G1说明文档

1. 布娃娃角色系统(https://deepmirror.atlassian.net/wiki/spaces/PG/pages/1590460428)
    a. 整体结构

        一般的关节结点结构包含（Collider、Rigidbody、ConfigurableJoint）

        i. 关节结点刚体相关设置

            关节结点的刚体质量设置部分参考了这份paper"https://xbdev.net/misc_demos/demos/fight_characters/paper.pdf"

            如果后续考虑不用目前这套物理动画结合的方案，则可以去除布娃娃角色的本体刚体，然后参考这份paper的布娃娃角色身体平衡算法和布娃娃角色在不同pose下的关节角度值的设置。

        ii. 身体骨骼

            1. Animator骨骼

                用于获取Humanoid AnimClip数据的动画骨骼

                另外，Unity只有Humanoid动画支持Unity原生的IK和动画重定向。目前基本上只有布娃娃角色的道具系统有使用到IK功能，下面有说到。（如果使用finalIK等插件或自己写一套IK系统，那么动画骨骼类型可以不受限制）

            2. Ragdoll骨骼

                物理骨骼，实际与整体游戏世界交互的骨骼，所有关节的旋转数据targetRotation都是由原始动画数据计算得到（不同物理关节结点的约束空间需要配置不同的值：即Axis与SecondaryAxis的设置，每个物理关节的结点会有所不同）

                P.S.:特别要注意的是，如果模型有修改，或者模型结点有增删，基本必定会需要重新指定每个物理关节结点的约束空间即Axis与SecondaryAxis的值(具体设置为什么值，需要根据实际自行计算)

            3. Generic骨骼

                用于获取Generic AnimClip数据的动画骨骼。之所有添加一套Generic数据，是因为，有些骨骼的动画数据在Humanoid骨骼中是获取不到的，以及有些Scale形变的动画，是不能直接作用在Ragdoll物理骨骼上的，会破坏关节间模拟的稳定性。因为需要这套Generic骨骼来读取这些骨骼的动作数据。

            4. Render骨骼

                整合了Animator骨骼、Ragdoll骨骼、Generic骨骼的数据，最后用于渲染的骨骼。

        iii. 初始化

布娃娃角色初始化时，记录了所有关节结点初始化时localRotation，用于根据运行时该关节结点在播放不同动画片段动画帧时的实时localRotation，计算出对应物理骨骼的同名结点的targetRotation的精确值。

另外，初始化时，为了使布娃娃角色的整体物理模拟的稳定性更强，布娃娃角色的物理关节中所有相邻的关节的碰撞都被忽略了。根据表现需求，还可以自行对其他会影响动作或稳定性的关节间的物理交互进行自定义设置。

b. 相关配置表

i. R_布娃娃动画片段帧事件表

为了使动作事件的执行更精确，用于在角色动画片段中插入各种动画事件。之后也可还原为最初动画与事件的逻辑完全分离的做法(只是需要精确计算出每个事件的实际播放时间填入配置表)。

相关脚本：AnimationEventCall.cs

ii. R_布娃娃动作事件表

诸如走跑跳、飞踢、头槌、越障等布娃娃行为的参数调整。

iii. R_布娃娃动作数据表

目前只有两个动作，两个双手持有道具的动作，为得是在拾取双手道具时，双手的动作状态都尽量逼近美术的原始动作数据。（用了新的双手道具拾取方案后，感觉可以不需要这个功能了）

iv. R_布娃娃额外骨骼表

目前有两种额外骨骼:

第一种是前面几列，bone类型的骨骼（作用是将bone的动作数据应用到Render渲染骨骼上，bone类型骨骼的数据只存在于Generic动画片段上）；

第二种是额外的物理骨骼，因为之前没有同步额外的物理骨骼，导致出现这些物理骨骼在客户端的表现有些抖动，所以目前这些额外的物理骨骼也都是在服务端模拟，然后同步关节的targetRotation数据到客户端进行渲染表现的。

额外骨骼的制作方式：直接对布娃娃角色的预制prefab进行操作。额外物理骨骼的最上层的物理结点必须绑定在布娃娃身体的某个物理关节结点上，然后根据需求设置该额外部件的所有关节的参数，基本上用的也都是ConfigurableJoint（可自由配置想要的效果）。

相关脚本：ragdollrendermanager、ragdollextrasyncpart

v. R_布娃娃关节角度限制表

对布娃娃所有身体结点的关节角度限制模式及具体的限制值进行设置，angularXMotion、angluarYMotion、angluarZMotion(关节旋转轴的限制)，包含三种模式，Locked(不能旋转)、Limited(有角度限制，可以指定范围)、Free(自由无限制)。

