**C语 言 项 目 报 告**

**项目名称：一个一无所有者的归途**

**项目成员：孙英凯 刘孙麟 齐思贤**

**填写日期：2021.7.13**

1. **摘要（Abstract）**

作为致敬Dark Souls III的作品，虽因时间和引擎源文件损坏等不可抗力因素所限未能填充内容，丰富体量，但本身游戏的开发秉持搭建框架的思想，基本实现了大体量内容填充下仍可有序运行，同时填充过程无需大量代码编写，代码模块化较为合理，各模块独立性较强。

1. **问题描述（Problem Statement）**

背景：国产游戏业时至今日终于厚积薄发，数款宣称3A级作品正在开发中，如Game Science的ACT游戏黑神话：悟空、飞燕群岛的FPS游戏光明记忆：无限。顺应这股浪潮，以兼容性好易上手的Unity引擎为工具，开发一款小体量游戏以致敬游戏界经典——宫崎英高的Dark Souls III意义：在致敬经典的同时锻炼个人的项目组织、代码编写、程序设计能力。

1. **组内分工（Group Division）**

糖化齐思贤：Quest&Dialogue————————————————20%糖化刘孙麟: State&Inventroy————————————————25%糖化孙英凯: 模型构建、Character、UI、Scene————————55%

1. **分析（Analysis）**

所谓致敬经典，不单单指纯粹的复刻，而是希望以崭新的面貌和内容，表现出以经典游戏精神内核为原点的艺术内涵，再加以延伸和演绎，在完成经典内核传承的同时进行艺术再创作，从而达到新生代与老一代经典思想的交融、传承与发展因此这款游戏需要具备游戏性与艺术表现力两方面的内容

1. 游戏性
   1. 富有挑战的关卡与敌人
   2. 多种策略构筑的高度可重玩性
   3. 挑战成功的反馈机制
2. 艺术表现
   1. 足够引人入胜的剧情
   2. 具有特色呼应主题的艺术风格

1. **设计（Design）**

五大模块：

* Input：涉及输入操作的功能——包括移动跳跃攻击和界面点击交互等
* State：状态相关功能——包括显示血蓝耐力经验值等级信息，敌人血量信息，同时涉及数据信息调用交互，调用数据前的处理，如根据武器伤害倍率和暴击率等数据计算实际伤害值等。为增加可玩性，游戏中大部分数据采用波动式，即设计最大值与最小值，core damage从范围中随机的方法。
* APC：涉及敌人对玩家的反馈，包括本身的四种状态——Guard、Patrol、Chase、Dead以及在攻击中的连击实现，投技实现、闪避实现、对峙实现、以及架势值为核心的受击反馈与硬直反馈。
* Inventory：基本背包UI的搭建、背包数据与相应物品数据的存储、捡拾世界中掉落物品、使用物品并达到相应效果。
* Quest&Dialogue: 其主体包括三个部分：生成器、对话框、触发器。整体上实现的功能是：主角作为信息接收者，到可互动单元附近时触发对话、任务故事、剧情等，其中对话双方既可以是主角与非玩家人物（NPC），又可以是NPC与NPC，以此丰富互动环节和故事内容，提高玩家游戏时的体验。

1. **实施（Implementation）**
2. Input
   1. Movement——共涉及9个脚本

由Preparation、Basis、Controller三部分组成，经多方参考学习，最终编写出一套实用性比较好的人物移动逻辑。主要涉及各种情况如移动速度，地面阻尼，滑落角度等等的参数，地面，悬崖等等检测与修正，以及对player参数的实时调控。

* + 1. Movement Basis——最底层的移动相关脚本，不必修改调整，涉及大量顶层脚本所用方法参数，代码量1024

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using SYK.Movement.Preparation;

namespace SYK.Movement.Basis

{

public class MovementBasis : MonoBehaviour

{

#region EDITOR FIELDS

[Header("Speed Limiters")]

[SerializeField]

private float \_maxLateralSpeed = 10.0f;

[SerializeField]

private float \_maxRiseSpeed = 20.0f;

[SerializeField]

private float \_maxFallSpeed = 20.0f;

[Header("Gravity")]

[SerializeField]

private bool \_useGravity = true;

[SerializeField]

private Vector3 \_gravity = new Vector3(0.0f, -30.0f, 0.0f);

[Header("Slopes")]

[Tooltip("玩家是否会从斜面上滑落" )]

[SerializeField]

private bool \_slideOnSteepSlope = true;

[Tooltip("玩家不会滑落的最大角度" )]

[SerializeField]

private float \_slopeLimit = 70.0f;

[Tooltip("玩家滑落时受到的重力倍数" )]

[SerializeField]

private float \_slideGravityMultiplier = 1.0f;

#endregion

#region FIELDS

//储存碰撞结果的缓存区

private static readonly Collider[] OverlappedColliders = new Collider[8];

private Coroutine \_lateFixedUpdateCoroutine;

private Vector3 \_normal;

private float \_referenceCastDistance;

private bool \_forceUnground;

private float \_forceUngroundTimer;

private bool \_performGroundDetection = true;

private Vector3 \_savedVelocity;

private Vector3 \_savedAngularVelocity;

#endregion

#region PROPERTIES

public float maxLateralSpeed

{

get { return \_maxLateralSpeed; }

set { \_maxLateralSpeed = Mathf.Max(0.0f, value); }

}

public float maxRiseSpeed

{

get { return \_maxRiseSpeed; }

set { \_maxRiseSpeed = Mathf.Max(0.0f, value); }

}

public float maxFallSpeed

{

get { return \_maxFallSpeed; }

set { \_maxFallSpeed = Mathf.Max(0.0f, value); }

}

public bool useGravity

{

get { return \_useGravity; }

set { \_useGravity = value; }

}

public Vector3 gravity

{

get { return \_gravity; }

set { \_gravity = value; }

}

public bool slideOnSteepSlope

{

get { return \_slideOnSteepSlope; }

set { \_slideOnSteepSlope = value; }

}

public float slopeLimit

{

get { return \_slopeLimit; }

set { \_slopeLimit = Mathf.Clamp(value, 0.0f, 89.0f); }

}

public float slideGravityMultiplier

{

get { return \_slideGravityMultiplier; }

set { \_slideGravityMultiplier = Mathf.Max(1.0f, value); }

}

public CapsuleCollider capsuleCollider

{

get { return groundDetection.capsuleCollider; }

}

private BasicGroundDetection groundDetection { get; set; }

//地面检测中射线与地面碰撞点

public Vector3 groundPoint

{

get { return groundDetection.groundPoint; }

}

public Vector3 groundNormal

{

get { return groundDetection.groundNormal; }

}

public Vector3 surfaceNormal

{

get { return groundDetection.surfaceNormal; }

}

public float groundDistance

{

get { return groundDetection.groundDistance; }

}

public Collider groundCollider

{

get { return groundDetection.groundCollider; }

}

public Rigidbody groundRigidbody

{

get { return groundDetection.groundRigidbody; }

}

public bool isGrounded

{

get { return groundDetection.isOnGround && groundDetection.isValidGround; }

}

//上一个物理步中人物的isGrounded值

public bool wasGrounded

{

get

{

return groundDetection.prevGroundHit.isOnGround && groundDetection.prevGroundHit.isValidGround;

}

}

//人物是否在任意平面上

public bool isOnGround

{

get { return groundDetection.isOnGround; }

}

//上一个物理步人物isOnGround值

public bool wasOnGround

{

get { return groundDetection.prevGroundHit.isOnGround; }

}

public bool isValidGround

{

get { return groundDetection.isValidGround; }

}

public bool isOnPlatform { get; private set; }

public bool isOnLedgeSolidSide

{

get { return groundDetection.isOnLedgeSolidSide; }

}

public bool isOnLedgeEmptySide

{

get { return groundDetection.isOnLedgeEmptySide; }

}

public float ledgeDistance

{

get { return groundDetection.ledgeDistance; }

}

public bool isOnStep

{

get { return groundDetection.isOnStep; }

}

public float stepHeight

{

get { return groundDetection.stepHeight; }

}

public bool isOnSlope

{

get { return groundDetection.isOnSlope; }

}

public float groundAngle

{

get { return groundDetection.groundAngle; }

}

public bool isValidSlope

{

get { return !slideOnSteepSlope || groundAngle < slopeLimit; }

}

public bool isSliding { get; private set; }

public Vector3 platformVelocity { get; private set; }

public Vector3 platformAngularVelocity { get; private set; }

public bool platformUpdatesRotation { get; set; }

public Vector3 velocity

{

get { return cachedRigidbody.velocity - platformVelocity; }

set { cachedRigidbody.velocity = value + platformVelocity; }

}

public float forwardSpeed

{

get { return Vector3.Dot(velocity, transform.forward); }

}

public Quaternion rotation

{

get { return cachedRigidbody.rotation; }

set { cachedRigidbody.MoveRotation(value); }

}

public GroundHit groundHit

{

get { return groundDetection.groundHit; }

}

public GroundHit prevGroundHit

{

get { return groundDetection.prevGroundHit; }

}

public LayerMask groundMask

{

get { return groundDetection.groundMask; }

}

public LayerMask overlapMask

{

get { return groundDetection.overlapMask; }

}

public QueryTriggerInteraction triggerInteraction

{

get { return groundDetection.triggerInteraction; }

}

public Rigidbody cachedRigidbody { get; private set; }

#endregion

#region METHODS

public void Pause(bool pause, bool restoreVelocity = true)

{

if (pause)

{

// Save rigidbody state, and make it kinematic

\_savedVelocity = cachedRigidbody.velocity;

\_savedAngularVelocity = cachedRigidbody.angularVelocity;

cachedRigidbody.isKinematic = true;

}

else

{

// Un-pause and restore saved rigidbody state (if desired)

cachedRigidbody.isKinematic = false;

if (restoreVelocity)

{

cachedRigidbody.AddForce(\_savedVelocity, ForceMode.VelocityChange);

cachedRigidbody.AddTorque(\_savedAngularVelocity, ForceMode.VelocityChange);

}

else

{

// If velocities should not be restored, zero out

var zero = Vector3.zero;

cachedRigidbody.AddForce(zero, ForceMode.VelocityChange);

cachedRigidbody.AddTorque(zero, ForceMode.VelocityChange);

}

cachedRigidbody.WakeUp();

}

}

public void SetCapsuleDimensions(Vector3 capsuleCenter, float capsuleRadius, float capsuleHeight)

{

capsuleCollider.center = capsuleCenter;

capsuleCollider.radius = capsuleRadius;

capsuleCollider.height = Mathf.Max(capsuleRadius \* 0.5f, capsuleHeight);

}

public void SetCapsuleDimensions(float capsuleRadius, float capsuleHeight)

{

capsuleCollider.center = new Vector3(0.0f, capsuleHeight \* 0.5f, 0.0f);

capsuleCollider.radius = capsuleRadius;

capsuleCollider.height = Mathf.Max(capsuleRadius \* 0.5f, capsuleHeight);

}

public void SetCapsuleHeight(float capsuleHeight)

{

capsuleHeight = Mathf.Max(capsuleCollider.radius \* 2.0f, capsuleHeight);

capsuleCollider.center = new Vector3(0.0f, capsuleHeight \* 0.5f, 0.0f);

capsuleCollider.height = capsuleHeight;

}

private void OverlapCapsule(Vector3 bottom, Vector3 top, float radius, out int overlapCount,

LayerMask overlappingMask, QueryTriggerInteraction queryTriggerInteraction)

{

var colliderCount = Physics.OverlapCapsuleNonAlloc(bottom, top, radius, OverlappedColliders,

overlappingMask, queryTriggerInteraction);

overlapCount = colliderCount;

for (var i = 0; i < colliderCount; i++)

{

var overlappedCollider = OverlappedColliders[i];

if (overlappedCollider != null && overlappedCollider != capsuleCollider)

continue;

if (i < --overlapCount)

OverlappedColliders[i] = OverlappedColliders[overlapCount];

}

}

public Collider[] OverlapCapsule(Vector3 position, Quaternion rotation, out int overlapCount,

LayerMask overlapMask, QueryTriggerInteraction queryTriggerInteraction = QueryTriggerInteraction.Ignore)

{

var center = capsuleCollider.center;

var radius = capsuleCollider.radius;

var height = capsuleCollider.height \* 0.5f - radius;

var topSphereCenter = center + Vector3.up \* height;

var bottomSphereCenter = center - Vector3.up \* height;

var top = position + rotation \* topSphereCenter;

var bottom = position + rotation \* bottomSphereCenter;

var colliderCount = Physics.OverlapCapsuleNonAlloc(bottom, top, radius, OverlappedColliders, overlapMask,

queryTriggerInteraction);

overlapCount = colliderCount;

for (var i = 0; i < colliderCount; i++)

{

var overlappedCollider = OverlappedColliders[i];

if (overlappedCollider != null && overlappedCollider != capsuleCollider)

continue;

if (i < --overlapCount)

OverlappedColliders[i] = OverlappedColliders[overlapCount];

}

return OverlappedColliders;

}

public bool ClearanceCheck(float clearanceHeight)

{

const float kTolerance = 0.01f;

var radius = Mathf.Max(kTolerance, capsuleCollider.radius - kTolerance);

var height = Mathf.Max(radius \* 2.0f + kTolerance, clearanceHeight - kTolerance);

var halfHeight = height \* 0.5f;

var center = new Vector3(0.0f, halfHeight, 0.0f);

var p = transform.position;

var q = transform.rotation;

var up = q \* Vector3.up;

var localBottom = center - up \* Mathf.Max(0.0f, halfHeight - kTolerance) + up \* radius;

var localTop = center + up \* halfHeight - up \* radius;

var bottom = p + q \* localBottom;

var top = p + q \* localTop;

int overlapCount;

OverlapCapsule(bottom, top, radius, out overlapCount, overlapMask, triggerInteraction);

return overlapCount == 0;

}

private void OverlapRecovery(ref Vector3 probingPosition, Quaternion probingRotation)

{

int overlapCount;

var overlappedColliders = groundDetection.OverlapCapsule(probingPosition, probingRotation, out overlapCount);

for (var i = 0; i < overlapCount; i++)

{

var overlappedCollider = overlappedColliders[i];

var overlappedColliderRigidbody = overlappedCollider.attachedRigidbody;

if (overlappedColliderRigidbody != null)

continue;

var overlappedColliderTransform = overlappedCollider.transform;

float distance;

Vector3 direction;

if (!Physics.ComputePenetration(capsuleCollider, probingPosition, probingRotation, overlappedCollider,

overlappedColliderTransform.position, overlappedColliderTransform.rotation, out direction,

out distance))

continue;

probingPosition += direction \* distance;

}

}

public bool ComputeGroundHit(Vector3 probingPosition, Quaternion probingRotation, out GroundHit groundHitInfo,

float scanDistance = Mathf.Infinity)

{

groundHitInfo = new GroundHit();

return groundDetection.ComputeGroundHit(probingPosition, probingRotation, ref groundHitInfo, scanDistance);

}

public bool ComputeGroundHit(out GroundHit hitInfo, float scanDistance = Mathf.Infinity)

{

var p = transform.position;

var q = transform.rotation;

return ComputeGroundHit(p, q, out hitInfo, scanDistance);

}

public void Rotate(Vector3 direction, float angularSpeed, bool onlyLateral = true)

{

if (onlyLateral)

direction = Vector3.ProjectOnPlane(direction, transform.up);

if (direction.sqrMagnitude < 0.0001f)

return;

var targetRotation = Quaternion.LookRotation(direction, transform.up);

var newRotation = Quaternion.Slerp(cachedRigidbody.rotation, targetRotation,

angularSpeed \* Mathf.Deg2Rad \* Time.deltaTime);

cachedRigidbody.MoveRotation(newRotation);

}

public void ApplyDrag(float drag, bool onlyLateral = true)

{

var up = transform.up;

var v = onlyLateral ? Vector3.ProjectOnPlane(velocity, up) : velocity;

var d = -drag \* v.magnitude \* v;

cachedRigidbody.AddForce(d, ForceMode.Acceleration);

}

public void ApplyForce(Vector3 force, ForceMode forceMode = ForceMode.Force)

{

cachedRigidbody.AddForce(force, forceMode);

}

public void ApplyVerticalImpulse(float impulse)

{

Vector3 up = transform.up;

cachedRigidbody.velocity = Vector3.ProjectOnPlane(cachedRigidbody.velocity, up) + up \* impulse;

}

public void ApplyImpulse(Vector3 impulse)

{

cachedRigidbody.velocity += impulse - Vector3.Project(cachedRigidbody.velocity, transform.up);

}

public void DisableGrounding(float time = 0.1f)

{

\_forceUnground = true;

\_forceUngroundTimer = time;

groundDetection.castDistance = 0.0f;

}

public void DisableGroundDetection()

{

\_performGroundDetection = false;

}

public void EnableGroundDetection()

{

\_performGroundDetection = true;

}

private void ResetGroundInfo()

{

groundDetection.ResetGroundInfo();

isSliding = false;

isOnPlatform = false;

platformVelocity = Vector3.zero;

platformAngularVelocity = Vector3.zero;

\_normal = transform.up;

}

private void DetectGround()

{

ResetGroundInfo();

if (\_performGroundDetection)

{

if (\_forceUnground || \_forceUngroundTimer > 0.0f)

{

\_forceUnground = false;

\_forceUngroundTimer -= Time.deltaTime;

}

else

{

groundDetection.DetectGround();

groundDetection.castDistance = isGrounded ? \_referenceCastDistance : 0.0f;

