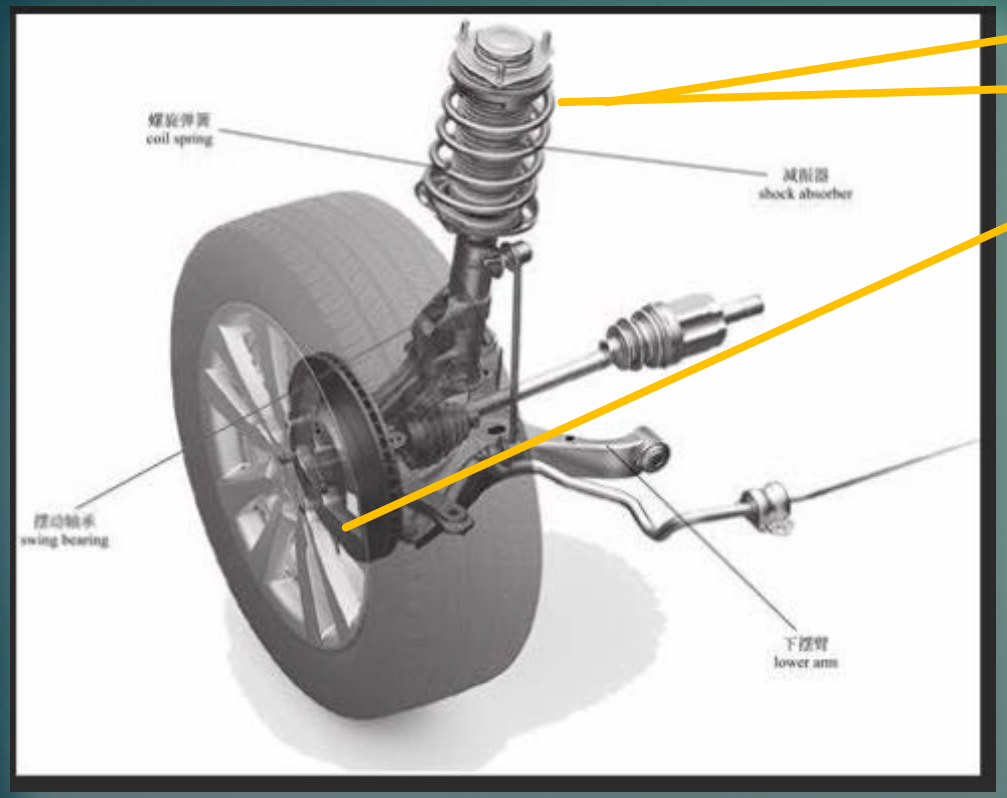


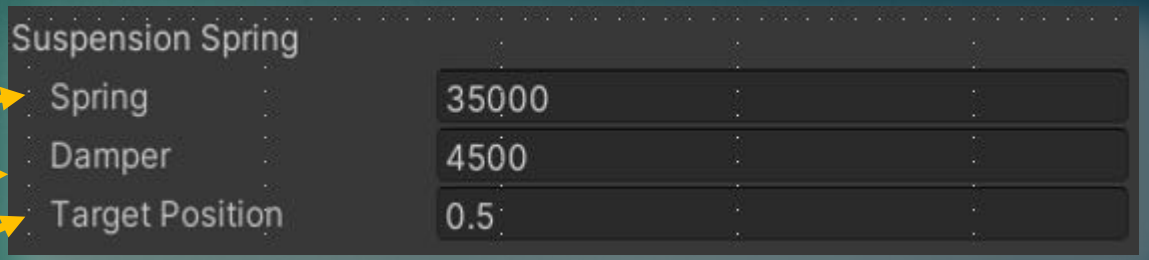


- Colliders have several very important properties
- Mass, center, forward friction, side friction, coefficient of friction, and so on.

Techniques in Cars - wheelColider

The suspension spring is one of the more difficult properties to understand.





The suspension spring as a whole is designed to simulate the vibration and jolting effect of the wheel.

