**红色拖拉机**

代码说明

2019年3月17日

**常微分方程组**

赵清锐 陆星存 刘思语 郭泓灏 李子明 杜鎔杰

目录

[概述 1](#_Toc3747800)

[几点值得说明的算法 2](#_Toc3747801)

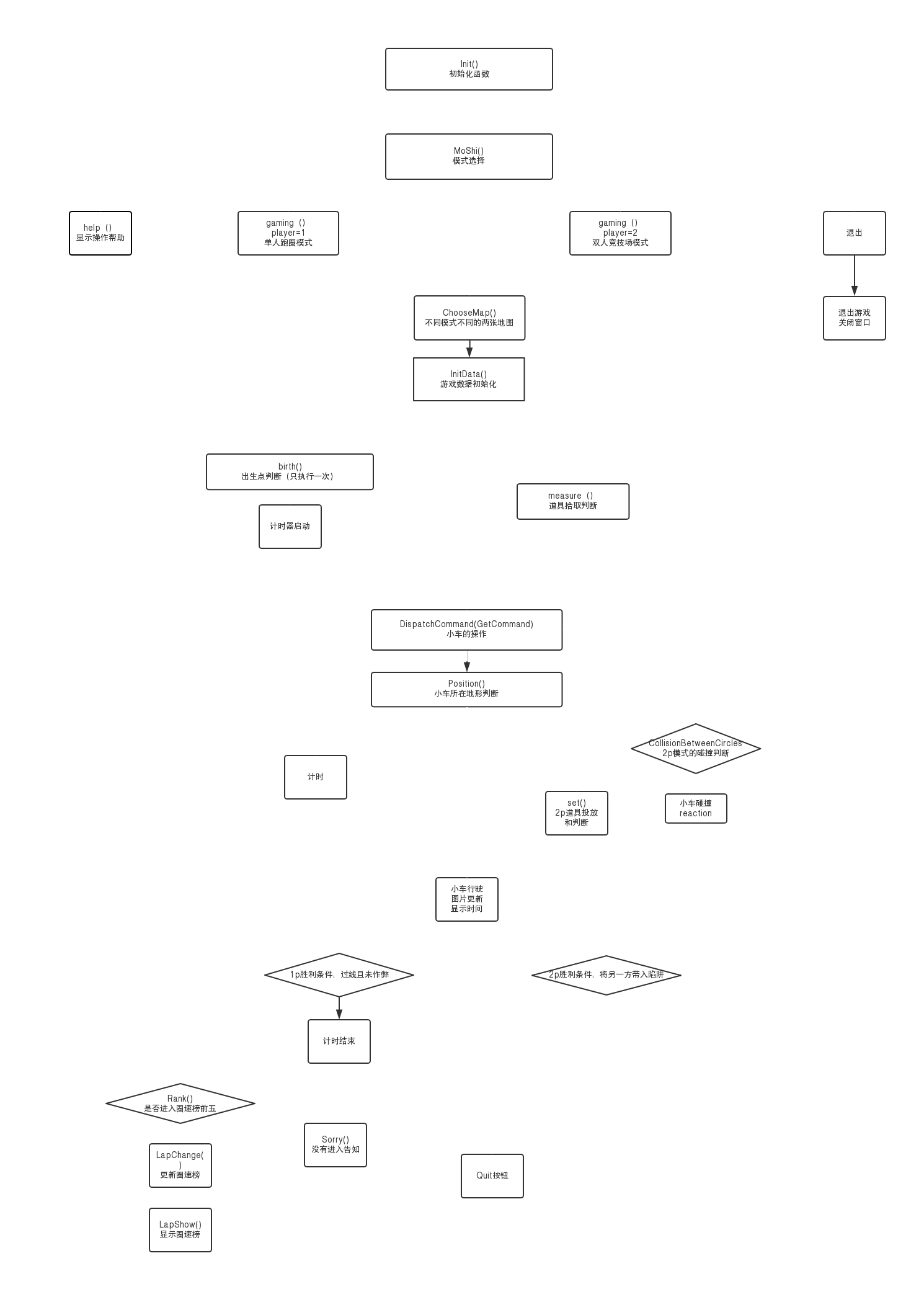
[对于赛车行驶的视觉效果的实现方案 2](#_Toc3747802)

