目录

1 概述

1.1 背景&目标

随着 RoboMaster 赛事影响力的扩大，招新工作的重要性也不断提升。校内赛作为招新、宣传的一大渠道，需要保证趣味性的同时，不断提高技术上限。更加贴合实际联盟赛的、对抗赛的情况，帮助新生快速了解比赛、产生兴趣。

在此背景下，一套有趣的裁判系统就变得尤为重要。这套系统拥有以下优点：低成本、高稳定、UI 贴近实际比赛，具有一定的可扩展性。能够在每车拥有四块装甲板的情况下，支持 2v2 的对抗赛。实时绘制各车辆血条，状况（有无 buff），复活倒计时，击杀提示并自动判定胜利，效果如图 1-1。

图 1-1 校内赛裁判系统直播截图

当前，其他学校的裁判系统主要以人工裁判为主，即使有自动判定也比较简单，仅为单终端或四五个终端有线连接，在比赛的观赏性上有待提升。而该系统能支持 6 至 15 个终端同时运行（已测试），理论上至多支持 20 套设备同时互联。本系统在 2023 年 11 月的校内赛中已经投入使用，经使用测试达到了设计目标。

1.2 裁判系统功能定义

裁判系统主要由主裁判端、选手端、车载终端、基地终端组成，系统模块图如下图 1-2 所示

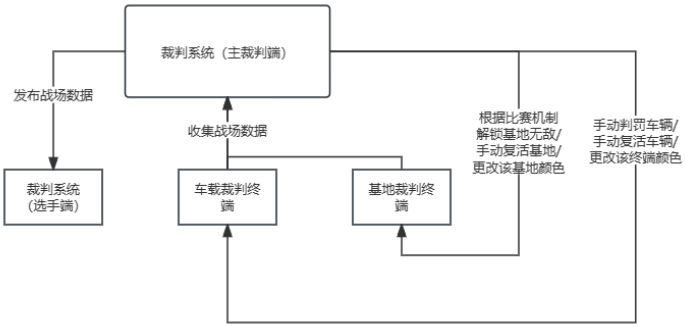


图 1-2 系统模块图

功能需求：

1) 主裁判系统作为消息中枢，起到了接收车载端、基地终端数据并转发给选手端。此外，还需要在大屏幕上实时绘制 ui，判罚，自动判定胜负等功能。
2) 选手端系统需要接收来自主裁判端转发的数据、实时绘制 ui，编码选手操作并发送。
3) 车载端和基地的端需要检测装甲板的打击扣除相应的血量，车载端还需要使用颜色传感器读取场地情况以获得不同增益效果。
4) 在一方任意单位被击杀后，基地解除无敌状态。若没被击杀，则最后 30 秒自动解除无敌状态

用户需求：提供可视化界面进行实时数据展示，提供技术暂停按钮，提供命令台对指定车辆判罚。

经济可行性：该系统软硬件成本都较低，车载裁判终端仅需 20，一块装甲板成本不超过 5 元，即使准备 10 套裁判系统也不超过 500 元，其低廉的成本适合各学校校内赛举办借鉴。而这套系统能大大提高赛事直播的观赏性趣味性，方便获取更多经费支持比赛。

技术可行性：该系统主要使用 python 与 c 两种语言，电脑端程序都使用 python 编写，调用标准库，后续可以使用 exe 封装便于使用。Python 面向对象的特点也便于实现更复杂的 ui 显示效果。

2 裁判系统概览

裁判系统主要由主裁判端、选手端、车载端、基地端组成。主裁判端、选手端是由电脑所允许的 python 程序，车载端、基地端。程序目录示意图如下。程序目录示意图如下 2-1 所示。