}

}

if (!isOnGround)

return;

var up = transform.up;

if (isValidGround)

\_normal = isOnLedgeSolidSide ? up : groundDetection.groundNormal;

else

{

\_normal = Vector3.Cross(Vector3.Cross(up, groundDetection.groundNormal), up).normalized;

}

var otherRigidbody = groundRigidbody;

if (otherRigidbody == null)

return;

if (otherRigidbody.isKinematic)

{

isOnPlatform = true;

platformVelocity = otherRigidbody.GetPointVelocity(groundPoint);

platformAngularVelocity = Vector3.Project(otherRigidbody.angularVelocity, up);

}

else

{

\_normal = up;

}

}

private void PreventGroundPenetration()

{

if (isOnGround)

return;

var v = velocity;

var speed = v.magnitude;

var direction = speed > 0.0f ? v / speed : Vector3.zero;

var distance = speed \* Time.deltaTime;

RaycastHit hitInfo;

if (!groundDetection.FindGround(direction, out hitInfo, distance))

return;

var remainingDistance = distance - hitInfo.distance;

if (remainingDistance <= 0.0f)

return;

var velocityToGround = direction \* (hitInfo.distance / Time.deltaTime);

var up = transform.up;

var remainingLateralVelocity = Vector3.ProjectOnPlane(v - velocityToGround, up);

remainingLateralVelocity = Preparation\_MathLibrary.GetTangent(remainingLateralVelocity, hitInfo.normal, up) \* remainingLateralVelocity.magnitude;

var newVelocity = velocityToGround + remainingLateralVelocity;

cachedRigidbody.velocity = newVelocity;

groundDetection.castDistance = \_referenceCastDistance;

}

private void ApplyMovement(Vector3 desiredVelocity, float maxDesiredSpeed, bool onlyLateral)

{

var up = transform.up;

if (onlyLateral)

desiredVelocity = Vector3.ProjectOnPlane(desiredVelocity, up);

if (isGrounded)

{

if (!slideOnSteepSlope || groundAngle < slopeLimit)

{

desiredVelocity = Preparation\_MathLibrary.GetTangent(desiredVelocity, \_normal, up) \* Mathf.Min(desiredVelocity.magnitude, maxDesiredSpeed);

velocity += desiredVelocity - velocity;

}

else

{

isSliding = true;

velocity += gravity \* (slideGravityMultiplier \* Time.deltaTime);

}

}

else

{

if (isOnGround)

{

var isBraking = desiredVelocity.sqrMagnitude < 0.000001f;

if (isBraking && onlyLateral)

{

desiredVelocity = velocity;

}

else

{

if (Vector3.Dot(desiredVelocity, \_normal) <= 0.0f)

{

var speedLimit = Mathf.Min(desiredVelocity.magnitude, maxDesiredSpeed);

var lateralVelocity = Vector3.ProjectOnPlane(velocity, up);

desiredVelocity = Vector3.ProjectOnPlane(desiredVelocity, \_normal) +

Vector3.Project(lateralVelocity, \_normal);

desiredVelocity = Vector3.ClampMagnitude(desiredVelocity, speedLimit);

}

}

}

velocity += onlyLateral

? Vector3.ProjectOnPlane(desiredVelocity - velocity, up)

: desiredVelocity - velocity;

if (useGravity)

velocity += gravity \* Time.deltaTime;

}

if (!isOnStep)

return;

var dot = Vector3.Dot(velocity, groundPoint - transform.position);

if (dot <= 0.0f)

return;

var angle = Mathf.Abs(90.0f - Vector3.Angle(up, velocity));

if (angle < 75.0f)

return;

var factor = Mathf.Lerp(1.0f, 0.0f, Mathf.InverseLerp(75.0f, 90.0f, angle));

factor = factor \* (2.0f - factor);

velocity \*= factor;

}

private void ApplyGroundMovement(Vector3 desiredVelocity, float maxDesiredSpeed, float acceleration,

float deceleration, float friction, float brakingFriction)

{

var up = transform.up;

var deltaTime = Time.deltaTime;

if (!slideOnSteepSlope || groundAngle < slopeLimit)

{

var v = wasGrounded ? velocity : Vector3.ProjectOnPlane(velocity, up);

var desiredSpeed = desiredVelocity.magnitude;

var speedLimit = desiredSpeed > 0.0f ? Mathf.Min(desiredSpeed, maxDesiredSpeed) : maxDesiredSpeed;

var desiredDirection = Preparation\_MathLibrary.GetTangent(desiredVelocity, \_normal, up);

var desiredAcceleration = desiredDirection \* (acceleration \* deltaTime);

if (desiredAcceleration.isZero() || v.isExceeding(speedLimit))

{

v = Preparation\_MathLibrary.GetTangent(v, \_normal, up) \* v.magnitude;

v = v \* Mathf.Clamp01(1f - brakingFriction \* deltaTime);

v = Vector3.MoveTowards(v, desiredVelocity, deceleration \* deltaTime);

}

else

{

v = Preparation\_MathLibrary.GetTangent(v, \_normal, up) \* v.magnitude;

v = v - (v - desiredDirection \* v.magnitude) \* Mathf.Min(friction \* deltaTime, 1.0f);

v = Vector3.ClampMagnitude(v + desiredAcceleration, speedLimit);

}

velocity += v - velocity;

}

else

{

isSliding = true;

velocity += gravity \* (slideGravityMultiplier \* Time.deltaTime);

}

}

private void ApplyAirMovement(Vector3 desiredVelocity, float maxDesiredSpeed, float acceleration,

float deceleration, float friction, float brakingFriction, bool onlyLateral = true)

{

var up = transform.up;

var v = onlyLateral ? Vector3.ProjectOnPlane(velocity, up) : velocity;

if (onlyLateral)

desiredVelocity = Vector3.ProjectOnPlane(desiredVelocity, up);

if (isOnGround)

{

if (Vector3.Dot(desiredVelocity, \_normal) <= 0.0f)

{

var maxLength = Mathf.Min(desiredVelocity.magnitude, maxDesiredSpeed);

var lateralVelocity = Vector3.ProjectOnPlane(velocity, up);

desiredVelocity = Vector3.ProjectOnPlane(desiredVelocity, \_normal) +

Vector3.Project(lateralVelocity, \_normal);

desiredVelocity = Vector3.ClampMagnitude(desiredVelocity, maxLength);

}

}

var desiredSpeed = desiredVelocity.magnitude;

var speedLimit = desiredSpeed > 0.0f ? Mathf.Min(desiredSpeed, maxDesiredSpeed) : maxDesiredSpeed;

var deltaTime = Time.deltaTime;

var desiredDirection = desiredSpeed > 0.0f ? desiredVelocity / desiredSpeed : Vector3.zero;

var desiredAcceleration = desiredDirection \* (acceleration \* deltaTime);

if (desiredAcceleration.isZero() || v.isExceeding(speedLimit))

{

if (isOnGround && onlyLateral)

{

}

else

{

v = v \* Mathf.Clamp01(1f - brakingFriction \* deltaTime);

v = Vector3.MoveTowards(v, desiredVelocity, deceleration \* deltaTime);

}

}

else

{

v = v - (v - desiredDirection \* v.magnitude) \* Mathf.Min(friction \* deltaTime, 1.0f);

v = Vector3.ClampMagnitude(v + desiredAcceleration, speedLimit);

}

if (onlyLateral)

velocity += Vector3.ProjectOnPlane(v - velocity, up);

else

velocity += v - velocity;

if (useGravity)

velocity += gravity \* Time.deltaTime;

}

private void ApplyMovement(Vector3 desiredVelocity, float maxDesiredSpeed, float acceleration,

float deceleration, float friction, float brakingFriction, bool onlyLateral)

{

if (isGrounded)

{

ApplyGroundMovement(desiredVelocity, maxDesiredSpeed, acceleration, deceleration, friction,

brakingFriction);

}

else

{

ApplyAirMovement(desiredVelocity, maxDesiredSpeed, acceleration, deceleration, friction,

brakingFriction, onlyLateral);

}

if (!isOnStep)

return;

var dot = Vector3.Dot(velocity, groundPoint - transform.position);

if (dot <= 0.0f)

return;

var angle = Mathf.Abs(90.0f - Vector3.Angle(transform.up, velocity));

if (angle < 75.0f)

return;

var factor = Mathf.Lerp(1.0f, 0.0f, Mathf.InverseLerp(75.0f, 90.0f, angle));

factor = factor \* (2.0f - factor);

velocity \*= factor;

}

private void LimitLateralVelocity()

{

var lateralVelocity = Vector3.ProjectOnPlane(velocity, transform.up);

if (lateralVelocity.sqrMagnitude > maxLateralSpeed \* maxLateralSpeed)

cachedRigidbody.velocity += lateralVelocity.normalized \* maxLateralSpeed - lateralVelocity;

}

private void LimitVerticalVelocity()

{

if (isGrounded)

return;

var up = transform.up;

var verticalSpeed = Vector3.Dot(velocity, up);

if (verticalSpeed < -maxFallSpeed)

cachedRigidbody.velocity += up \* (-maxFallSpeed - verticalSpeed);

if (verticalSpeed > maxRiseSpeed)

cachedRigidbody.velocity += up \* (maxRiseSpeed - verticalSpeed);

}

public void Move(Vector3 desiredVelocity, float maxDesiredSpeed, bool onlyLateral = true)

{

DetectGround();

ApplyMovement(desiredVelocity, maxDesiredSpeed, onlyLateral);

LimitLateralVelocity();

LimitVerticalVelocity();

PreventGroundPenetration();

}

public void Move(Vector3 desiredVelocity, float maxDesiredSpeed, float acceleration, float deceleration,

float friction, float brakingFriction, bool onlyLateral = true)

{

DetectGround();

ApplyMovement(desiredVelocity, maxDesiredSpeed, acceleration, deceleration, friction, brakingFriction, onlyLateral);

LimitLateralVelocity();

LimitVerticalVelocity();

PreventGroundPenetration();

}

private void SnapToPlatform(ref Vector3 probingPosition, ref Quaternion probingRotation)

{

if (\_performGroundDetection == false || \_forceUnground || \_forceUngroundTimer > 0.0f)

return;

GroundHit hitInfo;

if (!ComputeGroundHit(probingPosition, probingRotation, out hitInfo, groundDetection.castDistance))

return;

var otherRigidbody = hitInfo.groundRigidbody;

if (otherRigidbody == null || !otherRigidbody.isKinematic)

return;

var up = probingRotation \* Vector3.up;

var groundedPosition = probingPosition - up \* hitInfo.groundDistance;

var pointVelocity = otherRigidbody.GetPointVelocity(groundedPosition);

cachedRigidbody.velocity = velocity + pointVelocity;

var deltaVelocity = pointVelocity - platformVelocity;

groundedPosition += Vector3.ProjectOnPlane(deltaVelocity, up) \* Time.deltaTime;

if (hitInfo.isOnLedgeSolidSide)

groundedPosition = Preparation\_MathLibrary.ProjectPointOnPlane(groundedPosition, hitInfo.groundPoint, up);

probingPosition = groundedPosition;

if (platformUpdatesRotation == false || otherRigidbody.angularVelocity == Vector3.zero)

return;

var yaw = Vector3.Project(otherRigidbody.angularVelocity, up);

var yawRotation = Quaternion.Euler(yaw \* (Mathf.Rad2Deg \* Time.deltaTime));

probingRotation \*= yawRotation;

}

private IEnumerator LateFixedUpdate()

{

var waitTime = new WaitForFixedUpdate();

while (true)

{

yield return waitTime;

var p = transform.position;

var q = transform.rotation;

OverlapRecovery(ref p, q);

if (isOnGround && isOnPlatform)

SnapToPlatform(ref p, ref q);

cachedRigidbody.MovePosition(p);

cachedRigidbody.MoveRotation(q);

}

}

#endregion

#region MONOBEHAVIOUR

public void OnValidate()

{

maxLateralSpeed = \_maxLateralSpeed;

maxRiseSpeed = \_maxRiseSpeed;

maxFallSpeed = \_maxFallSpeed;

useGravity = \_useGravity;

gravity = \_gravity;

slideOnSteepSlope = \_slideOnSteepSlope;

slopeLimit = \_slopeLimit;

slideGravityMultiplier = \_slideGravityMultiplier;

}

public void Awake()

{

groundDetection = GetComponent<BasicGroundDetection>();

if (groundDetection == null)

{

Debug.LogError(

string.Format(

"CharacterMovement: No 'GroundDetection' found for '{0}' game object.\n" +

"Please add a 'GroundDetection' component to '{0}' game object",

name));

return;

}

\_referenceCastDistance = groundDetection.castDistance;

cachedRigidbody = GetComponent<Rigidbody>();

if (cachedRigidbody == null)

{

Debug.LogError(

string.Format(

"CharacterMovement: No 'Rigidbody' found for '{0}' game object.\n" +

"Please add a 'Rigidbody' component to '{0}' game object",

name));

return;

}

cachedRigidbody.useGravity = false;

cachedRigidbody.isKinematic = false;

cachedRigidbody.freezeRotation = true;

var aCollider = GetComponent<Collider>();

if (aCollider == null)

return;

var physicMaterial = aCollider.sharedMaterial;

if (physicMaterial != null)

return;

physicMaterial = new PhysicMaterial("Frictionless")

{

dynamicFriction = 0.0f,

staticFriction = 0.0f,

bounciness = 0.0f,

frictionCombine = PhysicMaterialCombine.Multiply,

bounceCombine = PhysicMaterialCombine.Average

};

aCollider.material = physicMaterial;

Debug.LogWarning(

string.Format(

"CharacterMovement: No 'PhysicMaterial' found for '{0}'s Collider, a frictionless one has been created and assigned.\n" +

"Please add a Frictionless 'PhysicMaterial' to '{0}' game object.",

name));

}

public void OnEnable()

{

if (\_lateFixedUpdateCoroutine != null)

StopCoroutine(\_lateFixedUpdateCoroutine);

\_lateFixedUpdateCoroutine = StartCoroutine(LateFixedUpdate());

}

public void OnDisable()

{

if (\_lateFixedUpdateCoroutine != null)

StopCoroutine(\_lateFixedUpdateCoroutine);

}

#endregion

}

}

* + 1. Preparation\_Extensions——Preparation是作为对底层脚本常调用方法的统一补充编写，此脚本为拓展方法，代码量123

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Movement.Preparation

{

public static class Preparation\_Extensions

{

//float值为零检查

public static bool isZero(this float value)

{

const float tolerance = 0.0000000001f;

return Mathf.Abs(value) < tolerance;

}

//清除yz轴分量

public static Vector3 onlyX(this Vector3 vector3)

{

vector3.y = 0.0f;

vector3.z = 0.0f;

return vector3;

}

//清除xz轴分量

public static Vector3 onlyY(this Vector3 vector3)

{

vector3.x = 0.0f;

vector3.z = 0.0f;

return vector3;

}

//清除xy轴分量

public static Vector3 onlyZ(this Vector3 vector3)

{

vector3.x = 0.0f;

vector3.y = 0.0f;

return vector3;

}

//清除竖直分量

public static Vector3 onlyXZ(this Vector3 vector3)

{

vector3.y = 0.0f;

return vector3;

}

//Vector3值为零检查

public static bool isZero(this Vector3 vector3)

{

return vector3.sqrMagnitude < 9.99999999999999E-11;

}

//Vector3精度检查

public static bool isExceeding(this Vector3 vector3, float magnitude)

{

// Allow 1% error tolerance, to account for numeric imprecision.

const float errorTolerance = 1.01f;

return vector3.sqrMagnitude > magnitude \* magnitude \* errorTolerance;

}

//复制向量并单位化

//同时返回长度

//若极小则返回零向量

public static Vector3 normalized(this Vector3 vector3, out float magnitude)

{

magnitude = vector3.magnitude;

if (magnitude > 9.99999999999999E-06)

return vector3 / magnitude;

magnitude = 0.0f;

return Vector3.zero;

}

//复制向量并单位化

//若本身长度小于1则直接返回

public static Vector3 clamped(this Vector3 vector3, out float magnitude)

{

magnitude = vector3.magnitude;

return magnitude > 1.0f ? vector3 / magnitude : vector3;

}

//将模大于指定值的Vector3缩放到指定值

//小于不变

public static Vector3 clampedTo(this Vector3 vector3, float maxLength)

{

if (vector3.sqrMagnitude > maxLength \* (double) maxLength)

return vector3.normalized \* maxLength;

return vector3;

}

//相对向量方向修正

public static Vector3 relativeTo(this Vector3 vector3, Transform target, bool onlyLateral = true)

{

var forward = target.forward;

if (onlyLateral)

forward = Vector3.ProjectOnPlane(forward, Vector3.up);

return Quaternion.LookRotation(forward) \* vector3;

}

}

}

* + 1. Preparation\_MathLibrary——Preparation是作为对底层脚本常调用方法的统一补充编写，此脚本为数学库，代码量30

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Movement.Preparation

{

public class Preparation\_MathLibrary

{

//返回调整方向使其与给定法线相切

//可定义相对向上方向

public static Vector3 GetTangent(Vector3 direction, Vector3 normal, Vector3 up)

{

var right = Vector3.Cross(direction, up).normalized;

var tangent = Vector3.Cross(normal, right);

return tangent.normalized;

}

//将点投影到面法线上

public static Vector3 ProjectPointOnPlane(Vector3 point, Vector3 planeOrigin, Vector3 planeNormal)

