软件设计文档

一、开发规划

开发方式

使用迭代方式进行开发

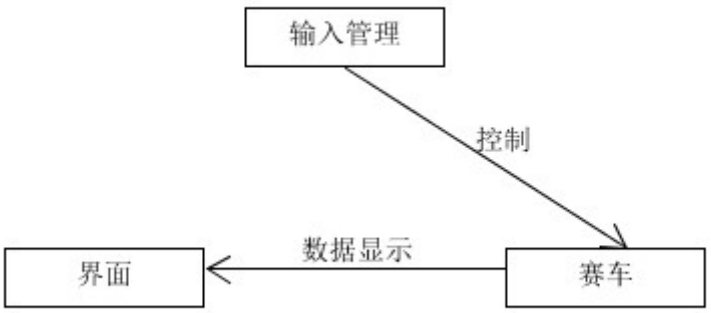
开发工具

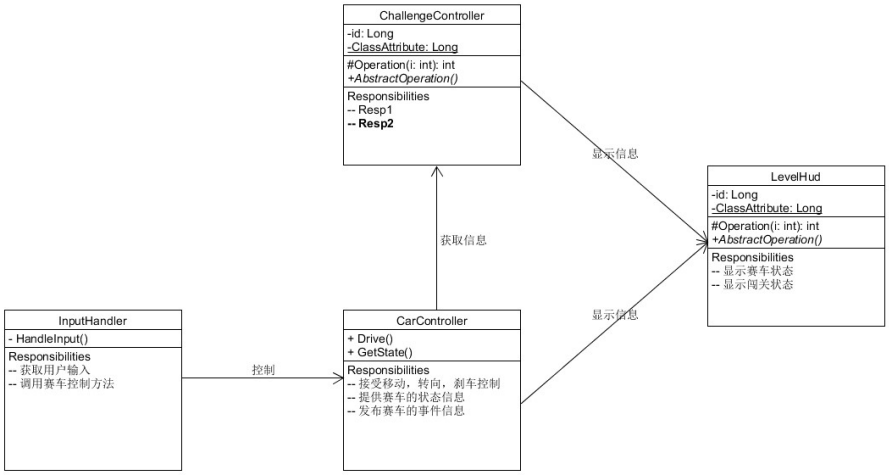
Unity， Visual Studio, Blender

二、软件设计技术

MVC 架构

例如对赛车的控制





闯关场景主要对象之间的关系

赛车控制器是一个提供赛车控制方法的类，可以接受外部调用来改变赛车的状态，可以获取赛车的状态。

```
//驾驶控制
public void Drive( float accel, float steer, float brake )
{
    //各车轮动力扭矩,超过最高速度则为0
    float motorTorque = accel * m_maxMotorTorque * Mathf.Clamp01( m_maxGear - m_curGear * 0.5f );
    float brakeTorque = brake * m_maxBrakeTorque;    //各车轮制动扭矩
    float steerAngle = m_maxSteer * steer * Mathf.Clamp01(1 - SpeedFactor * 0.25f);    //转向角

    //当赛车没有动力时，自动进行减速
    if ( motorTorque == 0 ||
        (GetCarState() == CarState.forward && motorTorque < 0) ||
        GetCarState() == CarState.backward && motorTorque > 0 )
    {
        brakeTorque += m_maxBrakeTorque * 0.15f;
        motorTorque = 0;
    }

    //四轮动力
    for ( int i = 0; i < 4; i++ )
    {
        m_wheelColliders[i].motorTorque = motorTorque / 4;
    }

    //四轮刹车
    for ( int i = 0; i < 4; i++ )
    {
        m_wheelColliders[i].brakeTorque = brakeTorque / 4;
    }

    //前轮转向
    for ( int i = 0; i < 2; i++ )
    {
        m_wheelColliders[i].steerAngle = steerAngle;
    }

    CalAccel();                 //计算加速度
    CapMaxSpeed();              //超速锁定
    SteerHelper( steer );       //转向帮助
    SyncWheelMeshes();          //同步车轮模型与碰撞器
    AdjustCarAttribute();       //调整赛车属性
    AdjustForce();              //赛车受力调整
    CalCurGear();               //计算当前档位
}
```

输入管理器可以获取用户输入来控制赛车:

```
//若为ture，处理输入
if ( m_acceptCarInput )
{
    m_accel = Input.GetAxis( "Vertical" );
    m_steer = Input.GetAxis( "Horizontal" );
    m_brake = Input.GetAxis( "Jump" );
    if( OnDriveInputChange != null)
    {
        OnDriveInputChange( m_accel, m_steer, m_brake );
    }
}
```