Techniques in Cars - wheelColider

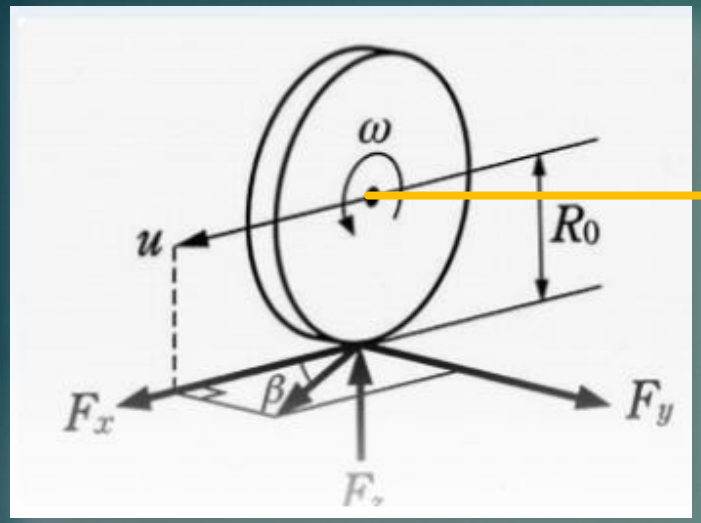
The collider not only simulates the effect of the tyer, but also the effect of the axle. This is the basis for the realization of car drive.

```
private void ApplyDrive()
{
    currentSpeed = moveInput * accleerationSpeed * Time.deltaTime;

    float motorTorqueValue = motorForce * currentSpeed;

    foreach (var wheel in wheels)
    {
        if (driveType == DriveType.FrontWheelDrive)
        {
            if (wheel.axelType == AxelType.Front)
                wheel.cd.motorTorque = motorTorqueValue;
        }
        else if (driveType == DriveType.RearWheelDrive)
        {
            if (wheel.axelType == AxelType.Back)
                wheel.cd.motorTorque = motorTorqueValue;
        }
        else
        {
            wheel.cd.motorTorque = motorTorqueValue;
        }
    }
}
```

```
private void ApplyBrakes()
{

    foreach (var wheel in wheels)
    {
        bool frontBrakes = wheel.axelType == AxelType.Front;
        float brakeSensetivity = frontBrakes ? frontBrakesSensetivity : backBrakesSensetivity;

        float newBrakeTorque = brakePower * brakeSensetivity * Time.deltaTime;
        float currentBrakeTorque = isBraking ? newBrakeTorque : 0;

        wheel.cd.brakeTorque = currentBrakeTorque;
    }
}
```

- MortorTorque and Braking Torque built into the collider.
- Creates friction between the wheels and the ground to stop the car.

Techniques in Cars - wheelColider



- The motorForce and braking force mentioned above actually act on the central position of the wheel. In fact, Colider's motorForce is a torque, which is calculated by:

```
float motorTorqueValue = motorForce * currentSpeed;
```

- The motorForce on the right is the power of the engine that we set.

- The WheelColider component is actually a part of the car that removes almost all of the outside of the car.
- In fact, when we remove the body of the car from the game, the wheels and the base drive forward without being affected.

Techniques in Cars - wheelColider

Also the steering…

```
private void ApplySteering()
{
    foreach (var wheel in wheels)
    {
        if (wheel.axelType == AxelType.Front)
        {
            float targetSteerAngle = steerInput * turnSensetivity;
            wheel.cd.steerAngle = Mathf.Lerp(wheel.cd.steerAngle, targetSteerAngle, .5f);
        }
    }
}
```

When we apply the drift method, we essentially change the lateral friction of the wheel in a certain period of time…

```
private void ApplyDrift()
{
    foreach (var wheel in wheels)
    {
        bool frontWheel = wheel.axelType == AxelType.Front;
        float driftFactor = frontWheel ? frontDriftFactor : backDriftFactor;

        WheelFrictionCurve sidewaysFriction = wheel.cd.sidewaysFriction;

        sidewaysFriction.stiffness *= (1 - driftFactor);
        wheel.cd.sidewaysFriction = sidewaysFriction;
    }
}
```

```
driftTimer -= Time.deltaTime;

if (driftTimer < 0)
    isDrifting = false;
```

Techniques in Cars Logitech steering wheel

```
private void FixedUpdate()
{
    if (carActive == false)
        return;
    if (!LogitechGSDK.LogiIsPlaying(0, LogitechGSDK.LOGI_FORCE_SPRING))
    {

        LogitechGSDK.LogiPlaySpringForce(0, 0, 30, 100);
    }
    if (LogitechGSDK.LogiUpdate() && LogitechGSDK.LogiIsConnected(0))
    {
        //CONTROLLER STATE
        LogitechGSDK.DIJOYSTATE2ENGINES rec = LogitechGSDK.LogiGetStateUnity(0);
        float steering = rec.lX / 32768f;
        float accel = -(rec.lY / 65536f - 0.5f);
        float brake = -(rec.lRz / 65536f - 0.5f);
        float back;
        if (System.Math.Abs(steering) < 0.01) steering = 0;
```

- core API: LogiGetStateUnity(int index)
- Return values for the various steering wheels.

