多人实时足球项目设计与实现

使用到的技术点

线程：多线程、互斥锁、条件变量、线程池

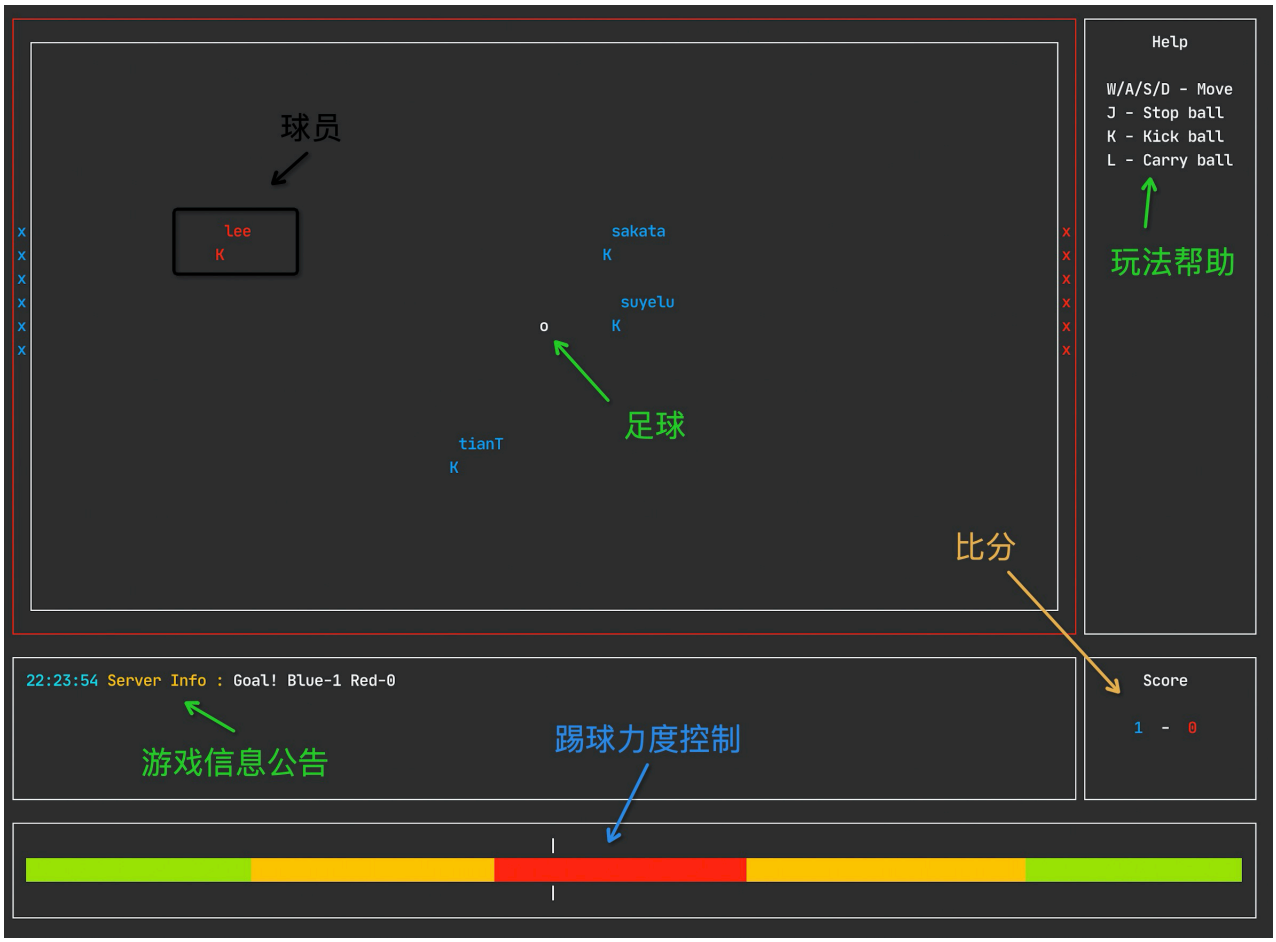
信号：SIGINT、SIGALARM、间隔定时器

IO：文件打开、文件读写、非阻塞IO、IO多路复用、Select、Epoll

网络: TCP、UDP、socket

需求说明

界面



整体框架

1. 本游戏是一个基于UDP网络编程的C/S架构的应用，需要独立完成 `server` 和 `client` 两端；
2. 玩游戏时，玩家启动 `client` 端，选择自己的队伍，登录 `server` 后，在球场外等待，之后自主进入球场，开始游戏；
3. 游戏过程中，由 `server` 端接收 `client` 的控制信息，决定玩家的移动，踢球，带球等游戏行为，再将游戏实时信息发送给所有客户端；
4. 玩家可以在游戏端发送聊天信息到客户端，同时也可以收到其他玩家的信息，当然也可以发送私聊信息给某位玩家