{

var toPoint = point - planeOrigin;

var toPointProjected = Vector3.Project(toPoint, planeNormal.normalized);

return point - toPointProjected;

}

}

}

* + 1. GroundHit——定义结构体，用于储存地面检测所得到的信息，是较之Unity自带射线检测碰撞储存信息更多更定制化的结构体，代码量95

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Movement.Basis

{

public struct GroundHit

{

#region FIELDS

private float \_ledgeDistance;

private float \_stepHeight;

#endregion

#region PROPERTIES

public bool isOnGround { get; set; }

public bool isValidGround { get; set; }

public bool isOnLedgeSolidSide { get; set; }

public bool isOnLedgeEmptySide { get; set; }

public float ledgeDistance

{

get { return \_ledgeDistance; }

set { \_ledgeDistance = Mathf.Max(0.0f, value); }

}

public bool isOnStep { get; set; }

public float stepHeight

{

get { return \_stepHeight; }

set { \_stepHeight = Mathf.Max(0.0f, value); }

}

public Vector3 groundPoint { get; set; }

public Vector3 groundNormal { get; set; }

public float groundDistance { get; private set; }

public Collider groundCollider { get; private set; }

public Rigidbody groundRigidbody { get; private set; }

public Vector3 surfaceNormal { get; set; }

#endregion

#region METHODS

public GroundHit(GroundHit other) : this()

{

isOnGround = other.isOnGround;

isValidGround = other.isValidGround;

isOnLedgeSolidSide = other.isOnLedgeSolidSide;

isOnLedgeEmptySide = other.isOnLedgeEmptySide;

ledgeDistance = other.ledgeDistance;

isOnStep = other.isOnStep;

stepHeight = other.stepHeight;

groundPoint = other.groundPoint;

groundNormal = other.groundNormal;

groundDistance = Mathf.Max(0.0f, other.groundDistance);

groundCollider = other.groundCollider;

groundRigidbody = other.groundRigidbody;

surfaceNormal = other.surfaceNormal;

}

public GroundHit(RaycastHit hitInfo) : this()

{

SetFrom(hitInfo);

}

public void SetFrom(RaycastHit hitInfo)

{

groundPoint = hitInfo.point;

groundNormal = hitInfo.normal;

groundDistance = Mathf.Max(0.0f, hitInfo.distance);

groundCollider = hitInfo.collider;

groundRigidbody = hitInfo.rigidbody;

}

#endregion

}

}

* + 1. BasicGroundDetection——基础地面检测方法，作为实际地面检测的父类，代码399

#if UNITY\_EDITOR

using UnityEditor;

#endif

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Movement.Basis

{

[RequireComponent(typeof(CapsuleCollider))]

public abstract class BasicGroundDetection : MonoBehaviour

{

#region EDITOR EXPOSED FIELDS

[SerializeField]

private LayerMask \_groundMask = 1;

[SerializeField]

private float \_groundLimit = 60.0f;

[SerializeField]

private float \_stepOffset = 0.5f;

[SerializeField]

private float \_ledgeOffset;

[SerializeField]

private float \_castDistance = 0.5f;

[SerializeField]

private QueryTriggerInteraction \_triggerInteraction = QueryTriggerInteraction.Ignore;

#endregion

#region FIELDS

private static readonly Collider[] OverlappedColliders = new Collider[16];

protected const float kBackstepDistance = 0.05f;

protected const float kMinCastDistance = 0.01f;

protected const float kMinLedgeDistance = 0.05f;

protected const float kMinStepOffset = 0.10f;

protected const float kHorizontalOffset = 0.001f;

private CapsuleCollider \_capsuleCollider;

protected GroundHit \_groundHitInfo;

private LayerMask \_overlapMask = -1;

private int \_ignoreRaycastLayer = 2;

private int \_cachedLayer;

#endregion

#region PROPERTIES

public LayerMask groundMask

{

get { return \_groundMask; }

set { \_groundMask = value; }

}

public float groundLimit

{

get { return \_groundLimit; }

set { \_groundLimit = Mathf.Clamp(value, 0.0f, 89.0f); }

}

public float stepOffset

{

get { return \_stepOffset; }

set { \_stepOffset = Mathf.Clamp(value, kMinStepOffset, capsuleCollider.radius); }

}

public float ledgeOffset

{

get { return \_ledgeOffset; }

set { \_ledgeOffset = Mathf.Clamp(value, 0.0f, capsuleCollider.radius); }

}

public float castDistance

{

get { return \_castDistance; }

set { \_castDistance = Mathf.Max(kMinCastDistance, value); }

}

public QueryTriggerInteraction triggerInteraction

{

get { return \_triggerInteraction; }

set { \_triggerInteraction = value; }

}

public CapsuleCollider capsuleCollider

{

get

{

if (\_capsuleCollider == null)

\_capsuleCollider = GetComponent<CapsuleCollider>();

return \_capsuleCollider;

}

}

public bool isOnGround

{

get { return \_groundHitInfo.isOnGround; }

}

public bool isValidGround

{

get { return \_groundHitInfo.isValidGround; }

}

public bool isOnLedgeSolidSide

{

get { return \_groundHitInfo.isOnLedgeSolidSide; }

}

public bool isOnLedgeEmptySide

{

get { return \_groundHitInfo.isOnLedgeEmptySide; }

}

public float ledgeDistance

{

get { return \_groundHitInfo.ledgeDistance; }

}

public bool isOnStep

{

get { return \_groundHitInfo.isOnStep; }

}

public float stepHeight

{

get { return \_groundHitInfo.stepHeight; }

}

public bool isOnSlope

{

get { return !Mathf.Approximately(groundAngle, 0.0f); }

}

public Vector3 groundPoint

{

get { return \_groundHitInfo.groundPoint; }

}

public Vector3 groundNormal

{

get { return \_groundHitInfo.groundNormal; }

}

public float groundDistance

{

get { return \_groundHitInfo.groundDistance; }

}

public Collider groundCollider

{

get { return \_groundHitInfo.groundCollider; }

}

public Rigidbody groundRigidbody

{

get { return \_groundHitInfo.groundRigidbody; }

}

public float groundAngle

{

get { return !isOnGround ? 0.0f : Vector3.Angle(surfaceNormal, transform.up); }

}

public Vector3 surfaceNormal

{

get { return \_groundHitInfo.surfaceNormal; }

}

public GroundHit groundHit

{

get { return \_groundHitInfo; }

}

public GroundHit prevGroundHit { get; private set; }

public LayerMask overlapMask

{

get { return \_overlapMask; }

set { \_overlapMask = value; }

}

#endregion

#region METHODS

protected virtual void InitializeOverlapMask()

{

var layer = gameObject.layer;

\_overlapMask = 0;

for (var i = 0; i < 32; i++)

{

if (!Physics.GetIgnoreLayerCollision(layer, i))

\_overlapMask |= 1 << i;

}

}

public Collider[] OverlapCapsule(Vector3 position, Quaternion rotation, out int overlapCount)

{

var center = capsuleCollider.center;

var radius = capsuleCollider.radius;

var height = capsuleCollider.height \* 0.5f - radius;

var topSphereCenter = center + Vector3.up \* height;

var bottomSphereCenter = center - Vector3.up \* height;

var top = position + rotation \* topSphereCenter;

var bottom = position + rotation \* bottomSphereCenter;

var colliderCount = Physics.OverlapCapsuleNonAlloc(bottom, top, radius, OverlappedColliders, \_overlapMask,

triggerInteraction);

overlapCount = colliderCount;

for (var i = 0; i < colliderCount; i++)

{

var overlappedCollider = OverlappedColliders[i];

if (overlappedCollider != null && overlappedCollider != capsuleCollider)

continue;

if (i < --overlapCount)

OverlappedColliders[i] = OverlappedColliders[overlapCount];

}

return OverlappedColliders;

}

protected bool Raycast(Vector3 origin, Vector3 direction, out RaycastHit hitInfo, float distance,

float backstepDistance = kBackstepDistance)

{

origin = origin - direction \* backstepDistance;

var hit = Physics.Raycast(origin, direction, out hitInfo, distance + backstepDistance, groundMask,

triggerInteraction);

if (hit)

hitInfo.distance = hitInfo.distance - backstepDistance;

return hit;

}

protected bool SphereCast(Vector3 origin, float radius, Vector3 direction, out RaycastHit hitInfo,

float distance, float backstepDistance = kBackstepDistance)

{

origin = origin - direction \* backstepDistance;

var hit = Physics.SphereCast(origin, radius, direction, out hitInfo, distance + backstepDistance,

groundMask, triggerInteraction);

if (hit)

hitInfo.distance = hitInfo.distance - backstepDistance;

return hit;

}

protected bool CapsuleCast(Vector3 bottom, Vector3 top, float radius, Vector3 direction, out RaycastHit hitInfo,

float distance, float backstepDistance = kBackstepDistance)

{

top = top - direction \* backstepDistance;

bottom = bottom - direction \* backstepDistance;

var hit = Physics.CapsuleCast(bottom, top, radius, direction, out hitInfo, distance + backstepDistance,

groundMask, triggerInteraction);

if (hit)

hitInfo.distance = hitInfo.distance - backstepDistance;

return hit;

}

public virtual bool SweepTest(Vector3 position, Quaternion rotation, Vector3 direction, out RaycastHit hitInfo,

float distance = Mathf.Infinity, float backstepDistance = kBackstepDistance)

{

var radius = capsuleCollider.radius;

var height = Mathf.Max(0.0f, capsuleCollider.height \* 0.5f - radius);

var bottomSphereCenter = capsuleCollider.center - Vector3.up \* height;

var topSphereCenter = capsuleCollider.center + Vector3.up \* height;

var bottom = position + rotation \* bottomSphereCenter;

var top = position + rotation \* topSphereCenter;

return CapsuleCast(bottom, top, radius, direction, out hitInfo, distance, backstepDistance);

}

protected virtual void DisableRaycastCollisions()

{

\_cachedLayer = gameObject.layer;

gameObject.layer = \_ignoreRaycastLayer;

}

protected virtual void EnableRaycastCollisions()

{

gameObject.layer = \_cachedLayer;

}

public virtual void ResetGroundInfo()

{

var up = transform.up;

prevGroundHit = new GroundHit(\_groundHitInfo);

\_groundHitInfo = new GroundHit

{

groundPoint = transform.position,

groundNormal = up,

surfaceNormal = up

};

}

public abstract bool ComputeGroundHit(Vector3 position, Quaternion rotation, ref GroundHit groundHitInfo,

float distance = Mathf.Infinity);

public void DetectGround()

{

DisableRaycastCollisions();

ComputeGroundHit(transform.position, transform.rotation, ref \_groundHitInfo, castDistance);

EnableRaycastCollisions();

}

public abstract bool FindGround(Vector3 direction, out RaycastHit hitInfo, float distance = Mathf.Infinity,

float backstepDistance = kBackstepDistance);

protected virtual void DrawGizmos()

{

#if UNITY\_EDITOR

if (!Application.isPlaying)

return;

if (!isOnGround)

return;

var color = new Color(0.0f, 1.0f, 0.0f, 0.25f);

if (!isValidGround)

color = new Color(0.0f, 0.0f, 1.0f, 0.25f);

Handles.color = color;

Handles.DrawSolidDisc(groundPoint, surfaceNormal, 0.1f);

Gizmos.color = isValidGround ? Color.green : Color.blue;

Gizmos.DrawRay(groundPoint, surfaceNormal);

if (groundNormal != surfaceNormal)

{

Gizmos.color = Color.yellow;

Gizmos.DrawRay(groundPoint, groundNormal);

}

if (!isOnStep)

return;

var stepPoint = groundPoint - transform.up \* stepHeight;

Gizmos.color = Color.black;

Gizmos.DrawLine(groundPoint, stepPoint);

Handles.color = new Color(0.0f, 0.0f, 0.0f, 0.25f);

Handles.DrawSolidDisc(stepPoint, transform.up, 0.1f);

#endif

}

#endregion

#region MONOBEHAVIOUR

protected virtual void OnValidate()

{

groundLimit = \_groundLimit;

stepOffset = \_stepOffset;

ledgeOffset = \_ledgeOffset;

castDistance = \_castDistance;

}

protected virtual void Awake()

{

InitializeOverlapMask();

\_ignoreRaycastLayer = LayerMask.NameToLayer("Ignore Raycast");

}

public void OnDrawGizmosSelected()

{

DrawGizmos();

}

#endregion

}

}

* + 1. GroundDetection——实际的地面检测，作为继承于basic的子类加以拓展，代码量246

#if UNITY\_EDITOR

using UnityEditor;

#endif

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Movement.Basis

{

public sealed class GroundDetection : BasicGroundDetection

{

#region METHODS

private bool BottomSphereCast(Vector3 position, Quaternion rotation, out RaycastHit hitInfo, float distance,

float backstepDistance = kBackstepDistance)

{

var radius = capsuleCollider.radius;

var height = Mathf.Max(0.0f, capsuleCollider.height \* 0.5f - radius);

var center = capsuleCollider.center - Vector3.up \* height;

var origin = position + rotation \* center;

var down = rotation \* Vector3.down;

return SphereCast(origin, radius, down, out hitInfo, distance, backstepDistance);

}

private bool BottomRaycast(Vector3 position, Quaternion rotation, out RaycastHit hitInfo, float distance,

float backstepDistance = kBackstepDistance)

{

var down = rotation \* Vector3.down;

return Raycast(position, down, out hitInfo, distance, backstepDistance) &&

SimulateSphereCast(position, rotation, hitInfo.normal, out hitInfo, distance, backstepDistance);

}

private bool SimulateSphereCast(Vector3 position, Quaternion rotation, Vector3 normal, out RaycastHit hitInfo,

float distance = Mathf.Infinity, float backstepDistance = kBackstepDistance)

{

var origin = position;

var up = rotation \* Vector3.up;

var angle = Vector3.Angle(normal, up) \* Mathf.Deg2Rad;

if (angle > 0.0001f)

{

var radius = capsuleCollider.radius;

var x = Mathf.Sin(angle) \* radius;

var y = (1.0f - Mathf.Cos(angle)) \* radius;

var right = Vector3.Cross(normal, up);

var tangent = Vector3.Cross(right, normal);

origin += Vector3.ProjectOnPlane(tangent, up).normalized \* x + up \* y;

}

return Raycast(origin, -up, out hitInfo, distance, backstepDistance);

}

private void DetectLedgeAndSteps(Vector3 position, Quaternion rotation, ref GroundHit groundHitInfo,

float distance, Vector3 point, Vector3 normal)

{

Vector3 up = rotation \* Vector3.up, down = -up;

var projectedNormal = Vector3.ProjectOnPlane(normal, up).normalized;

var nearPoint = point + projectedNormal \* kHorizontalOffset;

var farPoint = point - projectedNormal \* kHorizontalOffset;

var ledgeStepDistance = Mathf.Max(kMinLedgeDistance, Mathf.Max(stepOffset, distance));

RaycastHit nearHitInfo;

var nearHit = Raycast(nearPoint, down, out nearHitInfo, ledgeStepDistance);

var isNearGroundValid = nearHit && Vector3.Angle(nearHitInfo.normal, up) < groundLimit;

RaycastHit farHitInfo;

var farHit = Raycast(farPoint, down, out farHitInfo, ledgeStepDistance);

var isFarGroundValid = farHit && Vector3.Angle(farHitInfo.normal, up) < groundLimit;

if (farHit && !isFarGroundValid)

{

groundHitInfo.surfaceNormal = farHitInfo.normal;

RaycastHit secondaryHitInfo;

if (BottomRaycast(position, rotation, out secondaryHitInfo, distance))

{

groundHitInfo.SetFrom(secondaryHitInfo);

groundHitInfo.surfaceNormal = secondaryHitInfo.normal;

}

return;

}

if (isNearGroundValid && isFarGroundValid)

{

groundHitInfo.surfaceNormal =

(point - nearHitInfo.point).sqrMagnitude < (point - farHitInfo.point).sqrMagnitude

? nearHitInfo.normal

: farHitInfo.normal;

var nearHeight = Vector3.Dot(point - nearHitInfo.point, up);

var farHeight = Vector3.Dot(point - farHitInfo.point, up);

var height = Mathf.Max(nearHeight, farHeight);

if (height > kMinLedgeDistance && height < stepOffset)

{

groundHitInfo.isOnStep = true;

groundHitInfo.stepHeight = height;

}

return;

}

var isOnLedge = isNearGroundValid != isFarGroundValid;

if (!isOnLedge)

return;

groundHitInfo.surfaceNormal = isFarGroundValid ? farHitInfo.normal : nearHitInfo.normal;

groundHitInfo.ledgeDistance = Vector3.ProjectOnPlane(point - position, up).magnitude;

if (isFarGroundValid && groundHitInfo.ledgeDistance > ledgeOffset)

{

groundHitInfo.isOnLedgeEmptySide = true;

var radius = ledgeOffset;

var offset = Mathf.Max(0.0f, capsuleCollider.height \* 0.5f - radius);

var bottomSphereCenter = capsuleCollider.center - Vector3.up \* offset;

var bottomSphereOrigin = position + rotation \* bottomSphereCenter;

RaycastHit hitInfo;

if (SphereCast(bottomSphereOrigin, radius, down, out hitInfo, Mathf.Max(stepOffset, distance)))

{

var verticalSquareDistance = Vector3.Project(point - hitInfo.point, up).sqrMagnitude;

if (verticalSquareDistance <= stepOffset \* stepOffset)

groundHitInfo.isOnLedgeEmptySide = false;