- These values can be normalized, their value is converted to [0,1] into a coefficient.
- We treat rotation within a certain range as no input.

```
if (LogitechGSDK.LogiButtonIsPressed(0, 17))
{
    back = -1;
}
else if (LogitechGSDK.LogiButtonReleased(0, 17))
{
    back = 1;
}
else
{
    back = 1;
}
```

Set the car's gear. While Unity's setup is more complex than UE's, it's also more flexible

Techniques in Cars - Other

```
private void Explode()
{
    HashSet<GameObject> uniqeEntities = new HashSet<GameObject>();
    Collider[] colliders = Physics.OverlapSphere(explosionPoint.position,explosionRadius);

    foreach (Collider hit in colliders)
    {
        IDamagable damagable = hit.GetComponent<IDamagable>();

        if (damagable != null)
        {
            GameObject rootEntity = hit.transform.gameObject;
            if (uniqeEntities.Add(rootEntity) == false)
                continue;
            damagable.TakeDamage(explosionDamage);


            hit.GetComponentInChildren<Rigidbody>().AddExplosionForce(explosionForce,
                explosionPoint.position, explosionRadius, explosionUpwardsModifier, ForceMode.VelocityChange);

        }
    }
}
```

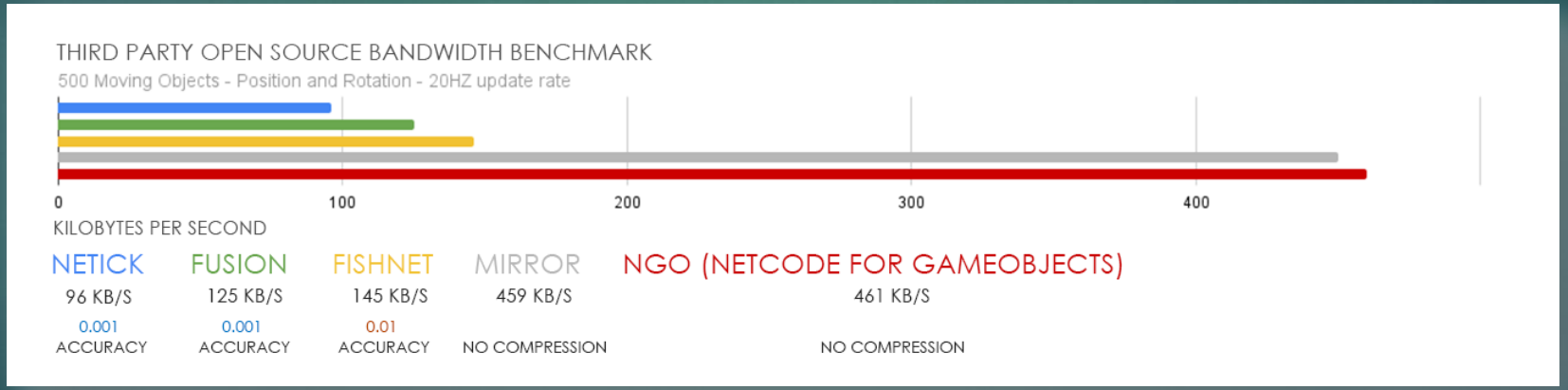
- Health is 0 – explode.
- Cause damage in a certain scope.
- Explode force.

