# 1概述

## 1.1系统功能描述

这是一个用C语言编写的拔河仿真程序，主要功能包括:基于物理仿真模拟拔河过程、

程序通过实时分析对手行为模式（分为摆烂型和进攻型）并动态调整自身策略、实现对抗性决策。

## 比赛状态和判定胜负

比赛区域；分为四个区域（零区、一区、二区、三区），根据滑块的位置判断比赛的进行和结束状态。

胜负规则：比赛未结束时：滑块进入零区或三区则判定胜负。比赛结束：时根据滑块在一区或二区的位置判定胜负。

1.1.2 物理上模拟拔河比赛中的物理模型

1、包括整个区间大小，边界，滑块的当前位置、速度和加速度的变化。

2、外力因素，如随机风力的影响。

3、计算摩擦力对滑块运动的影响。

## 1.2环境要求

开发环境：Microsoft Visual Studio 2022和Dev-C++及其VSCode

运行环境：Windows 10及以上操作系统

依赖库：标准C语言库（stdio.h/stdlib.h、math.h、time.h等）

# 2程序设计与实现

## 2.1 原理说明

运动方程：

real\_acc = wind + GetFriction(acc, vel) + acc

位置预测模型(不考虑摩擦力风力及其敌方拉力)：

predict\_pos = Pos + ((vel + (vel-count-1))\*count)/2

摩擦力模型：

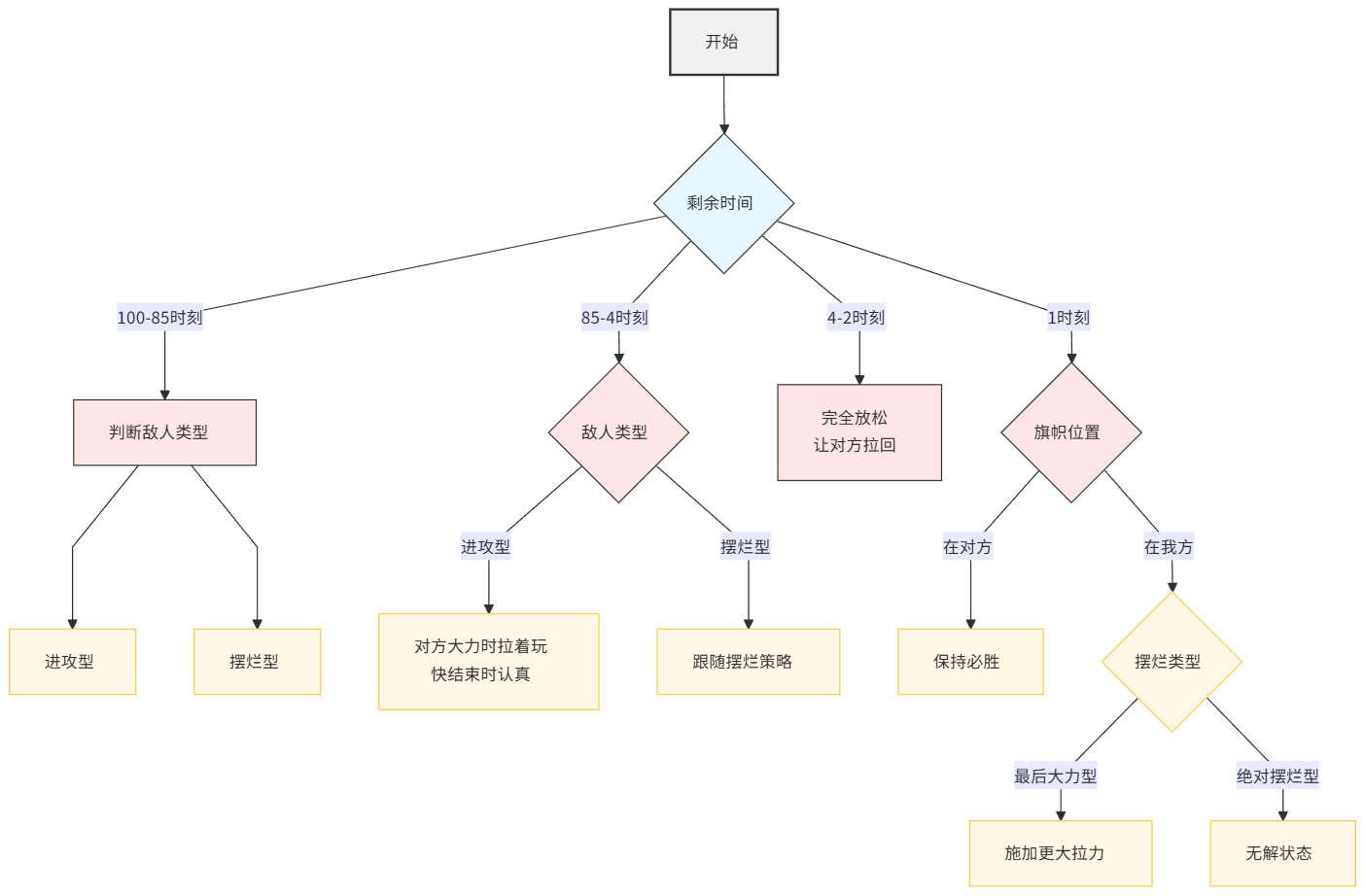
速度>0时：friction = -1.0

速度<0时：friction = 1.0

速度=0时：friction = -acc

## 2.2 总体设计

### 2.2.1流程图



### 2.2.2程序结构图：

Player.c

├── GetFriction(acc, vel)

├── GetEnemyType(struct war\_info \*info)

├── is\_Safe(Pos, acc, vel, wind, count, boundary)

└── GetEnemyLastAcc(My\_acc, vel)

### 2.2.3模块：

|  |  |
| --- | --- |
| 函数**GetFriction** | |
| 函数名 | GetFriction |
| 函数原形 | static double GetFriction(double acc, double vel) |
| 功能描述 | 获取当前摩擦力 |
| 输入参数 | acc: 当前加速度 vel: 当前速度 |
| 输出参数 | 无 |
| 返回值 | float类型摩擦力 |
| 被调用函数 | 无 |

|  |  |
| --- | --- |
