GameManager

CoroutineManager
协程控制器
该控制器可以不依赖于mono使用

- public GameCoroutine StartCoroutine(IEnumerator routine)
开启协程

- public void StopCoroutine(GameCoroutine co)
关闭协程

- public void StopAllCoroutines()
关闭所有协程

ResourceManager
资源控制器
主要用于AssetBundle资源的加载卸载和管理

待重新整理

InputManager
输入控制器
管理各种输入控制

- public bool HasInputControl(string inputName)
是否有指定输入控制

- public IInputControl GetInputControl(string inputName)
获取指定输入控制

- public void RegisterInputControl(IInputControl inputCtl)
注册输入控制

- public void UnregisterInputControl(string ctlName)
删除输入控制

DisableGameInput 事件
禁用游戏输入

EnableGameInput 事件
开启游戏输入

AudioManager
音频管理器
管理音乐，音效的播放

- public void LoadBgm(string audioPath)
加载背景音乐

- public void PlayBgm(bool loop)
播放背景音乐

- public float GetCurBgmTime()
获取当前背景音乐的总时长

AgentManager
角色控制器
管理所有角色

- public GameObject GetHeroNode()
获取英雄（玩家）的容器节点

- public GameObject GetMonsterNode()
获取怪物的容器节点

- public void LoadHero(uint herold)
加载英雄（玩家）角色

- public void RemoveHero()
删除英雄（玩家）角色

- public void LoadMonster(uint monsterId)
加载怪物角色

- public void RemoveMonster(uint monsterId)
删除怪物角色

UiManager
Ui控制器
控制所有Ui的加载和销毁

- public bool CheckPanelOpen<T>()
检查Panel是否已经打开【设计上一种panel同时只能被打开一个】

- public void OpenPanel<T>(string panelPath, Dictionary<string, object> openArgs = null) where T : UIPanel, new()
打开面板
T:面板类型
panelPath:prefab路径
openArgs:上下文参数

- public void ClosePanel<T>()
关闭面板

- public T AddControl<T>(UIEntity holder, string resPath, GameObject parent, Dictionary<string, object> openArgs = null) where T : UIControl, new()
添加UIControl（面板上的子模块）【UIControl可以添加多份】
Holder:control的持有方，可以是UIPanel，也可以是UIControl,即control还可可以添加control
resPath prefabPath
Parent：添加在哪个节点下
openArgs ：上下文参数

- public T BindControl<T>(UIEntity holder, GameObject ctlRootNode, Dictionary<string, object> openArgs = null) where T : UIControl, new()
绑定UIControl（面板上的子模块）【预制件中已有的UIControl，即control可以不动态加载】
Holder:control的持有方，可以是UIPanel，也可以是UIControl,即control还可可以添加control
ctlRootNode：UIControl的根节点
openArgs ：上下文参数

- public void RemoveControl(UIControl toRemove)
删除UIControl

GameSceneManager
场景控制器
控制Scene的加载卸载和切换

- public void LoadAndSwitchToScene(string sceneName)
切换到指定场景

MeterManager
节拍管理器
负责节拍的计算和节拍触发驱动

+ MeterIndex
当前节拍序号（int 始终自增）

- public float GetCurMeterPassedTime()
获取当前拍已经过的时间

- public int GetCurAudioTotalMeterLen()
获取当前背景音乐的总时长

- public void RegisterMeterHandler(IMeterHandler handler)
注册节拍处理器

- public void UnregisterMeterHandler(IMeterHandler handler)
删除节拍处理器

- public int GetCurrentMusicRhythmType()
获取当前音乐的拍型

- public bool CheckTriggerSceneBehaviour(int meterIndex)
检测节拍是否触发场景行为【目前对每个音乐支持设计一个场景触发序列】

- public bool CheckTriggerMeter(int meterIndex, float tolerance, float offset)
检测是否能触发指定拍（内部调用IsInMeterWithTolerance）
Tolerance：节拍的总触发范围比例
Offset：节拍的触发范围对节拍的偏移

- public bool IsInMeterWithTolerance(int meterIndex, float tolerance, float offset)
检测是否能在指定拍的指定范围内
Tolerance：节拍的总触发范围比例
Offset：节拍的触发范围对节拍的偏移

- public bool CheckTriggered(float tolerance, float offset, out int triggerMeter)
检测当前时间是否在节拍范围内，如果在范围内，triggerMeter是触发在哪一拍
Tolerance：节拍的总触发范围比例
Offset：节拍的触发范围对节拍的偏移