The same method to …


Online network

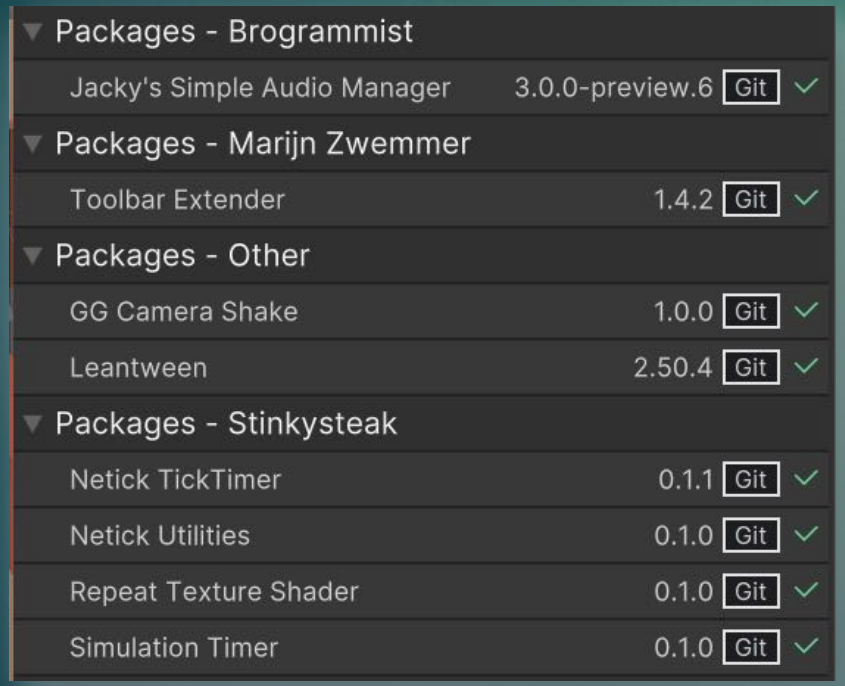
Online network – Test process

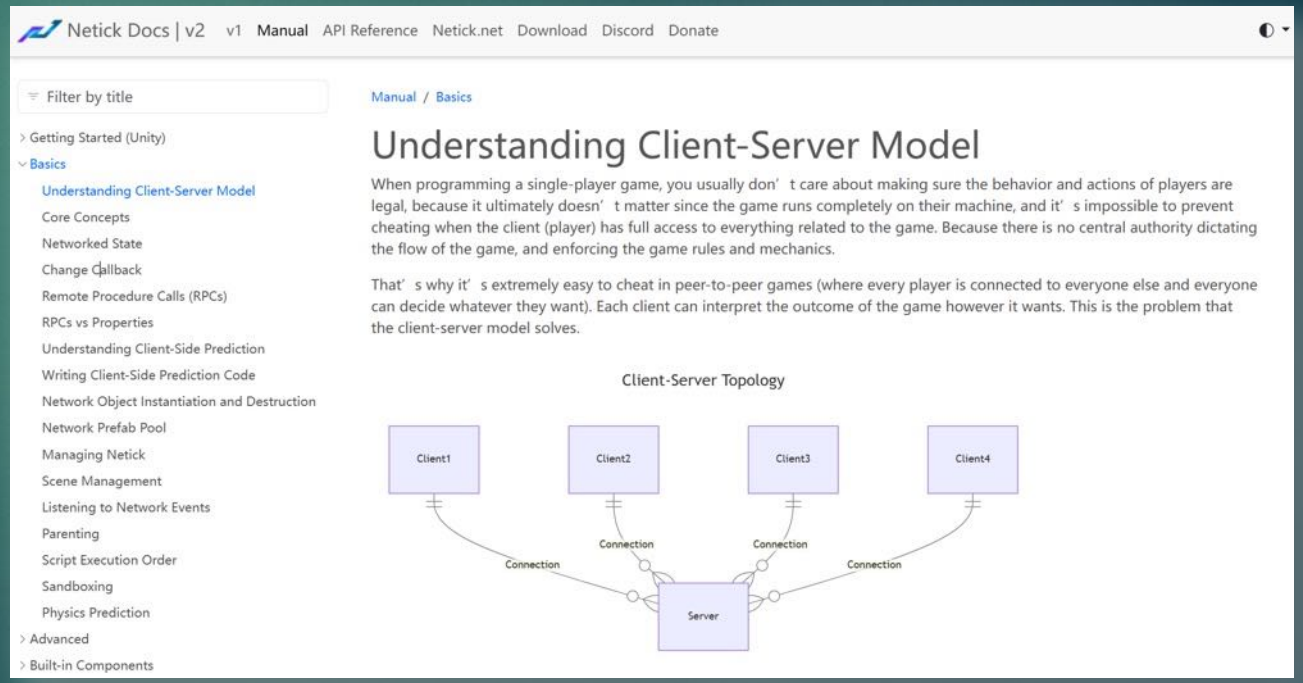
- **Netcode for GameObject**: suitable for cooperative games, not for FPS or TPS
- **Mirror**: requires a hosted server, expensive
- **Photon PUN2**: needs an external network link (VPN)
- **Netick**: low bandwidth usage and a fast and stable network connection



Online network – preparation

- Integrate a large number of third-party libraries
- View Developer documentation





Online network - general setting

- Level
- UI
- Player
- Damage system
- health system
- weapon system
- scoring system

