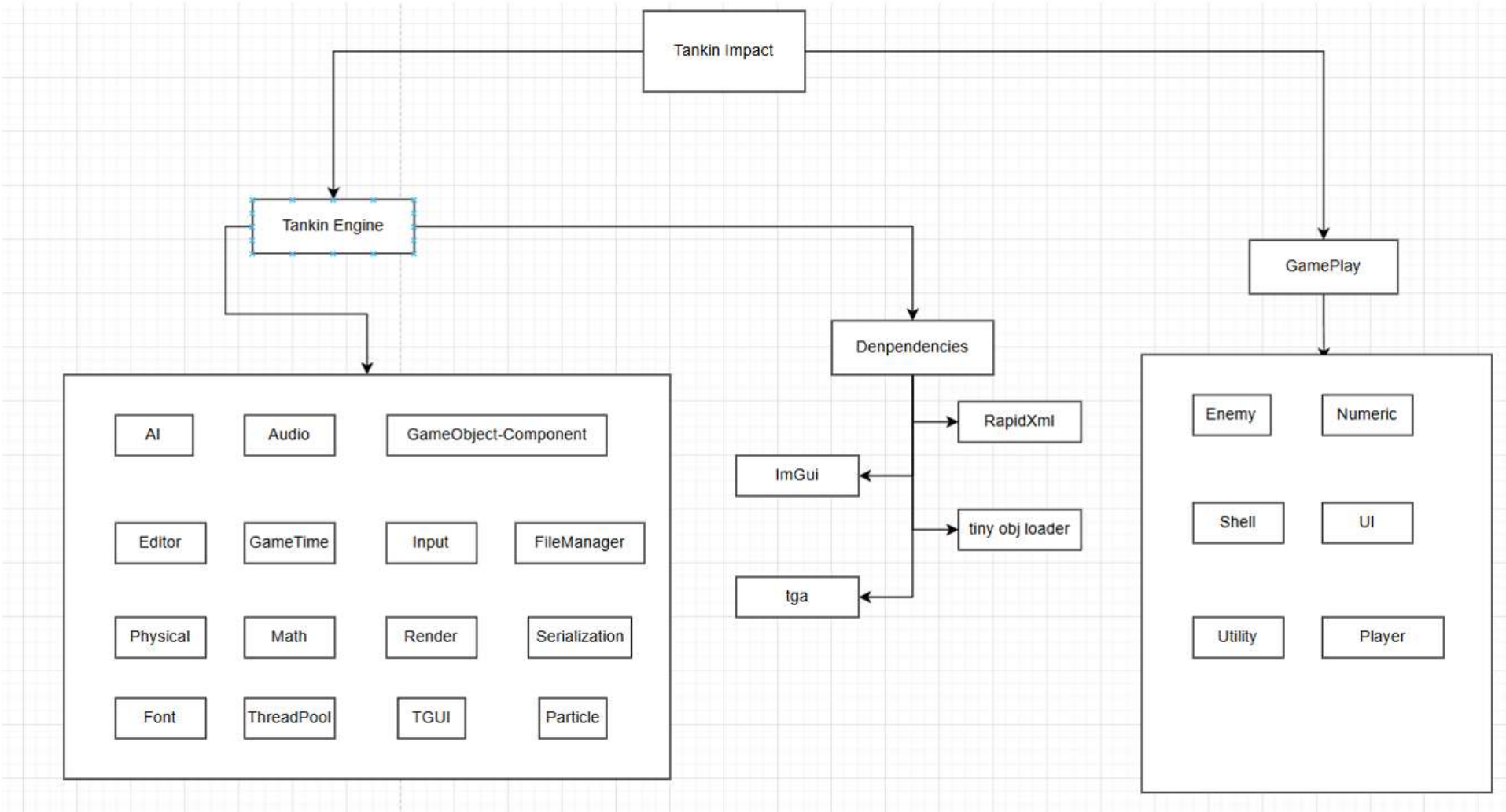
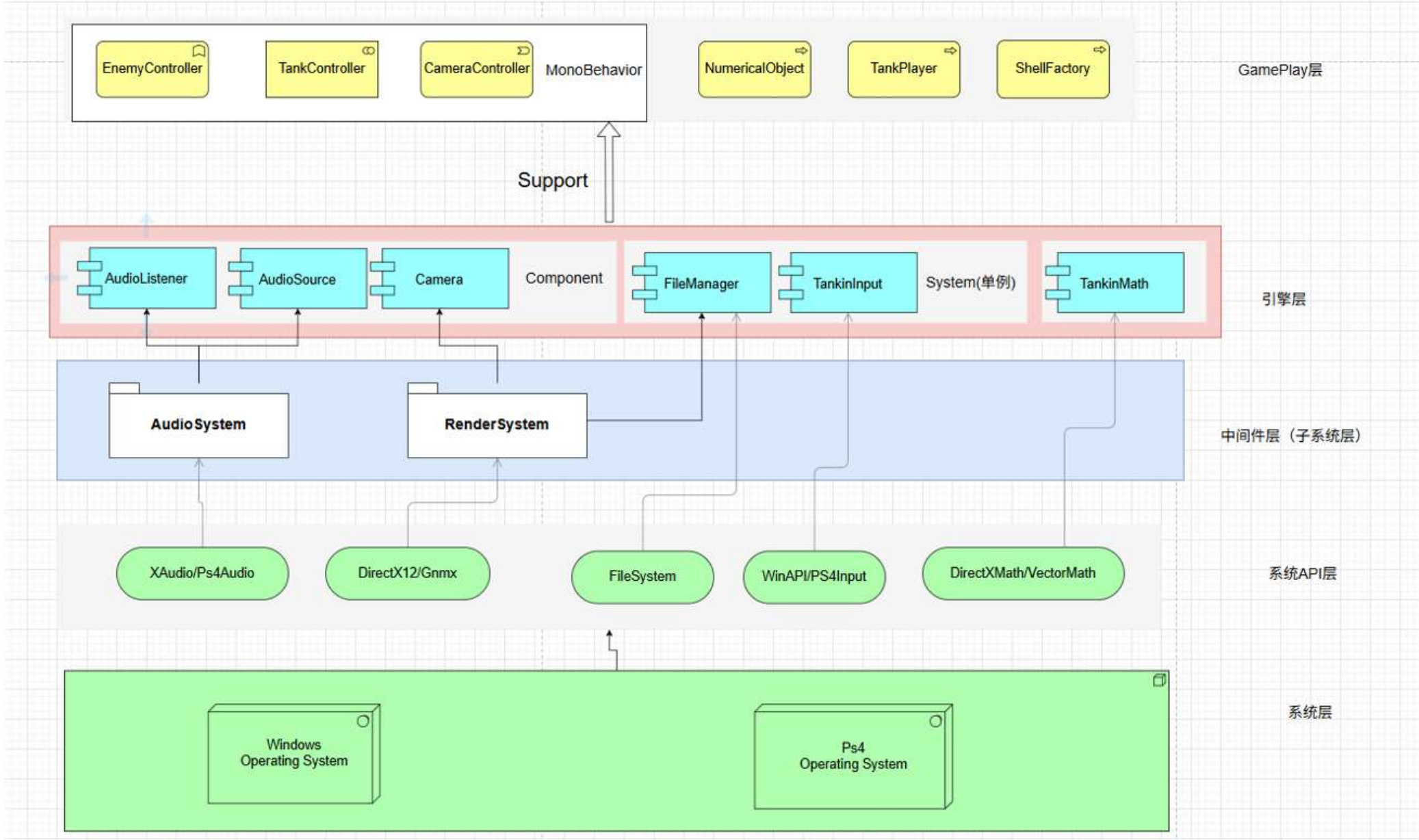
Game Engine Framework

Gameplay Framework

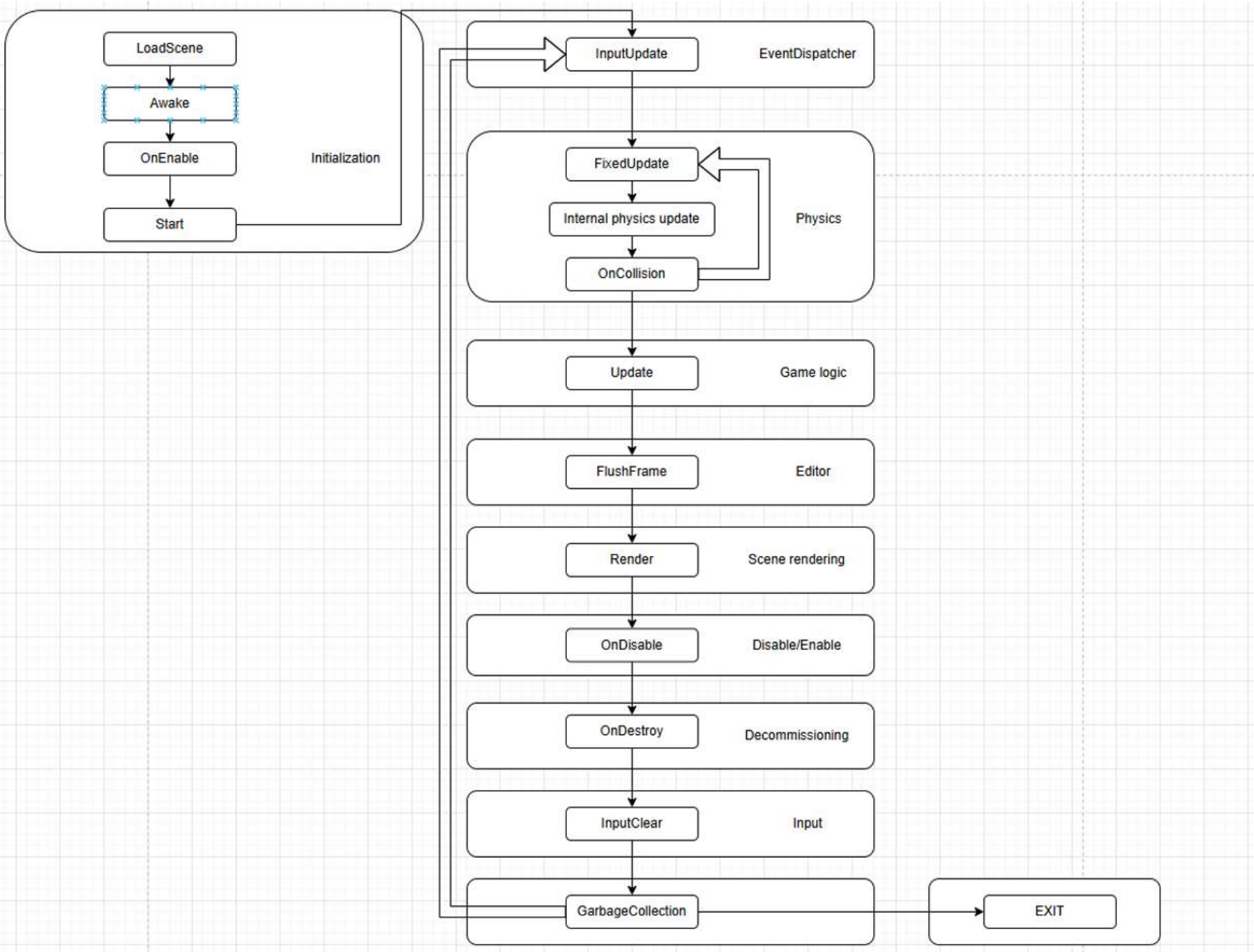
麻雀虽小，五脏俱全



Engine Structure



Game Engine Framework


LoadScene
Awake
OnEnable
Start
Initialization
InputUpdate
EventDispatcher
FixedUpdate
Internal physics update
OnCollision
Physics
Update
Game logic
FlushFrame
Editor
Render
Scene rendering
OnDisable
Disable/Enable
OnDestroy
Decommissioning
InputClear
Input
GarbageCollection
EXIT

Why GameObject-Component?