}

}

groundHitInfo.isOnLedgeSolidSide = !groundHitInfo.isOnLedgeEmptySide;

}

public override bool ComputeGroundHit(Vector3 position, Quaternion rotation, ref GroundHit groundHitInfo,

float distance = Mathf.Infinity)

{

var up = rotation \* Vector3.up;

RaycastHit hitInfo;

if (BottomSphereCast(position, rotation, out hitInfo, distance) && Vector3.Angle(hitInfo.normal, up) < 89.0f)

{

groundHitInfo.SetFrom(hitInfo);

DetectLedgeAndSteps(position, rotation, ref groundHitInfo, distance, hitInfo.point, hitInfo.normal);

groundHitInfo.isOnGround = true;

groundHitInfo.isValidGround = !groundHitInfo.isOnLedgeEmptySide &&

Vector3.Angle(groundHitInfo.surfaceNormal, up) < groundLimit;

return true;

}

if (!BottomRaycast(position, rotation, out hitInfo, distance))

return false;

groundHitInfo.SetFrom(hitInfo);

groundHitInfo.surfaceNormal = hitInfo.normal;

groundHitInfo.isOnGround = true;

groundHitInfo.isValidGround = Vector3.Angle(groundHitInfo.surfaceNormal, /\*Vector3.up\*/up) < groundLimit;

return true;

}

public override bool FindGround(Vector3 direction, out RaycastHit hitInfo, float distance = Mathf.Infinity,

float backstepDistance = kBackstepDistance)

{

var radius = capsuleCollider.radius;

var height = Mathf.Max(0.0f, capsuleCollider.height \* 0.5f - radius);

var center = capsuleCollider.center - Vector3.up \* height;

var origin = transform.TransformPoint(center);

var up = transform.up;

if (!SphereCast(origin, radius, direction, out hitInfo, distance, backstepDistance) ||

Vector3.Angle(hitInfo.normal, /\*Vector3.up\*/up) >= 89.0f)

return false;

var p = transform.position - transform.up \* hitInfo.distance;

var q = transform.rotation;

var groundHitInfo = new GroundHit(hitInfo);

DetectLedgeAndSteps(p, q, ref groundHitInfo, castDistance, hitInfo.point, hitInfo.normal);

groundHitInfo.isOnGround = true;

groundHitInfo.isValidGround = !groundHitInfo.isOnLedgeEmptySide &&

Vector3.Angle(groundHitInfo.surfaceNormal, /\*Vector3.up\*/up) < groundLimit;

return groundHitInfo.isOnGround && groundHitInfo.isValidGround;

}

protected override void DrawGizmos()

{

#if UNITY\_EDITOR

base.DrawGizmos();

var radius = capsuleCollider.radius;

var center = capsuleCollider.center;

var offset = Mathf.Max(0.0f, capsuleCollider.height \* 0.5f - radius);

if (!Application.isPlaying)

offset += castDistance;

var color = new Color(0.5f, 1.0f, 0.6f);

if (Application.isPlaying)

color = isOnGround ? (isValidGround ? Color.green : Color.blue) : Color.red;

Gizmos.matrix = Matrix4x4.TRS(transform.position, transform.rotation, transform.lossyScale);

Gizmos.color = color;

Gizmos.DrawWireSphere(center - Vector3.up \* offset, radius \* 1.01f);

Gizmos.matrix = Matrix4x4.identity;

Handles.matrix = Matrix4x4.TRS(transform.position, transform.rotation, transform.lossyScale);

var standingOnLedge = isOnLedgeSolidSide || isOnLedgeEmptySide;

if (standingOnLedge)

{

Handles.color = isOnLedgeSolidSide

? new Color(0.0f, 1.0f, 0.0f, 0.25f)

: new Color(1.0f, 0.0f, 0.0f, 0.25f);

Handles.DrawSolidDisc(Vector3.zero, Vector3.up, ledgeOffset);

}

Handles.matrix = Matrix4x4.identity;

#endif

}

#endregion

}

}

* + 1. BasicPlayerController——基本人物控制器，作为PlayerController的父类，代码量343

using SYK.Movement.Basis;

using SYK.Movement.Addons;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Serialization;

namespace SYK.Movement.Controller

{

public class BasicPlayerController : MonoBehaviour

{

#region EDITOR EXPOSED FIELDS

[Header("Movement")]

[SerializeField]

private float \_speed = 5.0f;

[SerializeField]

private float \_angularSpeed = 540.0f;

[SerializeField]

private float \_acceleration = 50.0f;

[SerializeField]

private float \_deceleration = 20.0f;

[SerializeField]

private float \_groundFriction = 8f;

[SerializeField]

private bool \_useBrakingFriction;

[SerializeField]

private float \_brakingFriction = 8f;

[SerializeField]

private float \_airFriction;

[Range(0.0f, 1.0f)]

[SerializeField]

private float \_airControl = 0.2f;

[SerializeField]

private float \_standingHeight = 1.8f;

[Header("Animation")]

[SerializeField]

private bool \_useRootMotion;

[SerializeField]

private bool \_rootMotionRotation;

#endregion

#region FIELDS

private Vector3 \_moveDirection;

private bool \_allowVerticalMovement;

private bool \_restoreVelocityOnResume = true;

#endregion

#region PROPERTIES

public MovementBasis movement { get; private set; }

public Animator animator { get; set; }

public RMController rootMotionController { get; set; }

public bool allowVerticalMovement

{

get { return \_allowVerticalMovement; }

set

{

\_allowVerticalMovement = value;

if (movement)

movement.useGravity = !\_allowVerticalMovement;

}

}

public float speed

{

get { return \_speed; }

set { \_speed = Mathf.Max(0.0f, value); }

}

public float angularSpeed

{

get { return \_angularSpeed; }

set { \_angularSpeed = Mathf.Max(0.0f, value); }

}

public float acceleration

{

get { return movement.isGrounded ? \_acceleration : \_acceleration \* airControl; }

set { \_acceleration = Mathf.Max(0.0f, value); }

}

public float deceleration

{

get { return movement.isGrounded ? \_deceleration : \_deceleration \* airControl; }

set { \_deceleration = Mathf.Max(0.0f, value); }

}

public float groundFriction

{

get { return \_groundFriction; }

set { \_groundFriction = Mathf.Max(0.0f, value); }

}

public bool useBrakingFriction

{

get { return \_useBrakingFriction; }

set { \_useBrakingFriction = value; }

}

public float brakingFriction

{

get { return \_brakingFriction; }

set { \_brakingFriction = Mathf.Max(0.0f, value); }

}

public float airFriction

{

get { return \_airFriction; }

set { \_airFriction = Mathf.Max(0.0f, value); }

}

public float airControl

{

get { return \_airControl; }

set { \_airControl = Mathf.Clamp01(value); }

}

public float standingHeight

{

get { return \_standingHeight; }

set { \_standingHeight = Mathf.Max(0.0f, value); }

}

public bool useRootMotion

{

get { return \_useRootMotion; }

set { \_useRootMotion = value; }

}

public bool useRootMotionRotation

{

get { return \_rootMotionRotation; }

set { \_rootMotionRotation = value; }

}

public bool applyRootMotion

{

get { return animator != null && animator.applyRootMotion; }

set

{

if (animator != null)

animator.applyRootMotion = value;

}

}

public bool isFalling

{

get { return !movement.isGrounded && movement.velocity.y < 0.0001f; }

}

public bool isGrounded

{

get { return movement.isGrounded; }

}

public Vector3 moveDirection

{

get { return \_moveDirection; }

set { \_moveDirection = Vector3.ClampMagnitude(value, 1.0f); }

}

public bool pause { get; set; }

public bool isPaused { get; private set; }

public bool restoreVelocityOnResume

{

get { return \_restoreVelocityOnResume; }

set { \_restoreVelocityOnResume = value; }

}

#endregion

#region METHODS

private void Pause()

{

if (pause && !isPaused)

{

movement.Pause(true);

isPaused = true;

}

else if (!pause && isPaused)

{

movement.Pause(false, restoreVelocityOnResume);

isPaused = false;

}

}

public void RotateTowards(Vector3 direction, bool onlyLateral = true)

{

movement.Rotate(direction, angularSpeed, onlyLateral);

}

public void RotateTowardsMoveDirection(bool onlyLateral = true)

{

RotateTowards(moveDirection, onlyLateral);

}

public void RotateTowardsVelocity(bool onlyLateral = true)

{

RotateTowards(movement.velocity, onlyLateral);

}

protected virtual Vector3 CalcDesiredVelocity()

{

if (useRootMotion && applyRootMotion)

return rootMotionController.animVelocity;

return moveDirection \* speed;

}

protected virtual void Move()

{

var desiredVelocity = CalcDesiredVelocity();

if (useRootMotion && applyRootMotion)

movement.Move(desiredVelocity, speed, !allowVerticalMovement);

else

{

var currentFriction = isGrounded ? groundFriction : airFriction;

var currentBrakingFriction = useBrakingFriction ? brakingFriction : currentFriction;

movement.Move(desiredVelocity, speed, acceleration, deceleration, currentFriction,

currentBrakingFriction, !allowVerticalMovement);

}

applyRootMotion = useRootMotion && movement.isGrounded;

}

protected virtual void Animate() { }

protected virtual void UpdateRotation()

{

if (useRootMotion && applyRootMotion && useRootMotionRotation)

{

movement.rotation \*= animator.deltaRotation;

}

else

{

RotateTowardsMoveDirection();

}

}

protected virtual void HandleInput()

{

if (Input.GetKeyDown(KeyCode.P))

pause = !pause;

moveDirection = new Vector3

{

x = Input.GetAxisRaw("Horizontal"),

y = 0.0f,

z = Input.GetAxisRaw("Vertical")

};

}

#endregion

#region MONOBEHAVIOUR

public virtual void OnValidate()

{

speed = \_speed;

angularSpeed = \_angularSpeed;

acceleration = \_acceleration;

deceleration = \_deceleration;

groundFriction = \_groundFriction;

brakingFriction = \_brakingFriction;

airFriction = \_airFriction;

airControl = \_airControl;

standingHeight = \_standingHeight;

}

public virtual void Awake()

{

movement = GetComponent<MovementBasis>();

movement.platformUpdatesRotation = true;

animator = GetComponentInChildren<Animator>();

rootMotionController = GetComponentInChildren<RMController>();

animator.SetBool("canRotate",true);

}

public virtual void FixedUpdate()

{

Pause();

if (isPaused)

return;

Move();

}

public virtual void Update()

{

HandleInput();

if (isPaused)

return;

UpdateRotation();

Animate();

}

#endregion

}

}

* + 1. PlayerController——作为继承于basic的子类，进行拓展，真正挂载在人物身上的脚本，代码量147

using SYK.Movement.Preparation;

using SYK.Movement.Controller;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Movement.Effective

{

public class PlayerController : BasicPlayerController

{

#region EDITOR FIELDS

[Header("CUSTOM CONTROLLER")]

[Tooltip("跟随相机" )]

public Transform playerCamera;

[Tooltip("角色Walk速度" )]

[SerializeField]

private float \_walkSpeed = 2.5f;

[Tooltip("角色Run速度" )]

[SerializeField]

private float \_runSpeed = 5.0f;

[Tooltip("角色Sprint速度" )]

[SerializeField]

private float \_sprintSpeed = 7.0f;

#endregion

#region PROPERTIES

public float walkSpeed

{

get { return \_walkSpeed; }

set { \_walkSpeed = Mathf.Max(0.0f, value); }

}

public float runSpeed

{

get { return \_runSpeed; }

set { \_runSpeed = Mathf.Max(0.0f, value); }

}

public float sprintSpeed

{

get { return \_sprintSpeed; }

set { \_sprintSpeed = Mathf.Max(0.0f, value); }

}

//步行输入控制

public bool walk { get; private set; }

public bool sprint { get; private set; }

#endregion

#region METHODS

//根据角色状态获取速度

private float GetTargetSpeed()

{

if(walk)

return walkSpeed;

else if(sprint)

return sprintSpeed;

else

return runSpeed;

}

//覆盖父类方法从而得到正确的状态速度

protected override Vector3 CalcDesiredVelocity()

{

//根据状态设置正确速度

speed = GetTargetSpeed();

//返回期望速度

return base.CalcDesiredVelocity();

}

//覆盖父类动画方法

protected override void Animate()

{

if (animator == null)

return;

var move = transform.InverseTransformDirection(moveDirection);

var forwardAmount = animator.applyRootMotion

? Mathf.InverseLerp(0.0f, sprintSpeed, move.z \* speed)

: Mathf.InverseLerp(0.0f, sprintSpeed, movement.forwardSpeed);

animator.SetFloat("Forward", forwardAmount, 0.1f, Time.deltaTime);

animator.SetFloat("Lateral",0.0f);

//animator.SetFloat("Lateral", (move.x/move.z)\*forwardAmount, 0.1f, Time.deltaTime);

animator.SetBool("OnGround", movement.isGrounded);

useRootMotionRotation = !animator.GetBool("canRotate");

}

//覆盖父类输入处理从而达到自定义效果

protected override void HandleInput()

{

moveDirection = new Vector3

{

x = Input.GetAxisRaw("Horizontal"),

y = 0.0f,

z = Input.GetAxisRaw("Vertical")

};

walk = Input.GetButton("Walk");

sprint = Input.GetButton("Sprint");

moveDirection = moveDirection.relativeTo(playerCamera);

}

#endregion

#region MONOBEHAVIOUR

//覆盖父类初始化

public override void OnValidate()

{

base.OnValidate();

walkSpeed = \_walkSpeed;

runSpeed = \_runSpeed;

sprintSpeed = \_sprintSpeed;

}

#endregion

}

}

* + 1. RMController——控制跟运动的脚本，代码量62

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Movement.Addons

{

[RequireComponent(typeof(Animator))]

public sealed class RMController : MonoBehaviour

{

#region FIELDS

private Animator \_animator;

#endregion

#region PROPERTIES

public Vector3 animVelocity { get; private set; }

public Vector3 animAngularVelocity { get; private set; }

public Quaternion animDeltaRotation { get; private set; }

#endregion

#region MONOBEHAVIOUR

public void Awake()

{

\_animator = GetComponent<Animator>();

if (\_animator == null)

{

Debug.LogError(

string.Format(

"RMController: There is no 'Animator' attached to the '{0}' game object.\n" +

"Please attach a 'Animator' to the '{0}' game object",

name));

}

}

public void OnAnimatorMove()

{

var deltaTime = Time.deltaTime;

if (deltaTime <= 0.0f)

return;

animVelocity = \_animator.deltaPosition / deltaTime;

animDeltaRotation = \_animator.deltaRotation;

float angleInDegrees;

Vector3 rotationAxis;

animDeltaRotation.ToAngleAxis(out angleInDegrees, out rotationAxis);

Vector3 angularDisplacement = rotationAxis \* angleInDegrees \* Mathf.Deg2Rad;

animAngularVelocity = angularDisplacement / Time.deltaTime;

}

#endregion

}

}

* 1. PlayerHandler——涉及其他模块与相关内容的交互，和人物移动以外的内容，代码量119

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerHandler : MonoBehaviour

{

private Animator anim;

public CharacterStats characterStats;

private bool isDeath;

private bool Restart;

public List<Transform> attackTargets;

public Rigidbody rigidbody;

public WeaponItem\_SO weapon;

private float enduranceRecoveryTimer;

public float forceMultiple;

void Awake()

{

anim = GetComponent<Animator>();

characterStats = GetComponent<CharacterStats>();

attackTargets = new List<Transform>();

rigidbody = transform.GetComponentInParent<Rigidbody>();

forceMultiple = 50.0f;

}

void Start()

{

GameManager.Instance.RegisterPlayer(characterStats);

}

void Update()

{

characterStats.characterData.currentPosture = Mathf.Min(Time.deltaTime+characterStats.characterData.currentPosture,characterStats.characterData.maxPosture);

enduranceRecoveryTimer -= Time.deltaTime;

isDeath = characterStats.CurrentHealth == 0;

anim.SetBool("Death",isDeath);

if (isDeath)

{

GameManager.Instance.NotifyObservers();

Invoke("RestartGame", 5);

}

AttackInput\_Anim();

Dash();

EnduranceRecovery();

}

void EnduranceRecovery()

{

if(characterStats.CurrentEndurance == characterStats.MaxEndurance)

return;

if(characterStats.CurrentEndurance != characterStats.MaxEndurance && characterStats.CurrentEndurance > 0 && enduranceRecoveryTimer <= 0)

characterStats.CurrentEndurance += 25 \* Time.deltaTime;

//if(characterStats.CurrentEndurance == 0 && enduranceRecoveryTimer >= -1.0f)

//动话

if(characterStats.CurrentEndurance == 0 && enduranceRecoveryTimer <= -1.0f)

characterStats.CurrentEndurance += 25 \* Time.deltaTime ;

characterStats.CurrentEndurance = Mathf.Min(characterStats.CurrentEndurance,100.0f);

}

void AttackInput\_Anim()

{

if(characterStats.CurrentEndurance > 0)

{

if(Input.GetMouseButtonDown(0))

{

anim.SetBool("isAttacking",true);

anim.SetTrigger("Attack");

characterStats.CurrentEndurance = Mathf.Max(characterStats.CurrentEndurance-18,0);

enduranceRecoveryTimer = 1.0f;

}

}

}

void Dash()

{

if(characterStats.CurrentEndurance > 0)