功能说明

1. 操作：
    - `space` 打开力度条，再次按下选中力度
    - `j` 停球
    - `k` 普通力度踢球
    - `l` 带球
    - `n` 显示球员姓名
    - `enter` 打开输入框，输入聊天信息
2. 移动
    - `w` 向上移动
    - `s` 向下移动
    - `a` 向左移动
    - `d` 向右移动
3. 规则判断：
    - 出界，直接在出界位置由对方球员发球
    - 进球，直接在球门前由守门员开球
    - 其他规则，暂无

详细说明

服务端

1. 启动
    - 启动时默认读取配置文件，获取端口 `port` ,游戏地图大小等参数
    - 同样也可以在启动时，通过指定选项，若指定选项，则配置文件中的配置不生效
2. 并发设置
    - 主线程
        - 主线程是一个 `acceptor` ，循环等待用户**登录**，登录后将用户按照队伍不同加到各自从反应堆中
        - 在主线程中，开启定时器，注册时钟信号处理函数，收到时钟信号，向 `client` 发送游戏数据
        - 主线程中注册 `sigint` 信号处理函数，收到 `ctrl + c` 时，告知客户端下线，结束运行

- 子线程
  - 两个子线程，每一个对应着一个队伍
  - 该线程为从反应堆，使用 `epoll` 等待用户控制信息，收到控制信息后，更新游戏数据

客户端

1. 启动
   - 启动时读取配置文件，获取服务端的 `ip` ， `port` 等信息，获取玩家用户名，队伍
   - 启动时也可以指定选项，若指定选项，则配置文件中的配置不生效
2. 并发设置
   - 主线程
     - 主线程向 `server` 发送登录信息，若收到响应，则成功登录，否则退出运行
     - 主线程中注册 `sigint` 信号处理函数，收到 `ctrl + c` 时，告知 `server` 下线，结束运行
     - 主线程负责接收键盘输入，判断游戏行为
   - 子线程
     - 子线程负责收服务端广播的信息，并做出解析
     - 重绘游戏界面

接口集合

数据接口

> 全部在common/datatype.h

球相关

文件： `datatype.h`

```
struct Bpoint{
    double x;
    double y;
};

struct Speed{
    double x;
    double y;
};

struct Aspeed{
    double x;
    double y;
};
```

```
15
16  struct BallStatus{
17      struct Speed v;
18      struct Aspeed a;
19      int who;//which队伍
20      char name[20];
21  };
22
23  struct Score{
24      int red;
25      int blue;
26  };
```

球员相关

文件: datatype.h

```
1   struct Point {
2       int x;
3       int y;
4   };
5   #define MAX 11 //每队球员数量
6   struct User {
7       int team; // 0 RED  1 BLUE
8       int fd; //该玩家对应的连接
9       char name[20];
10      int online;// 1 在线 0 不在线
11      int flag;  //未响应次数
12      struct Point loc;
13  };
```

数据交互相关

```
1   //登录相关的
2   struct  LogRequest {
3       char name[20];
4       int team;
5       char msg[512];
6   };
7
8   struct LogResponse{
9       int type; // 0 OK 1 NO
10      char msg[512];
11  };
12  //游戏期间交互
13  #define MAX_MSG 4096
14  //日常的消息交互，如聊天信息，统一为512长度
15
16  #define ACTION_KICK 0x01
17  #define ACTION_CARRY 0x02
18  #define ACTION_STOP 0x04
19
```

```c
struct Ctl {
    int action;
    int dirx;
    int diry;
    int strength; //踢球力度
};

#define FT_HEART 0x01 //心跳
#define FT_WALL 0x02 //公告
#define FT_MSG 0x04 //聊天
#define FT_ACK 0x08 //ack
#define FT_CTL 0x10 //控制信息
#define FT_GAME 0x20 //游戏场景数据
#define FT_SCORE 0x40 //比分变化
#define FT_GAMEOVER 0x80 //gameover
#define FT_FIN 0x100 //离场

struct FootBallMsg{
    int type;  // type & FT_HEART
    int size;
    int team;
    char name[20];
    char msg[MAX_MSG];
    struct Ctl ctl;
};
```