Online network - network tech

- Peer-to-peer network model
  - LAN
- Rewrite all methods
  - A lot of code


```
public override void NetworkRender()
{
    if (_timerMuzzleFlashLifetime.IsExpired(Sandbox))
    {
        _vfxMuzzleFlash.SetActive(false);
        _timerMuzzleFlashLifetime = AuthTickTimer.None;
    }
}
```

```
public override void NetworkFixedUpdate()
{
    PollReloadInput();

    if (_currentBullet <= 0 && !_timerReload.IsRunning)
    {
        _timerReload = TickTimer.CreateFromSeconds(Sandbox, _reloadSpeed);
    }

    if (_timerReload.IsExpired(Sandbox))
    {
        _currentBullet = _bulletPerMagazine;
        _timerReload = TickTimer.None;
    }

    ProcessShooting();
}
```

Online network - algorithm

Client-Side Prediction:

- allows a client to predict the location and state of a game object while waiting for a server response.

Snapshot Interpolation:

- technique for smooth transitions to the position and state of game objects on the client side.

Remote Procedure Calls (RPCs) :

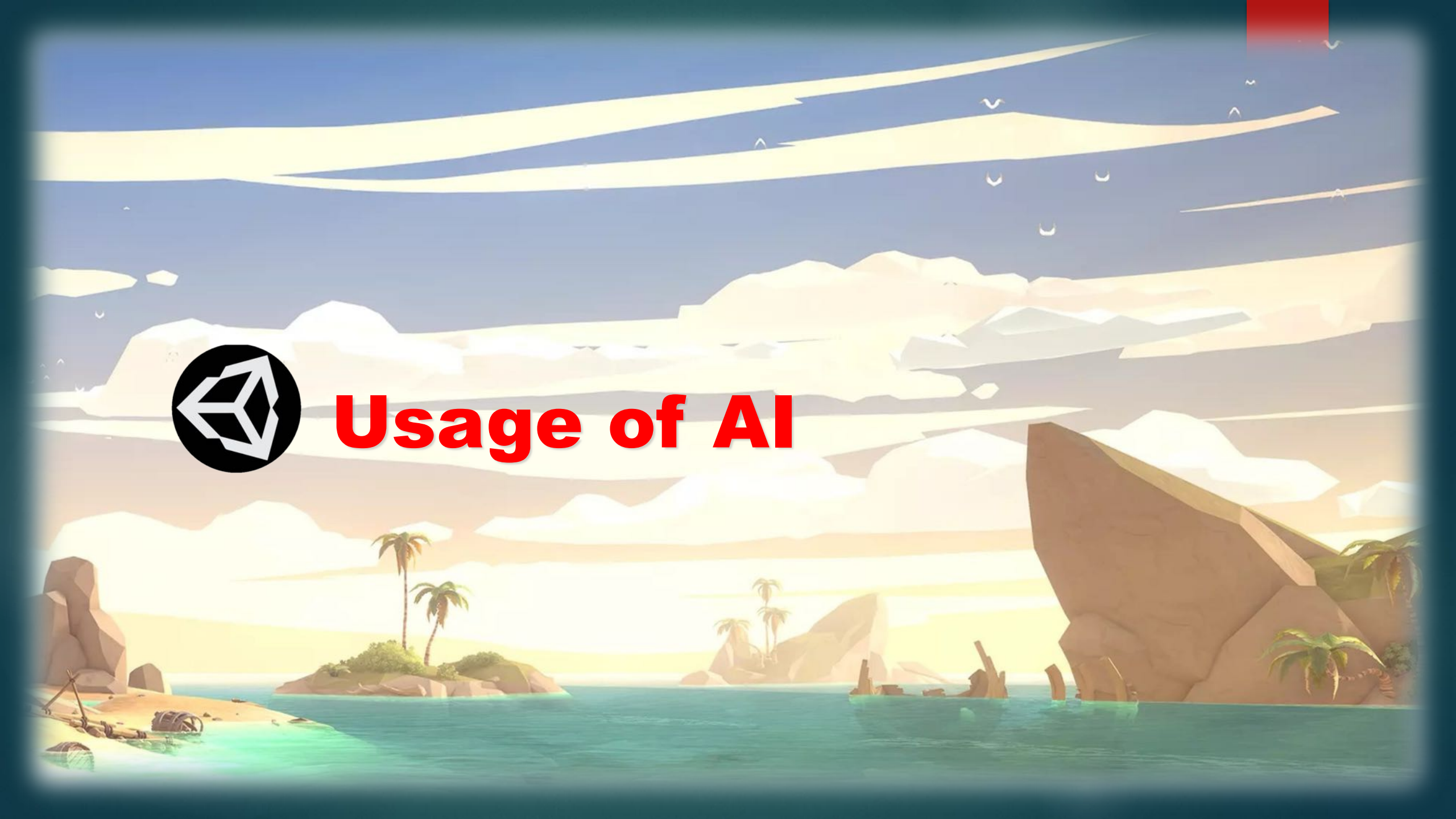
- allows a client to call a remote function or method located on the server to perform these operations.