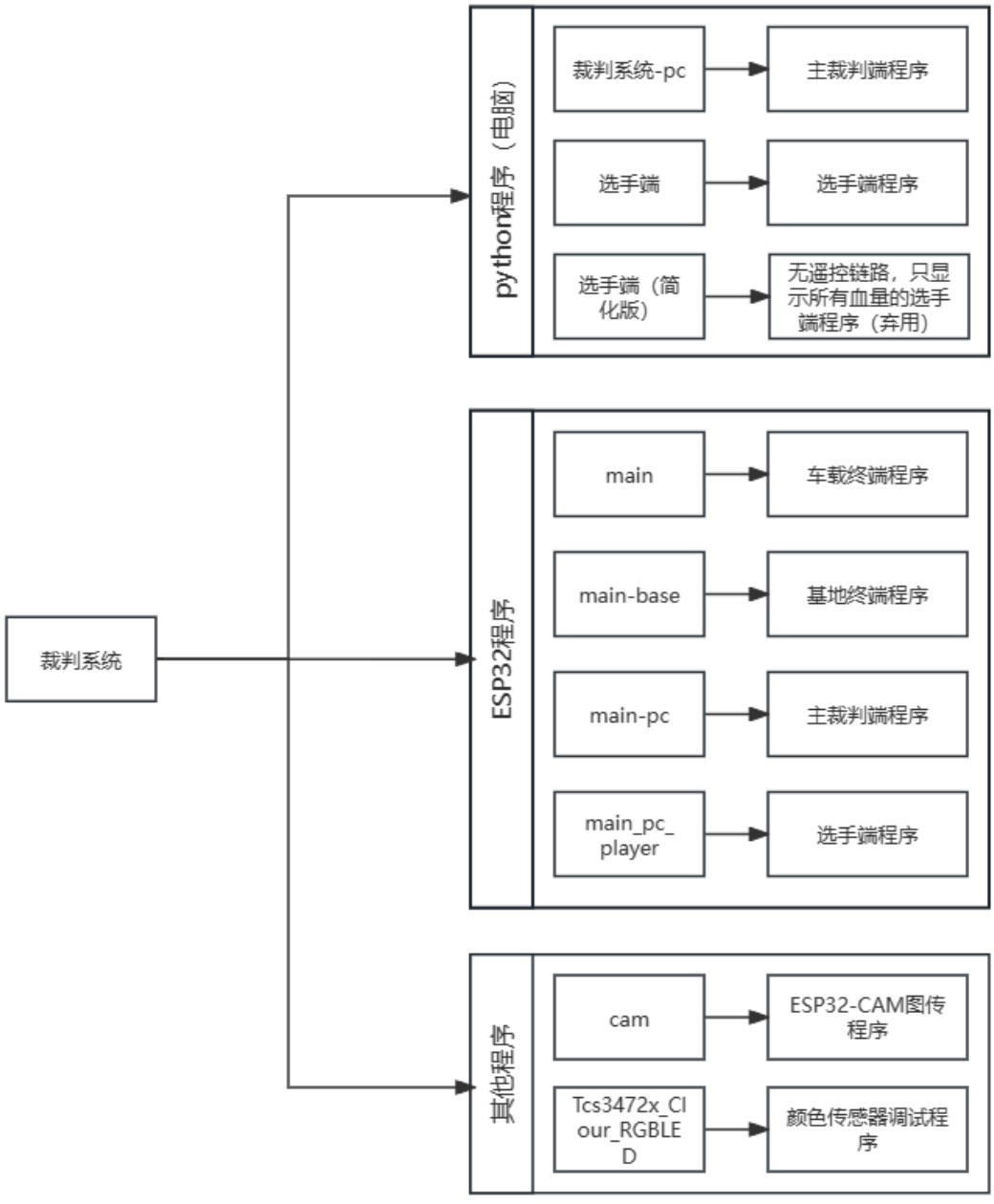


图 2-1 程序目录结构示意图

2.1 电脑端功能

2.1.1 主裁判端功能定义

裁判系统 pc 端主程序为 judgement-main.py，直接使用 python 环境运行，程序结构如图 2-2 所示。

环境需要：

pygame（ui 绘制库） opencv pyserial（串口通信库）

我的环境（仅供参考）：

Python	3.8
pygame	2.5.1
pyserial	3.5
opencv-python	4.4.0

建议使用 anaconda 配置虚拟环境

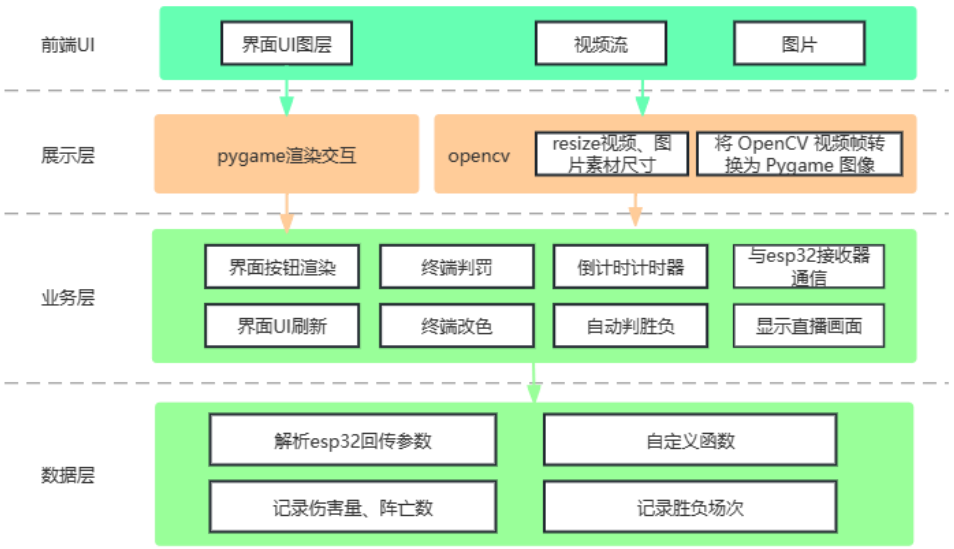


图 2-2 主裁判端程序结构图

2.1.2 选手端功能定义

选手端与主裁判端主体架构一致，区别仅限于无法判罚终端、增加了控制链路（蓝牙连接车辆）、增加了死亡时屏蔽画面和操作、左下角显示血量，注意：选手端 UI 的数据并非直接来自场上车辆，而由主裁判端进行重新编码转发以通过一串数据刷新所有单位的状态。同时，最终胜负判断也是由每个选手端接收的数据自行做出判罚（这点建议改进）。选手端功能定义：

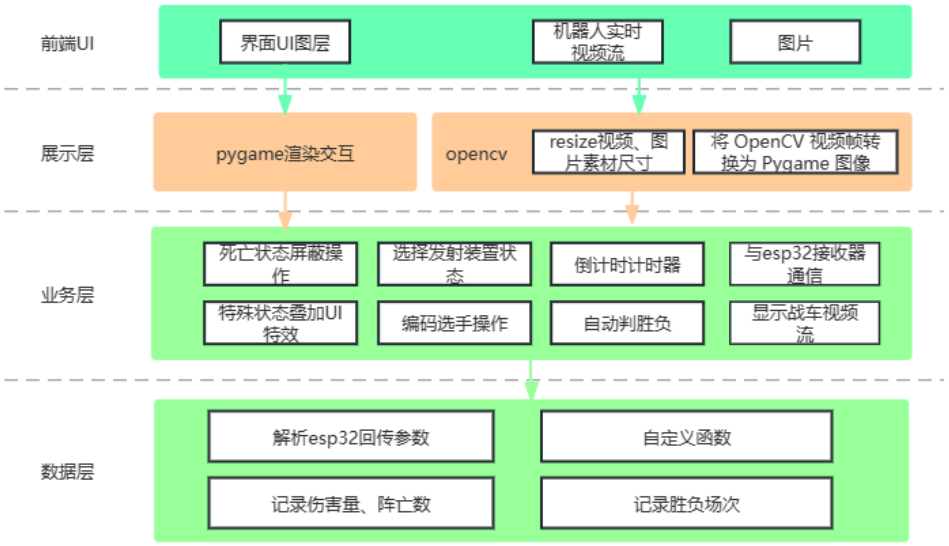


图 2-3 选手端程序结构图

2.2 ESP32 端功能

在 2v2 的情况下，ESP32 终端最少需要有 7 个：4 个车载端、一个主裁判、两个基地。若需要使用选手操作端，则需要再增加 4 个 ESP32 接收器。为了缩短准备时间，可以制作两组共 8 套车载终端，其中 4 套在场上使用，另外四套则利用这段时间拆除，并安装到下一场比赛的车辆上，ESP32 架构示意如图 2-4 所示。其中主裁判端 ESP32 作为核心网关，所有其他终端数据都会汇集到主裁判端 ESP32 核心网关进行处理，并转发给选手终端。主裁判核心网关所接收的数据会传输给主裁判电脑端绘制 UI，主裁判电脑做出的判罚也会通过核心网关下发到指定车载或基地终端。

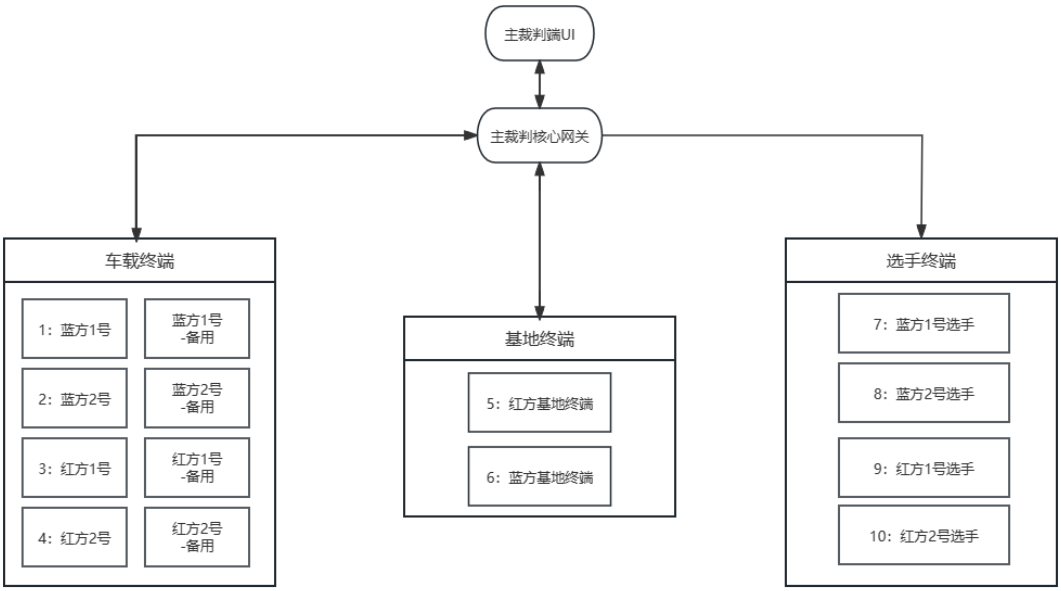


图 2-4 ESP32 通信架构示意图

ESP32 车载端需要连接四块装甲板、一个血量条与一个颜色传感器用于识别场地。基地端只需要连接两块装甲板与一个血量条，终端示意如下图 2-5。会在后续章节详细阐述。

图 2-5 裁判系统硬件示意图

2.2.1 主裁判端与场地端的通信

主裁判端接收来自车辆、基地的数据，数据格式如下表所示：

数据定义	数据类型	数据内容
车辆编号 myData.me	int	1-6
车辆状态 myData.state	int	正常 0，防御 buff 1
是否回血 myData.cure	bool	True/false
当前血量 myData.hp	int	车载 0-10、基地 0-30

数据结构体如下图所示

```
25  typedef struct struct_message {
26    int me;
27    int hp;
28    int state;
29    bool cure;
30  } struct_message;
```

主裁判端向车辆、基地发送的数据，数据格式如下表所示

数据定义	数据类型	数据内容
终端颜色 color	int	不亮 0 红 1 蓝 2
裁判指令（车） punish	int	杀死 1 满血 2 减半血 3
裁判指令（基地） punish	int	解除无敌 1 无敌回满血 2

数据结构体如下图所示

```
typedef struct judge_message {
  int color;
  int punish;
} judge_message;
judge_message judgement;
```

2.2.2 主裁判与选手端通信

主裁判端和选手端都需要连接 ESP32 作为信息收发器。

主裁判端的 ESP32 承担了全场最重的任务，在接收来自场上 6 个终端信息的同时，需要联合电脑端将数据整理编码，再转发给所有选手端。主裁判端拥有最高权限，因此未将其编号。

通信发送的内容为一个 6 列 2 行的二维数组，同一列的第一行表示该终端的血量，第二行表示该终端的状态。而二维数组的列数则表示终端的号码，例如：第一列索引为 0，表示为 1 号终端的信息；第 5 列索引为 4，表示为 5 号终端的信息。注意：在实际传输时，会将 6x2 的二维数组拉伸展开成 12x1 的一维数组。