球场数据

```c
struct Map{
    int width;
    int height;
    struct Point start;
    int gate_width;
    int gate_height;
};
```

全局变量

服务端

```c
struct Map court; //足球场，中间那个正式场地
struct Bpoint ball; //球的位置
struct BallStatus ball_status; //球的状态
struct Score score; //比分
int repollfd, bepollfd; //从反应堆 sub_reactor
```

客户端

```
int sockfd;

```

通用接口

get_conf_value

> 从配置文件中，根据key找到value

文件：common.c 、common.h

```
char *get_conf_value(const char *path, const char *key);
printf("name=%s\n", get_conf_value("./football.conf", "NAME"));
char *get_conf_value(const char *path, const char *key) {
    FILE *fp = NULL;
    //判断path 和 key  的合法性
    //调用fopen(path, "r");
    //while (getline(&line, &size, fp))
        //  if strstr(line, key) != NULL
        //判断下一个自符，是不是等于号
        //strncpy()

    return ans;  //inet_ntoa()
}
/*
NAME=suyelu
*/
```

make_non_block

make_block

客户端实现细节

文件: client.c

```
int main(int argc, char **argv) {
    int server_port;
    char server_ip[20] = {0};
    char name[20] = {0};
    int team;
    char msg[512] = {0};
    //参数解析
    // h:p:n:t:m:
    // Host_ip_of_server, Port_of_server, Name_of_player, Team_num(1:blue, 0:
red),Message_for_login

```

```
    //判断，如果上述参数，没有在选项中定义，则从配置文件中获取；

    //打印输出各变量
    return 0;
}
/*
Usage: ./a.out -h serverip -p port -t team -n name
*/
```

Socket接口

socket_create_udp

> 创建已个绑定确定端口的UDP套接字

文件: udp_server.c 、 udp_server.h

```
int socket_create_udp(int port);
int socket_create_udp(int port) {
    //创建SOCK_DGRAM套接字
    //设置地址重用
    //设置为非阻塞套接字
    //绑定INADDR_ANY & port
}
```

socket_udp

> 创建一个主动的UDP套接字

文件： udp_client.c 、 udp_client.h

```
int socket_udp();
int socket_udp() {
    //创建一个SOCK_DGRAM套接字
}
```

Epoll接口

add_to_sub_reactor

> 将主反应堆上接入的客户添加到一个从反应堆

文件: udp_epoll.c 、 udp_epoll.h

```
#define MAX 50

extern int port;
```

```c
extern struct User *rteam;
extern struct User *bteam;
extern int repollfd, bepollfd;

void add_event(int epollfd, int fd, int events);
void add_event_ptr(int epollfd, int fd, int events, struct User *user);
void del_event(int epollfd, int fd);
int udp_connect(int epollfd, struct sockaddr_in *serveraddr);
int udp_accept(int epollfd, int fd, struct User *user);
void add_to_sub_reactor(struct User *user);

void add_event(int epollfd, int fd, int events) {
    //注册epoll事件到epoll实例中
}

void add_event_ptr(int epollfd, int fd, int events, struct User *user) {
    //注册epoll实例，使用data.ptr保存用户user地址
}

void del_event(int epollfd, int fd) {
    //从epollfd中注销fd文件
}

```

共用接口

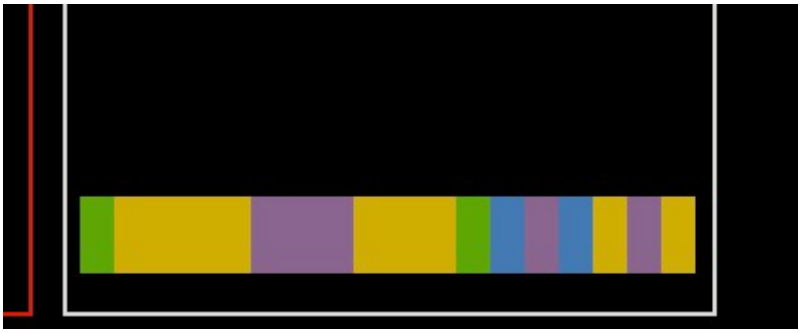
show_data_stream

> 显示收到的数据类型

文件：show_data_stream.h 、 show_data_stream.c

```c
extern char data_stream[20];
extern WINDOW *Help;
extern struct Map court;
//type:  'l','c','k','s','n','e'
//分别表示：login,  carry,  kick,  stop,  normal,  exit
void show_data_stream(int type);
void show_data_stream(int type) {
    //data_stream数组后移一位，将type加到第一位
    //根据type不同，使用wattron设置Help窗口的颜色
    //在合适位置打印一个空格
}