[对于赛道边界和不同地形的检测判断 2](#_Toc3747803)

[TWO-PLAYER模式下的道具算法 4](#_Toc3747804)

[阅读代码时的建议 5](#_Toc3747805)

# 流程图



# 概述

该游戏使用Visual Studio2017及EasyX编写。

游戏名称“红色拖拉机”（暂定）为法拉利车迷对法拉利跑车的昵称。

因上传文件的大小限制，我们无法上传完整的工程文件，因此我们只上传了2B.cpp文件和NieR.h头文件以及一个目前游戏的可执行文件。

 进入游戏界面后，有两种模式可供您选择，您可以通过单击界面右侧的“HELP”按钮来了解游戏两种模式的特点和游玩方式。如果您不想继续游戏，只需单击右下角的“QUIT”按钮即可退出。

“TOURNAMENT”模式是我们为赛车狂热爱好者提供的一种全新游戏体验，该模式中所有的赛道都是我们根据现实中F1的赛道而设计出来的。因此，该模式中，您的主要目的是尽全力跑出最快的圈速（注意：请尽量不要跑出赛道，否则在您冲线时可能会因被判抄近道而取消成绩）。我们会提供圈速榜以供您与其他玩家竞技，尽全力跑出最快圈速，占领榜首吧！

游戏主界面

“ARENA”模式是双人娱乐模式，类似于碰碰车。竞技场地不再是经典赛道，而是有我们设计的双人角斗场。在这个角斗场中，地形将会发生巨大的变化，甚至道路上会有设计好的陷阱，请您谨慎驾驶。“行车不规范，亲人两行泪。”与此同时，您的对手会操控另一辆小车与您进行竞技。请尽可能在安全的情况下将对方撞入红圈内，这就是您的胜利条件。此外，道路上还会随机出现增益道具，拾取后能产生特定的效果，为您带来优势。

更多详细内容请参见游戏文件内的“HELP“菜单。

最后，请尽情享受我们为您呈现的游戏之旅吧！

# 几点值得说明的算法

## 对于赛车行驶的视觉效果的实现方案

不同于其它游戏元素的W键控制向上、S键控制向下、A键控制向左和D键控制向右，赛车游戏的WASD键需要分别实现加速、左转、减速/倒车、右转的效果，因此我们使用类似积分的方法来绘制赛车的运动轨迹，代码如下：

Mx1 = Pspeed \* cos(1\*PForward) + Px;

My1 = Pspeed \* sin(1\*PForward) + Py;

Px = (int)ceil(Mx1);

Py = (int)ceil(My1);

该段代码约每25毫秒就执行一次，其中Pspeed代表赛车的速度，PForward代表赛车的偏转角，Mx1与Mx2将小车坐标暂存之后再以整型传回Px，Py。并在之后的代码中将小车的图像显示于Px，Py坐标。同时用

rotateimage(&Player1, &car1, -PForward, WHITE, true, false);

函数旋转小车图片使车头指向运动方向。以此在视觉效果上实现小车的行是而不是单纯平移。

## 对于赛道边界和不同地形的检测判断

为了实现丝滑而准确的赛道边界判断，我们使用的是EasyX中的GetImageBuffer函数。EasyX中对GetImageBuffer函数的定义和描述如下。

**GetImageBuffer**

这个函数用于获取绘图设备的显存指针。

WORD\* GetImageBuffer(IMAGE\* pImg = NULL);

**参数：**

pImg

绘图设备指针。如果为 NULL，表示默认的绘图窗口。

**返回值：**

返回绘图设备的显存指针。

**说明：**

获取到的显存指针可以直接读写。

在显存中，每个点占用 4 个字节，因此：显存的大小 = 宽度 × 高度 × 4 (字节)。像素点在显存中按照从左到右、从上向下的顺序依次排列。访问显存请勿越界，否则会造成难以预料的后果。

