**Tactics 项目开发文档**

**1 Scripts**

**1）ButtonManager**

**ButtonManager.cs**

使用该脚本的对象：MainMenu → Canvas → ButtonManager，Credit → ButtonManager

用途：主菜单场景和成员列表场景中按下不同按钮所执行的函数（主菜单中LoadGame按钮除外）

主菜单中：

Start按钮：QuickStart()，沿用用户上次的设置（如果没有则用默认值）快速进行一次仿真；用户上次的设置通过PlayerPrefs类储存。

GameSettings按钮：GameSet()，进入TrackSelect场景。

Credits按钮：Credits()，进入Credit场景。

Quit按钮：quit()，退出游戏。

成员列表场景中：

MainMenu按钮：MainMenu()，回到主菜单。

**MainMenuLoadButton.cs**

使用该脚本的对象：MainMenu → Canvas → ButtonManager

用途：主菜单场景中按下Load Game按钮后，呼出存档窗口；在存档窗口按下Back按钮后，关闭存档窗口。

**CompletePanelButton.cs**

使用该脚本的对象：RaceArea → Canvas → Panel complete → CompletePanelButtonManager

用途：在仿真结束界面，按下不同按钮所执行的函数。包括MainMenu()和Retry()。

**GamePauseButton.cs**

使用该脚本的对象：RaceArea → Canvas → Panel Pause → PauseButtonManager

用途：在仿真暂停界面，按下不同按钮所执行的函数。

Continue按钮：ContinueRace()，继续仿真。

Save and Load按钮：SaveGame()，打开存档读档窗口。

Retry按钮：Retry()，重新开始。

MainMenu按钮：BacktoMainMenu()，回到主菜单界面。

**GameSetting.cs**

使用该脚本的对象：TrackSelect → GameSettingManager

用途：在GameSetting界面，按下不同按钮所执行的函数。

**同时，该脚本还储存着用户对仿真的各项设置。之后在RaceArea场景中根据用户的设置对仿真进行初始化时，均需要调用该脚本中储存的变量。**

除此之外，用户在进入TrackSelect场景后，还会调用该脚本的Start()函数，对部分参数进行初始化。

用户在进行部分设置后，需要通过PlayerPrefs类储存用户的历史选择。

Monitor的设置是单独实现在MonitorSetting.cs里的。

**MonitorSetting.cs**

使用该脚本的对象：TrackSelect → Canvas →ButtonPanel → Panel → AddMonotors → MonitorDropDownManager