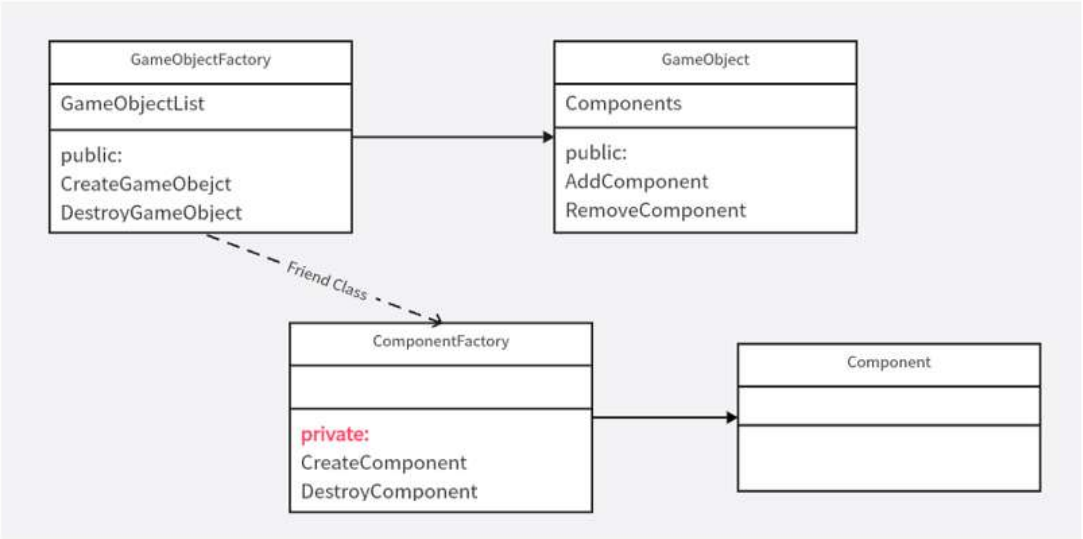
能够在极短的时间内（两个月），保证项目完成功能前提下，拥有很高的稳定性和可扩展性

对象生命周期管理

为什么一定要由引擎管理对象的生命周期?

如何实现由引擎掌控生命周期呢?

1、使用工厂模式
2、构造和析构函数都为私有
3、组件只能通过GameObject来创建和销毁



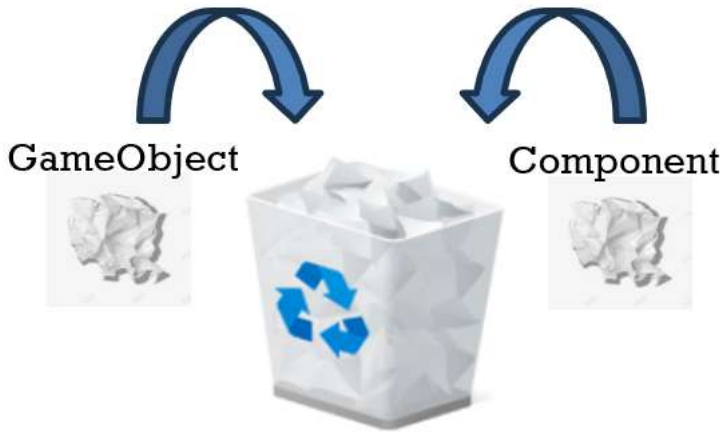
延迟销毁

Call

```
static void sDestroyGameObject(GameObject*& go);
```

```
static void sDestroyComponent(Component* component);
```

此时，对即将要销毁的Go和Component打上要被销毁标记



将这些即将销毁的内容放入GarbageList中

```
static void sGarbageCollect();
```

一定是先对Component进行回收，因为回收时调用OnDestroy可能会关系到GameObject

简单反射的实现

Why Reflection?

1、开发新的Component或MonoBehavior时不需要回去修改ComponentFactory，提高引擎的灵活性

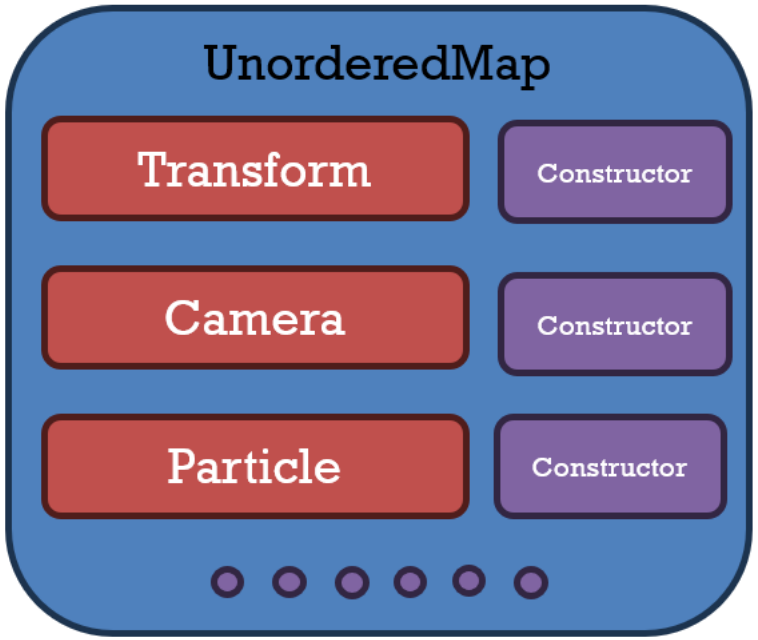
2、避免引擎需要引用GamePlay层的头文件，实现引擎层和GamePlay层的分离

```
if (name == "Transform")
{
    return new Transform();
}
else if (name == "RectTransform")
{
    return new RectTransform();
}
....
```

```
auto registry:TpUnorderedMap<string, function<Component*()>>
 = ComponentRegister::sGetInstance()->registry;

auto it = registry.find(name);
if (it != registry.end())
{
    return (it->second)();
}
```

简单反射的实现



遇到过的问题：静态变量初始化顺序

一开始没有想将ComponentRegister设计成单例，而是直接将registryList作为静态成员

但静态变量初始化顺序是不确定的，因此其他函数使用静态变量注册反射时， registryList可能还未被初始化，于是会出现未初始化的报错

解决方法：
将ComponentRegister设计成懒汉式单例

```
1  #include "RigidBody.h"
2
3  REGISTER_COMPONENT(RigidBody,"RigidBody")
```

等等，GameObject-Component同样还支持以下功能，助力游戏开发！

Layer系统

Layer系统主要通过宏定义实现，且每个Layer只有一个比特位为1，也就是说对一个Layer进行**且运算**就能快速得出是否有相同比特位为1

应用场景示例：UI相机只渲染Layer为UI的Go

Tag词条系统

对Go绑定一个string类型的tag，通过tag快速判断是什么类型的go

应用场景示例：子弹碰撞时，可以通过tag判断撞到的Player、Enemy或Obstacle

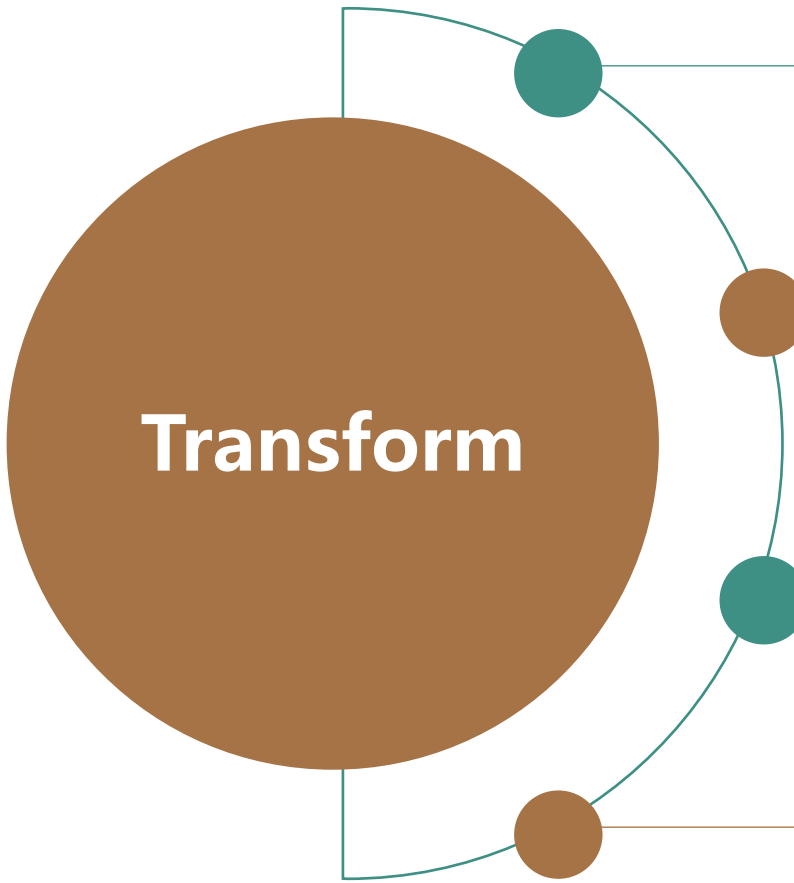
Active系统

一个Go是否时Active的决定他的组件是否需要调用生命周期函数，一个不激活的go是不会去调用他的生命周期函数的

应用场景示例：UI的开启和关闭

CoreComponent

Component——Transform



1

支持GameObject树形结构

1、固定渲染顺序、利好UI制作
2、父子联动，简化GamePlay开发，Go关系明确
3、快速查询某个Go

2

支持World-Local坐标系

1、分别描述相对行为和全局行为
2、父物体可以影响子物体，方便开发

3

统一三维世界信息

所有对三维世界的更新和修改都要通过Transform，避免二义性和数据重复存储

4

自动更新Model矩阵

通过脏标记优化渲染需要的Model矩阵，静态物体不需要每帧都运算一次

Component——Transform

通过std::function实现传入lambda表达式，通过前序或层序遍历某个Go下的所有Go，外部调用无需再写繁杂的遍历算法。

需要中序或者后序遍历也可以灵活扩展

```
//iterate all node in this tree
void foreachPreorder(const std::function<void(const Transform* transform)>& func);
void foreachActivePreorder(const std::function<void(const Transform* transform)>& func);
void foreachLevelOrder(const std::function<void(const Transform* transform)>& func);
void foreachActiveLevelOrder(const std::function<void(const Transform* transform)>& func);
```

Transform还提供丰富的旋转、移动等方法

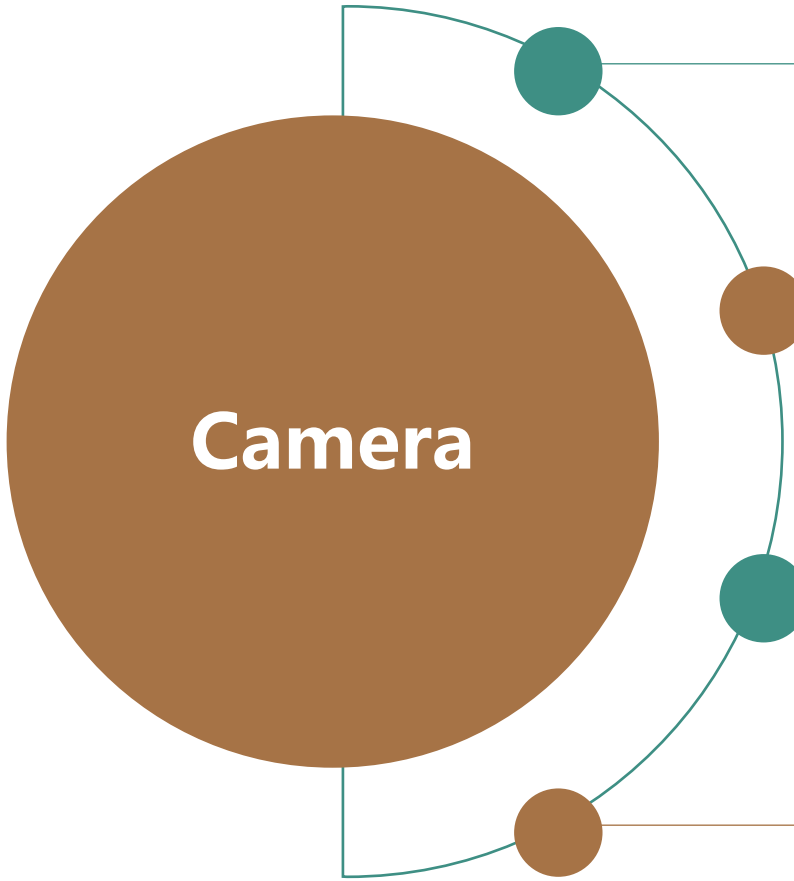
```
void setWorldPosition(const Vector3& newPosition);
void setLocalPosition(const Vector3& newPosition) {mLocalPosition = newPosition;}

void setWorldRotation(const Quaternion& newRotation);
void setLocalRotation(const Quaternion& newRotation) {mLocalRotation = newRotation;}

void rotateLocalPitchYawRoll(const Vector3& rotation);
void rotateAroundWorldAxis(const Vector3& axis, const float angle);
void rotateAroundLocalAxis(const Vector3& axis, const float angle);

void setLocalScale(const Vector3& newScale) {mLocalScale = newScale;};
```