输入控制器

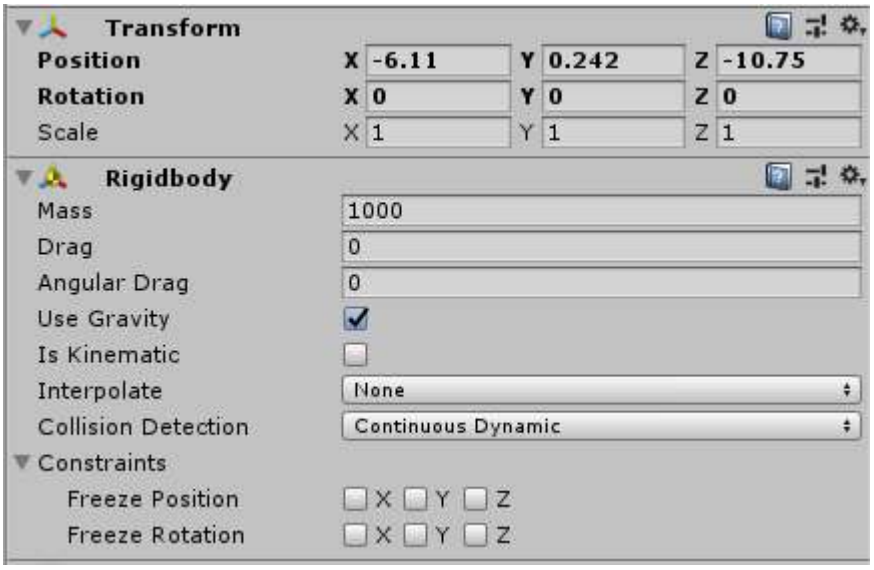
面向对象

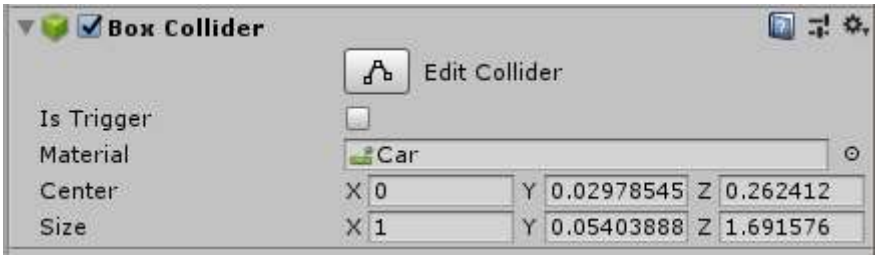
赛车控制，关卡管理，输入控制，闯关管理都是不同的对象，它们之间通过游戏控制器来连接通信，传输数据。