显存中的每个点对应 RGBTRIPLE 类型的结构体：

struct RGBTRIPLE {

BYTE rgbtBlue;

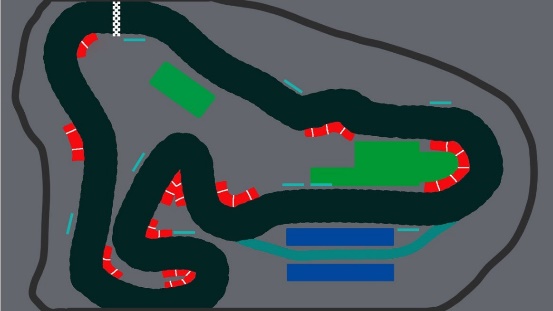
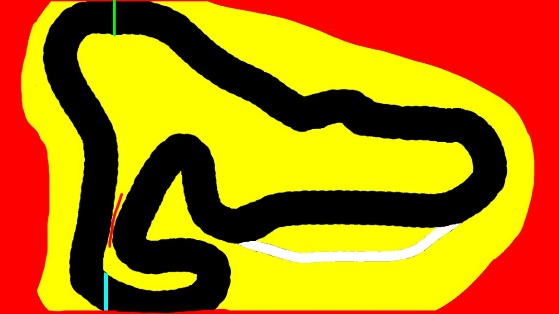
BYTE rgbtGreen;

BYTE rgbtRed;

}

RGBTRIPLE 在内存中的表示形式为：0xrrggbb (bb=蓝，gg=绿，rr=红)，而常用的 COLORREF 在内存中的表示形式为：0xbbggrr。注意，两者的红色和蓝色是相反的，请用 BGR 宏交换红色和蓝色。

如果操作绘图窗口的显存，请在操作完毕后，执行 FlushBatchDraw() 使操作生效。

在实际使用中，我们将绘制好的赛道地图转换成一副由红绿蓝三基色或以其简单混合来填色的副本（我们将其命名为pengzhuang，使用三基色及简单混合显然是因为好判断颜色）如下：