| 函数**GetEnemyType** | |
| 函数名 | GetEnemyType |
| 函数原形 | static void GetEnemyType(struct war\_info \*info) |
| 功能描述 | 获取敌人类型 |
| 输入参数 | 结构体war\_info的指针 |
| 输出参数 | 全局变量EnemyType |
| 返回值 | 无 |
| 被调用函数 | GetFriction |

|  |  |
| --- | --- |
| 函数**is\_Safe** | |
| 函数名 | is\_Safe |
| 函数原形 | static int is\_Safe(double Pos, double acc, double vel, double wind, int count, int boundary) |
| 功能描述 | 根据预测步长判断自身安全情况 |
| 输入参数 | Pos 当前位置, acc 当前加速度, vel 当前速度, wind当前风力, count预测步长, boundary 边界 |
| 输出参数 | 无 |
| 返回值 | 0：不安全 1：安全 |
| 被调用函数 | GetFriction |

|  |  |
| --- | --- |
| 函数**GetEnemyLastAcc** | |
| 函数名 | GetEnemyLastAcc |
| 函数原形 | static double GetEnemyLastAcc(double My\_acc, double vel, double wind) |
| 功能描述 | 估计上次敌人拉力 |
| 输入参数 | 自身拉力 My\_acc, 旗子速度vel, 风速 wind |
| 输出参数 | 无 |
| 返回值 | Enemy\_last\_acc敌人上次加速度 |
| 被调用函数 | 无 |

### 2.2.4数据结构&变量定义&宏函数&宏定义

|  |  |  |  |
| --- | --- | --- | --- |
| 结构体**war\_info** |  |  |  |
| pos | 滑块当前位置 | | |
| vel | 滑块当前速度 | | |
| wind | 实时风力 | | |
| ts\_left | 剩余时间 | | |
| zones[4] | 比赛区域大小 | | |
| player\_acc | 玩家输出 | | |
| \*player\_name | 玩家团队名称和成员姓名的数组指针 | | |

|  |  |  |
| --- | --- | --- |
| 全局变量 **EnemyType** |  |  |
| 变量类型 | static int | |
| 描述 | 敌人类型: 0:摆烂类,1:进攻型 | |