相关脚本：ragdollplayercontroller

vi. R_布娃娃关节驱动力表（https://deepmirror.atlassian.net/wiki/spaces/PG/pages/1590460445）

用于模拟推动布娃娃关节运动的主要力度参数（具体参数作用，可参照Unity的官方文档https://docs.unity3d.com/cn/current/Manual/class-ConfigurableJoint.html）

相关脚本：ragdollplayercontroller、ragdollactioncontroller

vii. R_布娃娃行为驱动表

对于不同动作行为，结合了多种布娃娃关节驱动力，用于创造出不同的动作表现。

主要函数：RagdollActionController:SetRagdollActionDriveMode(joint_action_id, delay_callback, instanly, deltaLerpTime)

在需要的位置调用上述函数，instanly == true代表即刻生效，instanly == false时需要设置deltaLerpTime，那么行为与行为间的物理模拟状态表现将类似二次贝塞尔曲线的缓动效果平滑过渡到目标状态。

相关脚本：ragdollactioncontroller

c. 编辑器制作工具

i. DM/Tools/GenrateRagdoll(第一步：选中美术原始模型执行)

用美术制作的初始模型骨骼生成布娃娃角色（含整体结构中的四套骨骼，但Ragdoll物理骨骼不包含任何物理组件信息，生成完成后，可能一些结点的物理Layer还得调整，一些fusion网络的component可能需要手动再添加）

ii. DM/Tools/GenerateRagdollPhysics(第二步：选中生成的布娃娃角色根结点，将原先制作好的物理骨骼改名为RagdollClone放在布娃娃角色的根结点下)

将之前制作好的物理组件按结点名称复制到新生成的布娃娃角色的物理骨骼上

iii. Assets/添加Player NameTableNode(最后一步)

上面都执行完了后，为所有骨骼的结点添加上NameTableNode的映射关系，方面使用

d. 布娃娃角色使用或包含的网络相关组件与结点

i. NetworkObject

fusion进行网络同步的必要组件，唯一NetworkId

ii. FusionCustomNetworkMecanimAnimator

之所以重写了NetworkMecanimAnimator => FusionCustomNetworkMecanimAnimator，是因为之前的fusion版本的NetworkMecanimAnimator不满足Server & Client或Host & Client其中一种的同步需求（具体忘了是哪一种），所以重写了部分结构体和该组件，实际改动很小。

iii. NetworkRigidbody

fusion进行物理刚体的同步组件

iv. FusionNetworkProperties

自定义的网络属性同步组件，根据需求和类型添加需要同步的数据

v. Root/InterpolationTarget

目标插值结点，Ragdoll物理骨骼的根结点Ragdoll上NetworkRigidbody的InterpolationTarget设置为该结点，客户端上所有玩家主刚体位置信息的插值同步数据，都从该结点获取。

2. 布娃娃角色战斗

a. 战斗攻击

基本上都由OnTriggerEnter或OnCollisionEnter触发

i. OnTriggerEnter

飞踢、头槌等由HitBox触发战斗事件的攻击

ii. OnCollisionEnter

普通攻击，道具攻击，投掷攻击等直接物理碰撞触发战斗事件的攻击

相关脚本：CollisionCallBack.cs、hitbox

b. 战斗交互

i. 拖拽

会在玩家布娃娃角色的双手物理结点上，与被拖拽的Object创建ConfigurableJoint，来实现拖拽的效果。

相关脚本：ragdollhandcontroller

ii. 举起玩家

暂未实现，基于拖拽，实现原理并不难，有些状态的转换或判断会复杂一点

实现原理：也是双手伸向被抓玩家，通过碰撞检测判断是否抓取到被抓玩家的身体关节，创建ConfigurableJoint。同时，自身进入举起玩家状态，设置双手的关节行为、关节力度、需要播放的动作蹬，被抓取玩家进入举起状态等等。

iii. 攀爬

暂未实现，同样可基于拖拽，进入的状态以及播放的动画与举起玩家不同

3. 布娃娃角色道具

a. 道具拾取

i. IK拾取

目前使用到的基本上都是Unity自带的简易IK接口，复杂的IK逻辑和系统基本都还没有实现，如果有需求需要自行实现

对于武器类型的道具，布娃娃角色会弯腰（HeadIK），然后进行拾取的那只手会伸向道具(HandIK)的绑定位置，如果与该道具的距离判定满足拾取要求，那么道具拾取成功，否则道具拾取失败（因为人物有可能处于高速移动状态，按下拾取键也有可能拾取不到）