赛道图

pengzhuang

具体实现方式如下。

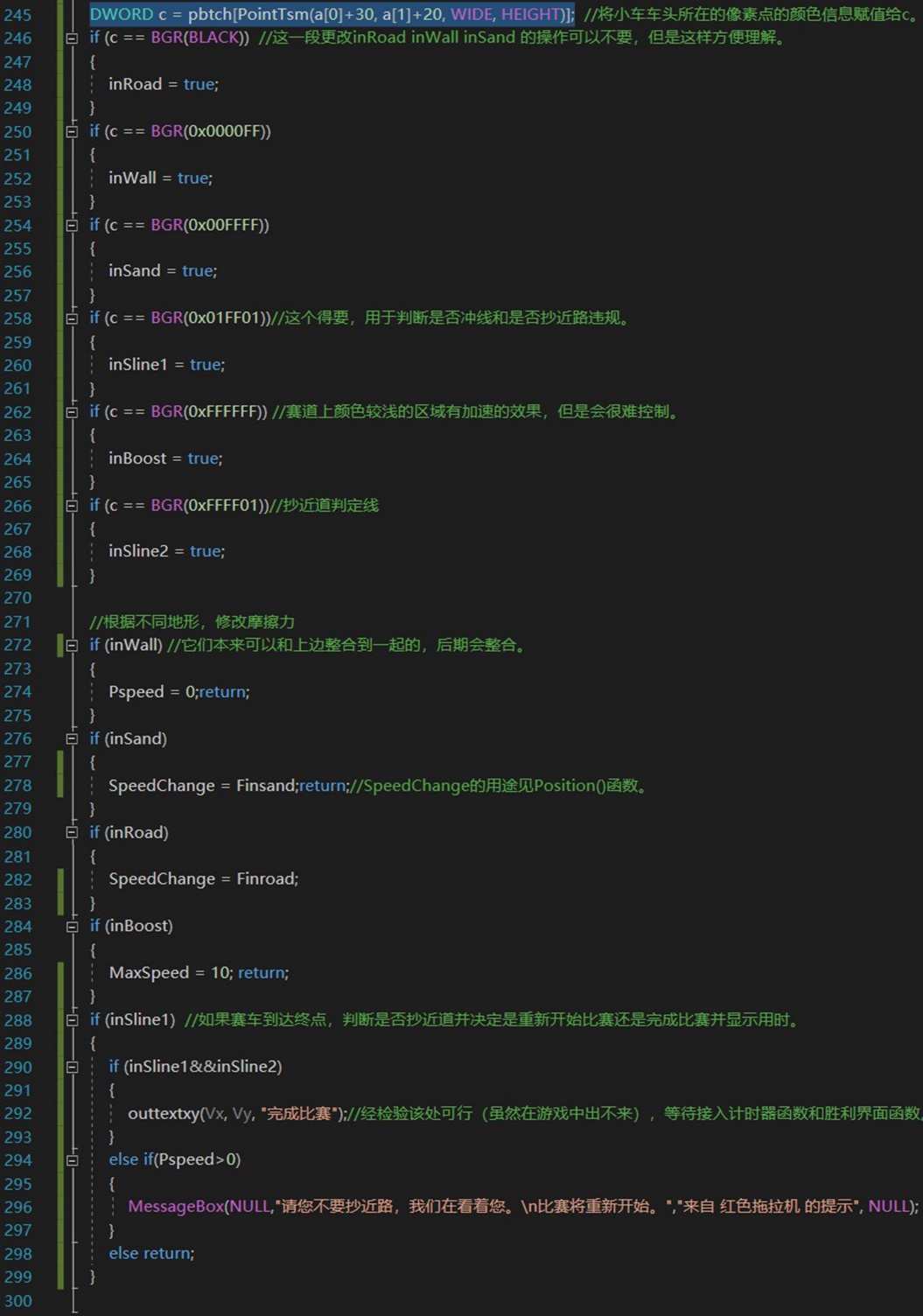
边界检测函数命名为MoveCheck()，在该函数中，首先定义

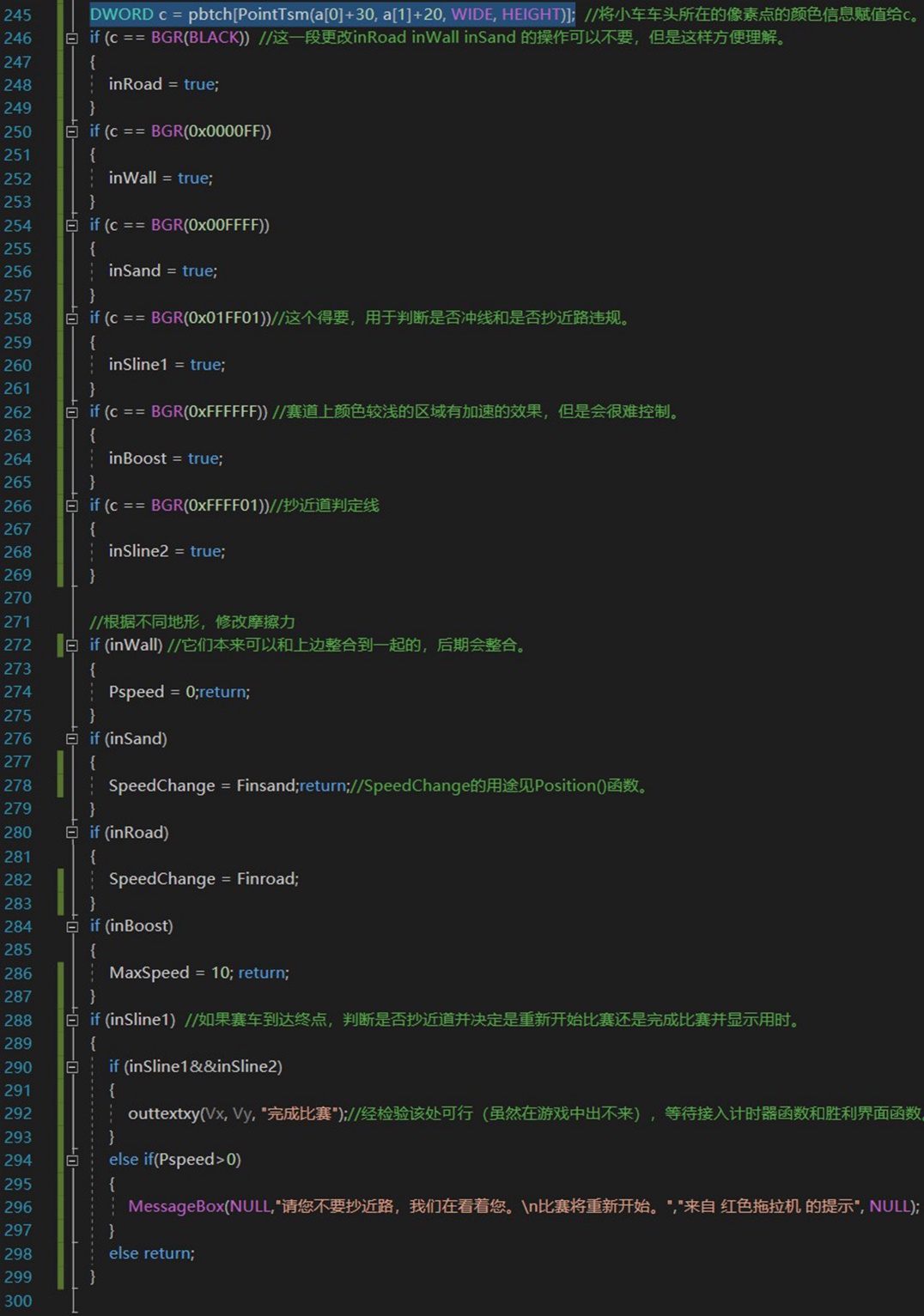
DWORD\* pbtch = GetImageBuffer(&pengzhuang);

即指向pengzhuang图片显存的指针pbtch。

之后计算出小车图片车头的坐标，因为若直接使用图片坐标容易出现小车卡进红测区域后就一直判定为inWall状态而无法运动的情况。使用车头坐标后小车可以通过转向来脱离该状态。之后再计算出该坐标所对应的图片显存数组位数，使用pbtch得到该位的颜色信息并赋值给变量c。

DWORD c = pbtch[PointTsm(a[0]+30, a[1]+20, WIDE, HEIGHT)];

后将变量c与BGR颜色信息进行比对从而得知小车目前所处的路面信息并做出进一步的操作。该部分代码如下：

在我们的游戏中，小车的出生点判断和是否抄近路的判断也是用此思路，详见2B.cpp文件的335-356行及285-295行。

## TWO-PLAYER模式下的道具算法

首先添加释放技能键位，根据最方便的位置选择两个玩家的道具按键，之后通过位运算，将按键与技能函数联系在一起，我们定义了三个结构体，分别储存道具设置，玩家道具数据，道具开关，建立了独立的道具投放函数，通过随机数据进行投放，在赛车移动的过程中即时计算两辆赛车与道具的距离，如果距离小于给定的距离，就判定该道具被拾取，道具图标被移至窗口外，同时判断该道具种类，玩家已获得道具数加一，在随后的道具被释放的时候，道具数相应减一，实现了拾取数与道具数相等。道具投放的时间间隔在之前已被定义，在后期测试时根据游戏节奏进行调整即可。

关于道具效果实现，使用独立函数，在已获得道具的前提下，按键并启动相关函数，改变小车位置数据。实现相关效果。

# 阅读代码时的建议

本说明中只简要展示了该游戏的大致框架和需要说明的算法，具体的细节方面您可以在源代码中得到详细的注释说明。

在阅读代码时，建议从main()函数看起，或参照上文中给出的流程图以便更好的了解到代码的整体框架。

**感谢您花费时间阅读本说明书和我们的代码，**