{

if(Input.GetButtonDown("Dash"))

{

Vector3 force = transform.forward \* forceMultiple;;

anim.SetTrigger("Dash");

rigidbody.velocity = force;

characterStats.CurrentEndurance = Mathf.Max(characterStats.CurrentEndurance-22,0);

enduranceRecoveryTimer = 1.0f;

}

}

}

#region AttackTargets List Formation

void OnTriggerEnter(Collider collider)

{

if(collider.gameObject.CompareTag("Enemy"))

attackTargets.Add(collider.transform);

}

void OnTriggerExit(Collider collider)

{

attackTargets.Remove(collider.transform);

}

#endregion

private void RestartGame()

{

Application.LoadLevel("StartScene\_Kingdom");

}

//Animation Event

void Attack\_AE()

{

foreach(Transform attackTarget in attackTargets)

{

if(attackTarget != null && Vector3.Distance(attackTarget.position, transform.position) <= characterStats.attackData.attackRange)

{

var targetStats = attackTarget.GetComponent<CharacterStats>();

targetStats.TakeDamage(characterStats, targetStats);

}

}

}

}

* 1. 武器与攻击处理
     1. Weapon Collider——挂载在武器上，需要Collider组件，涉及攻击判定与处理，代码量46

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Combat.Basis

{

public class WeaponCollider : MonoBehaviour

{

public Collider weaponCollider;

void Awake()

{

weaponCollider = GetComponent<Collider>();

weaponCollider.gameObject.SetActive(true);

weaponCollider.isTrigger = true;

weaponCollider.enabled = false;

}

public void EnableWeaponCollider()

{

weaponCollider.enabled = true;

}

public void DisableWeaponCollider()

{

weaponCollider.enabled = false;

}

private void OnTriggerEnter(Collider other)

{

CharacterStats enemyStats = other.gameObject.GetComponent<CharacterStats>();

CharacterStats playerStats = weaponCollider.GetComponentInParent<CharacterStats>();

if(other.gameObject.tag == "Enemy")

{

if(enemyStats != null && playerStats != null)

{

enemyStats.TakeDamage(playerStats,enemyStats);

}

}

if(other.gameObject.tag == "Player")

{

if(enemyStats != null && playerStats != null)

{

enemyStats.TakeDamage(playerStats,enemyStats);

}

}

}

}

}

* + 1. Weapon Slot——武器插槽脚本，控制武器出现与切换，籍此脚本可实现Sneath，Unsneath，切换武器与左右手和双手武器处理，代码量56

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using SYK.Combat.Basis;

namespace SYK.Combat.Basis

{

public class WeaponSlot : MonoBehaviour

{

public Transform parentPivot;

public bool isLeft;

public GameObject currentWeapon;

public void UnloadWeapon()

{

if(currentWeapon != null)

currentWeapon.SetActive(false);

}

public void UnloadAndDestroyWeapon()

{

if(currentWeapon != null)

Destroy(currentWeapon);

}

public void LoadWeapon(WeaponItem\_SO WeaponItem)

{

UnloadAndDestroyWeapon();

if(WeaponItem == null)

{

UnloadWeapon();

return;

}

Debug.Log(WeaponItem.Weapon.name);

GameObject weaponModel = Instantiate(WeaponItem.Weapon);

if(weaponModel != null)

{

if(parentPivot != null)

weaponModel.transform.parent = parentPivot;

else

weaponModel.transform.parent = transform;

}

weaponModel.transform.localPosition = Vector3.zero;

weaponModel.transform.localRotation = Quaternion.identity;

weaponModel.transform.localScale = Vector3.one;

currentWeapon = weaponModel;

}

}

}

* + 1. WeaponSlotManager——控制Slot，将武器信息向父级传递，代码量67

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using SYK.Combat.Basis;

namespace SYK.Combat.Basis

{

public class WeaponSlotManager : MonoBehaviour

{

public WeaponSlot rightHandSlot;

public WeaponSlot leftHandSlot;

public WeaponCollider rightHandWeaponCollider;

public WeaponCollider leftHandWeaponCollider;

private void Awake()

{

WeaponSlot[] weaponSlots = GetComponentsInChildren<WeaponSlot>();

foreach(WeaponSlot slot in weaponSlots)

{

if(slot.isLeft)

leftHandSlot = slot;

else

rightHandSlot = slot;

}

}

public void LoadWeaponOnSlot(WeaponItem\_SO weaponItem, bool isLeft)

{

if(isLeft)

{

leftHandSlot.LoadWeapon(weaponItem);

LoadLeftHandWeaponCollider();

}

else

{

rightHandSlot.LoadWeapon(weaponItem);

LoadRightHandWeaponCollider();

}

}

public void LoadRightHandWeaponCollider()

{

rightHandWeaponCollider = rightHandSlot.currentWeapon.GetComponentInChildren<WeaponCollider>();

}

public void LoadLeftHandWeaponCollider()

{

leftHandWeaponCollider = leftHandSlot.currentWeapon.GetComponentInChildren<WeaponCollider>();

}

public void OpenLeftCollider()

{

leftHandWeaponCollider.EnableWeaponCollider();

}

public void OpenRightCollider()

{

rightHandWeaponCollider.EnableWeaponCollider();

}

public void CloseLeftCollider()

{

leftHandWeaponCollider.DisableWeaponCollider();

}

public void CloseRightCollider()

{

rightHandWeaponCollider.DisableWeaponCollider();

}

}

}

* + 1. WeaponInstantiation——最高级脚本，与Player交互确定使用武器，本身也担负接受与发送信息给子级Manager的任务，代码量26

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using SYK.Combat.Basis;

namespace SYK.Combat.Basis

{

public class WeaponInstantiation : MonoBehaviour

{

public WeaponSlotManager weaponSlotManager;

public WeaponItem\_SO rightWeapon;

public WeaponItem\_SO leftWeapon;

private void Awake()

{

weaponSlotManager = GetComponentInChildren<WeaponSlotManager>();

}

private void Start()

{

weaponSlotManager.LoadWeaponOnSlot(rightWeapon,false);

weaponSlotManager.LoadWeaponOnSlot(leftWeapon,true);

}

}

}

* 1. 无敌帧处理——采用较为简单的StateMechineBehaviour操作。减少代码量，代码量36

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SuperArmorHandler : StateMachineBehaviour

{

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.transform.GetComponentInParent<Collider>().enabled = false;

}

// OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks

//override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

//

//}

//OnStateExit is called when a transition ends and the state machine finishes evaluating this state

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.transform.GetComponentInParent<Collider>().enabled = true;

}

// OnStateMove is called right after Animator.OnAnimatorMove()

//override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that processes and affects root motion

//}

// OnStateIK is called right after Animator.OnAnimatorIK()

//override public void OnStateIK(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that sets up animation IK (inverse kinematics)

//}

}

1. State——主要涉及Sciptable Object的编写和一个MonoBehaviour的处理
   1. Scriptable Object
      1. Character Data——存储角色基本信息的模板，代码量61

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "New Data",menuName = "Character Stats/Data")]

public class CharacterData\_SO : ScriptableObject

{

[Header("Stats Info")]

public int maxHealth;

public int currentHealth;

public int maxMana;

public int currentMana;

public float maxEndurance;

public float currentEndurance;

public int baseDefence;

public int currentDefence;

public int baseMagicDefence;

public int currentMagicDefence;

public int maxPosture;

public float currentPosture;

[Header("Kill")]

public int Exp;

public int Currency;

[Header("Level")]

public int currentLevel;

public int currentExp;

public int nowExp;

[Header("Player")]

public float expMultiplier;

public float currencyMultiplier;

public int levelPoint;

public int nowCurrency;

public void UpdateExp(int point,int currency)

{

currentExp += (int)(point \* expMultiplier);

while (currentExp >= nowExp)

LevelUp();

nowCurrency += currency;

}

private void LevelUp()

{

currentLevel++;

currentExp -= nowExp;

nowExp = (int)(nowExp \* 1.05f);

if(UnityEngine.Random.Range(0.0f,1.0f)<0.35f)

levelPoint += 3;

else

levelPoint += 2;

}

}

* + 1. Attack Data——存储角色攻击相关信息，代码量22

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "New Attack",menuName = "Attack/Attack Data")]

public class AttackData\_SO : ScriptableObject

{

[Header("Basic Info")]

public float coolDown;

public int minDamage;

public int maxDamage;

public float maxMagicDamageMultiplier;

public float minMagicDamageMultiplier;

public float criticalMultiplier;

public float criticalChance;

public int minPostureDamage;

public int maxPostureDamage;

[Header("Enemy")]

public float attackRange;

public float skillRange;

}

* 1. MonoBehaviour——Character Stats挂载在人物上，实际完成数据交互的脚本，代码量133

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CharacterStats : MonoBehaviour

{

public event Action<int, int> UpdateHealthBarOnAttack;

public CharacterData\_SO templateData;

public CharacterData\_SO characterData;

public AttackData\_SO attackData;

[HideInInspector]

public bool isCritical;

void Awake()

{

if (templateData != null)

characterData = Instantiate(templateData);

}

#region Read from Data\_SO

public int MaxHealth

{

get { if (characterData != null) return characterData.maxHealth; else return 0; }

set { characterData.maxHealth = value; }

}

public int CurrentHealth

{

get { if (characterData != null) return characterData.currentHealth; else return 0; }

set { characterData.currentHealth = value; }

}

public int BaseDefence

{

get { if (characterData != null) return characterData.baseDefence; else return 0; }

set { characterData.baseDefence = value; }

}

public int CurrentDefence

{

get { if (characterData != null) return characterData.currentDefence; else return 0; }

set { characterData.currentDefence = value; }

}

public int BaseMagicDefence

{

get { if (characterData != null) return characterData.baseMagicDefence; else return 0; }

set { characterData.baseMagicDefence = value; }

}

public int CurrentMagicDefence

{

get { if (characterData != null) return characterData.currentMagicDefence; else return 0; }

set { characterData.currentMagicDefence = value; }

}

public float MaxEndurance

{

get { if (characterData != null) return characterData.maxEndurance; else return 0; }

set { characterData.maxEndurance = value; }

}

public float CurrentEndurance

{

get { if (characterData != null) return characterData.currentEndurance; else return 0; }

set { characterData.currentEndurance = value; }

}

#endregion

#region CurrentData

private int CurrentMagicDamage()

{

//float coreDamage = UnityEngine.Random.Range(attackData.minMaDamage, attackData.maxDamage);

//if (isCritical)

//coreDamage \*= attackData.criticalMultiplier;

//return (int)coreDamage;

//魔法伤害

return 0;

}

private int CurrentDamage()

{

float coreDamage = UnityEngine.Random.Range(attackData.minDamage, attackData.maxDamage);

if (isCritical)

coreDamage \*= attackData.criticalMultiplier;

return (int)coreDamage;

}

private int CurrentPostureDamage()

{

float coreDamage = UnityEngine.Random.Range(attackData.minPostureDamage, attackData.maxPostureDamage);

//武器架势削减倍数

return (int)coreDamage;

}

#endregion

#region Character Combat

public void TakeDamage(CharacterStats attacker,CharacterStats defener)

{

int damage = Mathf.Max(attacker.CurrentDamage() - defener.CurrentDefence,1);

int magicDamage = Mathf.Max(attacker.CurrentMagicDamage() - defener.CurrentMagicDefence,1);

CurrentHealth = Mathf.Max(CurrentHealth - (damage + magicDamage), 0);

defener.characterData.currentPosture -= attacker.CurrentPostureDamage();

if(defener.gameObject.tag == "Player")

{

if(defener.characterData.currentPosture <= 0 || (damage + magicDamage)/(CurrentHealth + damage + magicDamage) > 0.75f )

defener.GetComponent<Animator>().SetTrigger("heavyHit");

else

defener.GetComponent<Animator>().SetTrigger("Hit");

}

else

{

if(defener.characterData.currentPosture <= 0)

defener.GetComponent<Animator>().SetTrigger("heavyHit");

else if(UnityEngine.Random.Range(0.0f,1.0f) < 0.23f)

defener.GetComponent<Animator>().SetTrigger("Hit");

}

if(defener.characterData.currentPosture <= 0)

defener.characterData.currentPosture = defener.characterData.maxPosture;

UpdateHealthBarOnAttack?.Invoke(CurrentHealth, MaxHealth);

if (CurrentHealth <= 0)

GameManager.Instance.playerStats.characterData.UpdateExp(characterData.Exp,characterData.Currency);

}

public void TakeDamage(int damage, CharacterStats defener)

{

int currentDamage = Mathf.Max(damage - defener.CurrentDefence, 0);

CurrentHealth = Mathf.Max(CurrentHealth - currentDamage , 0);

UpdateHealthBarOnAttack?.Invoke(CurrentHealth, MaxHealth);

}

#endregion

}

1. APC——包括小怪和BOSS两类
   1. 小怪——均为继承于Enemy Controller的子类，只添加了特殊的攻击效果，故此只展示父类Enemy Controller，代码量308

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public enum EnemyStates { GUARD, PATROL, CHASE, DEAD }

[RequireComponent(typeof(NavMeshAgent))]

[RequireComponent(typeof(CharacterStats))]

public class EnemyControllers : MonoBehaviour,IEndGameObserver

{

private EnemyStates enemyStates;

private NavMeshAgent agent;

private Animator anim;

private Collider coll;

protected CharacterStats characterStats;

private Rigidbody rb;

[Header("Basic Settings")]

public float sightRadius;

public bool isGuard;

protected float speed;

protected GameObject attackTarget;

public float lookAtTime;

private float remainLookAtTime;

private float lastAttackTime;

private Quaternion guardRotation;

[Header("patrol State")]

public float patrolRange;

private Vector3 wayPoint;

private Vector3 guardPos;

bool isWalk;

bool isChase;

bool isFollow;

bool isDeath;

bool playerDeath;

void Awake()

{

agent = GetComponent<NavMeshAgent>();

anim = GetComponent<Animator>();

characterStats = GetComponent<CharacterStats>();

coll = GetComponent<Collider>();

speed = agent.speed;

guardPos = transform.position;

guardRotation = transform.rotation;

remainLookAtTime = lookAtTime;

}

// Start is called before the first frame update

void Start()

{

if (isGuard)

{

enemyStates = EnemyStates.GUARD;

}

else

{

enemyStates = EnemyStates.PATROL;

GetNewWayPoint();

}

//�л������޸�

GameManager.Instance.AddObserver(this);

lastAttackTime = 0;

}

//�л���������

/\*void OnEnable()

{

GameManager.Instance.AddObserver(this);

}\*/

void OnDisable()

{

if (!GameManager.IsInitialized) return;

GameManager.Instance.RemoveObserver(this);

}

// Update is called once per frame

void Update()

{

if (characterStats.CurrentHealth == 0)

isDeath = true;

if (playerDeath == false )

{

anim.SetBool("Win", false);

SwitchStates();

SwitchAnimation();

lastAttackTime -= Time.deltaTime;

}

else

{

anim.SetBool("Win", true);

isChase = false;

isWalk = false;

attackTarget = null;

}

}

void SwitchAnimation()

{

anim.SetBool("Walk", isWalk);

anim.SetBool("Chase", isChase);

anim.SetBool("Follow", isFollow);

anim.SetBool("Critical", characterStats.isCritical);

anim.SetBool("Death", isDeath);

}

void SwitchStates()

{

if (isDeath == true)

enemyStates = EnemyStates.DEAD;

else if (FoundPlayer())

{

enemyStates = EnemyStates.CHASE;

}

switch (enemyStates)

{

case EnemyStates.GUARD:

isChase = false;

if(transform.position != guardPos )

{

isWalk = true;

agent.isStopped = false;

agent.destination = guardPos;

if (Vector3.SqrMagnitude(guardPos - transform.position) <= agent.stoppingDistance) ;

{

isWalk = false;

transform.rotation = Quaternion.Lerp(transform.rotation, guardRotation, 0.03f);

}

}

break;

case EnemyStates.PATROL:

isChase = false;

isWalk = true;

agent.speed = speed \* 0.4f;

//�ж��Ƿ����漴Ѳ�ߵ�

if (Vector3.Distance(wayPoint, transform.position) <= agent.stoppingDistance)

{

isWalk = false;

if (remainLookAtTime > 0)

remainLookAtTime -= Time.deltaTime;

else

GetNewWayPoint();

}

else

{

isWalk = true;

agent.destination = wayPoint;

}

break;

case EnemyStates.CHASE:

isWalk = false;

isChase = true;

agent.speed = speed;

if (!FoundPlayer())

{

isFollow = false;

if (remainLookAtTime > 0)

{

agent.destination = transform.position;

remainLookAtTime -= Time.deltaTime;

}

else if (isGuard)

enemyStates = EnemyStates.GUARD;

else

enemyStates = EnemyStates.PATROL;

}

else

{

if(lastAttackTime > 0)

{

isFollow = false;

isWalk = true;

agent.isStopped = false;

agent.speed = speed \* 0.4f;

agent.destination = attackTarget.transform.position;

}

else

{

isFollow = true;

isWalk = false;

agent.isStopped = false ;

agent.destination = attackTarget.transform.position;

}

}

if (TargetInAttackRange() ||TargetInSkillRange())

{

if(isFollow == true)

{

if(UnityEngine.Random.Range(0.0f,1.0f) < 0.25f)

anim.SetTrigger("RunningAttack");

}

if(lastAttackTime > 0)

{

if(UnityEngine.Random.Range(0.0f,1.0f)<0.15f)

