### 目录结构

res （引擎资源，取消client、server）

game （存放例子游戏）

sample （目前唯一的一个例子游戏）

lang （多语言，优先级高于media内的）

map （内含多张地图）

plugin （全部插件）

myplugin （目前唯一的插件）

block （方块）

stone （石头方块）

setting.json （配置）

triggers.bts （Trigger-Action脚本，非必须）

stone.png （相关贴图，0张或多张）

……（其它方块，结构同stone）

entity （单位，结构同block）

item （道具，结构同block）

region （区域，结构同block）

skill （技能，结构同block）

missile （子弹，结构同block）

buff （buff，结构同block）

task （任务，结构同block）

reward （奖励，结构同block）

crosshair （准星，结构同block）

setting.json

id\_mappings.json

game\_res （svn库存放正式游戏，非必须）

g2001、g2002、…… （每一个游戏与sample结构相同）

gameconfig.json （测试期间的重要配置）

lua （引擎脚本，所有游戏公用）

common （通用部分）

client （客户端部分）

server （服务端部分）

### 便捷调试

客户端控制台

单机模式（不开服务端即可）

小地图瞬移

debugport

连接本地client、server（F11、Telnet）

连接正式服/测试服的server（Telnet）

命令：

r 热更新（lua+大部分配置）

o 列出所有对象

d 执行（**p** = print、**pv** = Lib.pv、**w** = World.CurWorld、**o** = World.CurWorld:getObject）

a 通过server转接client

l 查文件[绝对路径、相对路径、序号、“.”、“..”]

lua报错vs下双击跳转

### 游戏对象

**场景对象（地图上有的）：**

Block（数据）

Region

Object => DropItem

=> Missile

=> Entity （=> Player）

**非场景对象（地图上没的）：**

Buff、Item、Team、Map、World、Map、……

（Skill可能不算对象）

### Object派生类关系

Logic

EntityServer

（增加AI、伤害计算等）

EntityClient

（增加渲染等功能）

EntityClientMainPlayer

（主角，增加控制、第一人称）

EntityServerPlayer

（增加网络连接、玩家数据等）

Entity

（包含Prop、Buff等功能）

Client

Missile

（C/S不同步）

DropItem

（与Item对应）

Server

Object

（基础碰撞对象）

### Object对象数据解决方案

**配置（Config）**

用于定义一类Object对象的常数值

由配置文件产生，不可运行中修改

定义： C++：ObjectConfig.h、EntityConfig.h、MissileConfig.h

Lua：无统一定义，看具体使用的代码

例如：viewDistance（可视距离）、canHitching（是否可搭乘）、canClick（是否可点击）

**属性（Prop）**

用于定义Entity的当前状态数据

初始值一般来源于配置文件，运行中一般由Buff进行修改

主要用于int和float数据类型，目前还支持集合（table）以及回调（函数）类型

定义： C++：EntityProp.h

Lua：entity.lua（三个） - Entity.EntityProp、游戏配置setting.json - customProp

例如：moveSpeed（移动速度）、damage（伤害）、headStar（头顶星星）

**Lua成员变量**

配置和属性无法满足时采取，常用方式有两种：

Object:data(“xxx”) （数据table块）

Object.xxx （任意数据）

定义：Object:initData()中或任意其它位置初始化

例如：Entity:data(“cfg”)、Entity.map

**C++成员变量**

配置和属性无法满足，且C++内部需要使用（一般有效率问题）

定义：Object.h、Entity.h、……

例如：Object::position、Entity::curHp

**Entity的getValue、setValue封装**

基于前两条Lua/C++成员变量，又加了一层封装，解决同步、初始化、存档等问题

定义：common/entity.lua - Entity.ValueDef

例如：Entity:getValue(“level”)、Entity:setValue(“teamId”)

**Action的GetObjectVar、SetObjectVar封装**

对于Object:data(“vars”)[key] = value的Action层面的封装，解决初始化和存档问题，服务器专用

定义：Entity配置的saveVars

例如：祥见bts填写

### 各方案对比

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 方式 | 实例 | 可变 | 同步 | 存档 | 效率 | 层级 | 类型 |
| 配置（Config） | × | × | 无需 | 无需 | 高 | 用户 | Any |
| 属性（Prop） | √ | Buff | Buff | Buff | 高 | 用户 | 受限 |
| Lua成员变量 | √ | √ | × | × | 中 | 开发 | Any |
| C++成员变量 | √ | √ | × | × | 高 | 开发 | Any |
| getValue | √ | √ | √ | √ | 低 | 开发 | Any |
| GetObjectVar | √ | √ | 无需 | √ | 低 | 用户 | Any |