### 

|  |  |
| --- | --- |
| 宏定义**GetTypeEnd** | |
| 定义 | 45 |
| 描述 | 判断敌人类型结束的游戏刻 |

|  |  |  |
| --- | --- | --- |
| 宏函数**I\_SAFE** |  |  |
| 函数原型 | I\_SAFE(MyAcc, Boundary) | |
| 定义 | (is\_Safe(info->pos, MyAcc, info->vel, info->wind, (int)(info->ts\_left), Boundary)) | |
| 描述 | 判断我方是否处于安全 | |

|  |  |  |
| --- | --- | --- |
| 宏函数**YOU\_DANGER** | |  |
| 函数原型 | YOU\_DANGER(YourAcc, Boundary) | |
| 定义 | (!is\_Safe(100 - info->pos - 1, -YourAcc, -info->vel, -info->wind, (int)(info->ts\_left), Boundary)) | |
| 描述 | 判断敌方是否处于危险 | |

## 2.3 详细设计

## 

// Fin\_player.c v114.514

// 风相对与自己，风是往自己吹的为正，所以抵消风力要加速度=-风力+...

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

// 安全检测宏定义（基于右侧玩家视角）

#define I\_SAFE(MyAcc, Boundary) (is\_Safe(info->pos, MyAcc, info->vel, info->wind, (int)(info->ts\_left), Boundary))

#define YOU\_DANGER(YourAcc, Boundary) (!is\_Safe(100 - info->pos - 1, -YourAcc, -info->vel, -info->wind, (int)(info->ts\_left), Boundary))

#define GetTypeEnd 45 // 对手类型识别结束时刻

static int EnemyType = -1; // 对手类型标识: -1=未识别, 0=摆烂型, 1=进攻型

/\*\*

\* @struct war\_info

\* @brief 游戏状态信息结构体（基于右侧玩家视角）

\*

\* 游戏引擎已经将我们的视角固定为右侧玩家，所有位置计算都基于此假设

\*/

struct war\_info {

// 比赛状态信息

double pos; ///< 滑块当前位置（0-100，右侧玩家视角）

double vel; ///< 滑块当前速度

double wind; ///< 实时风力（吹向右侧为正）

double ts\_left; ///< 剩余时间（整数时刻）

double zones[4]; ///< 比赛区域边界：[0]-零区, [1]-零区+一区, [2]-零区+一区+二区, [3]-比赛区总长

// 玩家输出信息

double player\_acc; ///< 玩家输出的加速度

char\* player\_name; ///< 玩家团队名称

};

/\*\*

\* @brief 计算摩擦力（基于物理规则）

\* @param acc 当前加速度

\* @param vel 当前速度

\* @return 摩擦力值

\*

\* 摩擦力规则:

\* - 速度为正时: -1.0

\* - 速度为负时: +1.0

\* - 速度为零时: 根据加速度大小决定

\*/

static double GetFriction(double acc, double vel) {

double friction = 0;

if (vel > 0) {

friction = -1.0; // 向右运动时向左的摩擦力

} else if (vel < 0) {

friction = 1.0; // 向左运动时向右的摩擦力

} else {

// 静止时摩擦力的特殊处理

if (acc > 1.0) {

friction = -1.0;

} else if (acc < -1.0) {

friction = 1.0;

} else {

friction = -acc; // 平衡外力

}

}

return friction;

}

/\*\*

\* @brief 位置安全检测（基于右侧玩家视角）

\* @param Pos 当前位置

\* @param acc 当前加速度

\* @param vel 当前速度

\* @param wind 当前风力

\* @param count 预测步长（小于等于0表示预测到速度为零）

\* @param boundary 安全边界值

\* @return 1-安全, 0-危险

\* 安全检测逻辑:

\* 1. 计算真实加速度（风力+摩擦力+玩家加速度）

\* 2. 更新速度

\* 3. 预测未来位置（等差数列求和）

\* 4. 比较预测位置与安全边界

\*/

static int is\_Safe(double Pos, double acc, double vel, double wind, int count, int boundary) {

double predict\_pos, real\_acc;

// 计算真实加速度 = 风力 + 摩擦力 + 玩家加速度

real\_acc = wind + GetFriction(acc, vel) + acc;

vel += real\_acc; // 更新速度

// 调整预测步长（确保不会过度预测）

if (count > 0 && vel - count <= 0) {

count = (int)vel; // 预测到速度为零

}

// 预测位置 = 当前位置 + 速度变化的等差数列和

// 公式: S = n\*(a1+an)/2

predict\_pos = Pos + ((vel + (vel - count - 1)) \* count) / 2;

// 检查是否超出安全边界（右侧玩家边界在右侧）

return (predict\_pos > boundary) ? 0 : 1;

}

/\*\*

\* @brief 识别对手类型（进攻型/摆烂型）

\* @param info 游戏状态指针

\*

\* 识别逻辑:

\* 1. 计算仅受风力影响的理论位置(OnlyWindPos)

\* 2. 在GetTypeEnd时刻比较实际位置与理论位置

\* - 实际位置 >= 理论位置 → 摆烂型(0)

\* - 实际位置 < 理论位置 → 进攻型(1)

\*/

static void GetEnemyType(struct war\_info\* info) {

static double accqq = 0, velqq = 0, OnlyWindPos;

// 初始化阶段（t=100）

if (info->ts\_left == 100) {

OnlyWindPos = 50; // 初始位置设为中点

}

// 更新纯风力位置（忽略我方拉力影响）

if (info->ts\_left != 100) {

OnlyWindPos += velqq;

}

// 计算纯风力作用下的运动

accqq = info->wind; // 仅风力作用

accqq += GetFriction(accqq, velqq); // 加入摩擦力

velqq += accqq; // 更新速度

// 在识别结束时刻确定对手类型

if (info->ts\_left == GetTypeEnd) {

// 比较实际位置与纯风力理论位置

if (info->pos - OnlyWindPos >= 0) {

EnemyType = 0; // 摆烂型（实际位置≥理论位置）

} else {

EnemyType = 1; // 进攻型（实际位置<理论位置）

}

}

}

/\*\*

\* @brief 获取对手上一次的加速度

\* @param My\_acc 我方当前加速度

\* @param vel 当前速度

\* @return 对手上一次加速度

\* 计算原理:

\* 基于速度变化反推对手动作:

\* Enemy\_acc = My\_last\_acc + last\_vel - vel + last\_wind

\* Enemy\_last\_acc+=GetFriction(Enemy\_last\_acc,last\_vel)

\*/

static double GetEnemyLastAcc(double My\_acc, double vel, double wind)

{

    static double Enemy\_last\_acc, last\_vel, My\_last\_acc = 1000, last\_wind;

    if (My\_last\_acc != 1000) // 跳过首次调用

        Enemy\_last\_acc = My\_last\_acc + last\_vel - vel + last\_wind;

        Enemy\_last\_acc+=GetFriction(Enemy\_last\_acc,last\_vel);

    My\_last\_acc = My\_acc;

    last\_vel = vel;

    last\_wind = wind;

    return Enemy\_last\_acc;

}

/\*\*

\* @brief 核心策略决策函数

\* @param info 游戏状态指针

\* 策略阶段:

\* 1. 100-45时刻: 对手类型识别

\* 2. 45-4时刻: 根据对手类型制定策略

\* 3. 4-2时刻: 诱导对手拉回旗帜

\* 4. 2时刻: 最终决胜策略

\*/

void player19(struct war\_info\* info) {

info->player\_name = "耐磨王"; // 设置玩家名称

static double Enemy\_acc, acc; // 策略使用的临时变量

// 获取对手加速度（用于策略决策）

Enemy\_acc = GetEnemyLastAcc(acc, info->vel, info->wind);