{

isFollow = false;

isWalk = false;

agent.isStopped = true;

lastAttackTime = characterStats.attackData.coolDown;

//�����ж�

characterStats.isCritical = Random.value < characterStats.attackData.criticalChance;

//ִ�й���

Attack();

}

}

else

{

isFollow = false;

isWalk = false;

agent.isStopped = true;

lastAttackTime = characterStats.attackData.coolDown;

//�����ж�

characterStats.isCritical = Random.value < characterStats.attackData.criticalChance;

//ִ�й���

Attack();

}

}

break;

case EnemyStates.DEAD:

agent.radius = 0;

coll.enabled = false;

Destroy(gameObject, 6f);

break;

}

}

void Attack()

{

transform.LookAt(attackTarget.transform);

if (TargetInAttackRange())

{

anim.SetTrigger("Attack");

anim.SetBool("Combo",UnityEngine.Random.Range(0.0f,1.0f)<0.3f);

}

if (TargetInSkillRange())

{

anim.SetTrigger("Skill");

}

}

bool FoundPlayer()

{

var colliders = Physics.OverlapSphere(transform.position, sightRadius);

foreach (var target in colliders)

{

if (target.CompareTag("Player"))

{

attackTarget = target.gameObject;

return true;

}

attackTarget = null;

}

return false;

}

bool TargetInAttackRange()

{

if (attackTarget != null)

return Vector3.Distance(attackTarget.transform.position, transform.position) <= characterStats.attackData.attackRange ;

else

return false;

}

bool TargetInSkillRange()

{

if (attackTarget != null)

return Vector3.Distance(attackTarget.transform.position, transform.position) <= characterStats.attackData.skillRange;

else

return false;

}

void GetNewWayPoint()

{

remainLookAtTime = lookAtTime;

float randomX = Random.Range(-patrolRange, patrolRange);

float randomZ = Random.Range(-patrolRange, patrolRange);

Vector3 randomPoint = new Vector3(guardPos.x + randomX, transform.position.y, guardPos.z + randomZ);

NavMeshHit hit;

wayPoint = NavMesh.SamplePosition(randomPoint, out hit, patrolRange, 1) ? hit.position : transform.position;

}

void OnDrawGizmosSelected()

{

Gizmos.color = Color.blue;

Gizmos.DrawWireSphere(transform.position, sightRadius);

}

void Hit()

{

if(attackTarget != null && transform.OA\_IsFacingTarget(attackTarget.transform) && Vector3.Distance(attackTarget.transform.position, transform.position) <= characterStats.attackData.attackRange)

{

var targetStats = attackTarget.GetComponent<CharacterStats>();

targetStats.TakeDamage(characterStats, targetStats);

}

}

public void EndNotify()

{

//��ʤ����

//ֹͣ�ƶ�

//ֹͣagents

anim.SetBool("Win", true);

playerDeath = true;

isChase = false;

isWalk = false;

attackTarget = null;

}

}

* 1. BOSS——由于BOSS的特殊性，每个BOSS均设置单独脚本，逻辑与小怪类似，只添加新的反应与处理方法，主要为投技，涉及身体部分，与武器攻击类似，故此展示身体攻击如Kick等的处理方法脚本，代码量59

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Combat.Basis

{

public class BodyAttackCollider : MonoBehaviour

{

public Collider weaponCollider;

public bool isHolding;

public Animator anim;

public EnemyBoss\_01\_OrcGaint BossScript;

void Awake()

{

weaponCollider = GetComponent<Collider>();

weaponCollider.isTrigger = true;

weaponCollider.enabled = false;

anim = GetComponentInParent<Animator>();

BossScript = GetComponentInParent<EnemyBoss\_01\_OrcGaint>();

}

void Update()

{

isHolding = anim.GetBool("HoldPlayer");

}

public void EnableWeaponCollider()

{

weaponCollider.enabled = true;

}

public void DisableWeaponCollider()

{

weaponCollider.enabled = false;

}

private void OnTriggerEnter(Collider other)

{

if(isHolding)

{

Vector3 nowPos = GetComponentInParent<Transform>().position;

BossScript.attackTarget.transform.Translate(nowPos);

}

CharacterStats enemyStats = other.gameObject.GetComponent<CharacterStats>();

CharacterStats playerStats = weaponCollider.GetComponentInParent<CharacterStats>();

if(other.gameObject.tag == "Enemy")

{

if(enemyStats != null && playerStats != null)

{

enemyStats.TakeDamage(playerStats,enemyStats);

}

}

if(other.gameObject.tag == "Player")

{

if(enemyStats != null && playerStats != null)

{

enemyStats.TakeDamage(playerStats,enemyStats);

}

}

}

}

}

* 1. 敌人攻击的处理——逻辑类似Player攻击处理，不过由于敌人攻击判定的单一性，可用更简便的AnimationEvent处理，脚本代码量可有效减少，代码量57

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace SYK.Combat.Basis

{

public class EnemyWeaponManager : MonoBehaviour

{

public WeaponCollider rightHandWeaponCollider;

public WeaponCollider leftHandWeaponCollider;

public BodyAttackCollider bodyAttackCollider;

private void Awake()

{

//LoadLeftHandWeaponCollider();

LoadRightHandWeaponCollider();

LoadBodyAttackCollider();

}

public void LoadRightHandWeaponCollider()

{

rightHandWeaponCollider = transform.GetComponentInChildren<WeaponCollider>();

}

public void LoadBodyAttackCollider()

{

bodyAttackCollider = transform.GetComponentInChildren<BodyAttackCollider>();

}

//public void LoadLeftHandWeaponCollider()

//{

// leftHandWeaponCollider = leftHandSlot.currentWeapon.GetComponentInChildren<WeaponCollider>();

//}

public void OpenBodyCollider()

{

bodyAttackCollider.EnableWeaponCollider();

}

public void CloseBodyCollider()

{

bodyAttackCollider.EnableWeaponCollider();

}

public void OpenLeftCollider()

{

leftHandWeaponCollider.EnableWeaponCollider();

}

public void OpenRightCollider()

{

rightHandWeaponCollider.EnableWeaponCollider();

}

public void CloseLeftCollider()

{

leftHandWeaponCollider.DisableWeaponCollider();

}

public void CloseRightCollider()

{

rightHandWeaponCollider.DisableWeaponCollider();

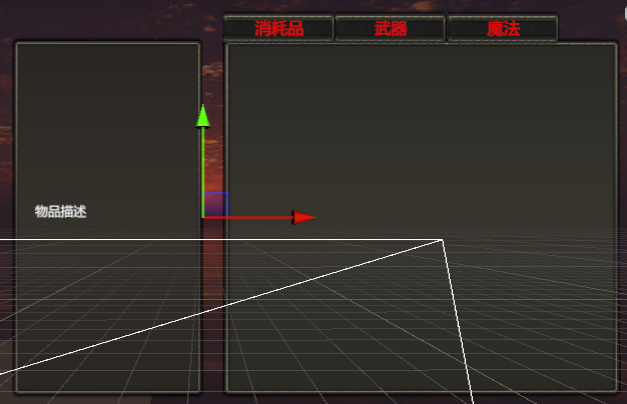
}

}

}

1. Inventory
   1. 基本背包UI的搭建

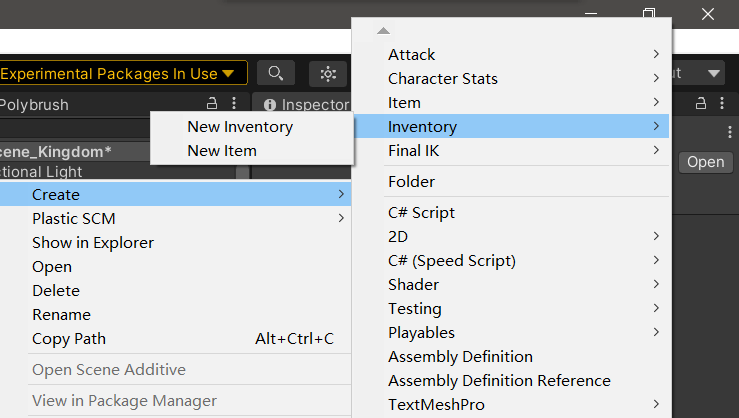
在背包UI的搭建过程中，主要使用了Unity自带的UI功能（Image Button等功能）以及Grid Layout Group 等插件来完成UI搭建，具体界面如下。



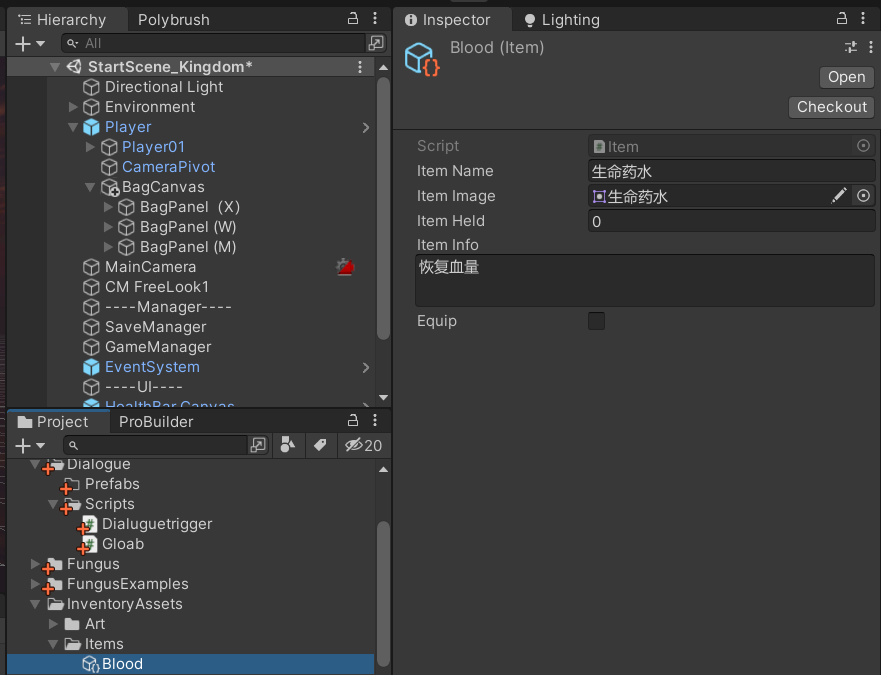
* 1. 背包数据与相应物品的存储

我们运用了Unity 自带的ScriptableObject 类来存储相应的数据。下面以Inventory类为例。首先，我们让这个类继承自ScriptableObject，接着我们将这个类作为一个Parent，在其中添加上List 方法，表明这一类作为Inventory 用来储存物品数据.它的子类则作为具体的Inventory 来储存某一些数据.所以我们使用了Unity 自带的CreateAssetMenu 方法来在右键菜单中创建出选项



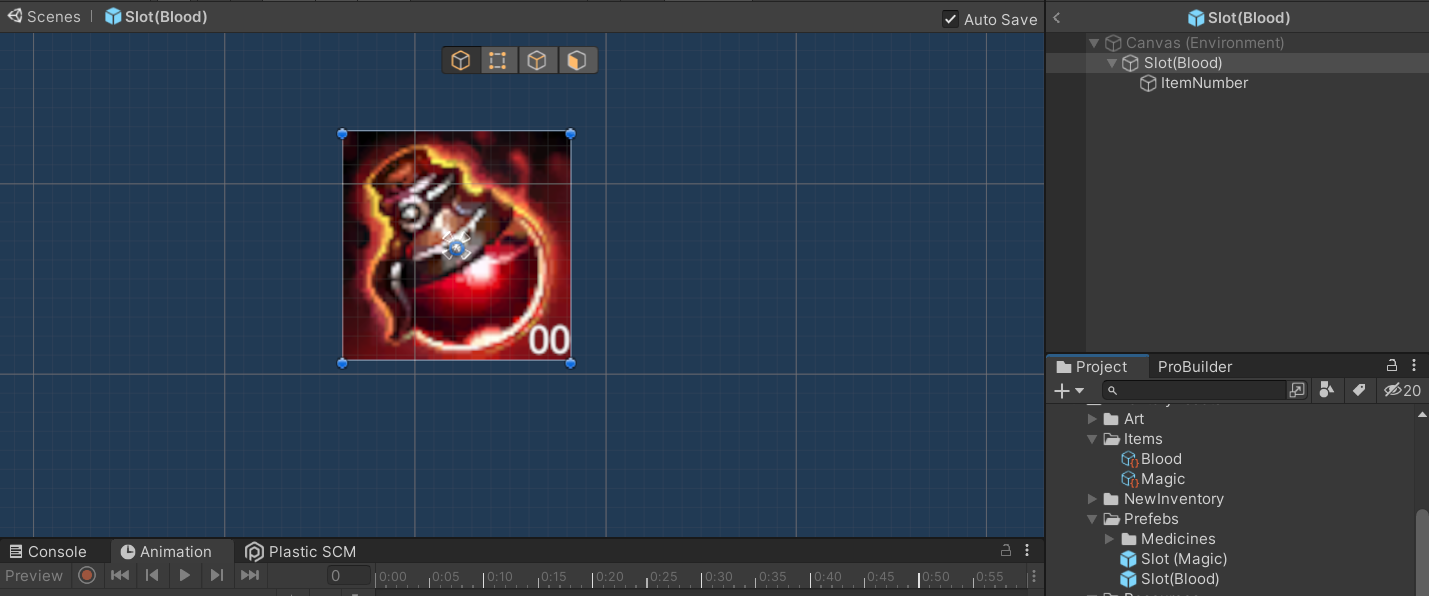


而对于具体的物品的信息,只需要在外部在Inspector中修改即可.



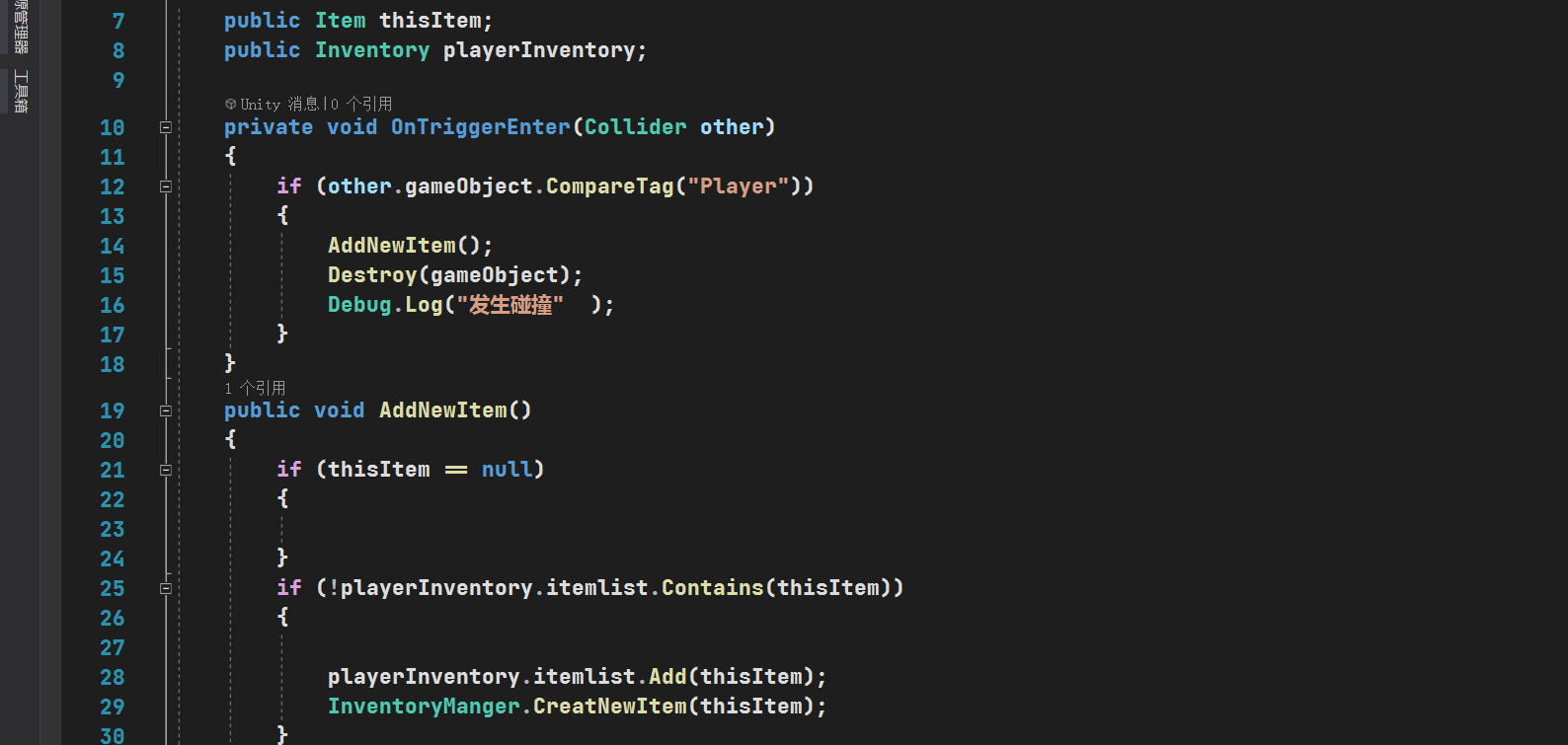
* 1. 捡拾世界中掉落的物品.