3 技术细节

3.1 电脑端程序

3.1.1 主裁判端程序

主裁判用例如下图 3-1 所示

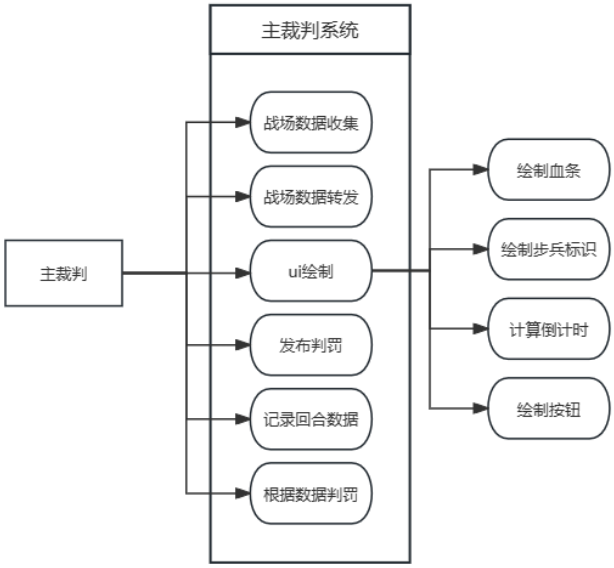


图 3-1 主裁判用例图

主程序为 judgement-main.py，其余四个类均在主程序中调用，如图 3-2 所示。

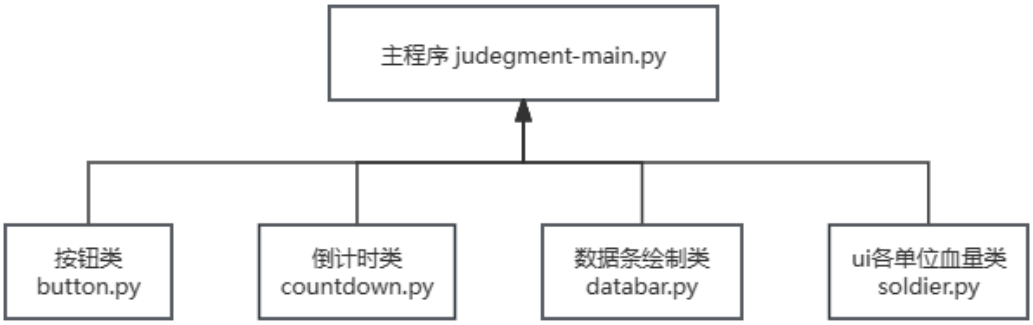


图 3-2 主程序与类关系图

主裁判端程序顺序图如下 3-3 所示

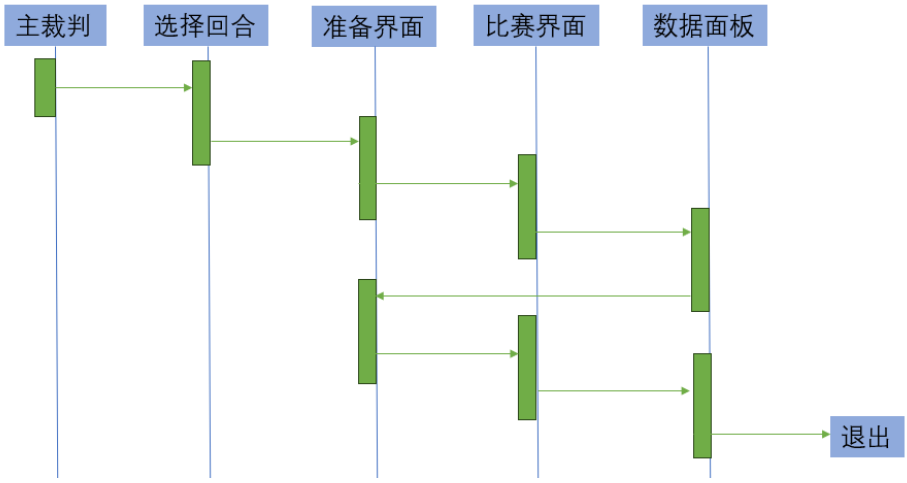


图 3-3 主裁判端程序顺序图

1) 按钮类用于绘制界面上所有的按钮，按钮具有以下属性：

```
btn2 = Button(l * 2, h * 2, window_width, window_height, "2回合", surBtnNormal, surBtnMove, surBtnDown, btn2CallBack,
              btnFont, (0, 113, 133))
```

依次为按钮在窗体上的 x 坐标、按钮在窗体上的 y 坐标、窗口宽度、窗口高度、按钮显示的字体、按钮常规样式、选中样式、按下样式、回调函数、显示字体、颜色。

其中窗口宽、高度在传入后会直接除以 5 作为按钮的横、纵大小。建议在新建对象时将所有坐标都设置为相对坐标，以适应不同比例大小的窗口。

按钮常规样式、选中样式、按下样式则使用 pygame 读入图片素材：

```
surBtnNormal = pygame.image.load("src/image/start_normal.png").convert_alpha()
surBtnMove = pygame.image.load("src/image/start_move.png").convert_alpha()
surBtnDown = pygame.image.load("src/image/start_down.png").convert_alpha()
```

可以进入对应路径文件夹替换为自己的图片素材。

2) 倒计时类用于显示比赛倒计时，该类具有以下属性：

```
class CountdownTimer:
    def __init__(self, total_seconds, size = 45,rgb = (0,130,130)):
```

依次为总时间、显示字号、显示颜色。类内部会自动将输入的秒数换算为“分:秒”的格式显示。可以通过在外部读取该对象的 total_seconds 属性得知倒计时剩余时间，用于触发不同任务。

3) 数据条绘制类用于绘制赛后数据面板的数据条，效果如下图 3-4 所示：

图 3-4 数据面板示意图

每个数据条类有以下属性：

```
bar_damage = DataBar("总伤害量",window_height*5/9,window_width / 2,window_height / 30,
                     blue_damage, red_damage,window_width,window_height)
```

数据条名称、在窗体上的 y 坐标（默认居中不需要 x 坐标）、数据条宽度、数据条高度、蓝方数据、红方数据、窗口宽度、窗口高度。

该类会根据传入的双方数据自动调整显示比例，使其不论数值大小都能在指定像素宽度下正确显示，展示出双方数据的正确比例。若传入的双方数据都为 0，则显示的数据条会变细，如“能量机关”数据条的显示。

4) Soldier.py 主要包括两个类：血条绘制与士兵类。血条绘制的原理与数据条类的原理相似。

血条类包含的属性如下

```
class HealthBar:
    def __init__(self, position, direction, total_health, current_health, bar_size, bar_color, invincible=False):
        self.position = position  # 血条位置 (x, y)
        self.direction = direction  # 血条方向，'left'表示从左向右消，'right'表示从右向左消
        self.total_health = total_health  # 总血量
        self.current_health = current_health  # 当前血量
        self.bar_size = bar_size  # 血条绘制尺寸 (width, height)
        self.bar_color = bar_color  # 血条颜色 (R, G, B)
        self.invincible = invincible  # 是否处于无敌状态
```

士兵类的效果包括根据编号显示步兵单位、显示步兵血条、右侧显示获得的 buff（获得 buff 时播放音效）。步兵死亡后会变灰并显示死亡倒计时，同时在大屏幕中央显示死亡信息，播放死亡音效，效果如图 3-5 所示。

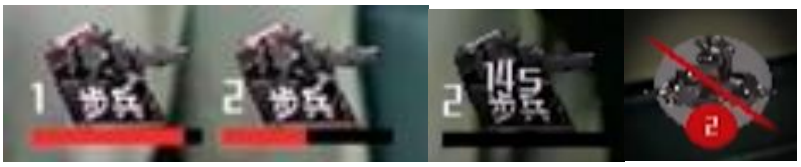


图 3-5 士兵 ui 显示效果图

包含的属性如下：

```
class Soldier:
    def __init__(self, team, position, size, terminal_number, soldier_number, state=0, hp=10):
        self.team = team  # 红方或蓝方
        self.position = position  # 显示位置 (x, y)
        self.size = size  # 显示大小 (width, height)
        self.terminal_number = terminal_number  # 终端编号 (1, 2, 3, 4, 5, 6)
        self.soldier_number = soldier_number  # 步兵编号 (1, 2, 3)
        self.state = state

        self.health = hp  # 当前血量
```

传入的参数如下：

```
Soldier('red', ((window_width / (3 * 6)) * (i + 1), window_height / 7.5),
        (window_width / 20, window_width / 30), red_terminal[i], i + 1))
```