// ====== 阶段1: 对手类型识别 (100-45时刻) ======

if (info->ts\_left >= GetTypeEnd && info->ts\_left <= 100) {

GetEnemyType(info); // 识别对手类型

return;

}

// ====== 阶段2: 类型化策略 (45-4时刻) ======

// 情况1: 进攻型对手处于危险位置 → 松手策略

if (YOU\_DANGER(Enemy\_acc, info->zones[2]) && info->ts\_left > 2 && EnemyType == 1) {

info->player\_acc = 0; // 完全松手

return;

}

// 情况2: 进攻型对手 & 旗帜在右侧 → 逐步增加拉力

else if (info->vel < 0 && info->pos <= 50 && EnemyType == 1 && info->ts\_left >= 4) {

// 寻找最大安全加速度（边界预留15单位防风吹飞）

for (acc = 0; I\_SAFE(acc, info->zones[2] - 15); acc += 0.01);

info->player\_acc = acc;

return;

}

// ====== 阶段3: 最终决胜策略 (t=2时刻) ======

if (info->ts\_left == 2) {

// 针对进攻型对手的策略

if (EnemyType == 1) {

/\* 决胜公式:

\* 所需加速度 = (安全边界 - 当前位置) - 风力 - 速度 - 2

\* 其中-2用于抵消摩擦力的基础值

\*/

info->player\_acc = (info->zones[2] - info->pos) - info->wind - 2 - info->vel;

}

// 针对摆烂型对手的策略

else if (EnemyType == 0) {

/\* 决胜公式:

\* 所需加速度 = 50(中点位置) - 风力 - 速度 - 2

\* 假设对手完全不发力时的保守策略

\*/

info->player\_acc = 40 - info->wind - 2 - info->vel;

}

return;

}

// 如果没有匹配任何策略，保持当前加速度（无额外处理）

}

## 2.4 使用说明

编译：将 player19.c 编译并链接至拔河比赛主程序框架中。

gcc Fin\_player.c -o player -lm

接口对接：在主程序添加标准函数 void playerX(struct war\_info\* info)

集成：作为拔河游戏AI模块使用

输入：通过war\_info结构体获取游戏状态

输出：设置player\_acc和player\_name

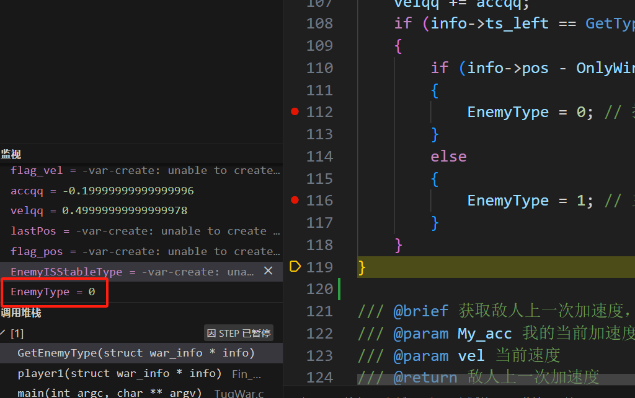
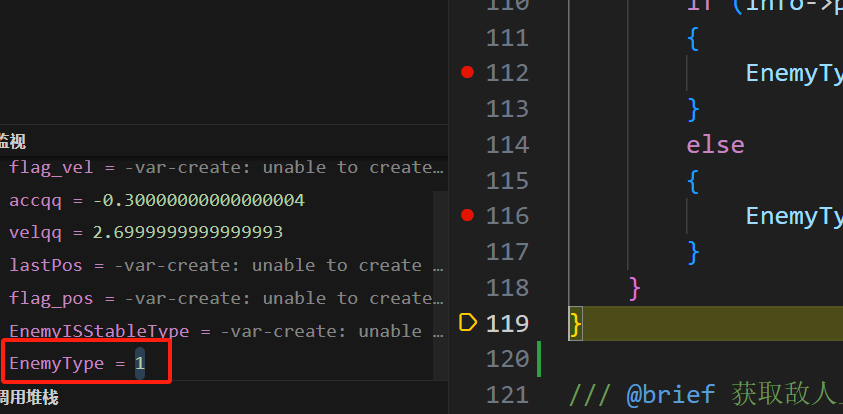
# 3测试分析