Component——Camera



1

提供渲染数据

1、计算View和Projection矩阵
2、根据Layer决定哪些相机能看到哪些物体

2

多相机叠加渲染

提供正交投影相机和透视投影相机

3

屏幕空间转换

提供屏幕空间和世界空间的相互转换
例：敌人血条UI

4

自定义渲染顺序

Camera的遍历顺序由depth决定，可以自己定义渲染顺序
例：游戏暂停界面的坦克细节展示就是正交相机先渲染，透视相机后渲染

Component——Camera

Component——MeshFilter、MeshRenderer

MeshFilter：
决定Go使用哪个Mesh进行渲染

MeshRenderer：
1、决定使用哪个Material进行渲染
2、设置颜色（R、G、B、A）
3、设置混合系数BlendFactor

GamePlay-3C

3C-Camera

对应的MonoBehavior：TankCameraController

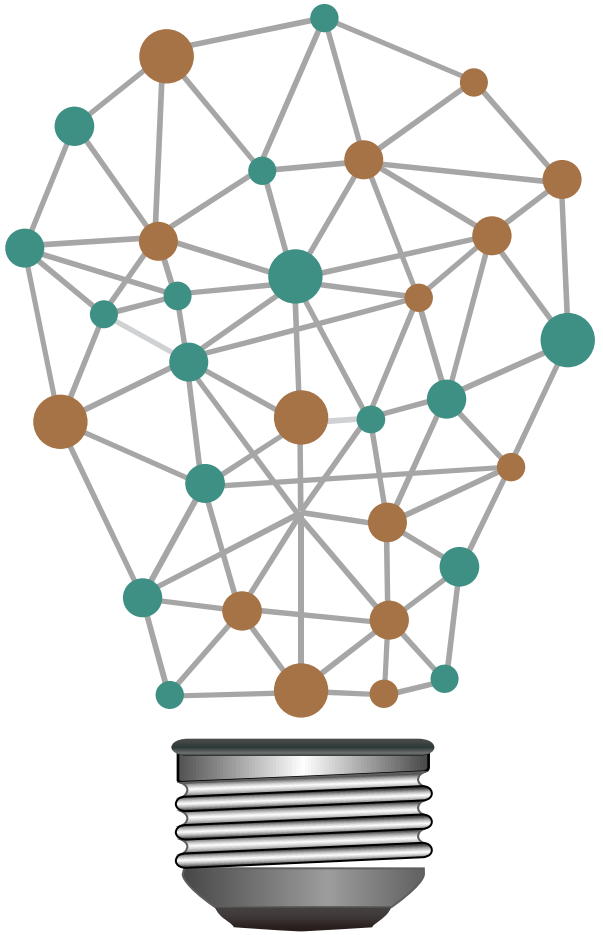
插值平滑过渡

使用逐帧线性插值，为相机做一个跟随过渡效果，实现软跟随，提升游玩手感

独立相机Go

相机并不作为坦克的子物体，而是记录一个相对坦克的位移，防止相机硬跟随

只需要设置相对位移就能自动实现软跟随效果



更改视野大小

坦克炮管抬起时攻击距离增加，此时应该将相机拉远

通过按键拉远拉近相机

相机抖动

开炮时附加一个随机抖动，增强沉浸感

3C-Camera

对应的MonoBehavior： TankCameraController

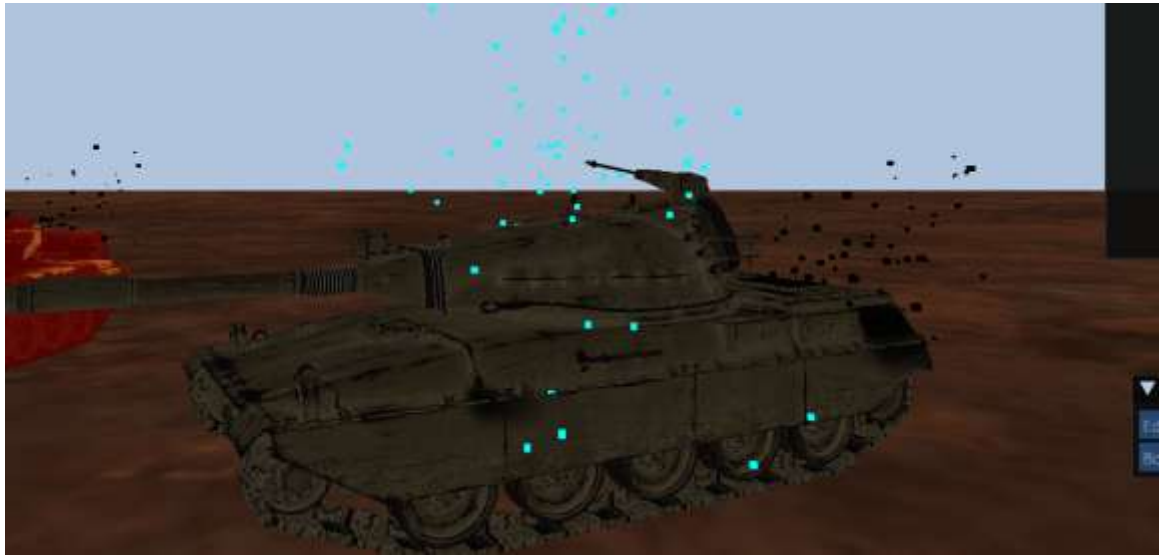
3C-Camera

逐帧插值

$$P(t) = (1 - t) \cdot \text{start} + t \cdot \text{end}$$

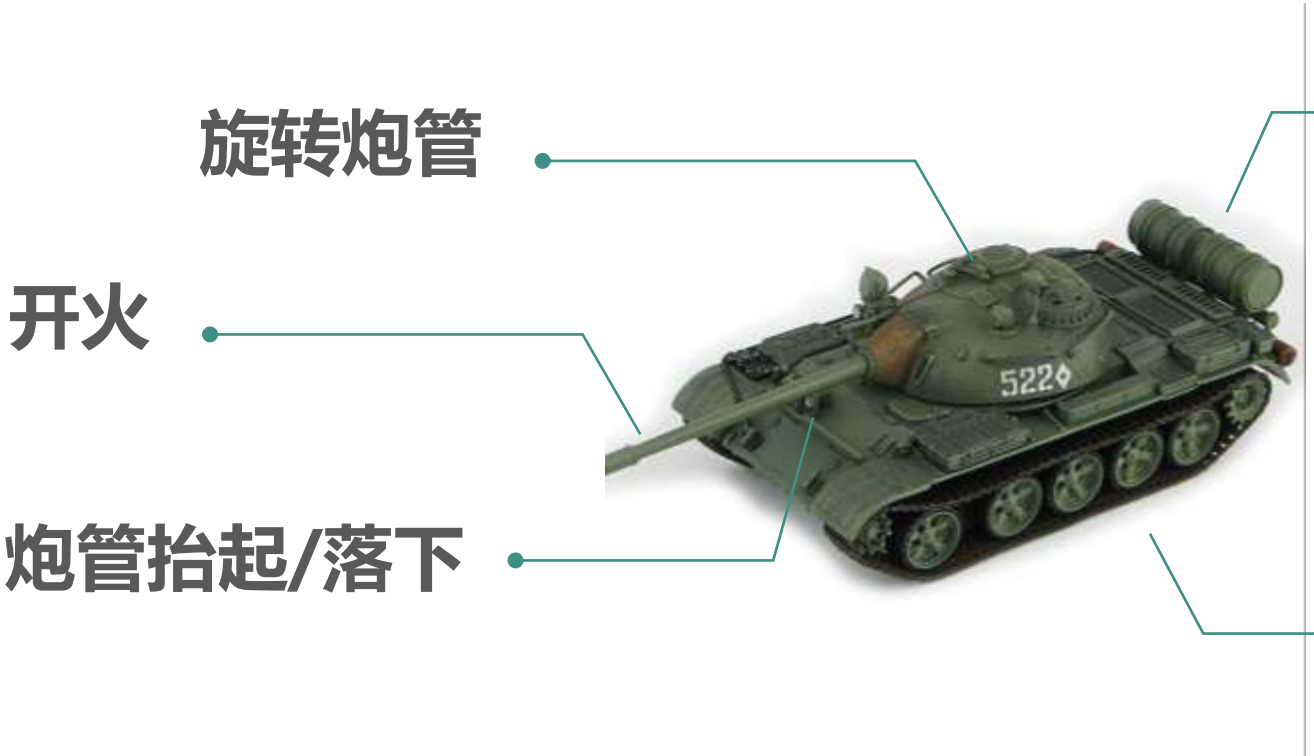
固定起点插值

$$P_{n+1} = (1 - \alpha) \cdot P_n + \alpha \cdot \text{target}$$



• Character

对应的MonoBehavior：TankController
1、根据输入控制坦克行为
2、根据坦克炮台方向决定移动方向
3、自动添加呼吸回血buff
4、计时器计算坦克开火CD
5、定义旋转速度、移动力大小等
6、判断是否死亡，更新死亡状态
7、根据坦克类型切换模型
8、播放玩家坦克的声音



3C-Character

对应单例类：TankPlayer

```
class TankPlayer
```

数据与控制分离

TankPlayer专门保存和设置Player所需要用到的数据，是Controller只能访问与更改，实现数据和控制的分离

唯一定义玩家信息

其他类可以通过该单例访问到玩家坦克的所有信息，防止信息被多重定义

组件通信

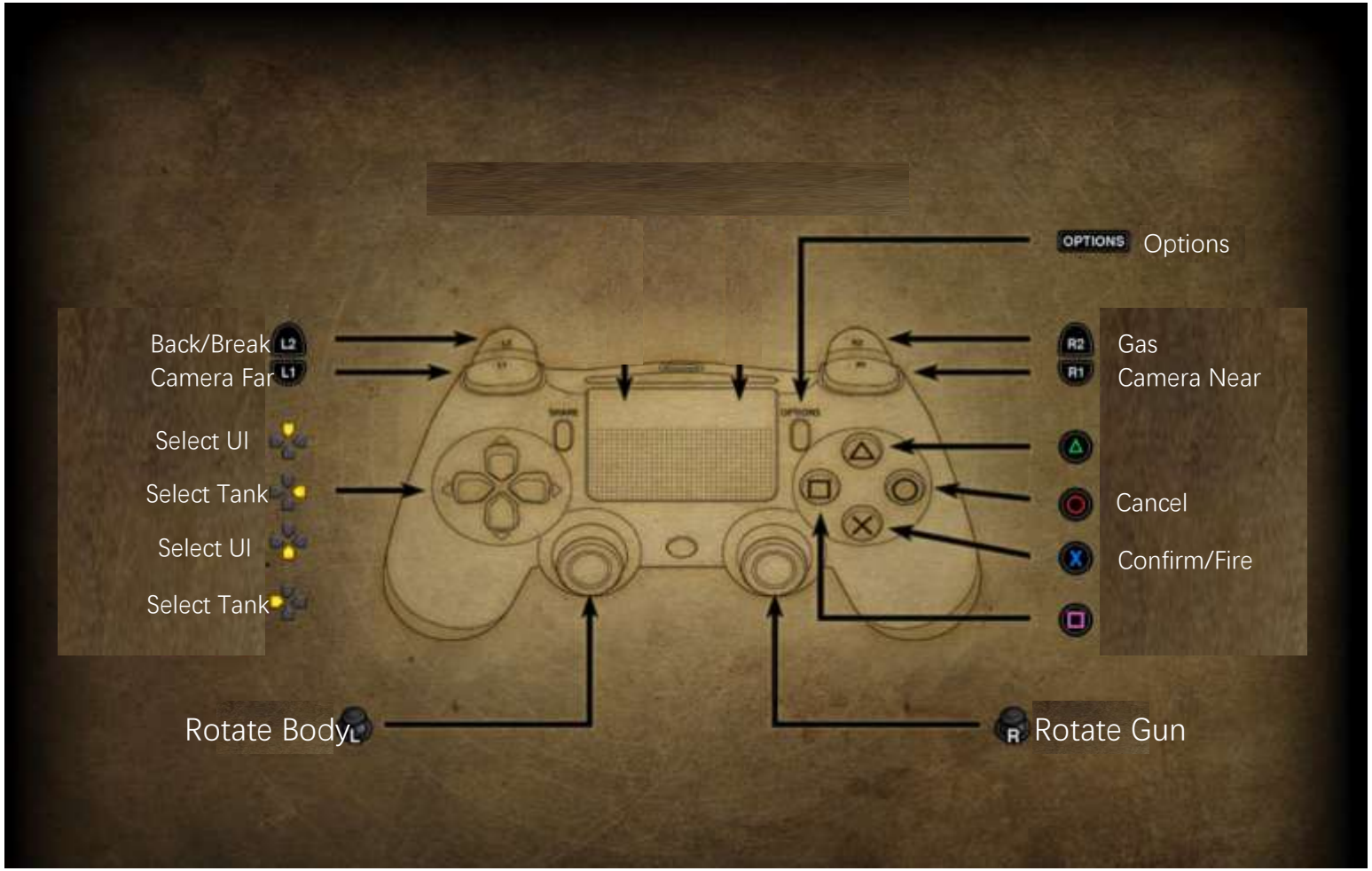
可以使用该单例实现组件之间的通信，不需要再让组件相互包含对方的引用，也避免了组件之间相互调用