用途：实现用户添加监视器的功能。储存用户对监视器的设置。

用户对监视器的设置包括：监视器数目（≤3），监视器目标，监视器的监视角度，都是通过DropDown实现，而非Button。

用户在进入TrackSelect场景后，会调用该脚本的Start()函数，根据用户的过往设置初始化相关的DropDown。

之后在RaceArea场景中MonitorManager.cs脚本根据用户的设置对监视器进行初始化时，需要获取该脚本中储存的参数。

**EscScene.cs**

使用该脚本的对象：TrackSelect → Esc， Credit → Esc

用途：在TrackSelect场景和Credit场景通过Esc键快速回到主菜单。

**quitGame.cs**

使用该脚本的对象：MainMenu → QuitGame

用途：在主菜单界面，按下Esc时，关闭仿真器

**quitRace.cs**

使用该脚本的对象：RaceArea → QuitRace

用途：在仿真场景下，按下Esc时，暂停仿真，并呼出暂停窗口

**2）Race**

**StartManager.cs**

使用该脚本的对象：RaceArea → StartManager

用途：控制巡线开始。

**进入巡线场景后，首先执行的便是该代码。**阅读代码可以从这里入手。

需要完成以下操作：

1）初始化Cpp接口；

2）判断本次运行是否需要读档

如果需要读档，则根据存档里的数据设置相应的参数；

如果不需要读档，则只需要进行一些初始化操作；

读档时，有些操作需要在巡线开始前进行，有些操作需要在巡线开始后进行，所以有loadBefore和loadAfter两个函数。

不过，目前读档功能还需要修改。

3）开启设置车辆颜色的代码；（CarColor.SetActive(true)）

4）开启设置不同巡线模式下UI的代码；（ModeManager.SetActive(true)）

5）开启切换视角功能；（ViewModeManager.SetActive(true)）

6）开启车辆控制的相关代码；（CarControl.SetActive (true);）

6）开启计时；（LapTimer.SetActive (true)）

7）开启车辆Speed、Steer、CruiseError的实时显示；

8）巡线开始前的倒计时；

**CarColorSetting.cs**

使用该脚本的对象：RaceArea → CarColor

用途：根据用户的选择，设置车辆的颜色。

对于每辆车，分别获取用户设置的颜色，再将对应颜色的车壳setActive(true)即可。

**carControlActive.cs**

使用该脚本的对象：RaceArea → CarControlActive

用途：根据用户的选择，开启每辆车的控制代码。

如果某辆车的ControlMethod为1，则为键盘控制，需要开启对应Car的CarUserControl组件；

如果某辆车的ControlMethod为2，则需要Cpp代码控制，将CallCppControl对象setActive，以此来调用Cpp代码。

**GameModeManager.cs**

使用该脚本的对象：RaceArea → ModeManager

用途：根据用户选择的模式，设置巡线时的UI以及相关GameObject

首先将车辆损伤、车辆得分等数据置零。

若选择TimeMode（也即RaceMode），则需要将TimeMode相关的UI开启，显示巡线时间和车辆损伤。

若选择ScoreMode，则需要将ScoreMode相关的UI开启，显示每辆车所得分数和损伤。同时，还要将赛道上的奖励宝石setActive（也即ScoreObjects）

同时还要考虑参与玩家的数量不同导致的UI差别。

**HalfPointTrigger.cs**

使用该脚本的对象：RaceArea → HalfPointTrigger

用途：记录车辆是否已经行驶了半圈，和LapComplete.cs配合使用。若车辆没有经过半途的检查点，即使车辆经过终点线也不会被判断为行驶了一圈。

每辆车对应一个叫HalfFlag的bool型变量，如果该车通过了半途的检查点则该变量值为true，否则为false。

**LapComplete.cs**

使用该脚本的对象：RaceArea → LapCompleteTrigger

用途：检测车辆是否通过起点，与HalfPointTrigger.cs配合使用，判断车辆是否行驶完了一圈。

车辆通过终点检查点时，若检测到之前已经通过了半途检查点（HalfFlag为true），则判断该车行驶完了一圈，圈数＋1，并设置对应的HalfFlag和LapFlag，开始下一圈的计数。

若车辆行驶的圈数达到目标圈数，巡线结束，将RaceFinish对象SetActive。

**LapTimeManager.cs**

使用该脚本的对象：RaceArea → LapTimeManager

用途：在TimeMode下，巡线计时，并在窗口左上角显示巡线时间。

**CruiseData.cs**

使用该脚本的对象：RaceArea → Car

用途：计算小车距离中心线的距离、赛道曲率等。

参数CarNum为当前小车的编号，不同Car中CruiseData组分的CarNum数值不同。

DistanceError储存小车距离中心线的距离；

Curvature储存当前位置赛道中心线曲率；

AngleError储存小车方向和赛道中心线方向的差距。

WaypointsData为当前赛道中心点的数据，数据为xml格式，是通过CarWaypoints插件生成的。该插件可以很方便地获取赛道中心路标点。

使用该插件时，需要将Editor文件夹中的CarWaypoints文件夹移动到Editor文件夹外；而使用完该插件后，需要将CarWaypoints文件夹放回Editor文件夹。该插件的使用教程可以在CarWaypoints文件夹中找到。

计算小车距离中心线的距离的方法：获取距离小车最近的两个路标点，以这两点作直线，求小车距离该直线的距离。

计算小车当前位置赛道中心线曲率的方法：获取离小车最近的一个路标点以及之后的第一个、第三个路标点，过这三个点做圆，该圆半径的倒数就为当前曲率。不过由于数值大小原因，返回的曲率值为半径倒数的十倍。