### 多地图

**概念改变：**

一个World（世界）包含多个Map（地图）

一个对象的完整定位需包含map和pos两项，特别情况允许map为空

方块、region等相关操作都移到了Map对象上，Object总管等还保留在World上

**默认地图：**

setting.json的defaultMap字段设置，默认地图（非必须）。

服务器的World.defaultMap变量保存了默认地图对象，客户端没有。

设置了默认地图的游戏，允许创建npc、dropitem等指令不填写具体地图。

**当前地图：**

客户端当前地图即为摄像机可见地图，一般来说主角就在这个地图上。

服务器没有当前地图概念。

客户端的World.CurMap用于记录当前地图对象。

**静态地图：**

对于某个地图资源，只会加载一份的情况，被称为静态地图。

静态地图多数时候可以忽视其创建过程，直接使用资源名表示地图，第一次使用此地图时会自动加载

非静态地图即地图副本，此情况一份地图资源可以被加载多次，必须用地图对象来表示具体地图。

**地图表达：**

table对象——完整的地图对象，action等上层接口的主要表达方式

string名——地图名称，仅对静态地图有效，一般用于填写配置文件，可用于上层接口

int数值——地图ID，仅用于协议同步、c++函数等底层接口

**获取Map的方法：**

Server：Object.map、World.defaultMap（不配置就不存在）、Trigger的上下文中的map

Client：Object.map、World.CurMap（一般应该存在）

### Buff系统

**主要目的：**

方便改变Prop数值

方便同步、存盘、定时

方便扩展，其它系统可以轻易加入Buff逻辑

**作用方式：**

增加Buff——Prop对应数值做“加法”

等待到达时间或被主动清除

消除Buff——Prop对应数值做“减法”

**待扩展：**

叠加规则

头顶图标、倒计时等显示效果

**应用场合举例：**

所有药瓶效果可以由Buff配置，装备属性由Buff配置

地图的安全区域（Region）配置无敌Buff，人物刚加入战场增加短时无敌Buff

踩到方块的减血、减速等可以由Buff自由扩展

被动技能即永久Buff，还支持存盘

用Buff增加一些“特殊”Prop，从而简化逻辑书写

**相关接口**

buff = entity:addBuff(cfgName, time)  
 entity:removeBuff(buff)

### 技能系统

**主要目的：**

同步、管理玩家的一般性行为

易于开发（利用技能框架开发新的行为）

**逻辑层次：**

SkillBase基类（common、client、server）

→类型派生(Click、MeleeAttack、BreakBlock、PlaceBlock、Missile、……)

→定义（配置文件）“派生”

×无实例对象

**逻辑接口：**

cfg:canCast(packet, from)

判断是否可释放、补充参数

cfg:cast(packet, from)

释放逻辑

cfg:getTouchTime(packet, from)

计算需要长按（吟唱）的时间

cfg:start(packet, from)

长按（吟唱）开始，【客户端】播放动作、特效、音效等

cfg:stop(packet, from)

长按（吟唱）中断，【客户端】停止动作、特效、音效等

cfg:preCast(packet, from)

【客户端】播放“起手”动作，包括CD、动作、特效、音效等

cfg:singleCast(packet, from)

【客户端】单机版直接调用cast

cfg:showIcon(show)