管理初始数值

虽然数值有单独的系统和组件，但仍然需要使用单例进行统一管理

3C-Control

参考知名游戏 Grand Theft Auto V 的载具驾驶操作，核心为R2键模仿油门，L2模拟刹车/倒车，再加上手柄震动，模拟驾驶感



3C-Control

摇杆-键盘映射

Vector2 (x, y)

~~轻按键盘~~

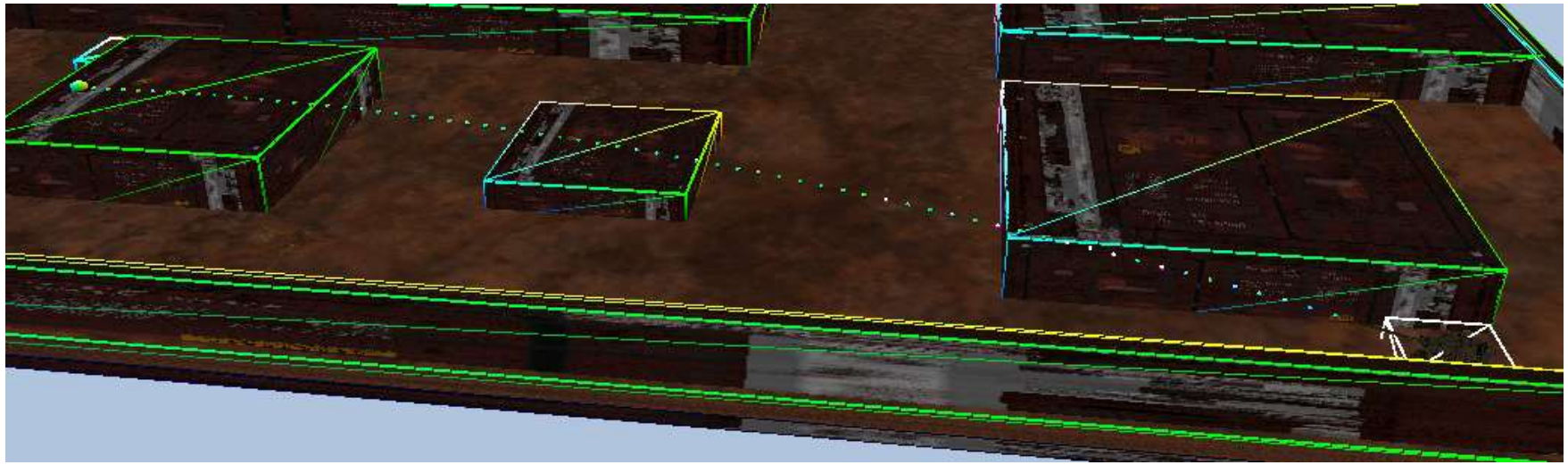
轻推摇杆

AmingLine

单独一个MonoBehavior：AmingLine

```
drawLine(const Vector3& startPos, const Vector3& startDir, float speed, float splitTime)
```

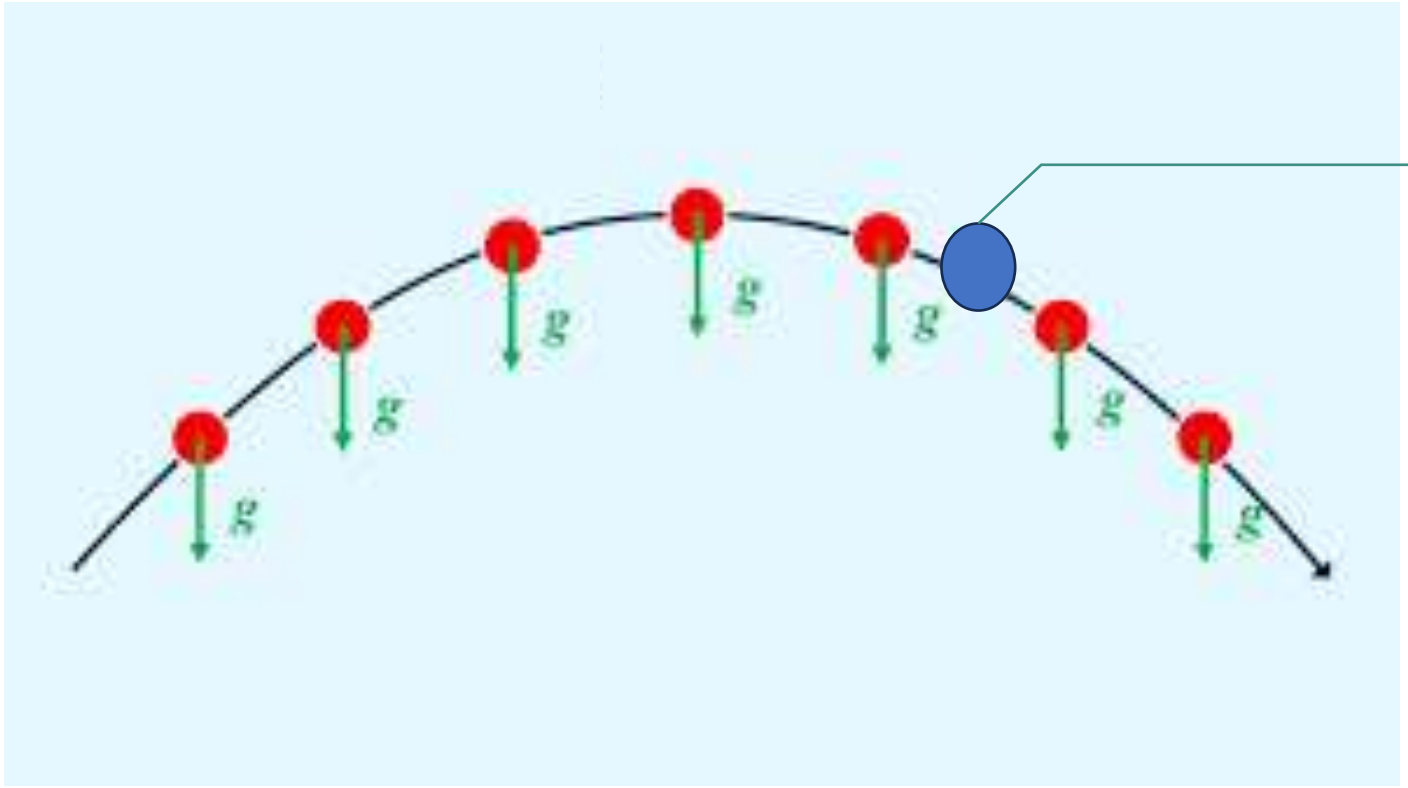
核心思路：使用DrawLine函数，接受上述四个参数，计算出有重力的抛物线，并在特定地点渲染小方块，达到一种类似虚线瞄准线的效果



优化：使用对象池，提前创建好所有小方块，然后每帧更改他们的位置就可以了
用不到的小方块（虚线已经检测到碰撞）就deactivate

AmingLine

检测瞄准线碰撞点逻辑：
1、小方块是有序排列的
2、每个小方块都有碰撞盒
3、每个小方块添加碰撞回调逻辑：碰到物体后就将当前帧的碰撞index设为自己
4、drawLine函数将碰撞index的小方块增大，index之后的小方块deactivate