士兵类的类图如下 3-6 所示

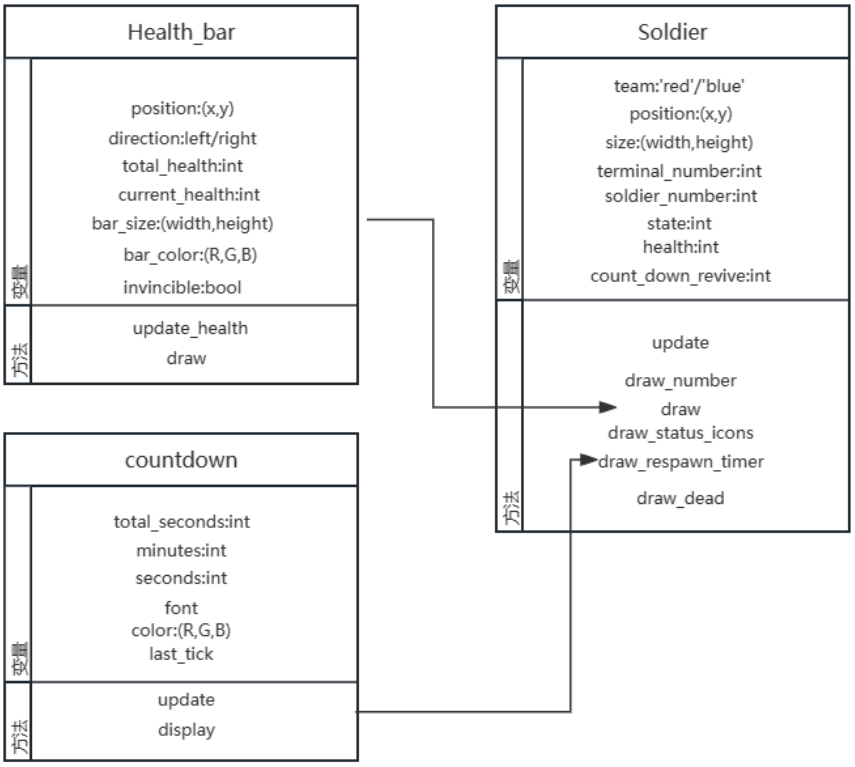


图 3-6 士兵类的类图

同样的，建议所有与坐标相关的参数都设置为相对坐标，方便后期随意调整窗口比例大小。

5) 函数：启动界面未封装函数！怕后续变量出现问题，有兴趣可以自己封装试试看。其他封装的函数名称与内容如下表所示。

choice()	选择回合数（两回合或三回合）
get_ready()	准备倒计时
count_down5()	五秒倒计时
race()	正赛内容，会自动调用 judge_winner()判断赢家
data_bar()	最终绘制赛后数据面板。

6) 手动判罚

当程序成功启动时就会显示：

```
Ready to send message!
```

若 ESP32 刚刚上电，自己会生成一些自检字符导致程序错误，重新启动电脑端主程序即可。

```
  File "D:\anancoder\envs\python38\lib\threading.py", line 932, in _bootstrap_inner
    self.run()
  File "D:\anancoder\envs\python38\lib\threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "C:\Users\DELL\Desktop\rm lite\esp32裁判系统\裁判系统-pc\judgement-main.py", line 111, in info_from_car
    character = int(numbers[0])
alueError: invalid literal for int() with base 10: 'ets Jul 29 2019 12:21:46'
```

当程序显示可以发送信息时，即可手动给每个终端判罚，

判罚数据结构为:《终端号，车辆颜色，判罚内容》

终端号不再赘述。终端颜色：0 为待机无颜色，1 为红色，2 为蓝色。

判罚内容：对于车载 0 为无判罚，1 为清空血量，2 为满血，3 为扣一半血。

对于基地 0 为无判罚，1 为解除无敌，2 为进入无敌（此时血条为绿色）。

3.1.2 主裁判端建议修改的参数

选择是否开启摄像头进行直播（建议开启）：

```
camera_use = True
```

提前定义队伍名称：

```
14  red_team = '红方'
15  blue_team = '蓝方'
```

终端号是记录于每个 esp32 裁判系统车载端内的编号，mac 地址太长，可以更直观的找到需要操作的对象车辆、终端

车载终端号，按顺序自动分配为 1 2 3 号车，运行前定义好。以红方举例，此时 2 号终端的消息对应 ui 上红方的一号步兵，3 号终端的消息对应 2 号步兵，4 号终端的消息对应 3 号步兵

```
red_terminal = [2,3,4]
blue_terminal = [1,5,6]
```

双方基地终端号，运行前定义好

```
red_base_termial = 8        #基地终端号
blue_base_termial = 7
```

能量机关的终端编号。

```
energy_mechanism = 9        #能量机关终端号
```

准备时间在调试中设置为 10 秒，建议设置为 90 秒。每回合时间建议设置为 300 秒。根据具体情况调整。

```
readytime = 10      #准备时间90（秒）
round_time = 10      #每回合时间 300秒
```

设置基地总血量，请不要把基地血量改为与步兵相同的 10！会导致程序判断错误

```
base_health = 30
```

提前调整好与 esp32 连接的串口号、波特率，需要在 esp32 上设置好，否则程序无法启动。

```
serial_port = 'COM4'  # 替换为你的串口名称
baud_rate = 115200  # 与 ESP32 配置的波特率相匹配
ser = serial.Serial(
    port='COM4',  # 串口设备名称，根据你的系统可能会有所不同
    baudrate=115200,  # 波特率，要与 ESP32 配置的波特率相匹配
    timeout=1  # 超时时间（秒）
)
```

复活血量设置：在 soldier 类下：self.health

```
117             else:
118                 self.health = 5  ##################################################################
119                 self.is_alive = True
120                 self.respawn_timer = self.count_down_revive
121                 self.alive_sound.play()
122
```

裁判终端：在 soldier.py 中的 Soldier 类复活时间与 buff 时间修改：

```
49          self.count_down_revive = 15             #复活时间：不建议小于buff时间 不然在吃了buff后 复活后buffui会继续显示
```

```
52          self.special_effect_duration = 15 * 1000  # 15 秒的特殊状态持续时间（以毫秒为单位）
```

其他变量说明：

score_red/blue	红蓝双方比分
energy_red/blue	双方能量机关激活次数，每回合归零
blue_kill/red_kill	蓝方击杀红方数/红方击杀蓝数
red_damage/blue_damage	红方所受伤害/蓝方所受伤害
red_base_health/blue_base_health	基地总血量