```



服务端接口

server_exit

> 服务端收到ctrl + c信号后的退出函数

文件: server_exit.c 、 server_exit.h

```c
#define MAX 50
extern struct User *rteam, *bteam;
void server_exit(int signum);
void server_exit(int signum) {
    struct FootBallMsg msg;
    msg.type = FT_FIN;
    for (int i = 0; i < MAX; i++) {
        if (rteam[i].online) send(rteam[i].fd, (void *)&msg, sizeof(msg), 0);
        if (bteam[i].online) send(bteam[i].fd, (void *)&msg, sizeof(msg), 0);
    }
    endwin();
    DBG(RED"Server stopped!\n"NONE);
    exit(0);
}
```

send_all

> 向所有在线的player发送一条消息

文件：server_send_all.c 、 server_send_all.h

```c
extern struct User *rteam, *bteam;
#define MAX 50

void send_to_team(struct User *team, struct FootBallMsg *msg) {
    for (int i = 0; i < MAX; i++) {
        if (team[i].online) send(team[i].fd, (void *)msg, sizeof(struct FootBallMsg), 0);
    }
}

void send_all(struct FootBallMsg *msg) {
    send_to_team(rteam, msg);
    send_to_team(bteam, msg);
}
```

thread_run

> 从反应堆线程池处理IO事件

文件: thread_pool.c 、 thread_poll.h

```c
void do_work(struct User *user);
void thread_run(void *arg);

void do_work(struct User *user) {

}

void *thread_run(void *arg) {
  struct task_queue *taskQueue = (struct task_queue *)arg;
  //分离线程
  //死循环弹出队列元素
  //调用do_work处理IO，传入参数为队列弹出的
}
```

heart_beat

> 服务端通过心跳机制判断客户端是否在线，单独线程执行

文件：heart_beat.c 、 heart_beat.h

```c
#define MAX 50

extern struct User *rteam, *bteam;
extern int repollfd, bepollfd;
```

```c
void heart_beat_team(struct User *team);
void *heart_beat(void *arg);

void heart_beat_team(struct User *team) {
    //遍历team数组，判断在线，则发送FT_TEST心跳包，flag--
  //判断palyer的flag是否减为0，减为0则判断为下线
    //数组中标记为offline
        //在从反应堆中注销IO
}
void heart_beat(void *arg) {
    //死循环，固定时间调用heart_beat_team
}
```

re_drew

> 服务端游戏刷新，定时执行，每次执行时，判断球的状态，沿着什么方向移动多远，人应该移动到什么位置

文件：server_re_drew.c 、server_re_drew.h

```c
extern struct User *rteam, *bteam;
extern WINDOW *Football, *Football_t;
extern struct BallStatus ball_status;
extern struct Bpoint ball;
#define MAX 50

void re_drew();
void re_drew_ball();
void re_drew_team(struct User *team);
void re_drew_palyer(int team, char *name, struct Point *loc);

void re_drew_ball() {
    //根据ball_status里记录的加速度，和上次re_drew时的速度，算出本次球应该移动的时间
  //加速度保持不变，速度更新
  //需要注意的是，当判读到速度减为0，ball_status里的速度和加速度都清空
  //同样需要注意的时，球如果超过球场边界，则判定为出界，球停止到边界即可
}

void  re_drew_player(int team, char *name, struct Point *loc) {
    //根据team，切换颜色
  //在loc位置打印player，并显示姓名
}

void re_drew_team(struct User *team) {
    //在team数组中，循环遍历用户，调用re_drew_palyer
}

void re_drew(){
    //分别调用re_drew_team、re_drew_ball
```

```
30  }
```