可能出现的问题：
1、瞄准线步长过长，障碍物太小，导致刚好在两个瞄准线点之间，会导致无法检测

2、fixedUpdate计时器如果不准，可能导致炮弹实际路径与瞄准线不符

3、如果步长太长，落点位置会不准确

4、需要平衡瞄准线步长和性能问题

Cutscene

直接实现在TankCameraController中

1、提前计算好子弹伤害和敌人血量，判断是否需要Cutscene

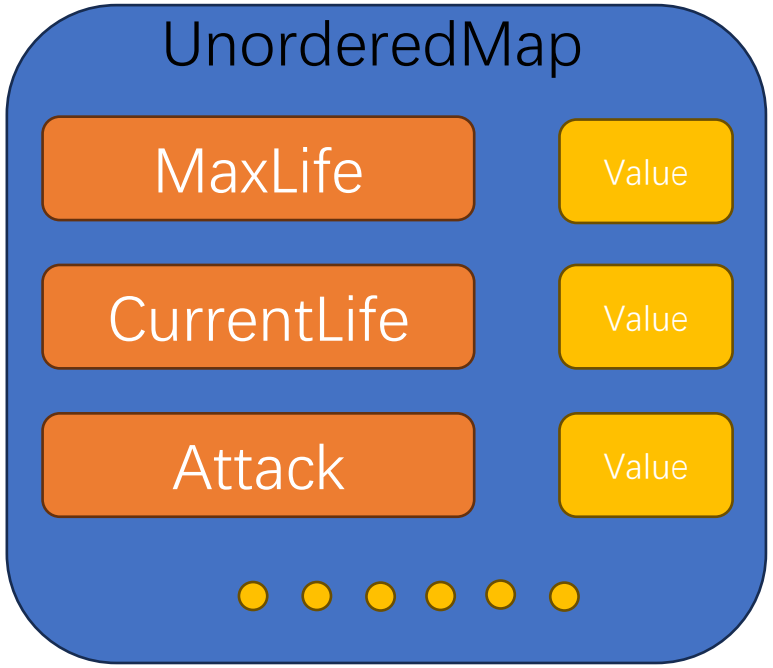
2、依赖AmingLine的对象检测

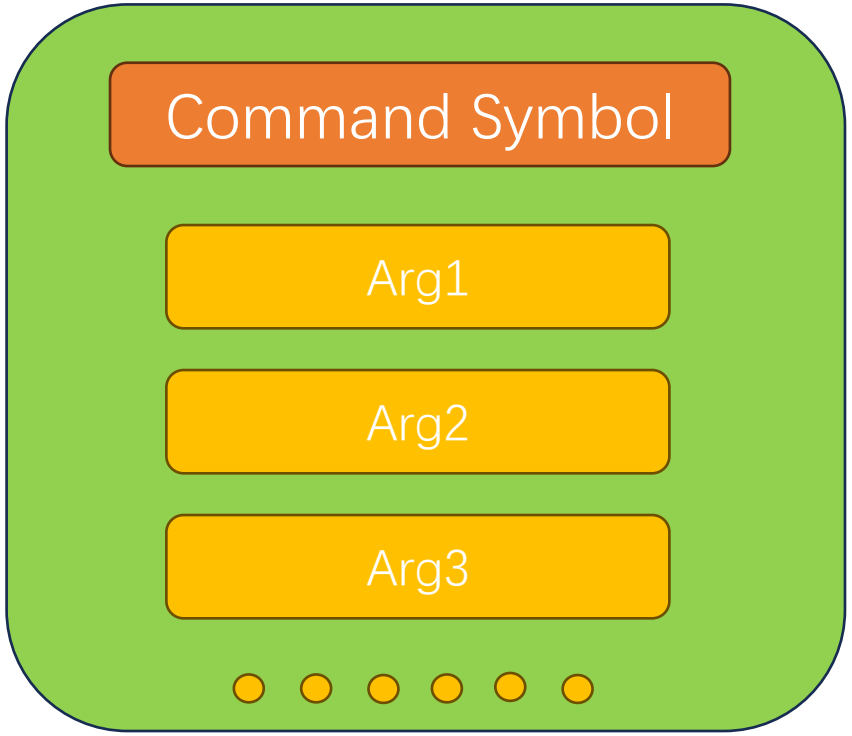
3、Cutscene时放慢游戏时间

4、相机跟随逻辑同样使用插值，实现相机位移平滑过渡

GamePlay-Numeric

- NumericalObject NumericalCommand





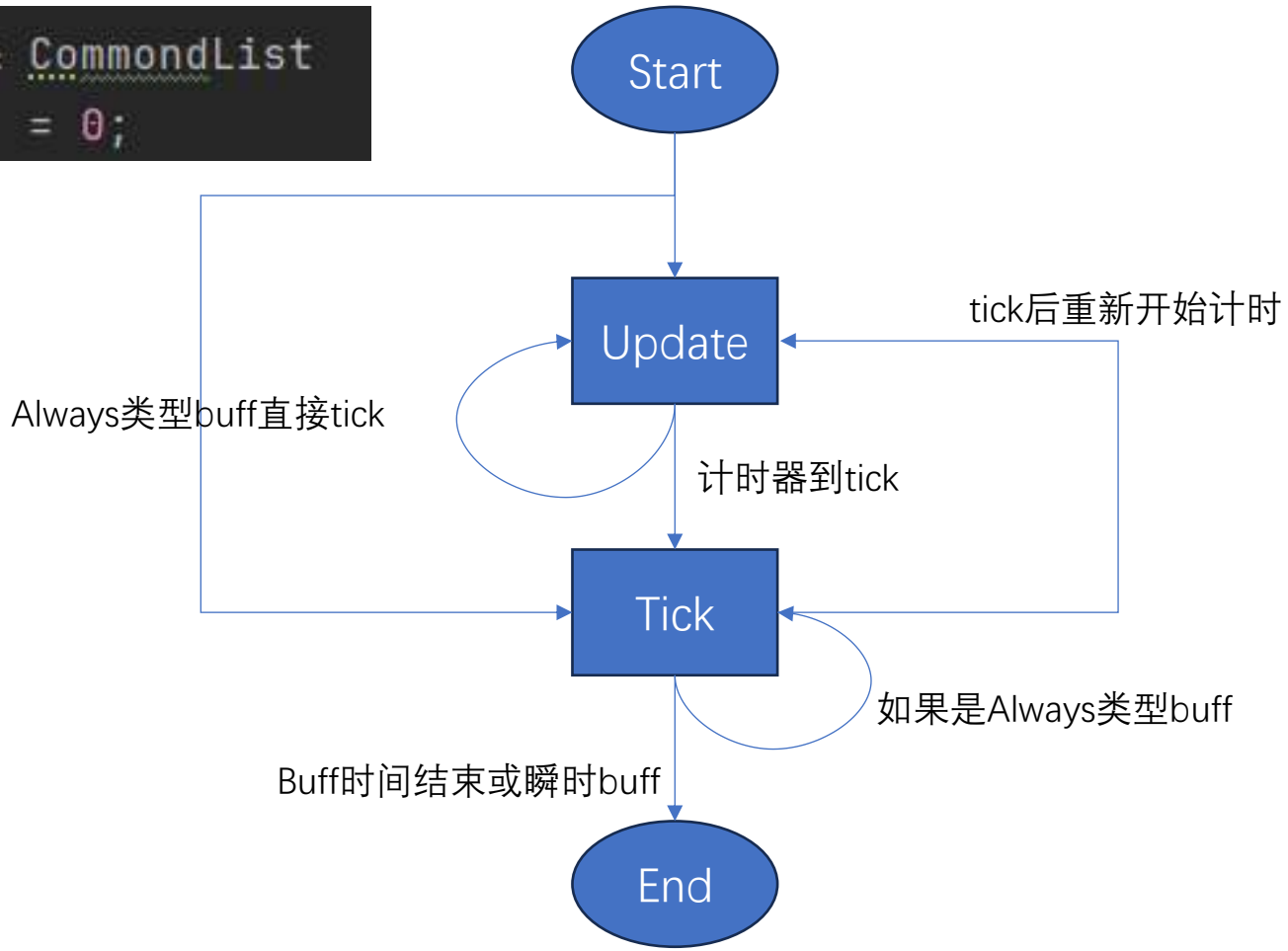
Numeric-BuffSystem

```
virtual void tick(TpList<NumericalCommand>& CommondList
    , const NumericalObject& computeResult) = 0;
```