3.1.3 选手端程序

选手端主体架构与主裁判端完全一致，仅多了以下几个功能：

选择自己的选手编号（会影响显示 ui 内容和判罚），获得 buff 时显示特效，低血量时警告效果，准星 ui，发射模式选择，操作信号编码。

由于选手端在最后并未使用，可能还存在许多 bug。

选手端用例图如下 3-7：

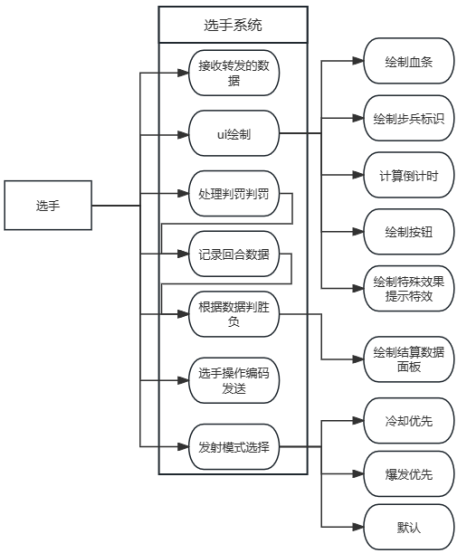


图 3-7 选手端用例示意图

选手端顺序图如下 3-8

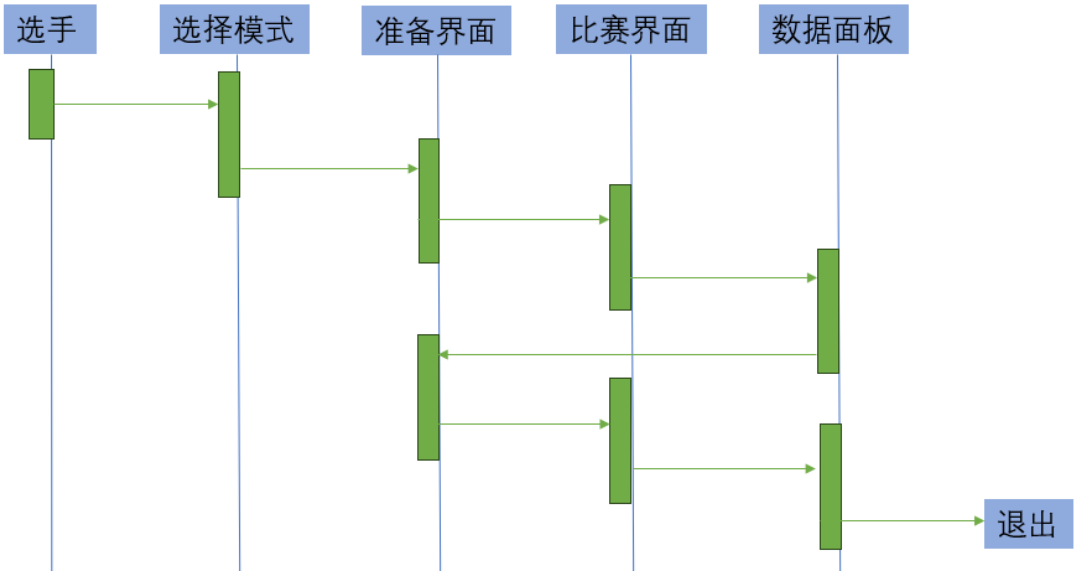


图 3-8 选手端顺序图

选择选手编号界面如 3-9 所示。

图 3-9 选手编号选择界面

选择发射模式如 3-10 所示：

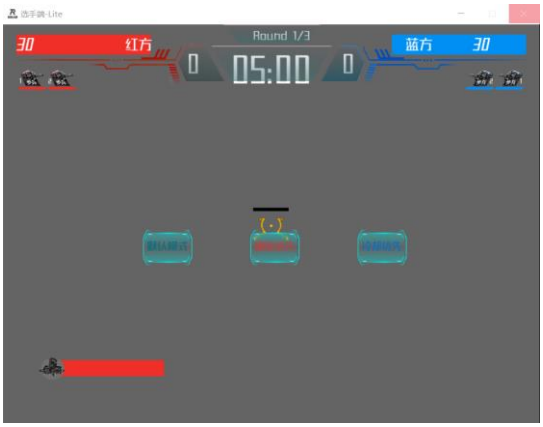


图 3-10 发射模式选择图

准星 ui 与死亡致盲效果如 3-11 所示（黑框为热量条）

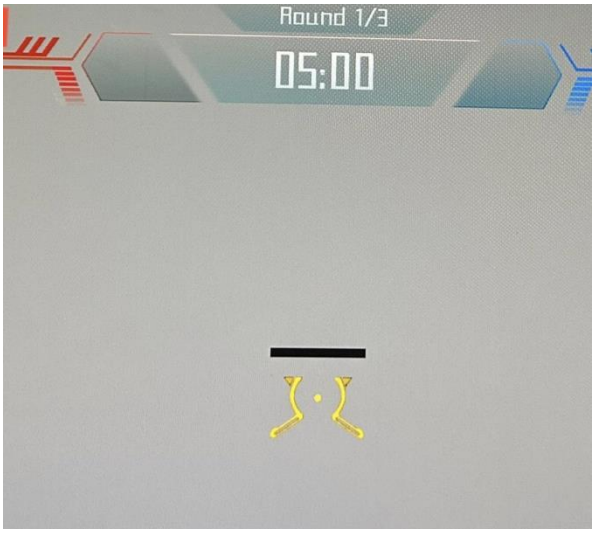


图 3-11 热量条示意图

获得 buff 与低血量效果如 3-12 所示。

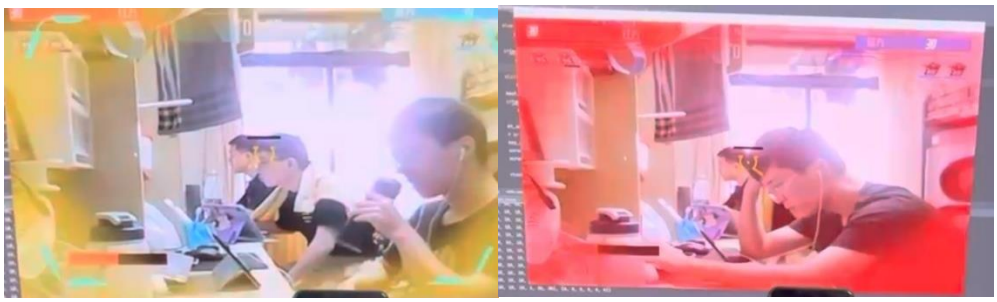


图 3-12 特效示意图

操作信号编码：

传统的蓝牙遥控同时只能发送一个信号，导致无法灵活操作。在选手端中，将选手的键鼠操作编码为一串数据一起发送，并在小车端解码，以实现多向移动同时打弹的效果。

选手端使用键鼠操作，支持的按键为 wasd、shift、鼠标左右键，务必切换为英语输入法输入。所有输入的信息将被编码，发送至 arduino 小车，信息编码格式为：

鼠标横向速度、鼠标纵向速度，wasd，shift 键，鼠标左右键。

鼠标速度范围从 0-100，中间值 50 表示鼠标没有移动，数值越小，表示向左移动越快；越大，向右移动越快。

当没有按键按下，数值为 5050N，其中 N 表示消息发送完毕的截止符号。以下是部分编码示例表。

按下按键	编码
w	5050wN
s	5050sN
wa	5050waN
左 shift	5050lN
组合：wa，shift	5050walN
鼠标左键	5050(N
鼠标右键	5050)N
组合:wa，shift，左键	5050wal(N

获取并解析鼠标速度，限制鼠标速度上限：

```
mx, my = pygame.mouse.get_pos()  # 获得鼠标坐标
mouse_speed_x= int(((mx - previous_mouse_position_x) / (window_width / 2)) * mouse_sensitivity) + 50
mouse_speed_y= int(((my - previous_mouse_position_y) / (window_height / 2)) * mouse_sensitivity) + 50


if mouse_speed_x<0:
    mouse_speed_x=0
elif mouse_speed_x>99:
    mouse_speed_x=99
if mouse_speed_y<0:
    mouse_speed_y=0
elif mouse_speed_y>99:
    mouse_speed_y=99
if int(mouse_speed_x / 10)==0:
    key_count+="0"
key_count+=str(mouse_speed_x)
if int(mouse_speed_y / 10)==0:
    key_count+="0"
key_count+=str(mouse_speed_y)
```

根据键盘按钮编码：

```
if key[K_w]:
    key_count+="w"
if key[K_s]:
    key_count+="s"
if key[K_a]:
    key_count+="a"
if key[K_d]:
    key_count+="d"
if key[pygame.K_LSHIFT]:
    key_count+="l"
```

热量计算与冷却：

```
if mouse[0]==True:
    key_count+='('
    if(gun_heat<heat_sum + 20):      #计算热量
        if(fire_mode == 0):           #普通
            gun_heat += 10
        if (fire_mode == 1):          #爆发优先
            gun_heat += 5
        if (fire_mode == 2):          #冷却优先
            gun_heat += 20

elif(gun_heat > 0):
    if (fire_mode == 0):          #冷却
        gun_heat -= 2
    if (fire_mode == 1):
        gun_heat -= 1
    if (fire_mode == 2):
        gun_heat -= 8
```

使用 `heat_bar.update_health(gun_heat)` 更新枪口热量

当超热量时，自动阻止鼠标左键的标识符发送，再使用 bt_ser.write 发送蓝牙编码。

```
if(gun_heat > heat_sum):
    key_count = key_count.replace('(','')
bt_ser.write(key_count.encode())
```

3.1.4 选手端建议修改的参数

是否开图传（图传在后文中会提到）：

```
camera_use = False
```

鼠标灵敏度：

```
mouse_sensitivity = 150
```

蓝牙串口：

```
bt_serial_port = 'COM3'
```

总热量限制：

```
45          heat_sum = 150
```

选手端复活时间在 computer_mouse.py 中修改：

```
13          revive_time = 15
```

Buff 时间在 soldier.py 中修改：

```
52              self.special_effect_duration = 15 * 1000  # 15 秒的特殊状态持续时间（以毫秒为单位）
```

其余需要修改的变量就与主裁判端完全一致了。

3.2 ESP32 端

3.2.1 ESP32 车载端

ESP32 车载端需要连接四块装甲板、一个血量条 WS2812B（10 颗灯珠）与一个颜色传感器 Tcs3472x 用于识别场地。车载终端程序逻辑图如下图 3-13。

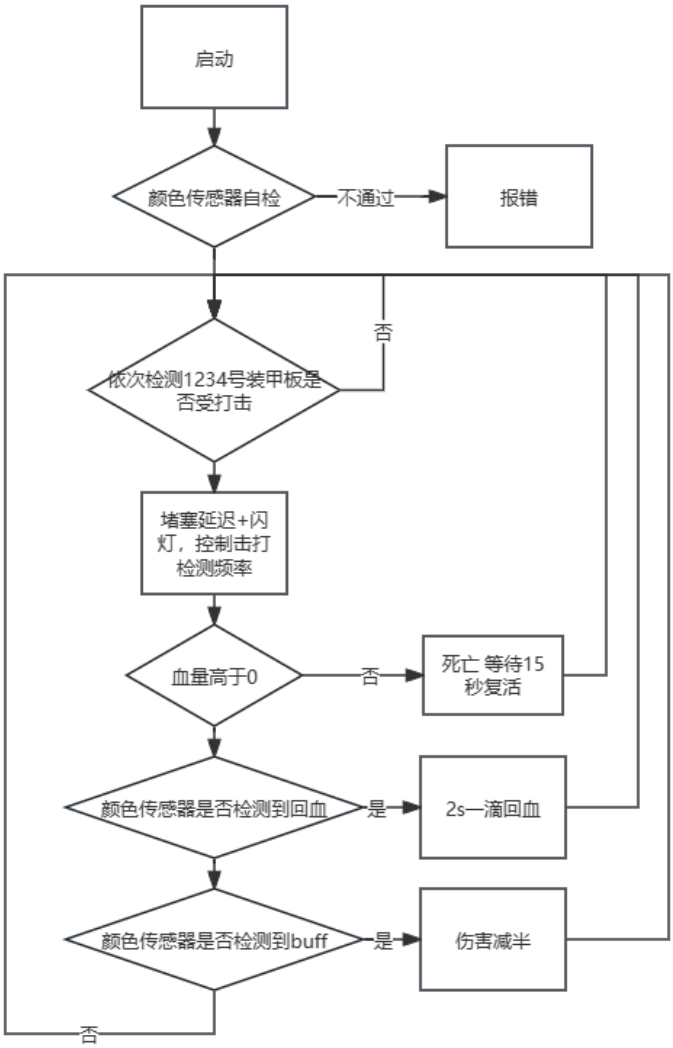


图 3-13 车载终端逻辑图

车载端的程序名称为 main/main.ino

需要提前自行更改所述颜色（1 为红，2 为蓝）以及终端编号（1，2，3，4，5，6）

```
45    int mycolor = 1;
46    int my_number = 2;
```