- public float GetTimeToMeter(int offset)
计算现在到向后offset拍需要多长时间

- public int GetMeterIndex(int from, int offset))
计算从from拍向后offset拍的节拍index

- public int GetMeterOffset(int from, int to)
计算to拍和from拍之间差几拍

- public float GetTotalMeterTime(int from, int to)
计算发rom拍到to拍的总时长

- public float GetCurrentMeterTotalTime()
获取当前拍的总时长

- public float GetCurrentMeterProgress()
获取现在所处的拍子，已经走过了这一拍的百分比

DataCenter
数据中心

+ TableDataCenter
所有配表数据

+ AgentStatusGraphCenter
所有角色状态配置数据

+ AgentComboGraphCenter
所有角色combo配置数据

+ BehaviurTreeCenter
所有行为树配置数据

CameraManager
相机控制器
控制所有相机

- public Camera GetMainCam()
获取主相机

- public Camera GetUiCam()
获取Ui相机

- public void MainCameraMoveTo(Vector3 pos)
设置主相机位置

- public void MainCameraMoveTo(Vector3 pos, float duration)
主相机移动到（插值移动） 暂未实现

- public void MainCamZoomTo(float size)
主相机缩放

- public void MainCamZoomTo(float size, float duration)
主相机缩放到（插值缩放） 暂未实现

GameEventSystem
事件系统
事件的注册和广播

- public void Fire(string evtName)
- public void Fire<T1>(string evtName, T1 arg1)
- public void Fire<T1, T2>(string evtName, T1 arg1, T2 arg2)
- public void Fire<T1, T2, T3>(string evtName, T1 arg1, T2 arg2, T3 arg3)
- public void Fire<T1, T2, T3, T4>(string evtName, T1 arg1, T2 arg2, T3 arg3, T4 arg4)
- public void Fire<T1, T2, T3, T4, T5>(string evtName, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5)
- public void Fire<T1, T2, T3, T4, T5, T6>(string evtName, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6)
- public void Fire<T1, T2, T3, T4, T5, T6, T7>(string evtName, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7)
即时事件，监听者在事件发出后立刻收到通知，目前支持最多7个任意类型参数

- public void Post(string evtName)
- public void Post<T1>(string evtName, T1 arg1)
- public void Post<T1, T2>(string evtName, T1 arg1, T2 arg2)
- public void Post<T1, T2, T3>(string evtName, T1 arg1, T2 arg2, T3 arg3)
- public void Post<T1, T2, T3, T4>(string evtName, T1 arg1, T2 arg2, T3 arg3, T4 arg4)
- public void Post<T1, T2, T3, T4, T5>(string evtName, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5)
- public void Post<T1, T2, T3, T4, T5, T6>(string evtName, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6)
- public void Post<T1, T2, T3, T4, T5, T6, T7>(string evtName, T1 arg1, T2 arg2, T3 arg3, T4 arg4, T5 arg5, T6 arg6, T7 arg7)
非即时事件，根据配置每帧处理一定数量的事件，目前支持最多7个任意类型参数

事件监听者参考GameEventListerner

TimeMgr
时间管理器

+ int FrameIndex
当前所在帧

+ float Now
当前游戏时间

GameColliderManager
碰撞管理器

- public void RegisterGameCollider(ConvexCollider2D collider)
注册碰撞体

- public void UpdateColliderPos(ConvexCollider2D collider, Vector3 newAnchorPos)
更新碰撞体位置

- public void UpdateColliderRotateAngle(ConvexCollider2D collider, float newAngle)
更新碰撞体角度

- public void UpdateColliderScale(ConvexCollider2D collider, Vector3 newScale)
更新碰撞体缩放

- public bool UnRegisterGameCollider(ConvexCollider2D collider)
删除碰撞体

- public bool CheckCollideHappenWithShape(IConvex2DShape shape, IGameColliderHandler handler, ConvexCollider2D exceptCollider, out Dictionary<ConvexCollider2D, Vector2> tgtsWithLeaveV2Dict)
- public bool CheckCollideHappenWithShape(IConvex2DShape shape, IGameColliderHandler handler, out Dictionary<ConvexCollider2D, Vector2> tgtsWithLeaveV2Dict)
- public bool CheckCollideHappen(ConvexCollider2D srcCollider, IGameColliderHandler handler, out Dictionary<ConvexCollider2D, Vector2> tgtsWithLeaveV2Dict)
检测是否发生碰撞
Handler：碰撞处理
exceptCollider：不进行碰撞检测的碰撞体
tgtsWithLeaveV2Dict：分离向量

- public bool CheckCollideHappenWithShape(IConvex2DShape shape, IGameColliderHandler handler, ConvexCollider2D exceptCollider, out List<ConvexCollider2D> tgtColliders)
- public bool CheckCollideHappenWithShape(IConvex2DShape shape, IGameColliderHandler handler, out List<ConvexCollider2D> tgtColliders)
- public bool CheckCollideHappenWithShape(ConvexCollider2D collider, IGameColliderHandler handler, out List<ConvexCollider2D> tgtColliders)
使用形状体检测是否发生碰撞
Handler：碰撞处理
exceptCollider：不进行碰撞检测的碰撞体
tgtColliders:检测到碰撞的碰撞体

SkEftEntityManager
技能效果管理器

- public void RegisterNewSkEftEntity(SkEftEntity newSkEntity)
注册技能效果

- public void UnRegisterSkEftEntity(SkEftEntity removeSkEntity)
删除技能效果

GameMapManager
地图管理器

+ MapRootNode
地图根节点

+ BaseLayerNode
BaseLayer节点

+ EventLayerNode
EventLayer节点

- public bool IsPosOnGround(Vector3 worldPos)
是否在地表上

UpdateCenter
updater中心
后面可能会改个名字

- public void RegisterUpdater(IGameUpdate updater)
注册update

- public void UnregisterUpdater(IGameUpdate updater)
删除update

MeterTimerCenter
节拍定时器中心
申请节拍定时器

- public MeterTimer SetTimer(int meterOffset, int loopTime)
开启定时器

- public MeterTimer SetTimer(int meterOffset, int loopTime, Action cb)
开启定时器

- public void RemoveTimer(MeterTimer timer)
删除定时器

TimerCenter
定时器中心
申请定时器

- public GameTimer SetTimer(float time, int loopTime)
开启定时器

- public GameTimer SetTimer(float time, int loopTime, Action cb)
开启定时器

- public void RemoveTimer(GameTimer timer)
删除定时器