**CollisionDamage.cs**

使用该脚本的对象：RaceArea → Car

用途：检测小车的碰撞和损伤。

当小车收到碰撞时，获取小车抵消该碰撞所需的冲力，并记录小车碰撞次数和碰撞损伤程度。

**CurrentScore.cs**

使用该脚本的对象：无

用途：在ScoreMode仿真中，储存当前各车辆的得分。

**BlueScore.cs**

使用该脚本的对象：RaceArea → ScoreObjects → BlueScore

用途：在ScoreMode仿真中，当车辆触碰到蓝色宝石后，给该车辆加分。

**RedScore.cs**

使用该脚本的对象：RaceArea → ScoreObjects → RedScore

用途：在ScoreMode仿真中，当车辆触碰到红色宝石后，给该车辆加分。

**YellowScore.cs**

使用该脚本的对象：RaceArea → ScoreObjects → YellowScore

用途：在ScoreMode仿真中，当车辆触碰到黄色宝石后，给该车辆加分。

**RaceFinish.cs**

使用该脚本的对象：RaceArea → RaceFinish

用途：巡线结束时执行的代码，主要功能为关闭车辆控制相关的代码，并呼出巡线结束的UI。其中ScoreMode和TimeMode的结束UI不完全相同。

**3）CameraManage**

**CameraStable.cs**

使用该脚本的对象：RaceArea → Car → Cube

用途：稳定相机。相机为Cube的子对象，在Cube上附加该代码后， Cube在X方向和Z方向不偏转，相机的拍摄角度也因此稳定。

**MonitorManager.cs**

使用该脚本的对象：RaceArea → Canvas → Monitors → MonitorManager

用途：根据用户的设置，仿真时在屏幕上方添加小窗口显示监视器中的画面。

**ViewModeManager.cs**

使用该脚本的对象：RaceArea → ViewModeManager

用途：在仿真界面，用户按下c键可以切换视角，按下x键可以切换当前观察的车辆。

关于c键和x键的输入，可以参看Edit → Project Settings → Input Manager。

每辆车有四个摄像头，分别为普通视角、第一人称视角、远视角、俯视角，参数ViewMode表示当前视角；

用户还可以在不同车辆之间切换视角，参数CamNum表示当前观察的车辆编号。

**4）Display**

**LapNumDisplay.cs**

使用该脚本的对象：RaceArea → Canvas → UIRight → PanelRight → LapNumDisplayManager

用途：巡线时，在窗口右上角显示当前视角对应车辆的已行驶圈数。

**SpeedDisplay.cs**

使用该脚本的对象：RaceArea → Canvas → UIBottom → SpeedDisplay → SpeedDisplayManager

用途：获取每辆车的行驶速度，并显示在屏幕下方的UI上。

Speed数组储存各车辆的行驶速度，但屏幕下方只显示当前视角跟随的车辆的速度值。

**SteerDisplay.cs**

使用该脚本的对象：RaceArea → Canvas → UIBottom → SteerDisplay → SteerDisplayManager

用途：获取每辆车的方向盘转角，并显示在屏幕下方的UI上。在第一人称视角模式下，还会显示方向盘的贴图。

和SpeedDisplay.cs不同（有一个Speed数组，分别储存每辆车的速度），只有一个steer参数储存当前视角所跟随车辆的方向盘转角。

**ErrorDisplayManager.cs**

使用该脚本的对象：RaceArea → Canvas → UIBottom → ErrorDisplay → ErrorDisplayManager

用途：获取每辆车距离赛道中心线的距离，并显示在屏幕下方的UI上。

和SpeedDisplay.cs不同（有一个Speed数组，分别储存每辆车的速度），只有一个CruiseError参数储存当前视角所跟随车辆的巡线误差。

**DamageDisplay.cs**

使用该脚本的对象：RaceArea → Canvas → UILeft → ……DamageDisplay → DamageDisplayManager

用途：仿真过程中，在窗口左上角显示各车辆的损伤程度。

ExtentOfDamage数组和CollisionNum数组分别储存各车辆的损伤程度和碰撞次数。

参数CarNum表示该脚本要显示几号车辆的损伤值。若脚本是显示一号车辆损伤程度的UI的组件，该值则设为0；若脚本是显示二号车辆损伤程度的UI的组件，该值则设为1……以此类推。