· BuffBase基类实现自动更新

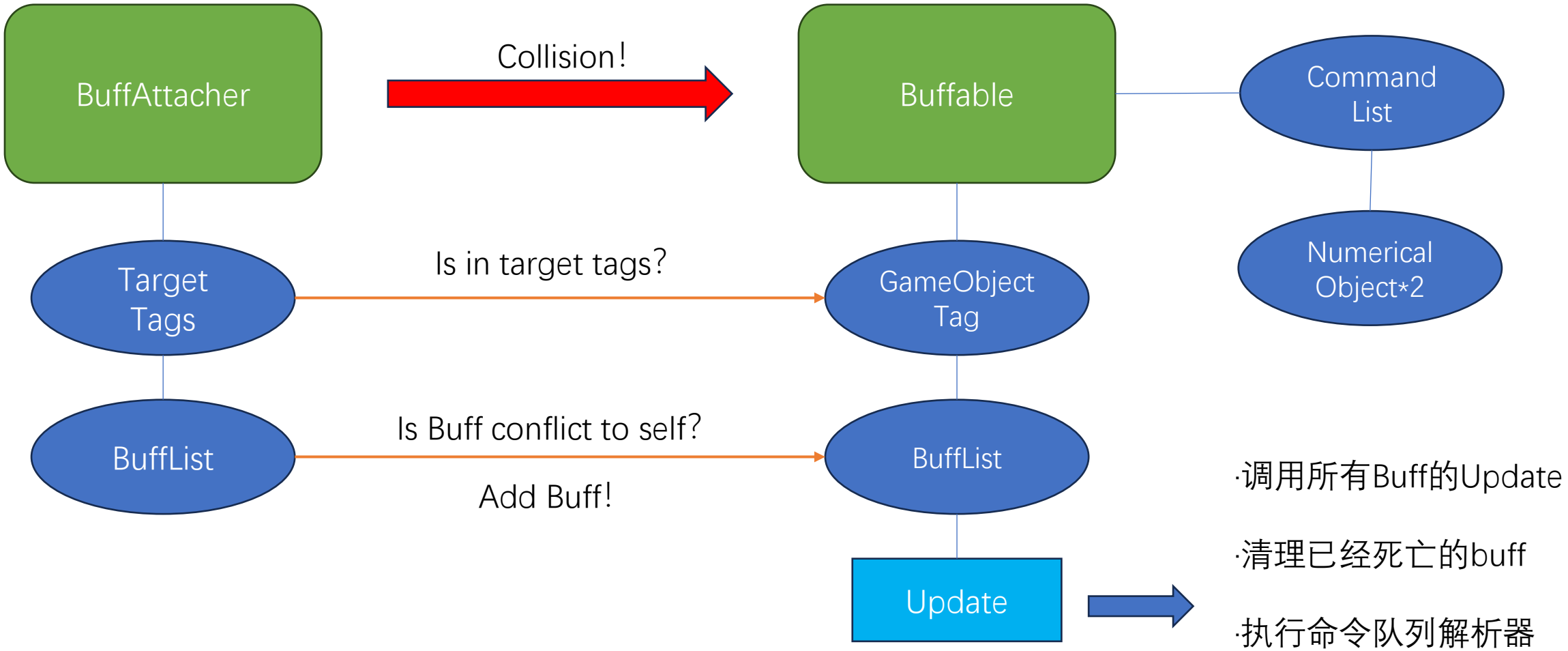
· Buff继承BuffBase并重写tick方法

· 不同tick方法向CommondList中添加不同数值命令，实现灵活度和可扩展性高的buff系统



Numeric-BuffAttacher，Buffable

两个Monobehavior：Buffattacher向拥有Buffable组件的Go挂在buff，Buffable根据buff每帧更新自己的数值



TankinGUI（TGUI）

GameTime

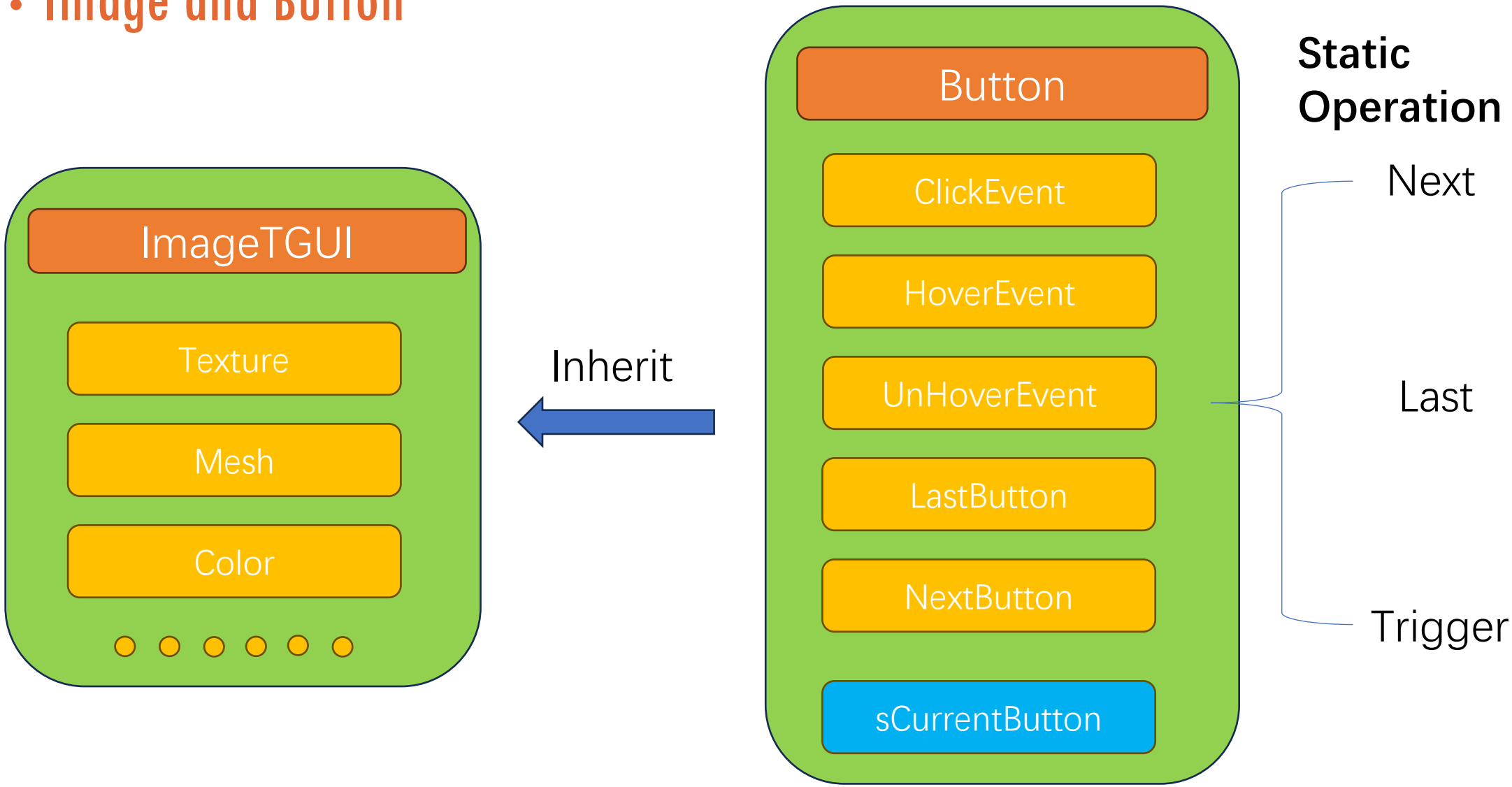
1、可以限制帧率FPS

2、提供获取帧间隔时间的接口

3、计时器计时FixedUpdate

4、让坦克和敌人的移动逻辑加入DeltaTime，就能通过设置TimeScale来控制游戏时间

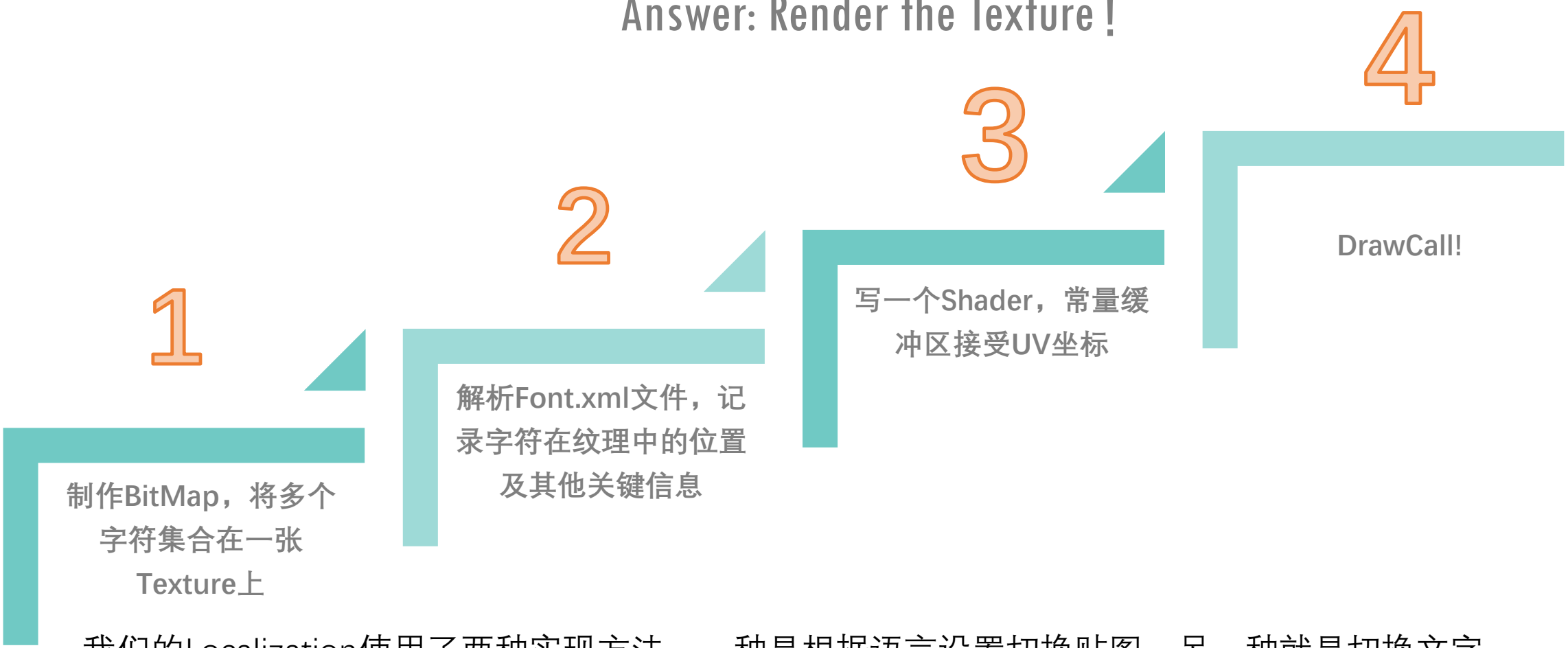
- Image and Button



- Front

Question: How to render the front?

Answer: Render the Texture !



我们的Localization使用了两种实现方法，一种是根据语言设置切换贴图，另一种就是切换文字

- Implement

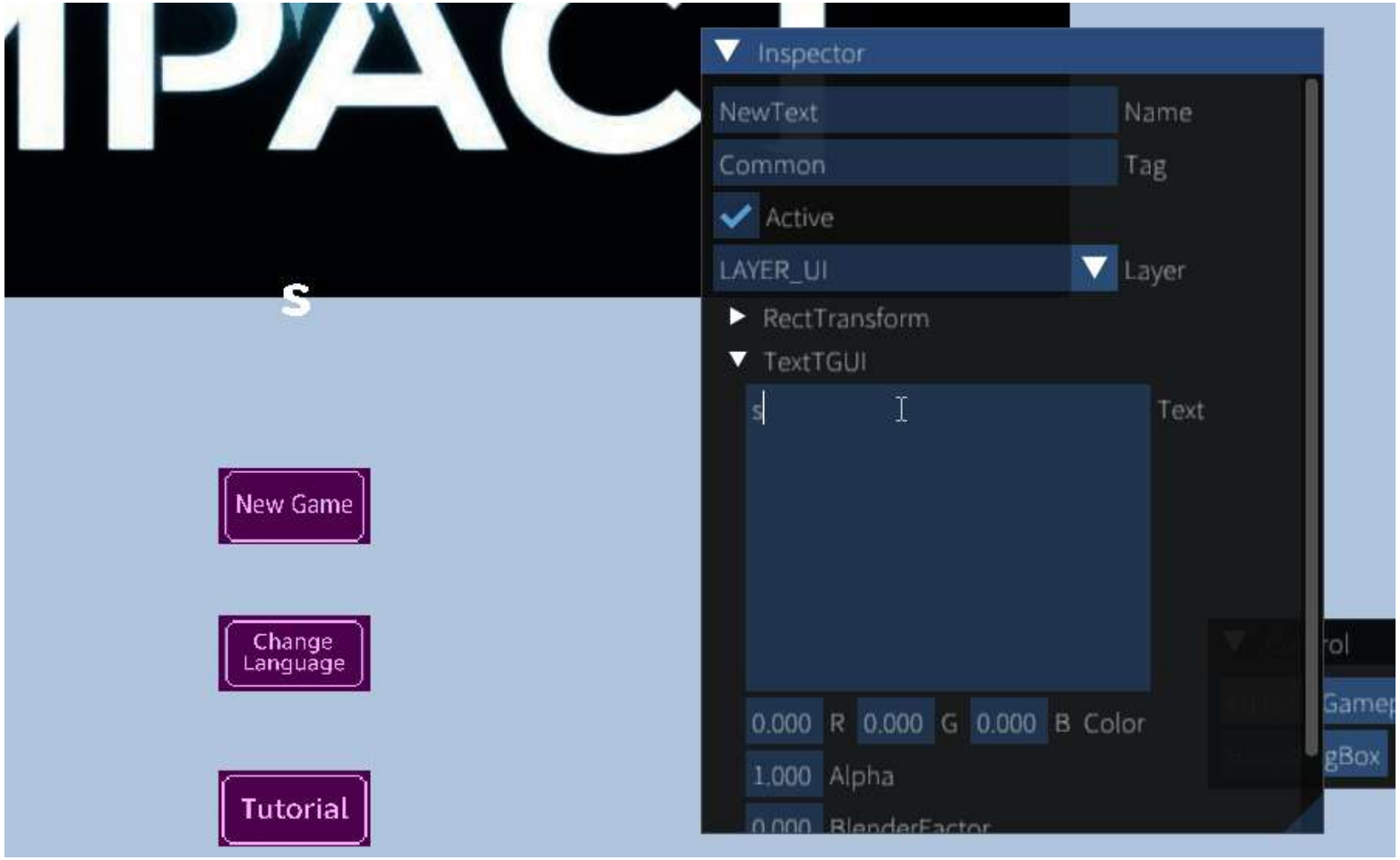
我们正好有
XML解析器

放一张bitMap大概长啥样

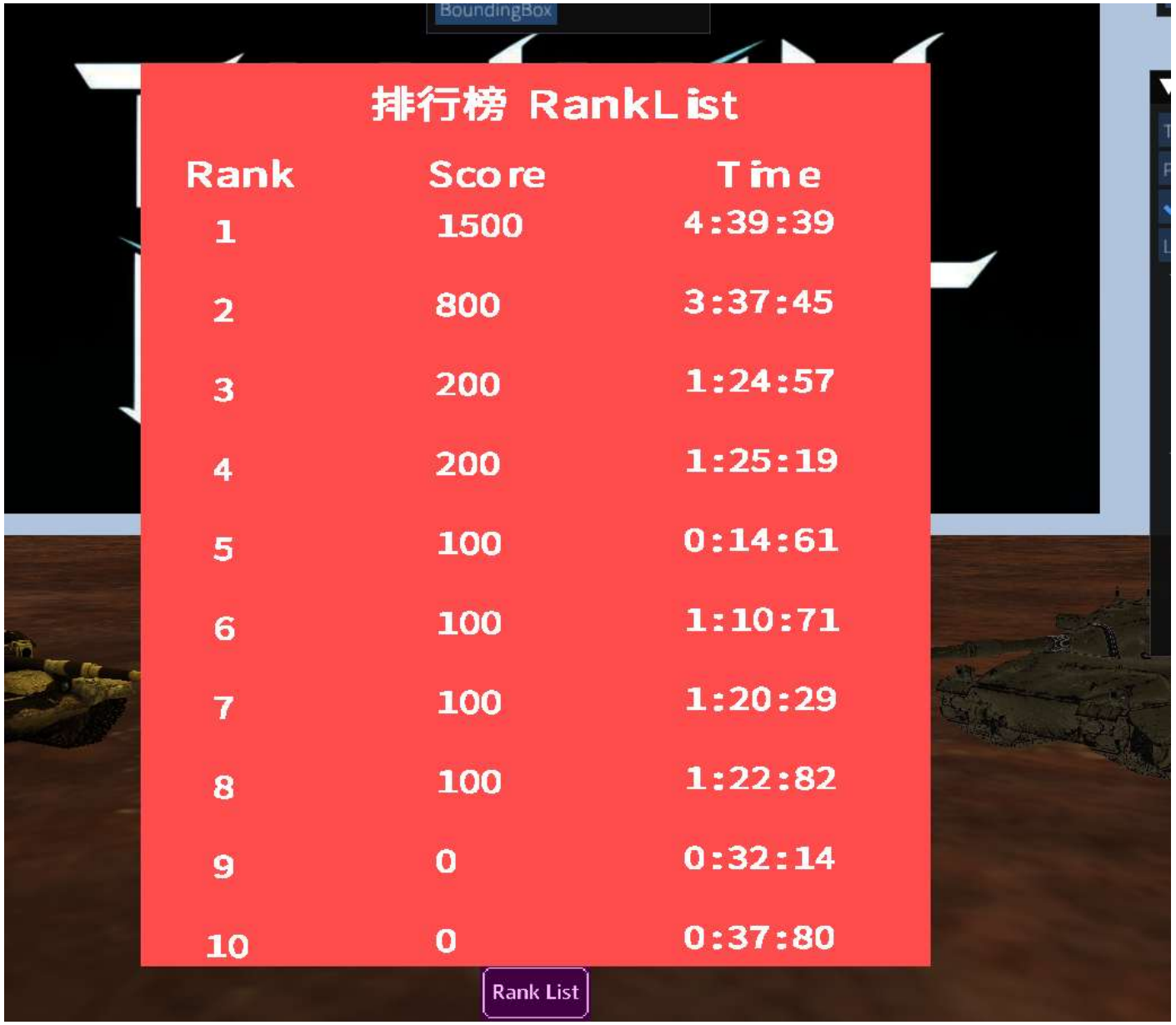
```
<char id="256" x="210" y="81" width="19" height="24" xoffset="-1" yoffset="2" xadvance="18" page="0" chnl="15" />
<char id="257" x="68" y="174" width="14" height="19" xoffset="0" yoffset="7" xadvance="15" page="0" chnl="15" />
<char id="258" x="91" y="0" width="19" height="26" xoffset="-1" yoffset="0" xadvance="18" page="0" chnl="15" />
<char id="259" x="45" y="153" width="14" height="20" xoffset="0" yoffset="6" xadvance="15" page="0" chnl="15" />
<char id="260" x="0" y="81" width="20" height="25" xoffset="-1" yoffset="6" xadvance="18" page="0" chnl="15" />
<char id="261" x="89" y="131" width="16" height="20" xoffset="0" yoffset="11" xadvance="15" page="0" chnl="15" />
<char id="262" x="131" y="0" width="19" height="26" xoffset="1" yoffset="0" xadvance="20" page="0" chnl="15" />
<char id="263" x="173" y="148" width="13" height="20" xoffset="0" yoffset="6" xadvance="14" page="0" chnl="15" />
<char id="264" x="71" y="0" width="19" height="26" xoffset="1" yoffset="0" xadvance="20" page="0" chnl="15" />
<char id="265" x="159" y="148" width="13" height="20" xoffset="0" yoffset="6" xadvance="14" page="0" chnl="15" />
<char id="266" x="190" y="81" width="19" height="24" xoffset="1" yoffset="2" xadvance="20" page="0" chnl="15" />
<char id="267" x="155" y="171" width="13" height="18" xoffset="0" yoffset="8" xadvance="14" page="0" chnl="15" />
<char id="268" x="111" y="0" width="19" height="26" xoffset="1" yoffset="0" xadvance="20" page="0" chnl="15" />
<char id="269" x="145" y="150" width="13" height="20" xoffset="0" yoffset="6" xadvance="14" page="0" chnl="15" />
<char id="270" x="75" y="27" width="17" height="26" xoffset="2" yoffset="0" xadvance="20" page="0" chnl="15" />
<char id="271" x="0" y="132" width="18" height="20" xoffset="0" yoffset="6" xadvance="17" page="0" chnl="15" />
<char id="272" x="197" y="106" width="20" height="20" xoffset="-1" yoffset="6" xadvance="20" page="0" chnl="15" />
<char id="273" x="72" y="132" width="16" height="20" xoffset="0" yoffset="6" xadvance="15" page="0" chnl="15" />
<char id="274" x="53" y="107" width="16" height="24" xoffset="1" yoffset="2" xadvance="18" page="0" chnl="15" />
<char id="275" x="36" y="174" width="15" height="19" xoffset="0" yoffset="7" xadvance="15" page="0" chnl="15" />
<char id="276" x="105" y="54" width="16" height="26" xoffset="1" yoffset="0" xadvance="18" page="0" chnl="15" />
<char id="277" x="155" y="127" width="15" height="20" xoffset="0" yoffset="6" xadvance="15" page="0" chnl="15" />
<char id="278" x="36" y="107" width="16" height="24" xoffset="1" yoffset="2" xadvance="18" page="0" chnl="15" />
<char id="279" x="139" y="171" width="15" height="18" xoffset="0" yoffset="8" xadvance="15" page="0" chnl="15" />
<char id="280" x="237" y="54" width="17" height="25" xoffset="1" yoffset="6" xadvance="18" page="0" chnl="15" />
<char id="281" x="187" y="127" width="15" height="20" xoffset="0" yoffset="11" xadvance="15" page="0" chnl="15" />
```