## 3.1模块测试

**GetEnemyType**

1.前期：第一个测试将示例敌人大魔王不做处理，第二个测试将其前期拉力设置为0。在类型赋值处打断点。

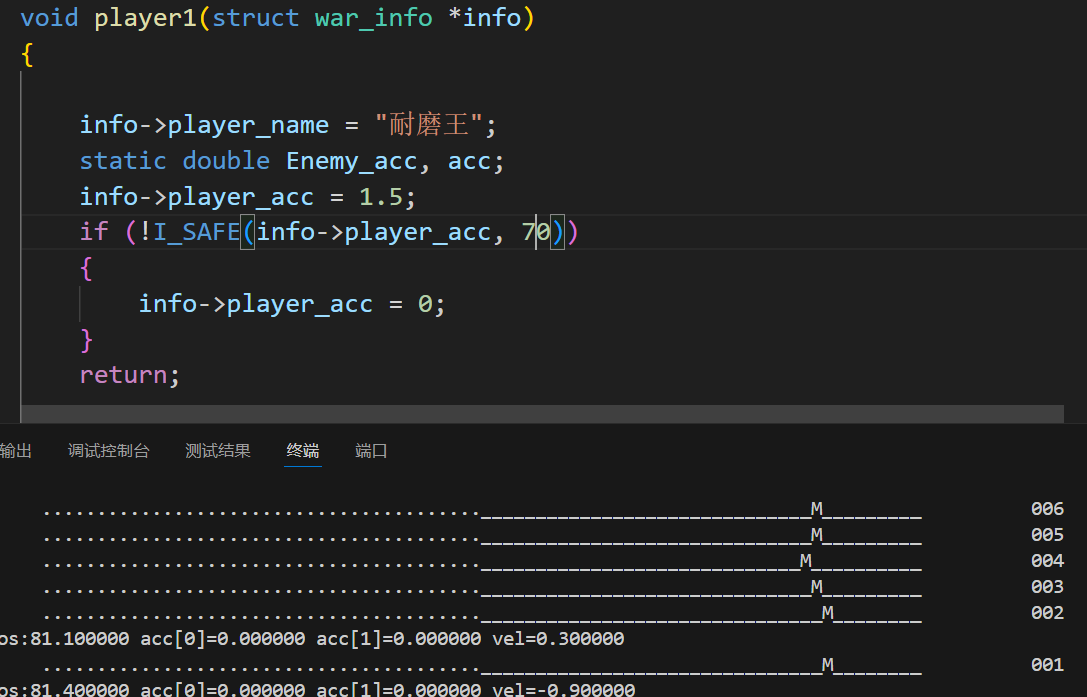
2.调试：运行，运行到断点。

3.结果：左为测试点1，右为测试点2故判断正常

**I\_Safe**

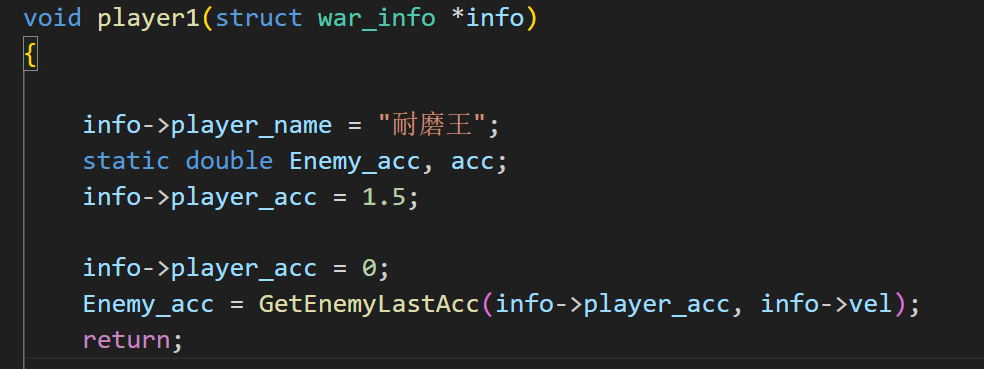
1.前期：写如图代码运行。

2.调试：运行，观察运动趋势

3.结果：旗帜在70左右来回徘徊（风速变化），故预测有效但是粗略。

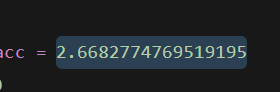
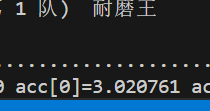
**GetEnemyLastAcc**