**ScoreDisplay.cs**

使用该脚本的对象：RaceArea → Canvas → UILeft → ScoreModeDisplay → ScoreModeDisplay

用途：ScoreMode仿真过程中，在窗口左上角显示各车辆的得分。

参数CarNum表示该脚本要显示几号车辆的得分。若脚本是显示一号车辆得分的UI的组件，该值则设为0；若脚本是显示二号车辆得分的UI的组件，该值则设为1……以此类推。

**MiniMap.cs**

使用该脚本的对象：RaceArea → Canvas → UIRight → MiniMap → MarkPlayerCar

用途：控制小地图的车辆标识位置和方向。

参数CarNum表示该脚本控制的是几号车的小地图标志。

**5）CarControlCpp**

**CppControl.cs**

使用该脚本的对象：无

用途：实现Cpp代码和Unity的接口。

API的详细情况请参考CppControl.cs和CppCarControl.h中的定义。

**CallCppControl.cs**

使用该脚本的对象：RaceArea → CppControl

用途：调用外部cpp代码，控制车辆运行。

为保证仿真确定性，不使用Update()函数而使用FixedUpdate()函数。

仿真开始时，在Start()内会调用CppControl.InitializeCppControl()，进行仿真初始化。

每次在FixedUpdate()中调用CppControl.CarControlCpp()函数。

InitializeCppControl()函数和CarControlCpp()函数定义在CppControl.cs脚本中，但实际内容在外部cpp代码中实现。

用户在CarControlCpp()中编写代码进行运算，将运算得到的控制参数（即steering、accel、footbrake、handbrake）通过CarMove函数传递到CppControl.cs脚本中的steering[]、accel[]、footbrake[]、handbrake[]数组中，再在CppControl.cs脚本中使用CarController.Move函数控制车辆运行。

**6）CarControl（Unity自带）**

**CarController.cs**

使用该脚本的对象：RaceArea → Car

用途：实现车辆模型和车辆控制，目前使用的是Unity自带的代码。

若需要修改车辆动力学模型，应该从这入手。

**CarUserControl.cs**

使用该脚本的对象：RaceArea → Car

用途：实现车辆的键盘控制，是根据Unity自带的代码修改的。

参数CarNum表示该脚本控制的是几号车辆。若脚本为一号车辆的组件，则将该值设为0；若脚本为二号车辆的组件，则将该值设为1……以此类推。

**7）Save**

**SaveTactic.cs**

使用该脚本的对象：无

用途：定义SaveTactic类，明确存档时需要储存哪些内容。

由于存档功能大改，因此该类也需要修改。

**SaveCondition.cs**

使用该脚本的对象：无

用途：定义SaveCondition类，记录各个存档位的使用情况。

**SaveConditionManager.cs**

使用该脚本的对象：RaceArea → Canvas → Panel Save → SaveConditionManager

用途：根据存档位使用情况，修改UI。

若存档没有被使用，在仿真场景中查看该存档则需要显示Save按钮以供用户存档；在主菜单中查看该存档则不能有任何按钮。

若存档已经被使用，在仿真场景中查看该存档则需要显示Save按钮和Load按钮，分别提供存档和读档的功能；在主菜单中查看该存档则需要显示Load按钮和Delete按钮，分别提供读档和删档功能。

**SaveButton.cs**

使用该脚本的对象：RaceArea → Canvas → Panel Save → PanelRoll → Panel → Save Savebutton

用途：实现存档功能。

在存档窗口按下Save按钮后，储存当前的状况。将当前的状况储存为一个SaveTactic类对象，并将其转化为二进制文件储存在相应的文件夹中，并修改存档使用情况。

**LoadButton.cs**

使用该脚本的对象：RaceArea → Canvas → Panel Save → PanelRoll → Panel → Save Savebutton，MainMenu → Canvas → Panel Load → PanelRoll → Panel → Save Button → Delbutton/Loadbutton