TGUI-Text

注意wchar和char， wstring和string， ASCII和UTF-8的转换



TGUI-Text


BoundingBox
排行榜 RankList
Rank Score Time
1 1500 4:39:39
2 800 3:37:45
3 200 1:24:57
4 200 1:25:19
5 100 0:14:61
6 100 1:10:71
7 100 1:20:29
8 100 1:22:82
9 0 0:32:14
10 0 0:37:80
Rank List

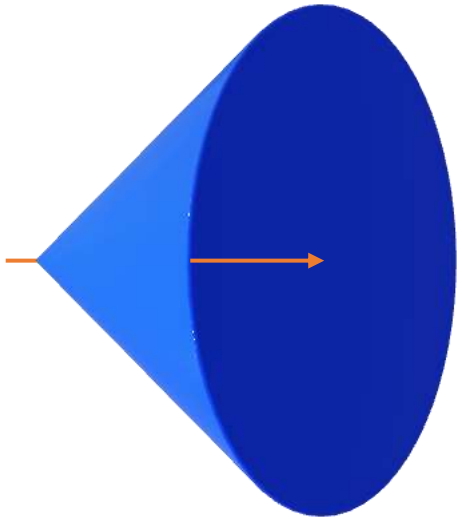
Other Implement-JJW

Component-Particle

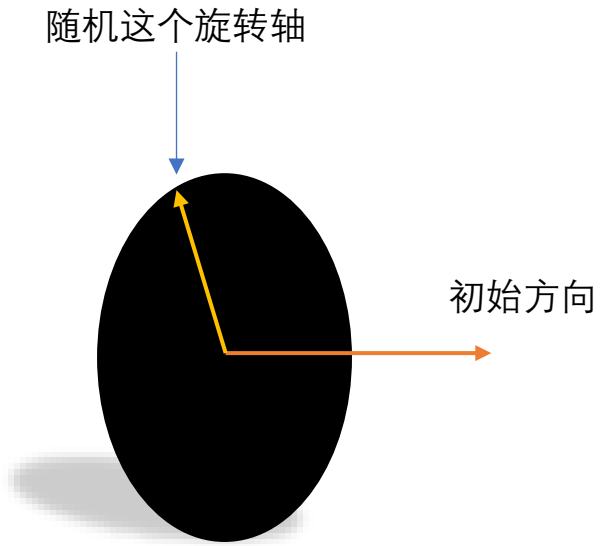
· Particle掌管自己的生命周期，自己计算更新下一帧的位置
· ParticleSystem使用对象池技术，根据每个Particle的位置提交渲染对象