【客户端】显示/隐藏技能图标

**常见技能配置：**

"type":"XXX" （技能类型）

"startAction":"XXX" （起始动作）

"startEffect": { （起始特效）

"effect": "XXX",（特效文件）

"once":true, （只播放一次）

"pos":{...}, （偏移位置）

"yaw":0 （水平旋转角度）

}

"startSound": { （起始音效）

"sound": "XXX",（音效文件）

"loop":false, （循环播放）

"volume":1 （音量）

}

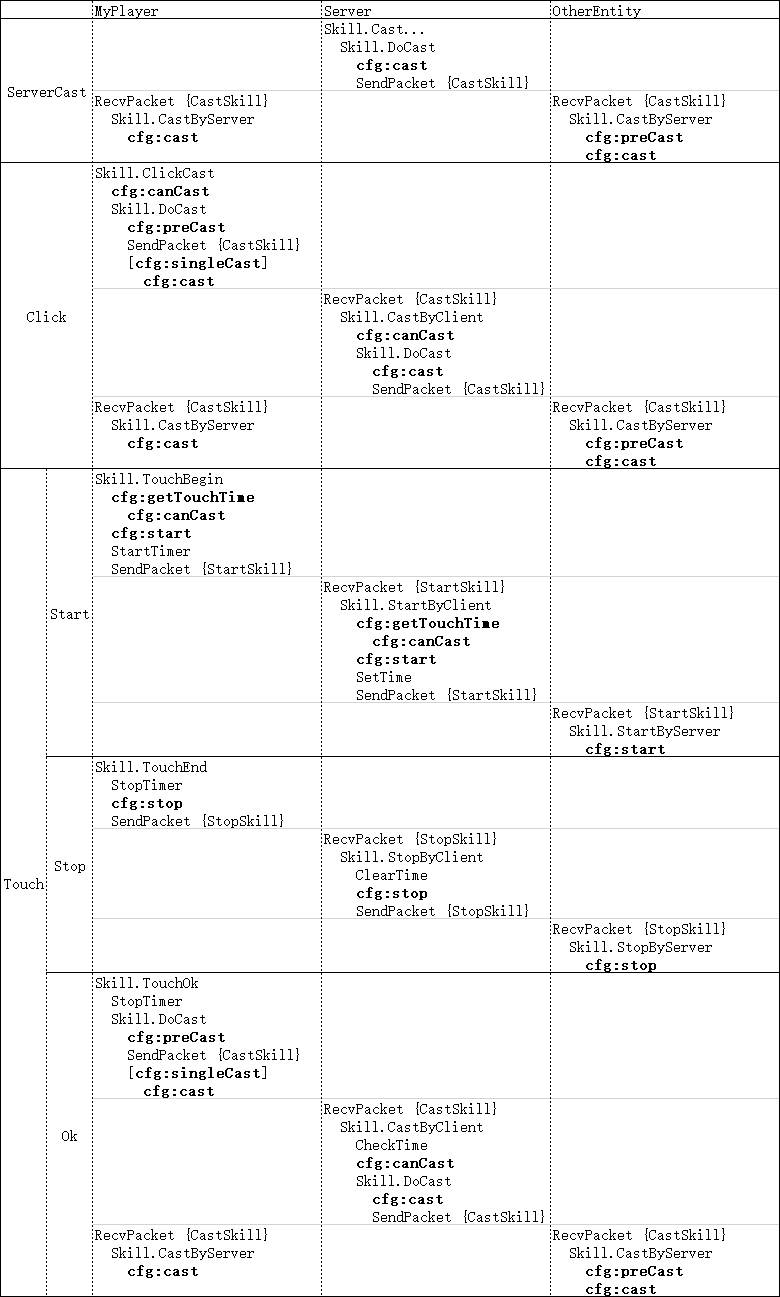
"castAction"、"castEffect"、"castSound" （释放技能的动作、特效、声音）

"icon":”XXX” （技能图标）

"cdTime"、"cdKey" （CD时间和CD类型）

"isClick"、"isTouch"（可否点击释放、可否长按释放）

### 技能调用



### 子弹系统

**主要目的：**

实现一种可以自由“飞行”的载体丰富技能表现

定义多种触发技能，实现可嵌套的技能-子弹-技能逻辑扩充

特别的碰撞检测方法和逻辑实现不同的命中需求

**典型的配置方法：**

配置子弹的**释放技能** => 配置**子弹** => 配置子弹的**触发技能**

**子弹释放技能常用配置项：**

"type":"Missile" （技能类型是释放子弹）

"missileCfg":"XXX" （释放的子弹配置）

"missileCount":1 （子弹数量）

"targetType":"XXX" （子弹目标，常用目标如下）

CameraYaw 镜头水平方向 Camera 镜头方向

BodyYaw 身体水平朝向 Block 选中方块

Entity 选中单位 FrontEntity 前方范围内的一个单位

Self 自身 Any 任意（Entity or Block or CameraYaw）

"startPos":{...} （子弹起始位置）

"startWait":0 （子弹起始延迟）

"startYaw":0 （子弹起始水平偏角）

"startPitch":0 （子弹起始上偏角）

**子弹常用配置：**

"modelMesh"、"modelBlock"、"modelPicture" （外观三选一：模型、方块、图片）

"moveSpeed":0.5 （速度）

"moveAcc":0 （加速度）

"gravity":0.5 （重力加速度）

"rotateSpeed":0 （自旋速度）

"followTarget":false （跟踪目标）

"lifeTime":100 （存在时间）

"vanishTime":0 （消亡等待时间）

"collideBlock":1 （碰撞方块：0.穿透，1.消失，2.反弹，3.阻挡，4.命中）

"collideEntity":1 （碰撞单位：0.穿透，1.命中任何单位，2.仅命中目标）

"hitCount":1 （碰撞数，不填不限制）

"hitInterval":10 （碰撞间隔）

"startSkill"、"startEffect"、"startSound" （起始技能、特效、声音）

"hitSkill"、"hitEffect"、"hitSound" （碰撞技能、特效、声音）

"vanishSkill"、"vanishEffect"、"vanishSound"（消亡技能、特效、声音）

### 整体框架风格

所有功能可配置可扩充，支持不同层次的需求：

仅会摆放方块和单位等

能看懂配置说明，对方块、单位等进行简单的配置修改

可以理解Trigger-Action，会制作美术资源，可以用组织复杂、独特的玩法逻辑

可以脱离编辑器，手写bts文件，可以用电子表格等工具高效编辑配置文件，会使用高级配置功能

熟悉lua脚本编写，理解底层原理，可以团队开发商业游戏

引擎（C++、Lua）中不出现具体游戏需求，只做通用功能

对象/资源的标识方式：

Lua中的静态资源（配置）一般用字符串

在确保安全的前提下，实例对象一般直接用对象（table、指针）

C++对Lua接口以及协议，一般用数字ID表示实例对象

Action要确保安全性，不提供底层有风险的数据直接访问

游戏暂无UI定制的能力，但有通用UI的支持，系统UI也尽量提供较好的配置能力。

也不要过度设计