Sandboxing:

- a security mechanism used to restrict the access of code or programs to the system at runtime.

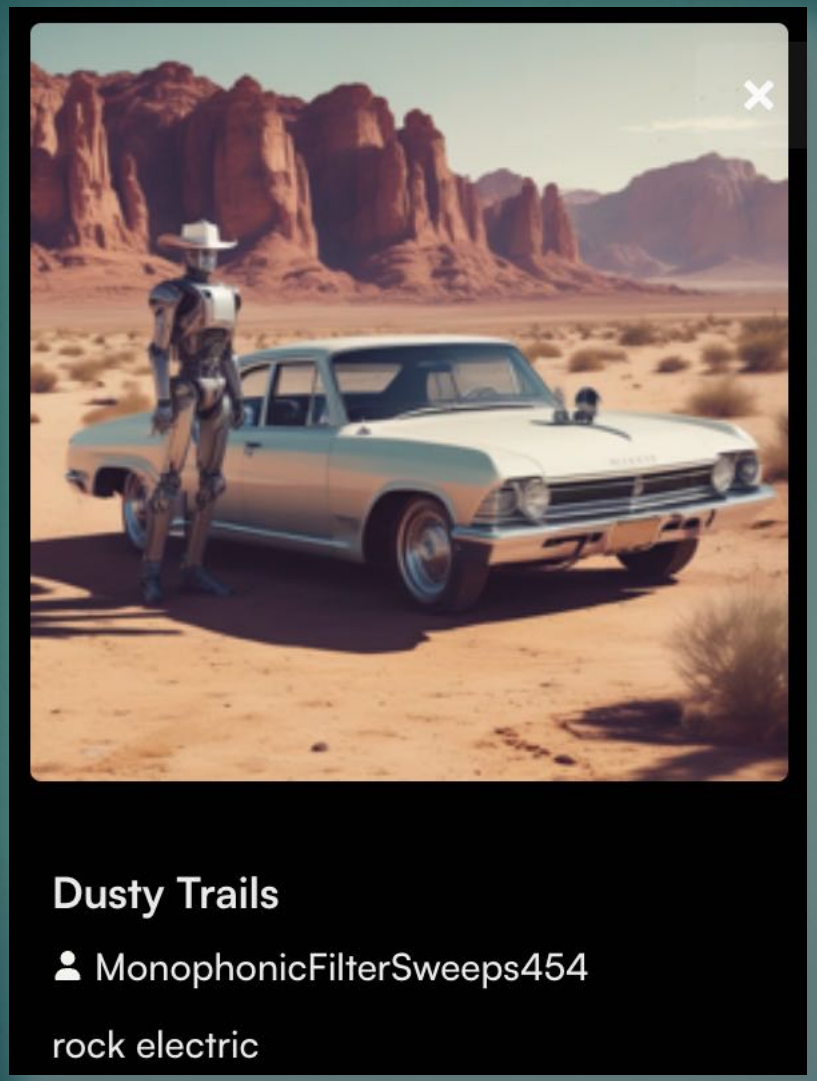
LZ4 algorithm :

- lossless data compression algorithm.


Usage of AI

Usage of AI – suno.ai

We create music for our game by using suno.ai.



[Verse]
Dusty roads and blazing sun
Robot rides with wheels of steel
Metal heart and circuits run
In the desert where the wild feels real

[Verse 2]
Sparks fly as engines roar
Favorite car a beast untamed
Through sand and dust they explore
Cowboy bot no need for fame

[Chorus]
Robot cowboy in the desert fight
Metal justice burning bright
Favorite car under the moonlight
Together strong a fearless sight

Usage of AI – In game Ai assistant

- Training tutor identity
- By Integrating large language models
- Pre-established knowledge of AI
- Customizing AI's moods and personalities
- Customized language scenarios for AI