以及电脑端 esp32 的 mac 地址用作接受端，在此更改

```
11    uint8_t broadcastAddress[] = {0xA8, 0x42, 0xE3, 0x4C, 0x2F, 0x0C};  //  ESP32 接收器 MAC 地址
```

回血颜色在此更改：

```
  uint16_t clear, red, green, blue;
  tcs.getRGBC(&red, &green, &blue, &clear);
  tcs.lock();
  uint32_t sum = clear;
  float r, g, b;
  r = red; r /= sum;
  g = green; g /= sum;
  b = blue; b /= sum;
  r *= 256; g *= 256; b *= 256;
  if(((int)r > 150 && hp<10 && mycolor==1) || ((int)b > 120 && hp<10 && mycolor==2))     //检测到红色且不是满血，回血
  {
    t1 = millis();      //获取当前时间
    if(t1 - t2 >2000)
    {
      hp++;
      myData.hp = hp;
      myData.cure = true;
      esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
      t2 = millis();
    }
  }
  if((int)g>124)
  {
    myData.state = 1;
    esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
  }

  // Serial.println(hp);
}
```

目前根据己方颜色自行判断：红方，在红色区域回血。蓝方，在蓝色区域回血。绿色：在绿色区域获得防御加成 15 秒。

```
  if(sheild_buff == 1 && (millis()-t_sheild < buff_duration *1000))
  {
    if(sheild_buff ==0)
      sheild_buff = 1;
    else
      sheild_buff = 0;
  }
  else
  {
    sheild_buff = 0;
  }
  // Serial.println(hp);
}
```

其原理为：激活 buff 后每两下的攻击中会有一次攻击不被判定，如此循环间隔。

每当车辆有任意一个状态改变（激活 buff、掉血、回血）就会给电脑端 esp32 发送一次目前状态的数据。

3.2.2 电脑端 esp32

电脑端为 main-pc/main-pc.ino

场上有几个终端，就有注册几个从机

```
esp_now_peer_info_t peerInfo1;

esp_now_peer_info_t peerInfo2;

esp_now_peer_info_t peerInfo3;

esp_now_peer_info_t peerInfo4;

esp_now_peer_info_t peerInfo5;

esp_now_peer_info_t peerInfo6;
```

设置从机 mac 地址，建议按照 1，2，3，4，.....的顺序来，方便记忆。

```
37    uint8_t broadcastAddress1[] = {0xA8, 0x42, 0xE3, 0x4C, 0x29
38    uint8_t broadcastAddress2[] = {0xB0, 0xA7, 0x32, 0x32, 0x5C
39    uint8_t broadcastAddress3[] = {0xA1, 0x42, 0xE3, 0x4C, 0x29
40    uint8_t broadcastAddress4[] = {0xA2, 0x42, 0xE3, 0x4C, 0x29
41    uint8_t broadcastAddress5[] = {0xA3, 0x42, 0xE3, 0x4C, 0x29
42    uint8_t broadcastAddress6[] = {0xA4, 0x42, 0xE3, 0x4C, 0x29
```

每注册一个从机，就要写一段

```
memcpy(peerInfo1.peer_addr, broadcastAddress1, 6);   // Register peer注册接受端机器
peerInfo1.channel = 0;         //两个默认参数 不用管
peerInfo1.encrypt = false;
esp_now_add_peer(&peerInfo1);
memcpy(peerInfo2.peer_addr, broadcastAddress2, 6);   // Register peer注册接受端机器
peerInfo2.channel = 0;         //两个默认参数 不用管
peerInfo2.encrypt = false;
esp_now_add_peer(&peerInfo2);
memcpy(peerInfo3.peer_addr, broadcastAddress3, 6);   // Register peer注册接受端机器
peerInfo3.channel = 0;         //两个默认参数 不用管
peerInfo3.encrypt = false;
esp_now_add_peer(&peerInfo3);
memcpy(peerInfo4.peer_addr, broadcastAddress4, 6);   // Register peer注册接受端机器
peerInfo4.channel = 0;         //两个默认参数 不用管
peerInfo4.encrypt = false;
esp_now_add_peer(&peerInfo4);
memcpy(peerInfo5.peer_addr, broadcastAddress5, 6);   // Register peer注册接受端机器
peerInfo5.channel = 0;         //两个默认参数 不用管
peerInfo5.encrypt = false;
esp_now_add_peer(&peerInfo5);
memcpy(peerInfo6.peer_addr, broadcastAddress6, 6);   // Register peer注册接受端机器
peerInfo6.channel = 0;         //两个默认参数 不用管
peerInfo6.encrypt = false;
esp_now_add_peer(&peerInfo6);
```

使用 strtok 函数分割来自电脑的信息，根据电脑的信息判断剩下的内容给谁发。

```
if(character == "1")          //判断发送的目标是几号终端
  esp_now_send(broadcastAddress1, (uint8_t *) &judgement, sizeof(judgement));
else if(character == "2")
  esp_now_send(broadcastAddress2, (uint8_t *) &judgement, sizeof(judgement));
else if(character == "3")
  esp_now_send(broadcastAddress3, (uint8_t *) &judgement, sizeof(judgement));
else if(character == "4")
  esp_now_send(broadcastAddress4, (uint8_t *) &judgement, sizeof(judgement));
else if(character == "5")
  esp_now_send(broadcastAddress5, (uint8_t *) &judgement, sizeof(judgement));
else if(character == "6")
  esp_now_send(broadcastAddress6, (uint8_t *) &judgement, sizeof(judgement));
```

1. 色彩采集。在布置场地时，需要使用 Tcs3472x_Clour_RGBLED，调用颜色传感器对场上的特殊区域颜色进行采样：包括：己方回血区，buff 加成区。绿色为防御加成区。场地示意如图 3-14 所示。

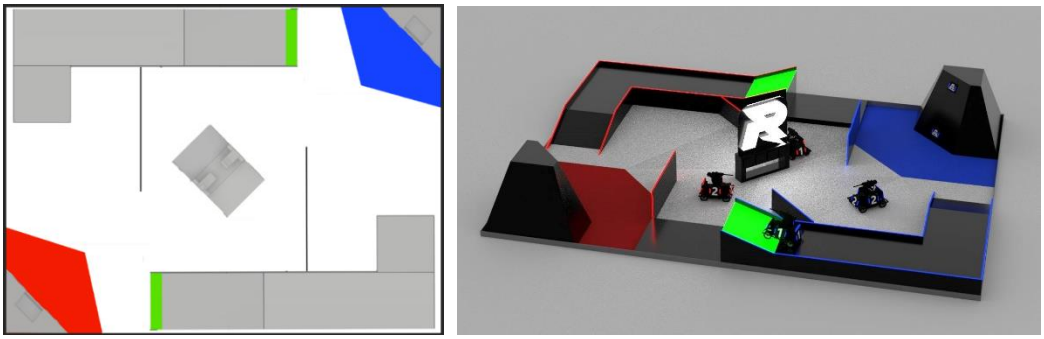


图 3-14 场地示意图

3.2.3 ESP32 端建议修改的参数

1) 死亡复活时间

车载终端 main.ino：目前为 15s 复活，3s 无敌

```
235     if(hp == 0)     //没血，则装甲板熄灯 进入堵
236     {
237       fill_solid(armor_1, 6, CRGB::Black);
238       fill_solid(armor_2, 6, CRGB::Black);
239       fill_solid(armor_3, 6, CRGB::Black);
240       fill_solid(armor_4, 6, CRGB::Black);
241       HP_display(0);
242       FastLED.show();
243       delay(15000);    //15s复活 3s无敌
244       myData.hp = 5;
245       hp = 5;
246       fill_solid(armor_1, 6, NOW_COLOR);
247       fill_solid(armor_2, 6, NOW_COLOR);
248       fill_solid(armor_3, 6, NOW_COLOR);
249       fill_solid(armor_4, 6, NOW_COLOR);
250       HP_display(hp);
251       FastLED.show();
252       delay(3000);
253     }
254
```