相关脚本：ragdollikcontroller、AnimatorIKController.cs

ii. 非IK拾取

拾取后，人物不会做出任何相关IK的操作动作，道具或相关物品直接挂在布娃娃角色的某个物理结点上

相关脚本：ragdollitemcontroller、itembase

b. 道具挂点

i. 单手道具

IK拾取判定成功后，将瞬移绑定在布娃娃角色的某个物理关节结点上，在该道具的配置表上有配置。

ii. 双手道具

第一只手的拾取判定逻辑与单手道具一模一样，重点在第二只手的拾取设置。

目前第一只手拾取成功后，第二只手也会同时与道具的刚体进行绑定，通过创建一个x,y,z轴移动限制为Limited模式的ConfigurableJoint。创建的同时，取消勾选AutoConfigureConnectedAnchor，并清零Anchor与ConnectedAnchor两个向量。接着设置ConnectedAnchor的值为双手道具在配置表中的ConnectedAnchor参数的值。设置好x,y,z轴移动限制Limited模式下的LinearLimit限制值，缓慢过渡到0，然后设置x,y,z轴的移动限制模式为Locked，至此，双手道具拾取绑定成功。

双手道具绑定点的确定：将相应道具的拾取idle动作放入场景内部，然后将被拾取的道具设为拾取挂点的子物体，调整道具到想要的位置，保证双手的与该道具的相对位置都满足要求，然后在道具的根结点下创建两个空物体，移动到双手的位置，并分别改名为："lefthand"，"righthand"。这两个位置就是双手道具的左右手绑定点。

双手道具参数ConnectedAnchor的确定：同样把道具的拾取idle动作放入场景内部，然后将拾取道具设为拾取挂点的子物体，给双手都添加上一个静态的刚体，然后双手都添加一个ConfigurableJoint组件。保证两个ConfigurableJoint组件的Anchor与ConnectedAnchor都为0，然后运行场景。在运行时，将道具的刚体赋值给ConfigurableJoint组件，先赋值第一只手的ConfigurableJoint的ConnectedBody，然后赋值给第二个只手的ConfigurableJoint的ConnectedBody。记录下第二只手此时的ConnectedAnchor，即为该双手道具的ConnectedAnchor，填写到道具的配置表中。

相关脚本：clientphysicsmanager、ragdollitemcontroller、itembase

相关函数：ClientPhysicsManager:CreateConfigurableJoint(rig, connected_rig, limit_pos, anchor, limiter, limit_rot)

4. fusion
   a. FusionAssetbundleManager.cs

      Fusion用AB模式加载NetworkPrefabTable中的资源时，管理记录当前加载的AB数量，某些初始化或重置操作需要等待所有AB都卸载完毕才可执行。

   b. FusionBindLua.cs

      用于客户端client在lua层创建与服务端对应的Fusion NetworkObject的lua类

      相关脚本：networkclassdefine

   c. FusionManager.cs

      Fusion的全局管理单例，lua层写得一些ShutDown、掉线Disconnected的逻辑在该脚本中的Fusion回调中执行。

   d. FusionNetworkProperties.cs

      挂载在需要同步数据的NetworkObject上，定义了不同类型的需要同步的网络数据类型。

      如果该NetworkObject在客户端client存在逻辑需要处理，那么lua类名也通过该脚本同步。

      玩家角色的输入数据由该脚本的getInputAction传入lua层进行处理。

   e. FusionNetworkRoomManager.cs

      Fusion Game Room Manager，包含一些当前房间的状态信息和玩家信息。

      目前只支持4人，增加人数需要扩展该脚本的房间角色容量。

      之前版本的fusion因为自定义的RoomPlayer NetworkStruct，在创建List后，无法正确读取到结构体当中的数据，因此写死了4个RoomPlayer。后续可以看看目前支不支持创建List<RoomPlayer> roomPlayers这种方式。

   f. FusionNetworkStart.cs

      Fusion create room game(single mode、sever mode、host mode、client join game等等)

   g. FusionRagdollPartInit.cs

      记录所有关节结点的初始awakeLocalRotation

   h. FusionRpcManager.cs

      Rpc协议管理

   i. PaFusionNetworkInput.cs

      网络角色输入管理

   j. PaObjectSpawner.cs

网络object(ai player、item、sceneobj)的创建销毁管理

k. PaPlayerSpawner.cs

网络角色创建绑定管理，继承自PaSpawnerPrototype

l. PaSpawnerPrototype.cs

角色创建原型

5. 特效

特效的生成与播放针对fusion的server & client做了一些简单的特殊处理。

比如有些需要挂载在fusion networkobejct上的特效，则需要通过带额外传参的rpc接口去调用实现。

6. AI

比较简单，创建AI的过程与玩家角色创建一样，只不过操作输入由简易的ai逻辑读取ai相关配置表自行去处理。

相关目录：game/pa/server/ai/...