Component-Particle

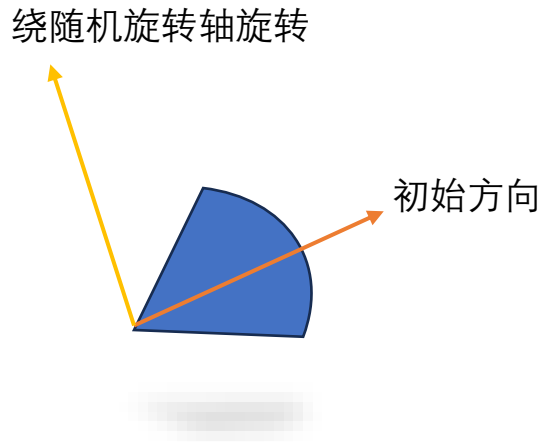
· 随机发射角算法



提供初始方向，一个开角为θ度的锥体，随机一个锥体中的发射方向



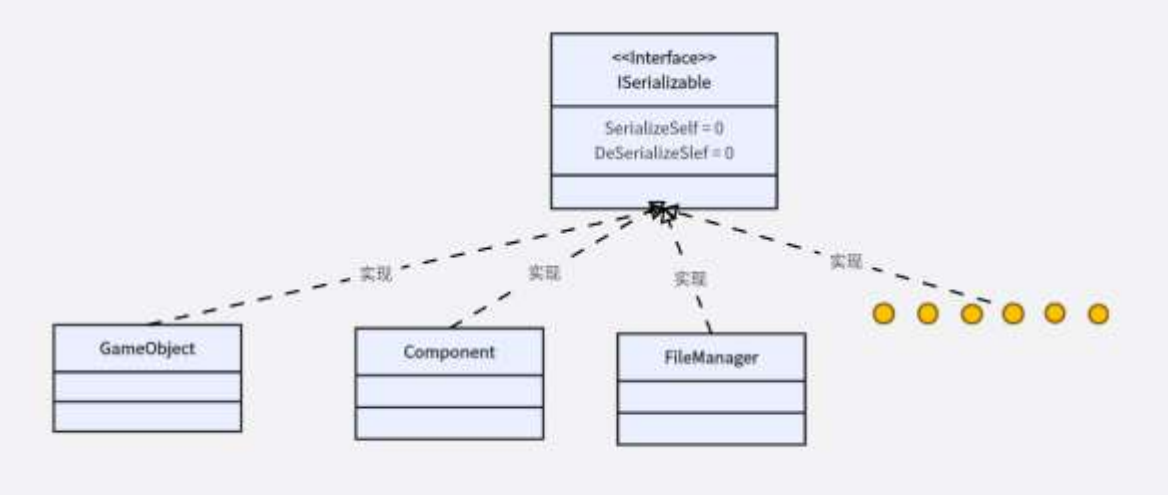
在法线为初始方向的平面中，随机一个方向作为旋转轴



再让初始方向绕这个角度旋转随机[-θ/2, θ/2]度，完成随机选择发射方向

Serialization

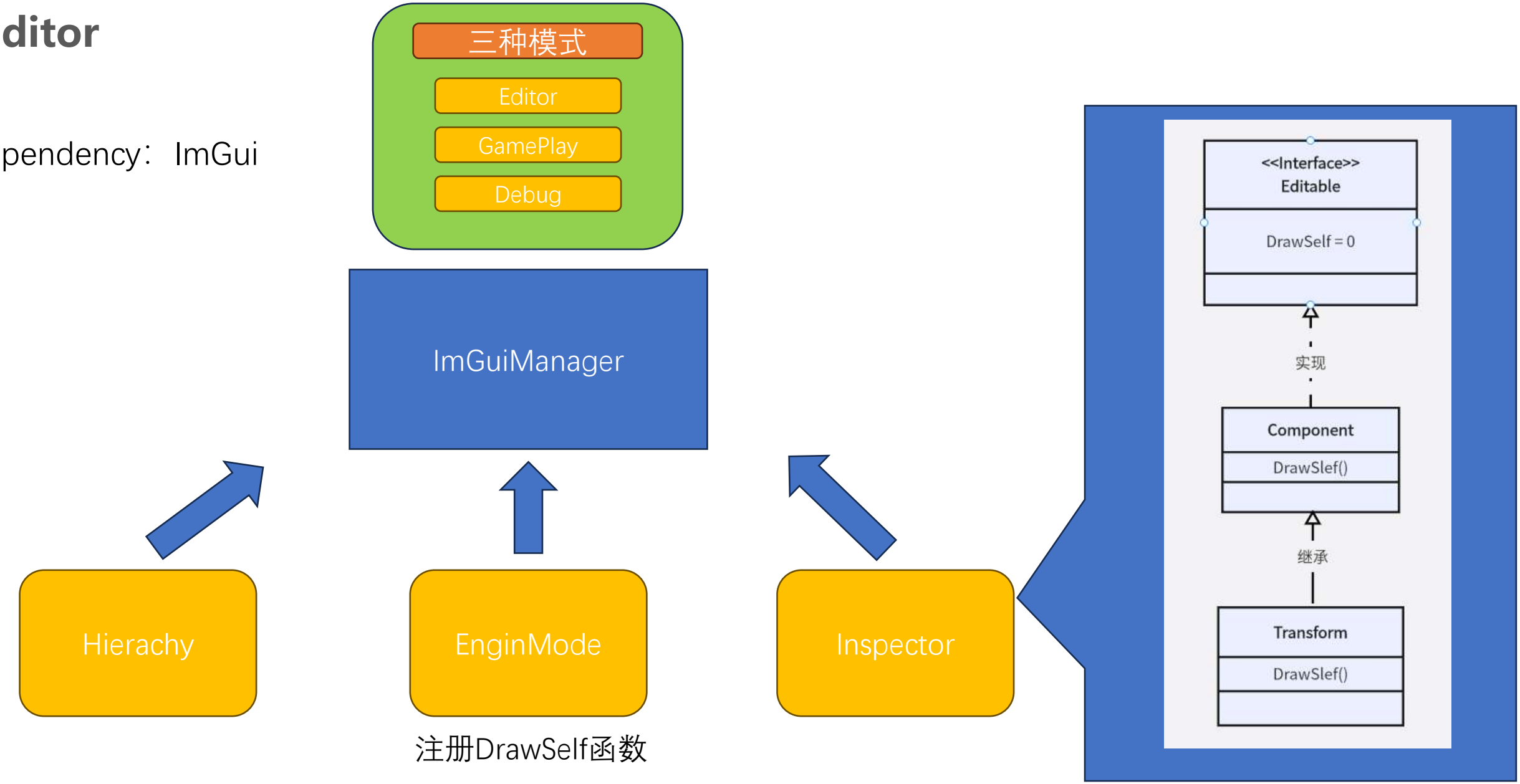
Dependency： rapidXml



场景文件截图

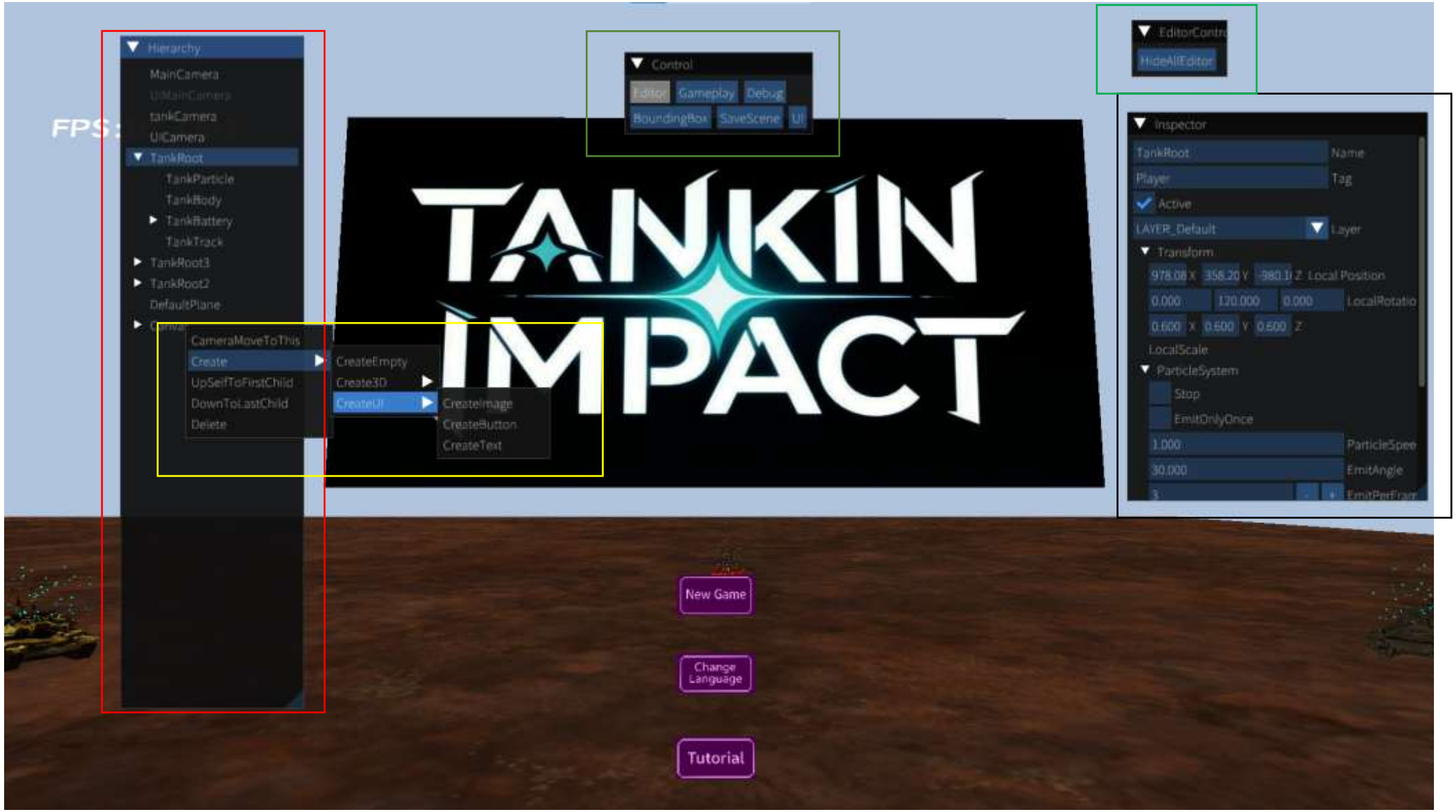
Editor

Dependency： ImGui



地图搭建、粒子效果调整、UI搭建都是使用Editor做的

Editor

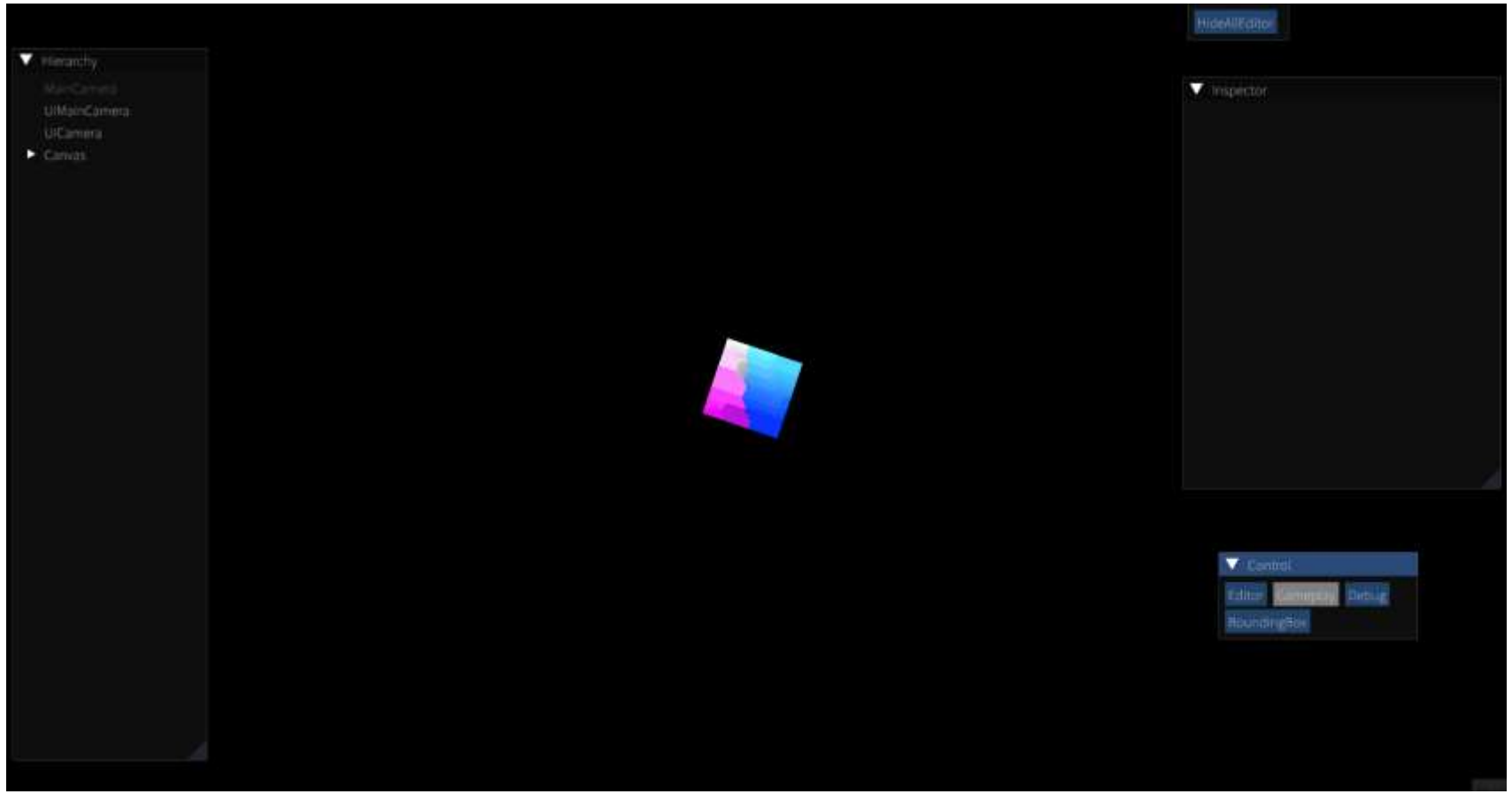

Hierarchy
MainCamera
tankCamera
UICamera
TankRoot
TankParticle
TankBody
TankBattery
TankTrack
TankRoot3
TankRoot2
DefaultPlane
CameraMoveToThis
Create
UpSelfToFirstChild
DownToLastChild
Delete
CreateEmpty
Create3D
CreateUI
CreateImage
CreateButton
CreateText
Control
Editor
Gameplay
Debug
BoundingBox
SaveScene
UI
EditorContro
HideAllEditor
Inspector
TankRoot
Name
Player
Tag
Active
LAYER_Default
Layer
Transform
Local Position
LocalRotatio
LocalScale
ParticleSystem
Stop
EmitOnlyOnce
ParticleSpee
EmitAngle
EmitPerFram
TANKIN IMPACT
New Game
Change Language
Tutorial

FileManager

· 使用模板特化加载不同类型资源

· 多线程加载和上传资源

· 开始时加载Loading界面必要的资源，之后Loading界面渲染时再多线程加载其他资源

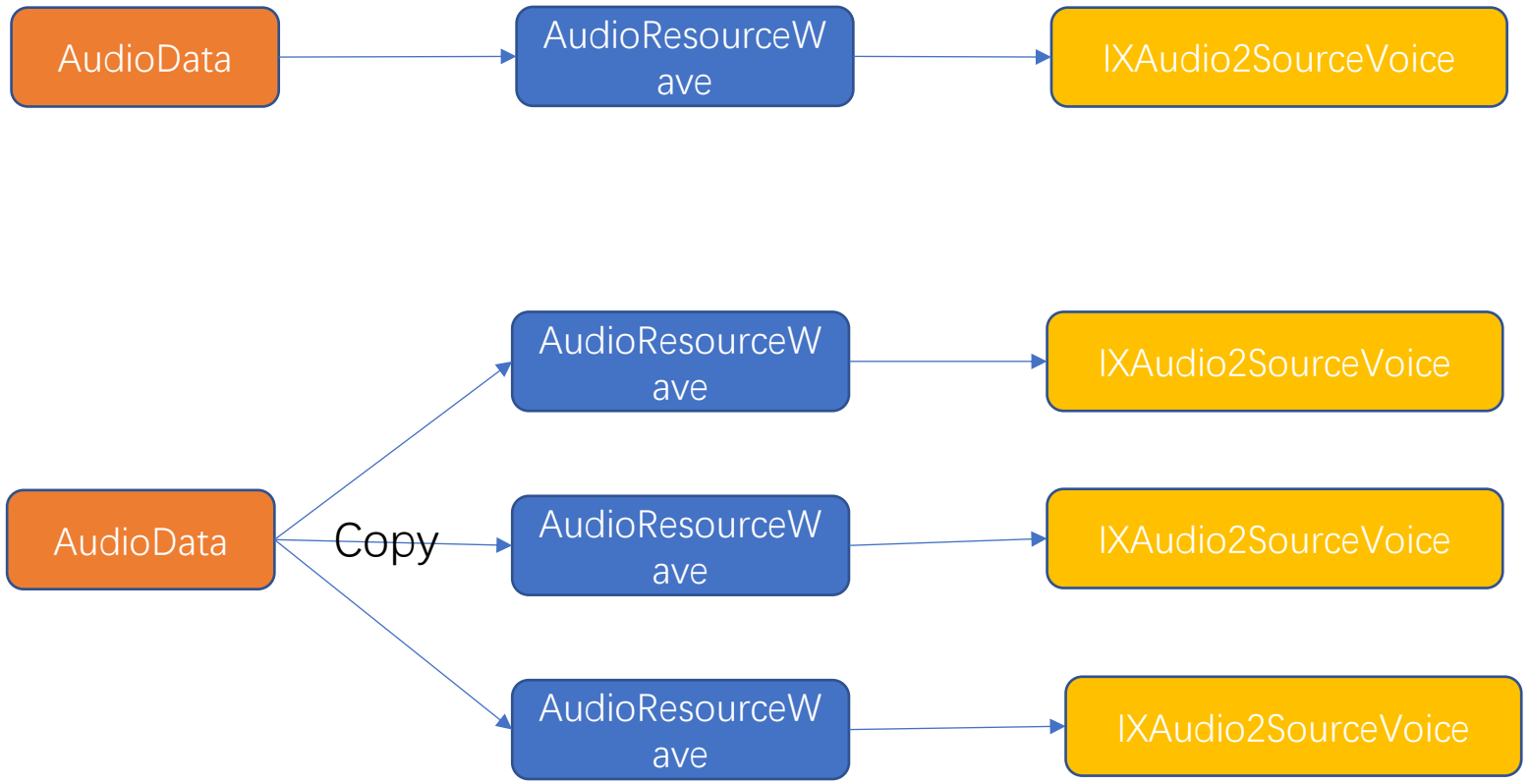


AudioSystem

· 直接使用了项目提供框架中的Audio，但存在问题：无法同时播放多个相同的声音

PC

同一个data无法创建多个AudioResourceWave

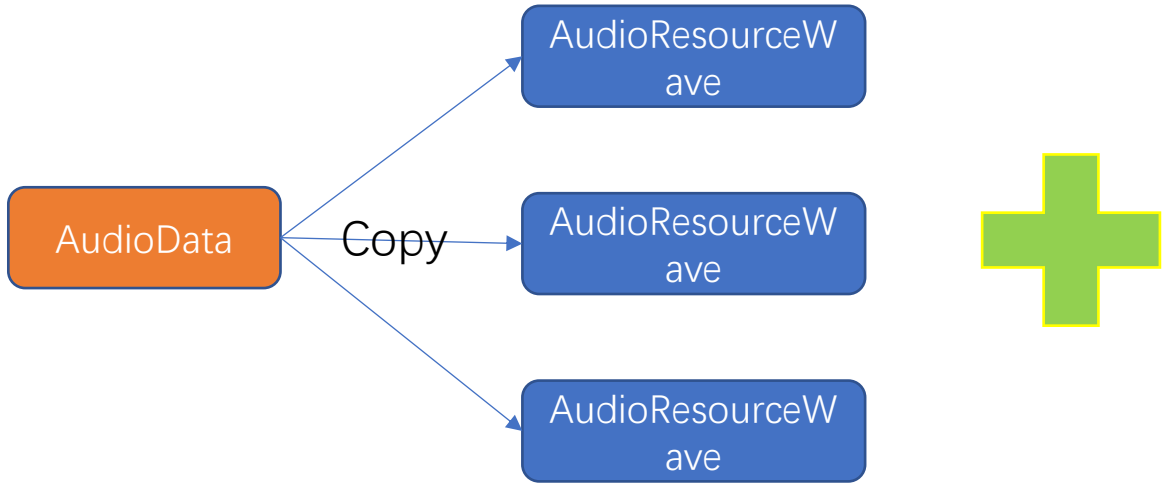


AudioSystem

· 直接使用了项目提供框架中的Audio，但存在问题：无法同时播放多个相同的声音

PS4

同一个data无法创建多个AudioResourceWave +
AudioPS4Context数量有限，无法同时播放超过数量的声音



AI敌人判断与玩家距离，
距离过近才会播放声音

成长与收获

光有功能不行，还要思考如何展示

Xaudio问题和反射问题，让我再一次体会到了C++基础**真的真的真的很重要**

C++写起来熟练了很多

写代码习惯将遇到的问题记录下来

写TDD能力明显增加了

团队协作更加顺畅，习惯使用VCS了

体会到了CodingStyle的重要性