2) buff 时间

车载终端：main.ino

```
50    int sheild_buff = 2;
51    int buff_duration = 15;    //buff持续15秒
52    long int t1 = millis();
```

3) 车辆血量

Main.ino 车载终端：

```
46    int hp = 10;            //初始血量10
```

初始血量

```
244       myData.hp = 5;
245       hp = 5;
```

复活后血量

4) 基地血量

基地血量：基地端修改 main-base.ino

```
42    int hp = 30;            //基地初始血量30
```

5) MAC 地址修改

选手端修改：需要统一设置为裁判 pc 端 esp32 接收器的 MAC 地址，在 main.ino 中修改

```
9   uint8_t broadcastAddress[] = {0xA8, 0x42, 0xE3, 0x4C, 0x2F, 0x0C};  //  ESP32 接收器 MAC 地址
10
```

场上基地端修改，同理，在 main-base.ino 中修改：

```
9   uint8_t broadcastAddress[] = {0xA8, 0x42, 0xE3, 0x4C, 0x2F, 0x0C};  //  ESP32 接收器 MAC 地址
```

《注意：裁判端接收器修改：需要修改的终端比较多，谨慎操作！，必须严格按照顺序！！！》

车载终端修改：顺序分别为 1-蓝 1，2-蓝 2，3-红 1，4-红 2

```
53   uint8_t broadcastAddress1[] = {0xA8, 0x42, 0xE3, 0x4C, 0x29, 0xE0};  //  ESP32 接收器 MAC 地址
54   uint8_t broadcastAddress2[] = {0xB0, 0xA7, 0xD3, 0x32, 0x5C, 0x62};
55   uint8_t broadcastAddress3[] = {0xA1, 0x43, 0xE9, 0x4D, 0x29, 0xE1};
56   uint8_t broadcastAddress4[] = {0xA2, 0x42, 0xE1, 0x4F, 0x20, 0xE3};
```

基地终端修改：5-红色基地　6-蓝色基地

```
57   uint8_t broadcastAddress5[] = {0xA3, 0x42, 0xE3, 0x4C, 0x29, 0xE3};   //基地两个(现在假的)
58   uint8_t broadcastAddress6[] = {0xA4, 0x42, 0xE3, 0x4C, 0x29, 0xE4};
```

选手端 mac 修改：7-蓝 1，8-蓝 2，9-红 1，10-红 2：

```
59   uint8_t broadcastAddress7[] = {0xB0, 0xA7, 0x32, 0x32, 0x5C, 0x60};    /
60   uint8_t broadcastAddress8[] = {0xC4, 0x42, 0xE3, 0x4C, 0x29, 0xE4};
61   uint8_t broadcastAddress9[] = {0xF3, 0x42, 0xE3, 0x4C, 0x29, 0xE3};
62   uint8_t broadcastAddress10[] = {0xD4, 0x42, 0xE3, 0x4C, 0x29, 0xE4};
63
```

3.3 硬件部分

3.3.1 装甲板

装甲板使用震动传感器判断击打，经实测较为可用，但是会出现误判情况。

零件	备注
震动传感器	压电片+敲击感应，单价 2.64，每块装甲板 1 个
灯带	2812 灯带，左右两侧各三个，共 6 个
4p 线	1.25 间距，4p 公母线方便插拔
装甲板壳	本体黑色打印，灯条处白色打印.

中央部分为震动传感器，两侧为灯条，均使用胶枪固定，如图 3-15 所示

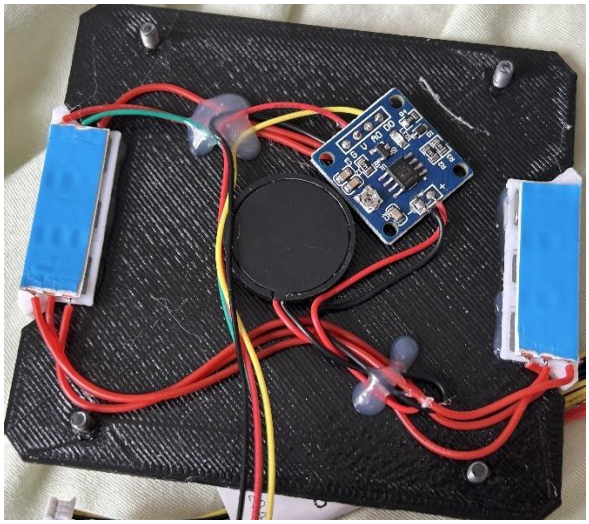


图 3-15 装甲板固定示意图

3.3.2 颜色传感器（场地交互模块）

颜色传感器型号为 Tcs3472x，单价为二十左右，离地 1cm 左右时检测效果较好。颜色校准：提前使用 Tcs3472x_Clour_RGBLED 程序读取实际 RGB 数值，扩大一点范围写死在车载端的程序中。

3.3.3 血量条

使用 108 灯/m 的 2812 灯条，每 10mm 一颗灯珠，10cm 对应步兵的十滴血量。基地只是将灯珠改为 30 颗，灯条发光效果如图 3-16 所示。

图 3-16 发光效果示意图

3.3.4 ESP32 终端

ESP32 终端上需要焊接一根 2p 用于供电，4 根 4p 用于连接装甲板，1 根 4p 用于连接颜色传感器，1 根 3p 用于连接血条。效果如下图所示。具体焊接串口定义根据程序修改。终端使用单节 18650 直接供电，经测试拥有较好的续航能力，终端所用线材如图 3-17.

图 3-17 所用线材清单图

车载终端成品如下图 3-18

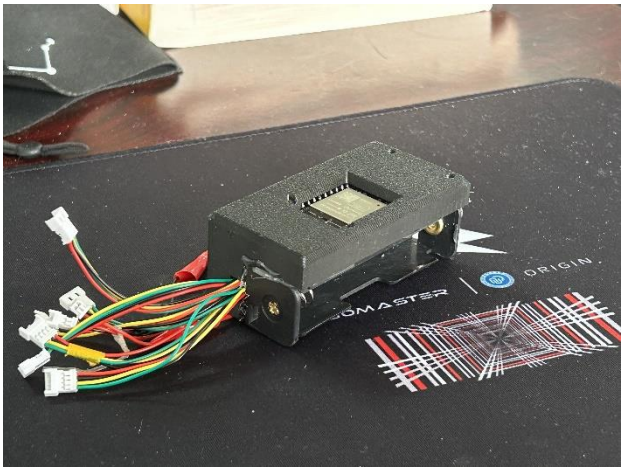
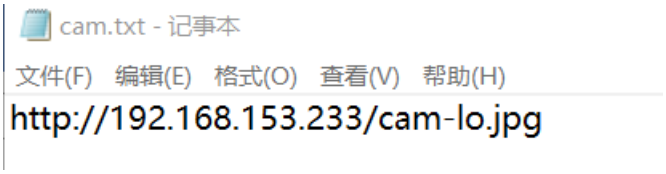


图 3-18 车载终端成品图

3.3.5 图传

图传选择 esp32-cam 方案，单价 20 左右，延迟为 0.2s，且其视频流可以通过 python 获取并传输到选手端。代码部分参考 cam.ino。每次运行后会生成一段固定的 ip 地址，将这段 ip 地址写入选手端路径下的 cam.txt 中，格式为：



需要将选手端电脑和图传接入同一个 wifi 下，具体实现原理自行搜索，这里不再赘述。

4 界面使用

4.1 主裁判端

启动程序后的主页显示如图 4-1

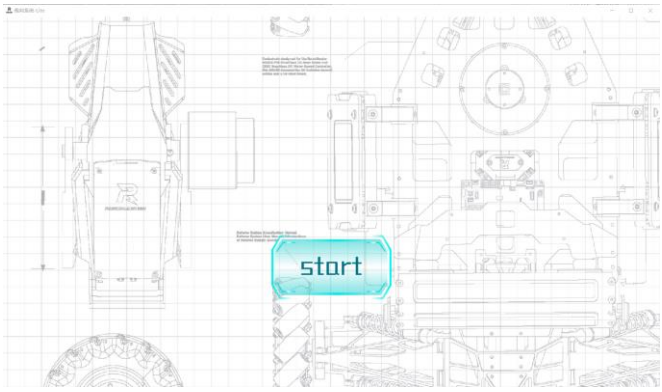


图 4-1 主界面显示图

选择比赛回合如图 4-2.