首先,在场景中创建两个物品,并将其制成预制体



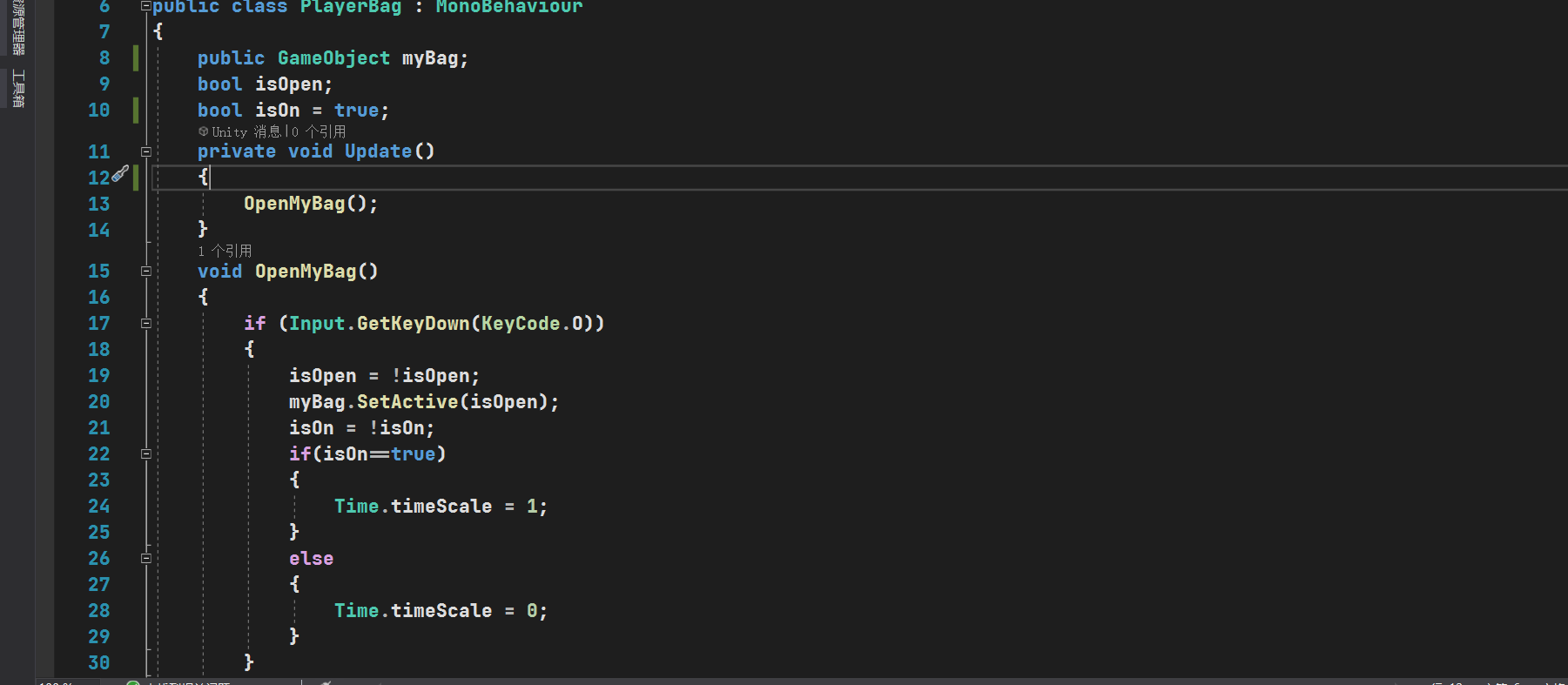
然后对于这些世界中的消耗品,创建脚本itemOnWorld并挂在预制体上

对于这些物体,当它们与玩家发生碰撞时,它们会消失并添加到玩家背包中.所以碰撞时用Destroy函数销毁物品并将这些物品的Colider设置为Trigger. 若玩家背包中已经存在这些物品,就将它们的数量+1,若不存在i就在玩家背包中加入新物品



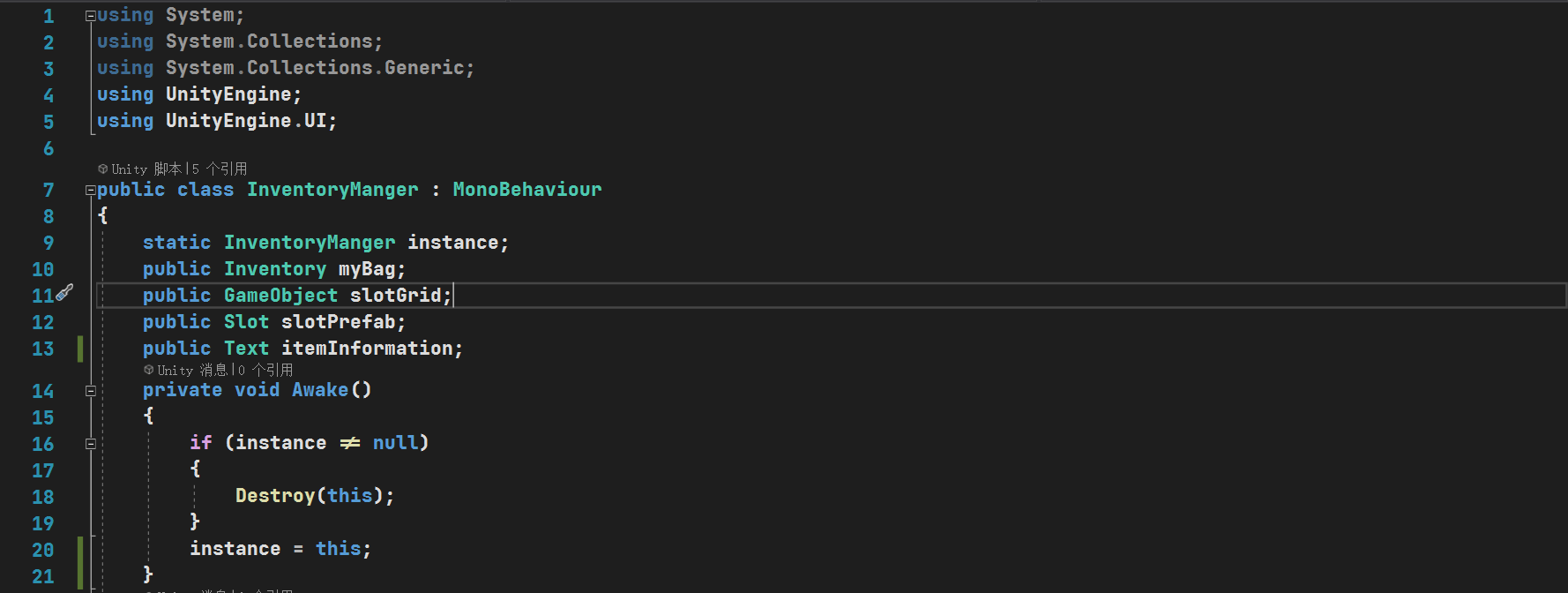
接着我们来创建玩家储存数据背包PlayerBag()

玩家背包主要是控制背包界面打开以及在背包界面打开时时间暂停,运用两个bool变量实现



最后是背包内管理InventoryManger,挂在玩家背包上

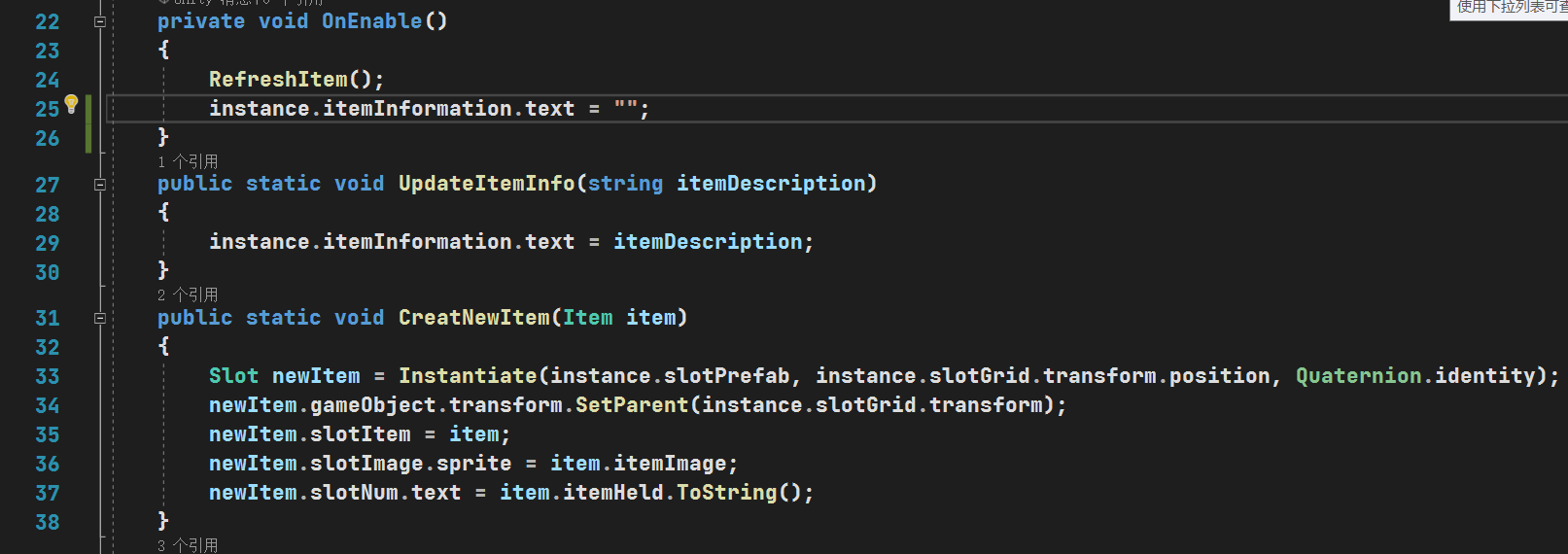
首先声明相应的变量,通过外部拖拽进行赋值并构建单例模式(在游戏中同一时间只会存在一个InventoryManger)

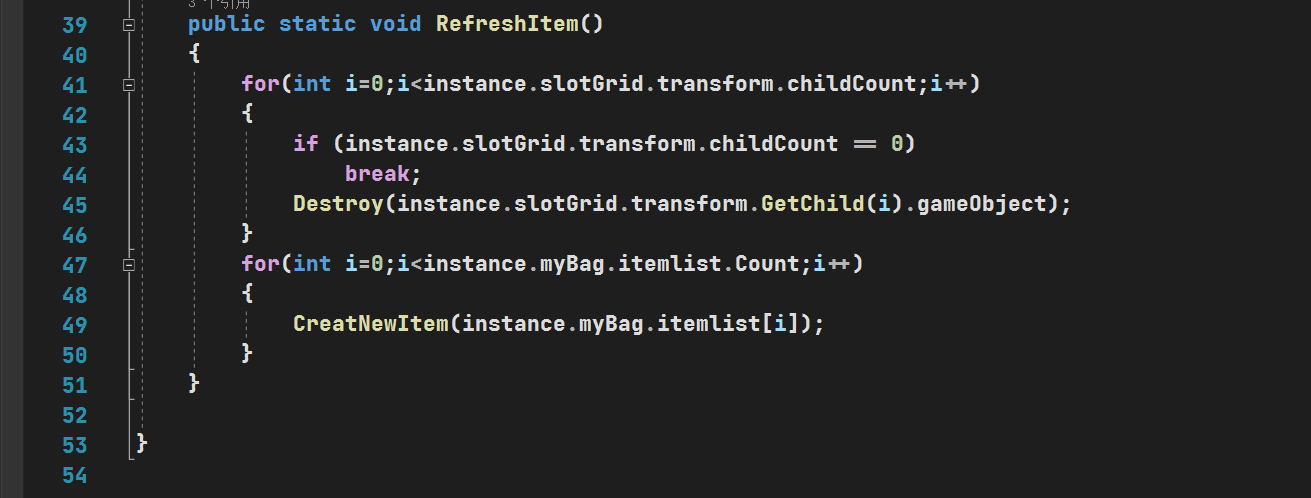


在每次背包启动时刷新背包并清空物品信息栏中相关信息(RefreshItem)

在刷新背包时,首先通过for循环遍历列表中每一个物体并将它们的物体与子物体删除,本部完成后将它们再按照本地Inventory中储存的顺序通过CreatNewItem创造出新物体.

CreatNewItem的作用是在相应的位置进行一次实例化,将选中的继承自Item类的物体的预制体实例化到相应的位置(背包格子中)

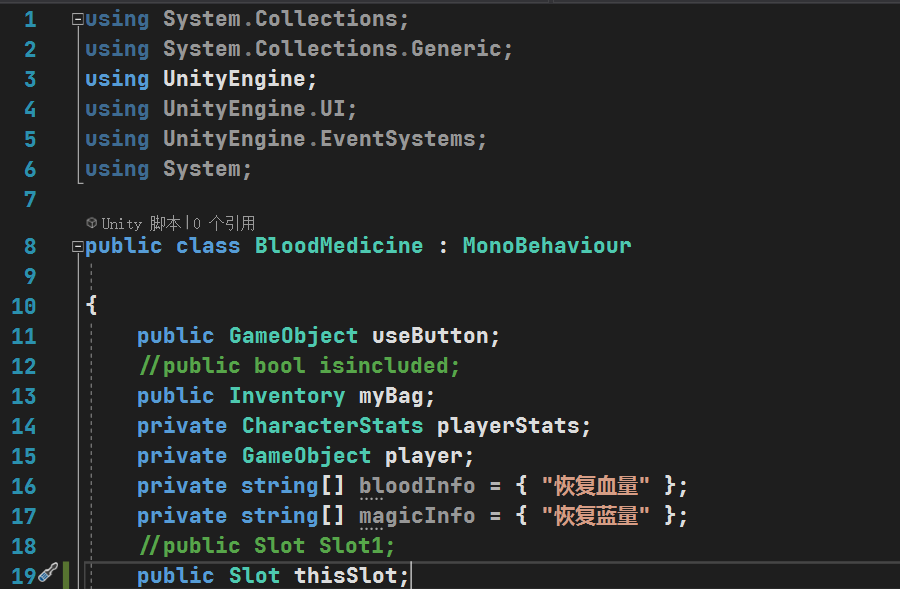




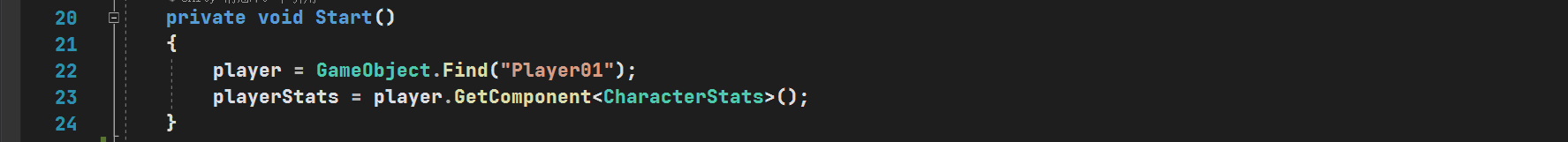
* 1. 使用物品并达到相应效果

我们在本项目种创建的预制体类型均为Button,故均能通过点击的方式进行事件触发.接下来以血药加血功能的实现为例,详解物品使用功能的实现

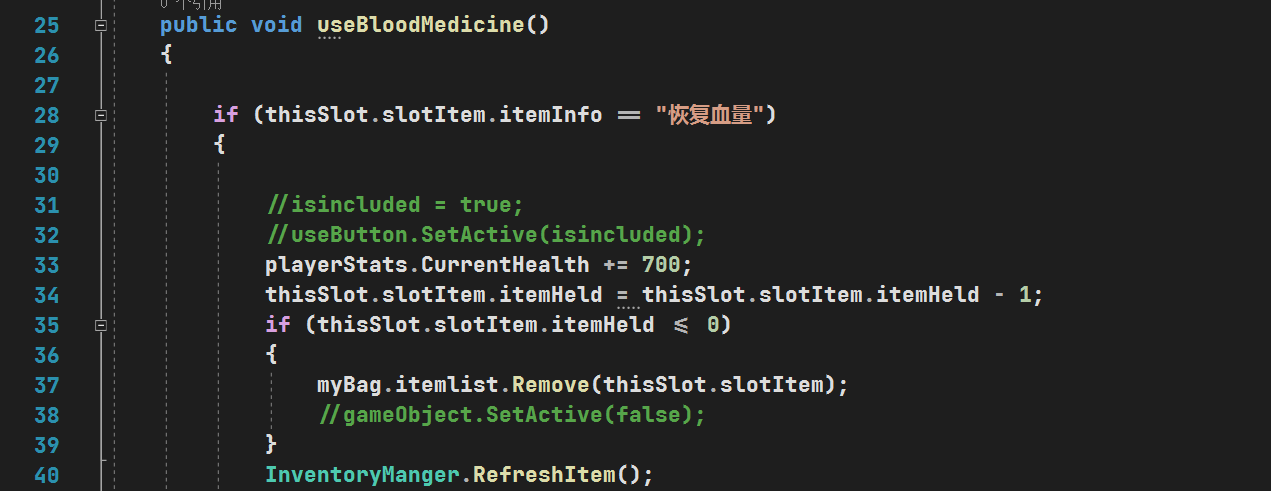
首先,声明相应的变量并通过拖拽对相应变量进行赋值.