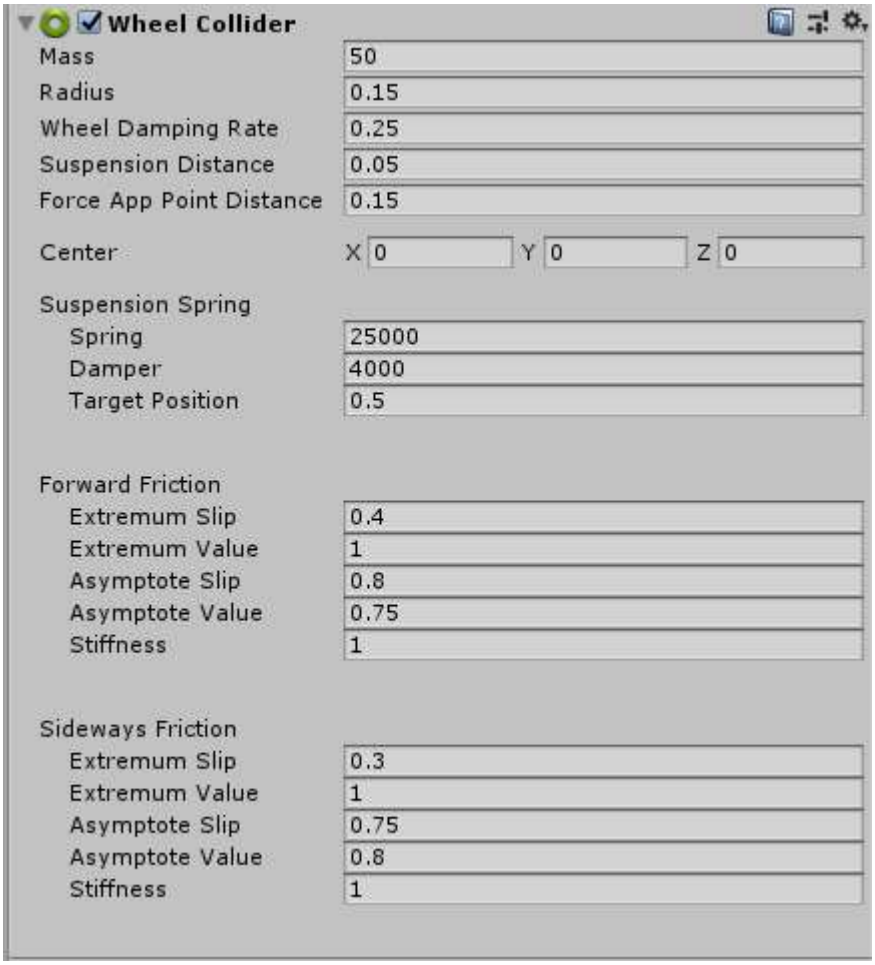
组合模式

赛车由不同的组件构成，如车轮碰撞器，碰撞体，刚体，渲染等:

赛车结构







观察者模式

发生碰撞时发送消息，播放音效

```
public delegate void Collid(float factor);

public event Collid OnCollid;        //发送碰撞事件
```

定义事件

```
//发生碰撞时，检测是否达到碰撞要求
private void OnCollisionEnter(Collision collision)
{
    if ( collision.impulse.magnitude > m_collisionImpulseLimit )
    {
        if ( OnCollid != null )
        {
            OnCollid(SpeedFactor);
        }
    }
}
```

发布碰撞事件

```
private void Start()
{
    StartSound();   //建立音源
    m_carcontroller.OnCollid += PlayCollideSound;   //订阅碰撞事件
}
```

订阅碰撞事件

```
//播放碰撞音效
public void PlayCollideSound(float speedFactor)
{
    m_collsion.Stop();
    m_collsion.volume = speedFactor;
    m_collsion.Play();
}
```

播放碰撞音效

闯关控制器通过发布各阶段的事件来进行闯关控制

```
public class ChallengeController : MonoBehaviour {

    public static ChallengeController Instance { get; private set; }

    [Header( "=== 关卡控制器配置属性 ===" )]
    [SerializeField] private float m_timeLimit;       //过关时间限制
    [SerializeField] private float m_timeCount;       //当前用时
    [SerializeField] private int m_countDown;         //读秒
    [SerializeField] private int m_turnLimit;         //过关圈数限制
    [SerializeField] private int m_turnCount;         //当前圈数
    [SerializeField] private ChallengeState m_challengeState;          //闯关状态

    [Header( "=== 外部组件 ===" )]
    [SerializeField] private LevelInfoList m_levelInfoList;      //所有关卡信息表
    private LevelInfo m_curInfo;     //当前关卡信息
    private CheckPointController m_checkPointController;         //检查点控制器

    //事件
    public event Action OnChallengePrepare;      //准备阶段调用
    public event Action OnChallengeStart;        //开始阶段调用
    public event Action OnChallengeGoingOn;      //进行阶段调用
    public event Action OnChallengeSucceed;      //成功阶段调用
    public event Action OnChallengePaused;       //暂停阶段调用
    public event Action OnChallengeFailed;       //失败阶段调用
    public event Action OnTurnFinished;          //当完成一圈时
    public event Action OnTimeCountChanged;      //当计时进行时

    //公有获取只读数据方法,
    public float TimeLimit { get { return m_timeLimit; } }
    public float TimeCount { get { return m_timeCount; } }
    public float TurnLimit { get { return m_turnLimit; } }
    public float TurnCount { get { return m_turnCount; } }
    public int   CountDown { get { return m_countDown; } }
```

模块划分

关卡控制模块
声音控制模块
相机控制模块
赛车控制模块
界面显示模块

三、用例

1. 用例文本:

[1] 玩家控制赛车移动，转向，刹车
[2] 相机始终跟随赛车，视角朝着赛车行驶方向
[3] 玩家点击相应关卡，从关卡选择界面场景切换到相应游戏关卡场景。再点击返回主界面，回到主界面场景。
[4] 玩家在主界面点击开始游戏，进入关卡选择界面

[5] 进入游戏关卡场景后，系统通过玩家赛车零件数据调整赛车各项属性。

[6] 当玩家选择关卡后，进入游戏场景，开始闯关前进行 5 秒倒计时，显示通过时间限制，在正式开始后，记录闯关时间。

[7] 玩家点开游戏后，如果是第一次运行，则可以开始新游戏，如果是之后运行，则可以选择之前的用户数据继续游戏或开始新游戏清空之前的用户数据。

[8] 玩家在退出游戏后可以保存闯关进度和赛车零件数据

[9] 在关卡选择界面，玩家可以查看各关卡通过状态，可以重复挑战

[10] 当用户点击退出游戏后，玩家的关卡通过数据仍然能保存

[11] 当用户开启游戏后，载入之前保存的关卡信息。如果是第一次开启，则会创建新的关卡信息并在退出时保存。