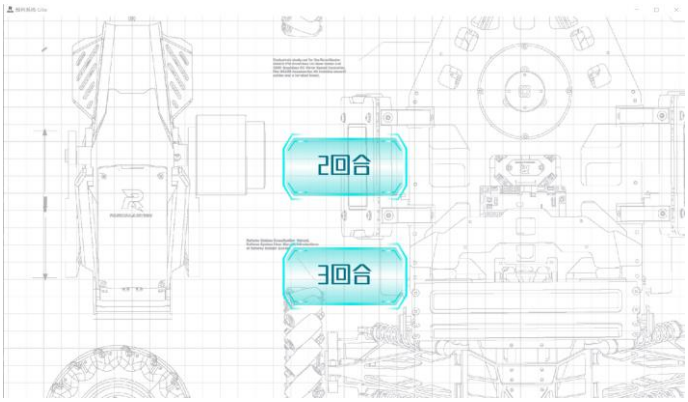


图 4-2 回合选择界面

准备倒计时界面：点击下方倒计时区域会发起技术暂停如图 4-3

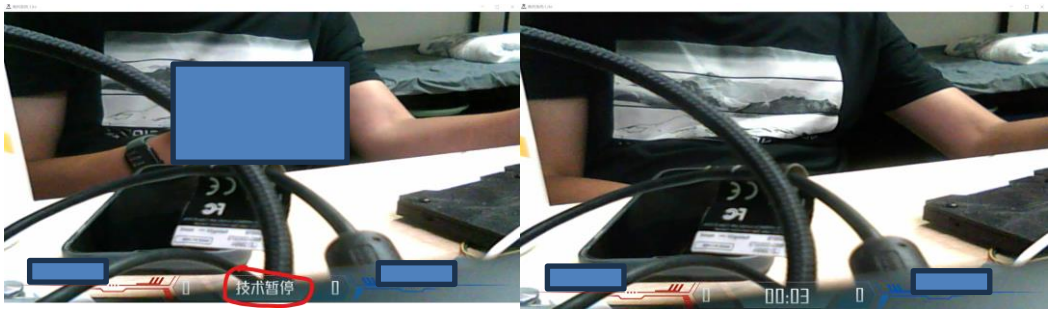


图 4-3 技术暂停示意图

经过五秒倒计时后，会进入正式比赛。同样的，点击倒计时区域进行技术暂停如图 4-4

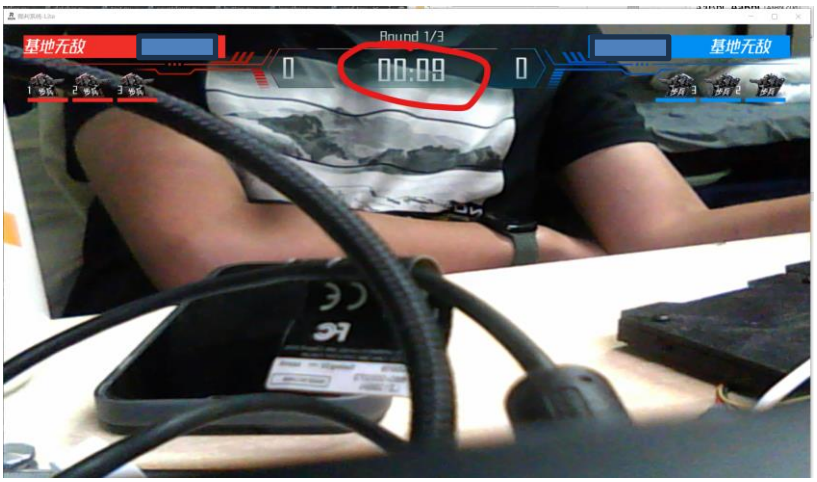


图 4-4 技术暂停点击区域图

当车辆激活 buff 时会有图标显示如图 4-5

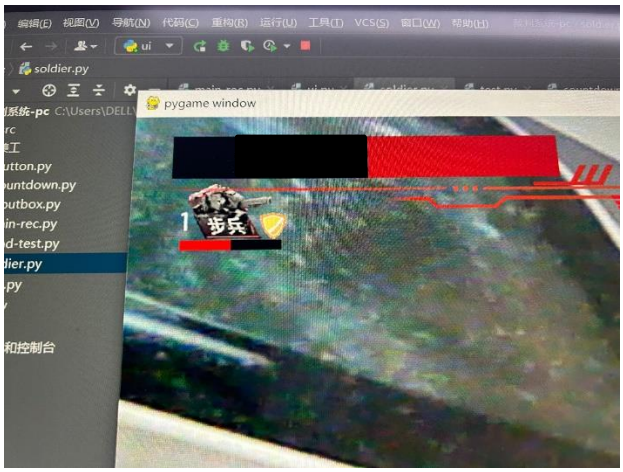


图 4-5 获得 buff 标志示意图

比赛结束后会进入胜负判决页面如图 4-6。

图 4-6 判决示意图

等待十秒后会显示比赛数据面板，点击 vs 开始下一场比赛（进入倒计时准备时间）。若回合比完了，点击 vs 直接退出程序。

4.2 选手端

测试界面如图 4-7

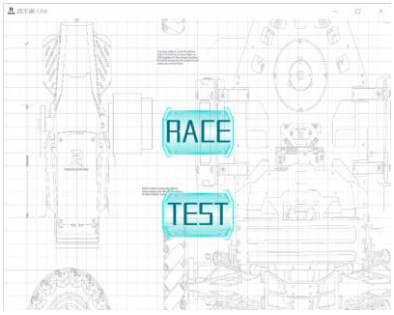


图 4-7 模式选择示意图

在选择前，请确保电脑已经连接了战车的蓝牙且填写了正确的端口号。

在选择前，将窗口调整到适合的大小。

在测试时，选择 test 进入测试界面，如图 4-8 所示。

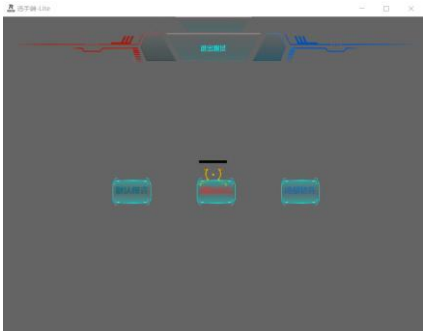


图 4-8 测试界面

虚拟枪口热量限制：连续开火后，枪口会过热，需要停止开火冷却。

进入界面后，有三种发射模式需要选择，对应了三种虚拟枪口热量限制：

默认模式：普通的热量上限和冷却速度

爆发优先：较高的热量上限和较慢的冷却速度

冷却优先：较低的热量上限和较快的冷却速度

2) 比赛界面

比赛现场连接了 ESP32 选手端裁判系统，选择 race 后，选择自己对应的身份，如图 4-9 所示。

图 4-9 身份选择界面

进入系统后，左下角显示自己操作兵种的实时血量，上方显示双方血量和基地血量，比分，倒计时等信息如图 4-10 所示。

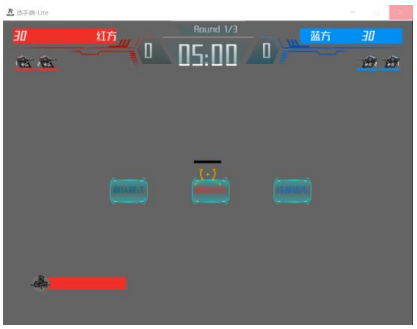


图 4-10 比赛模式下操作手界面示意图

在倒计时归零或一方胜出时，系统自动判断胜负。

注意：倒计时仅供参考，在选择完发射模式后，按下任意键盘按键，倒计时将启动。

5 总结

5.1 系统总结

该系统的四大组成部分经过测试具有较好的稳定性，在为期一天共九十多局的比赛中并仅出现了三次基地无法解除无敌的 bug（通过主裁判端手动操作解除无敌），一个车载终端由于电源反接烧毁，其余并未出现问题。

同时，车载端和基底端还具有低成本的优势，每个车载终端的成本仅为 20 元的 ESP32，每块装甲板成本预计不超过 5 元，每个血量条成本不超过 3 元，便于大量制作以支撑起完整的校内赛。、

5.2 改进

系统依然有众多 bug 需要完善，主要可改进的方向如下：

1. 增加终端 mac 号动态分配功能，通过可视化界面直接指定终端对应哪个编号的车。
2. 增加可视化判罚界面，当前在命令台中输入判罚编码效率较低，建议参考官方裁判系统服务器制作一个判罚界面。
3. 增加消息队列功能：当两个终端同时向主裁判端发送信号时，主裁判端会忽略其中一个信号导致系统出现 bug（怀疑基地不解除无敌就是这个原因）。
4. 装甲板模块的震动传感器会出现误检测的情况，可以选择其他传感器测试效果。
5. 增加中央资源岛、能量机关，丰富比赛内容。