开始时, 将获取主角数据并将其赋值给相应变量

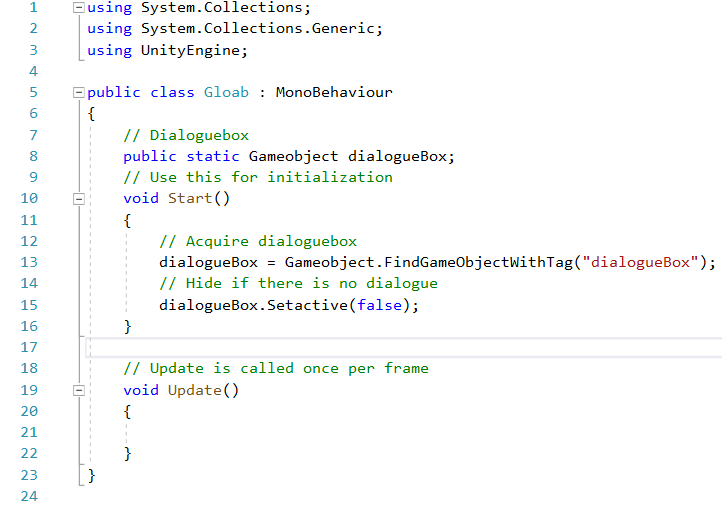


设置点击事件.首先进行判断,若背包中有血瓶,则血瓶数量减一,若血瓶数量小于等于0,则从背包列表中移除血瓶.同时,将人物的血量数值增加700,若达到上限则不再继续增加.同时在使用血瓶后刷新背包,确保数量及时变更.

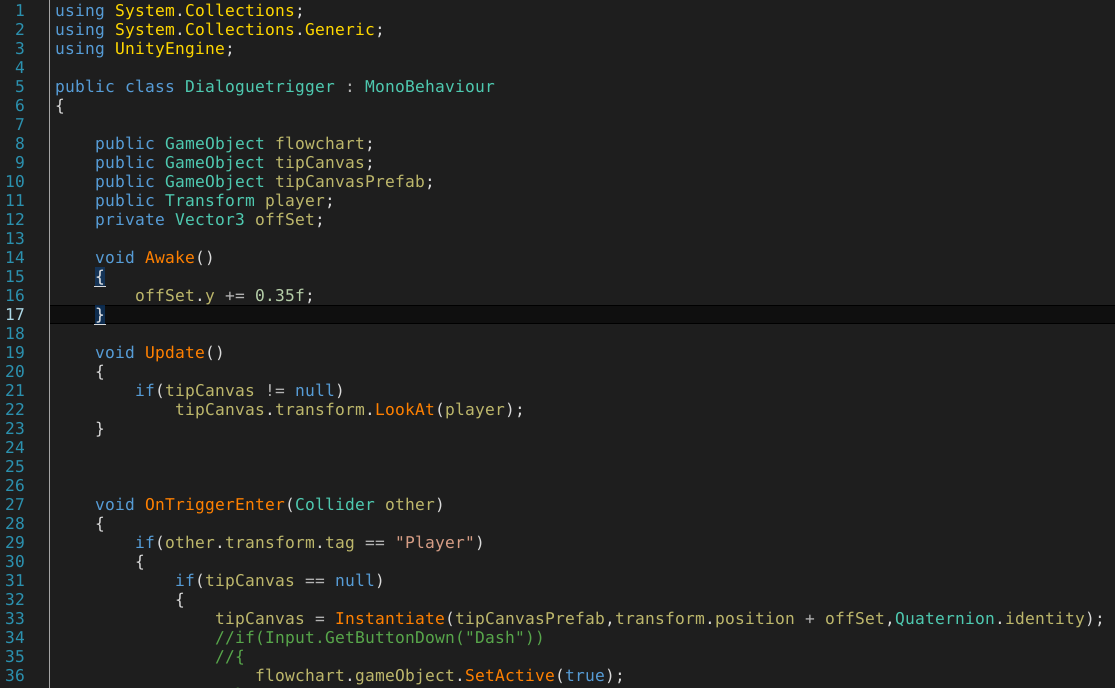


1. Quest&Dialogue
   1. 对话框的生成

首先生成一个xml文件作为框架，用于装载对话内容，随后再编写函数生成对话框，并调用文件中的文本内容。

然后再将对话框与场景中角色相衔接。

* 1. 触发器的设计

先在游戏场景里构建一个碰撞模型Collider，编写触发函数，即主角触及Collider时弹出选项按钮，点击按钮生成对话框，同时引用向量矫正主角与NPC的位置，使每一帧画面中NPC都看向主角。

1. 其他
   1. 传送功能（传送门配对）涉及三个脚本
      1. Scene Controller，代码量42

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.AI;

public class SceneController : Singleton<SceneController >

{

GameObject player;

NavMeshAgent playerAgent;

public void TransitionToDestination(TransitionPoint transitionPoint)

{

switch (transitionPoint.transitionType)

{

case TransitionPoint.TransitionType.SameScene:

StartCoroutine(Transition(SceneManager.GetActiveScene().name, transitionPoint.destinationTag));

break;

case TransitionPoint.TransitionType.DifferentScene:

break;

}

}

IEnumerator Transition(string sceneName,TransitionDestination.DestinationTag destinationTag)

{

player = GameManager.Instance.playerStats.gameObject;

playerAgent = player.GetComponent<NavMeshAgent>();

playerAgent.enabled = false;

player.transform.SetPositionAndRotation(GetDestination(destinationTag).transform.position, GetDestination(destinationTag).transform.rotation);

playerAgent.enabled = true;

yield return null;

}

private TransitionDestination GetDestination(TransitionDestination.DestinationTag destinationTag)

{

var entrances = FindObjectsOfType<TransitionDestination>();

for (int i=0 ; i<entrances.Length ; i++)

{

if (entrances[i].destinationTag == destinationTag)

return entrances[i];

}

return null;

}

}

* + 1. TransitionPoint，代码量47

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SaveManager : Singleton<SaveManager>

{

protected override void Awake()

{

base.Awake();

DontDestroyOnLoad(this);

}

void Update()

{

if(Input.GetKeyDown(KeyCode.S))

{

SavePlayerData();

}

if (Input.GetKeyDown(KeyCode.L))

{

LoadPlayerData();

}

}

public void SavePlayerData()

{

Save(GameManager.Instance.playerStats.characterData, GameManager.Instance.playerStats.characterData.name);

}

public void LoadPlayerData()

{

Load(GameManager.Instance.playerStats.characterData, GameManager.Instance.playerStats.characterData.name);

}

public void Save(Object data,string key)

{

var jsonData = JsonUtility.ToJson(data,true );

PlayerPrefs.SetString(key, jsonData);

PlayerPrefs.Save();

}

public void Load(Object data, string key)

{

if(PlayerPrefs.HasKey(key))

{

JsonUtility.FromJsonOverwrite(PlayerPrefs.GetString(key), data);

}

}

}

* + 1. TransitionDestination，代码量12

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class TransitionDestination : MonoBehaviour

{

public enum DestinationTag

{

ENTER,A,B,C

}

public DestinationTag destinationTag;

}

* 1. UI相关（只涉及人物血条和敌人血条）
     1. 人物血条，代码量63

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class PlayerHealthUI : MonoBehaviour

{

Text levelText;

Text Health;

Text Exp;

Text Endurance;

Text Mana;

Image healthSlider;

Image manaSlider;

Image enduranceSlider;

Image expSlider;

void Awake()

{

Health = transform.GetChild(3).GetComponent<Text>();

Exp = transform.GetChild(4).GetComponent<Text>();

levelText = transform.GetChild(2).GetComponent<Text>();

healthSlider = transform.GetChild(0).GetChild(0).GetComponent<Image>();

expSlider = transform.GetChild(1).GetChild(0).GetComponent<Image>();

Endurance = transform.GetChild(6).GetComponent<Text>();

enduranceSlider = transform.GetChild(5).GetChild(0).GetComponent<Image>();

Mana = transform.GetChild(8).GetComponent<Text>();

manaSlider = transform.GetChild(7).GetChild(0).GetComponent<Image>();

}

void Update()

{

levelText.text = "Level " + GameManager.Instance.playerStats.characterData.currentLevel.ToString("00");

Health.text = GameManager.Instance.playerStats.CurrentHealth +"/"+ GameManager.Instance.playerStats.MaxHealth;

Exp.text = GameManager.Instance.playerStats.characterData.currentExp + "/" + GameManager.Instance.playerStats.characterData.nowExp;

Endurance.text = (int)GameManager.Instance.playerStats.CurrentEndurance +"/"+ GameManager.Instance.playerStats.MaxEndurance;

Mana.text = GameManager.Instance.playerStats.characterData.currentMana + "/" + GameManager.Instance.playerStats.characterData.maxMana;

UpdateHealth();

UpdateExp();

UpdateEndurance();

UpdateMana();

}

void UpdateHealth()

{

float sliderPercent = (float)GameManager.Instance.playerStats.CurrentHealth / GameManager.Instance.playerStats.MaxHealth;

healthSlider.fillAmount = sliderPercent;

}

void UpdateExp()

{

float sliderPercent = (float)GameManager.Instance.playerStats.characterData.currentExp / GameManager.Instance.playerStats.characterData.nowExp;

expSlider.fillAmount = sliderPercent;

}

void UpdateMana()

{

float sliderPercent = (float)GameManager.Instance.playerStats.characterData.currentMana / GameManager.Instance.playerStats.characterData.maxMana;

manaSlider.fillAmount = sliderPercent;

}

void UpdateEndurance()

{

float sliderPercent = (float)GameManager.Instance.playerStats.CurrentEndurance / GameManager.Instance.playerStats.MaxEndurance;

enduranceSlider.fillAmount = sliderPercent;

}

}

* + 1. 敌人血条，代码量56

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class HealthBarUI : MonoBehaviour

{

public GameObject healthUIPrefab;

public Transform barPoint;

public bool alwaysVisible;

public float visibleTime;

private float timeLeft;

Image healthSlider;

Transform UIbar;

public Transform cam;

CharacterStats currentStats;

void Awake()

{

currentStats = GetComponent<CharacterStats>();

currentStats.UpdateHealthBarOnAttack += UpdateHealthBar;

}

void OnEnable()

{

foreach(Canvas canvas in FindObjectsOfType<Canvas>())

{

if (canvas.renderMode == RenderMode.WorldSpace)

{

UIbar = Instantiate(healthUIPrefab, canvas.transform).transform;

healthSlider = UIbar.GetChild(0).GetComponent<Image>();

UIbar.gameObject.SetActive(alwaysVisible);

}

}

}

private void UpdateHealthBar(int currentHealth,int maxHealth)

{

if (currentHealth <= 0)

Destroy(UIbar.gameObject);

UIbar.gameObject.SetActive(true);

timeLeft = visibleTime;

float sliderPercent = (float)currentHealth / maxHealth;

healthSlider.fillAmount = sliderPercent;

}

void LateUpdate()

{

if(UIbar !=null )

{

UIbar.position = barPoint.position;

UIbar.forward = -cam.forward;

if (timeLeft <= 0 && !alwaysVisible)

UIbar.gameObject.SetActive(false);

else

timeLeft -= Time.deltaTime;

}

}

}

* 1. 单例模式运用——涉及泛型单例构建和Manager搭建
     1. 泛型单例，便于构建Manager，减少重复代码，代码量28

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Singleton<T> : MonoBehaviour where T:Singleton <T >

{

private static T instance;

public static T Instance

{

get { return instance; }

}

protected virtual void Awake()

{

if (instance != null)

Destroy(gameObject);

else

instance = (T)this;

}

public static bool IsInitialized

{

get { return instance != null; }

}

protected virtual void OnDestroy()

{

if (instance == this)

instance = null;

}

}

* + 1. GameManager——控制广播者模式，代码量30

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GameManager : Singleton<GameManager>

{

public CharacterStats playerStats;

List<IEndGameObserver> endGameObservers = new List<IEndGameObserver>();

public void RegisterPlayer(CharacterStats player)

{

playerStats = player;

}

public void AddObserver(IEndGameObserver observer)

{

endGameObservers.Add(observer);

}

public void RemoveObserver(IEndGameObserver observer)

{

endGameObservers.Remove(observer);

}

public void NotifyObservers()

{

foreach (var observer in endGameObservers )

{

observer.EndNotify();

}

}

}

* + 1. SaveManager——控制信息存储，主要涉及人物相关数据，代码量45

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SaveManager : Singleton<SaveManager>

{

protected override void Awake()

{

base.Awake();

DontDestroyOnLoad(this);

}

void Update()

{

if(Input.GetKeyDown(KeyCode.S))

{

SavePlayerData();

}

if (Input.GetKeyDown(KeyCode.L))

{

LoadPlayerData();

}

}

public void SavePlayerData()

{

Save(GameManager.Instance.playerStats.characterData, GameManager.Instance.playerStats.characterData.name);

}

public void LoadPlayerData()

{

Load(GameManager.Instance.playerStats.characterData, GameManager.Instance.playerStats.characterData.name);

}

public void Save(Object data,string key)

{

var jsonData = JsonUtility.ToJson(data,true );

PlayerPrefs.SetString(key, jsonData);

PlayerPrefs.Save();

}

public void Load(Object data, string key)

{

if(PlayerPrefs.HasKey(key))

{

JsonUtility.FromJsonOverwrite(PlayerPrefs.GetString(key), data);

}

}

}

* 1. 广播者模式接口，代码量4

public interface IEndGameObserver

{

void EndNotify();

}

* 1. 扩展方法，代码量40

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public static class ExtensionMethod

{

private const float OA\_dotThreshold = 0.5f;

private const float SA\_dotThreshold = 0.2f;

private const float OS\_dotThreshold = 0.6f;

public static bool OA\_IsFacingTarget( this Transform transform,Transform target)

{

var vectorToTarget = target.position - transform.position;

vectorToTarget.Normalize();

float dot = Vector3.Dot(transform.forward, vectorToTarget);

return dot >= OA\_dotThreshold;

}

public static bool OS\_IsFacingTarget(this Transform transform, Transform target)

{

var vectorToTarget = target.position - transform.position;

vectorToTarget.Normalize();

float dot = Vector3.Dot(transform.forward, vectorToTarget);

return dot >= OS\_dotThreshold;

}

public static bool SA\_IsFacingTarget(this Transform transform, Transform target)

{

var vectorToTarget = target.position - transform.position;

vectorToTarget.Normalize();

float dot = Vector3.Dot(transform.forward, vectorToTarget);

return dot >= SA\_dotThreshold;

}

public static bool boss\_IsFacingTarget(this Transform transform, Transform target, float dotResult )

{

var vectorToTarget = target.position - transform.position;

vectorToTarget.Normalize();

float dot = Vector3.Dot(transform.forward, vectorToTarget);

return dot >= dotResult;

}

}

* 1. 动画状态机相关脚本
     1. 攻击时硬直实现——三个脚本，基本上通过最简单的开关RM操控，敌人则直接控制Agent组件
        1. RM

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RootMotionHandler : StateMachineBehaviour

{

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.SetBool("canRootMotion",false);

}

// OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks

override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.SetBool("canRootMotion",false);

}

// OnStateExit is called when a transition ends and the state machine finishes evaluating this state

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.SetBool("canRootMotion",true);

}

// OnStateMove is called right after Animator.OnAnimatorMove()

//override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that processes and affects root motion

//}

// OnStateIK is called right after Animator.OnAnimatorIK()

//override public void OnStateIK(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that sets up animation IK (inverse kinematics)

//}

}

* + - 1. RMRotation

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class RotateHandler : StateMachineBehaviour

{

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.SetBool("canRotate",false);

}

// OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks

override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.SetBool("canRotate",false);

}

// OnStateExit is called when a transition ends and the state machine finishes evaluating this state

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.SetBool("canRotate",true);

}

// OnStateMove is called right after Animator.OnAnimatorMove()

//override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that processes and affects root motion

//}

// OnStateIK is called right after Animator.OnAnimatorIK()

//override public void OnStateIK(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that sets up animation IK (inverse kinematics)

//}

}

* + - 1. Agent

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class StopAgent : StateMachineBehaviour

{

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.GetComponent<NavMeshAgent>().isStopped = true;

}

// OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks

override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.GetComponent<NavMeshAgent>().isStopped = true;

}

// OnStateExit is called when a transition ends and the state machine finishes evaluating this state

//override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// animator.GetComponent<NavMeshAgent>().isStopped = false ;

//}

// OnStateMove is called right after Animator.OnAnimatorMove()

//override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that processes and affects root motion

//}

// OnStateIK is called right after Animator.OnAnimatorIK()

//override public void OnStateIK(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that sets up animation IK (inverse kinematics)

//}

}

* + 1. 连击判定——敌人Player共用逻辑

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Combo : StateMachineBehaviour

{

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

//override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

//

//}

// OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks

//override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

//

//}

// OnStateExit is called when a transition ends and the state machine finishes evaluating this state

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.SetBool("isAttacking",false);

}

// OnStateMove is called right after Animator.OnAnimatorMove()

//override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that processes and affects root motion

//}

// OnStateIK is called right after Animator.OnAnimatorIK()

//override public void OnStateIK(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that sets up animation IK (inverse kinematics)

//}

}

* 1. 武器实例化数据（Scriptable Object）

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "New Weapon",menuName = "Item/Weapon")]

public class WeaponItem\_SO : Item\_SO

{

public GameObject Weapon;

[Header("Attack Info")]

public float damageMultiplier;

public float damageMagicMultiplier;

public float damagePostureMultiplier;

public float enduranceMultiplier;

}

1. **测试（Test）**

详见视频