用途：实现读档和删档功能。

在存档窗口按下Load按钮后，进入该存档对应的赛道场景中，并令参数LoadNum为对应的存档号（不等于0）。当LoadNum不等于0时，进入仿真场景后，StartManager.cs脚本会读取对应的存档并进行相应的设置。

在存档窗口按下Delete按钮后，删除对应存档，并修改档位使用情况。

**8）Others**

**SkyBoxRot.cs**

使用该脚本的对象：RaceArea和TrackSelect场景中的任意一个对象均可

用途：令RaceArea和TrackSelect场景中背景的天空缓缓旋转。

**9）TrackData**

该文件夹储存道路中心路标点文件，这些文件目前都是通过CarWaypoints插件生成的。这些数据在CruiseData.cs脚本中被使用。

**10）WayPointData**

CarWaypoints插件插件所需使用的部分脚本，可以参考CarWaypoints插件教程理解。教程可以在Editor/CarWaypoints文件夹中找到。

**2 GameObjects（RaceArea）**

**1）Car**

即用户控制的车辆。

装载刚体组件（Rigidbody），还装载着CarController.cs、CarUserControl.cs、CarAudio.cs、ColisionDamage.cs、CruiseData.cs脚本。

其子对象中，Cube用于稳定摄像头；

SkyCar中为车辆的外观模型，CarColorSetting.cs脚本设置车辆颜色时，就是将对应颜色的车壳setActive，而不同颜色的车壳就包含在SkyCar中。

**2）Canvas**

用户看到的UI。

UILeft：左上角的UI，又分为TimeMode的UI和ScoreMode的UI。

UIRight：右上角的UI，主要显示当前圈数和小地图。

UIBottom：下方UI，主要显示当前车辆的速度、方向盘方向和巡线误差。

Monitors：当用户添加了监视器时，在上方显示的监视器窗口。

CountDown：巡线开始时的倒计时UI。

Panel Save：存档窗口的UI。

Panel complete：结束窗口的UI。

Panel Pause：暂停窗口的UI。

**3）StartManager**

控制仿真开始的GameObject，装载StartManager.cs脚本。

在执行完一定的操作后，会让对象ModeManager、LapTimeManager、CarControlActive、CarColor、ViewModeManager激活。

**4）ModeManager**

由StartManager激活，根据用户选择的模式（TimeMode或ScoreMode）设置UI。装载GameModeManager.cs。

**5）LapTimeManager**

由StartManager激活，计时。装载LapTimeManager.cs。

**6）CarControlActive**

由StartManager激活，根据用户选择的控制方式，分别激活每辆车的键盘控制功能或代码控制功能。装载CarControlActive.cs。

**7）ViewModeManager**

由StartManager激活。装载ViewModeManager.cs，实现用户切换视角和车辆的功能。

**8）CarColor**

由StartManager激活。装载CarColorSetting.cs，根据用户的选择设置各车辆的颜色。

**9）CppControl**

由ModeManager激活。当仿真要使用外部cpp代码进行车辆控制时，该对象会被激活。装载CallCppControl.cs脚本，实现cpp控制功能。

**10）ScoreObjects**

由ModeManager激活。ScoreMode中场地上的宝石。在ScoreMode下，该对象会被激活。

**11）HalfPointTrigger**

触发器，不渲染但是有碰撞箱。判断车辆是否通过半途检查点。装载HalfPointTrigger.cs脚本。

**12）LapCompleteTrigger**

触发器，不渲染但是有碰撞箱。判断车辆是否通过终点线。装载LapComplete.cs脚本。若车辆通过终点线，仿真结束，则会激活RaceFinish。

**13）RaceFinish**

由LapCompleteTrigger激活。当仿真结束时，该对象会被激活。装载RaceFinish.cs脚本，完成仿真结束时所需的操作。

**14）QuitRace**

装载QuitRace.cs，按下Esc后暂停游戏并呼出暂停窗口。

**15）RaceScene**

赛道的物理模型和环境的模型等。这里面的GameObject主要是场景中的地形和装饰等。