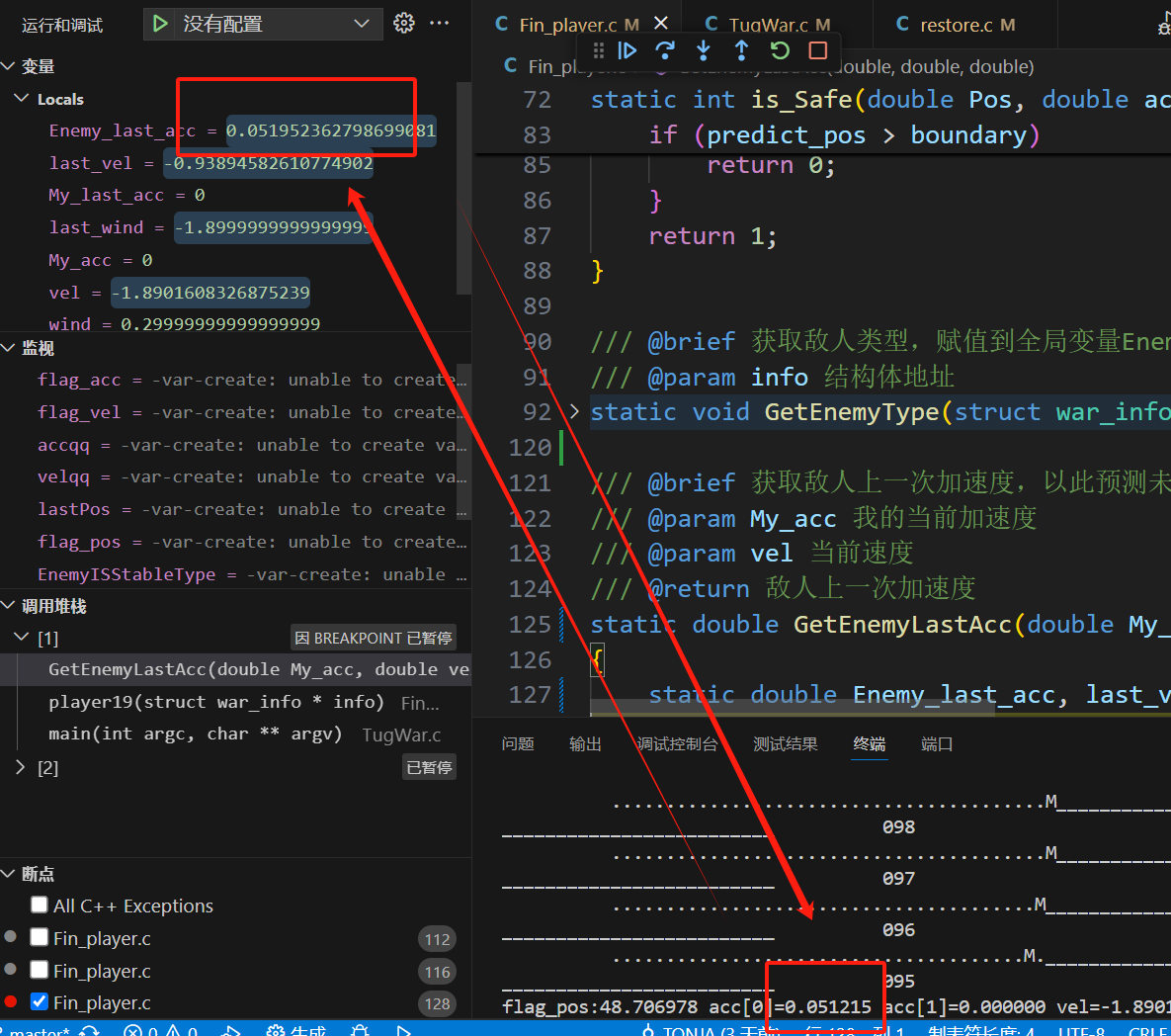
1.前期：写如图代码运行。在赋值处打断点。监测对方拉力及其估计的拉力。



2.调试：运行到断点。

3.结果：多次运行，可以发现估计值和实际值偏差在1左右（未考虑风力摩擦力）。



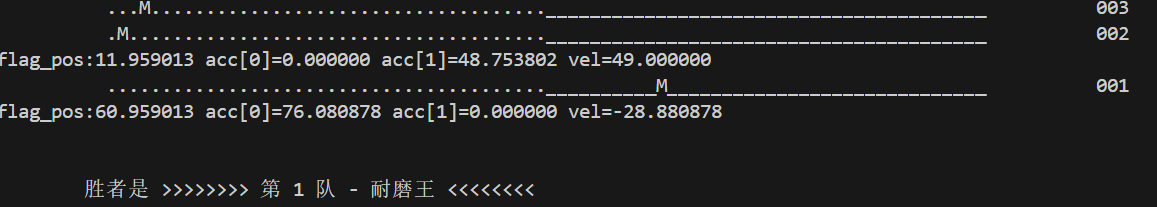
4.修正：补偿了风力和摩擦力，结果更加精准。

## 3.2联调测试

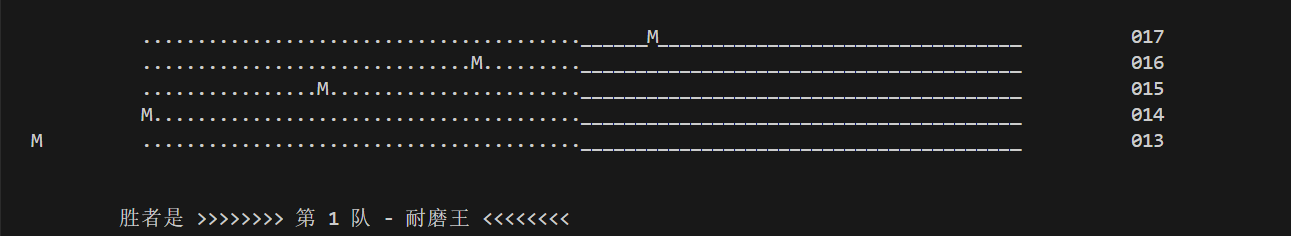
1.前期：将上一代玩家（有最后大力拉）与最终玩家测试。

2.调试：多次比赛。

3.结果：多次运行，可以发现上一代要么把自己拉出去要么最后时刻没拉赢，有几次自己拉出去了，对部分代码修正，增加了稳定性。

最后大力拉成功时刻：

放手机制成功时刻：



# 4结论与问题讨论

### 1、总结完成设计要求的程度

（1）成功实现了滑块轨迹预测的策略。通过动态评估未来可能的位置状态，优化发力，避免滑块进入边缘。

（2）根据对手策略灵活变化拉力，程序识别对方策略，快速调整，从而实现反制。

（3） 风险控制：加入了安全边际机制，避免滑块进入或三区右侧，提高了容错性和安全性。

### 2、遇到的问题及解决办法

1.第一版是使用PID法，在遇见PID多次过冲后我们去除了PID，设计了第二版，并加入了针对PID功能的放手机制,了解了PID算法后使对方过冲。基本是自我对抗总结最优解。[[1]](#A)

# 参考资料

[1] [CSDN博主 Kisorge. 模糊PID控制算法（C语言实现）[EB/OL]. CSDN博客, 2024. [2025-06-16].](https://blog.csdn.net/qq_42681425/article/details/144018034)