can_kick

> 判断player是否可以踢球，若成功，则更新球的运行方向，加速度，初速度

文件： ball_status.c、 ball_status.h

```
1   #define PI 3.1415926
2   extern WINDOW *Message;
3   extern struct Bpoint ball;
4   extern struct BallStatus ball_status;
5
6   int can_kick(struct Point *loc, int strength);
7
8   int can_kick(struct Point *loc, int strength){
9       //palyer和ball坐标对此
10      //判断palyer和ball的坐标在上下左右2个单位距离内，则可踢球
11      //根据player和ball的相对位置，计算球的运动方向，加速度方向，假设球只能在palyer和ball的延
    长线上运动
12      //假设player踢球的接触时间为0.2秒，默认加速度为40，力度增加，加速度也增加
13      //可踢返回1，否则返回0
14  }
```

can_access

> 判断停球，带球范围是否可达

文件： ball_status.c、 ball_status.h

```
1   #define PI 3.1415926
2   extern WINDOW *Message;
3   extern struct Bpoint ball;
4   extern struct BallStatus ball_status;
5
6   int can_access(struct Point *loc);
7
8   int can_access(struct Point *loc) {
9       //修正坐标 判断player是否在ball的2*2范围内
10      //可达返回1，否则返回0
11  }
```

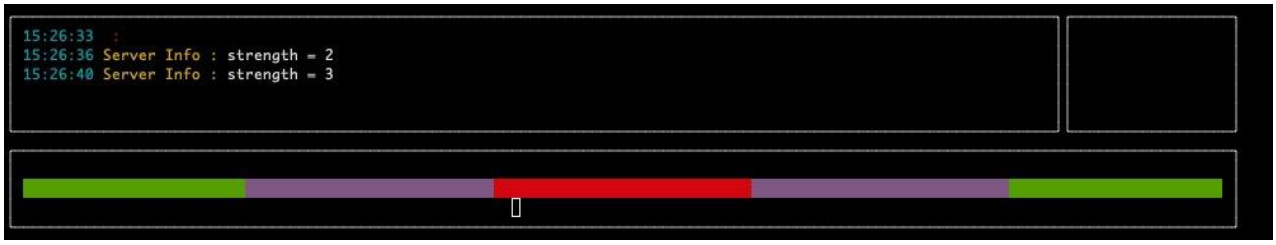
客户端接口

show_strength

> 显示踢球时的力度，并通过控制按键时机达到控制踢球力度的效果

文件: show_strength.c 、show_strength.h

```c
extern WINDOW *Write;
extern int sockfd;
void show_strength();
void show_strength() {
    //在Write窗口中，显示踢球力度条，光标在进度条上快速移动
    //设置0为非阻塞IO
    //while 等待空格或者'k'键的按下，如果按下退出，取得当前的strength
    //通过sockfd向服务端发送控制信息，踢球
}
```



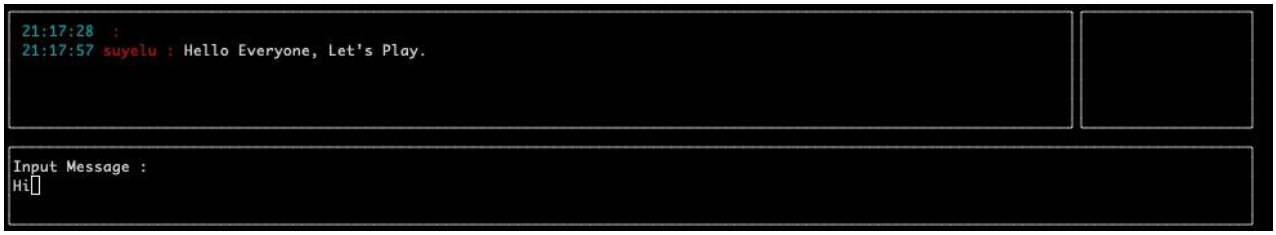
send_chat

> 用户输入聊天信息，并发送给服务端

文件：send_chat.c 、send_chat.h

```c
extern int sockfd;
extern WINDOW *Write;
extern struct FootBallMsg chat_msg;

void send_chat();
void send_chat() {
    //打开echo回显
    //打开行缓冲
    //在Write窗口中输入数据并读入
    //判断读入信息非空，发送
    //重绘Write
    //关闭echo
    //关闭行缓冲
}
```



send_ctl

> 客户端发送控制信息到服务端

文件：send_ctl.c 、 send_ctl.h

```c
extern int sockfd;
extern struct FootBallMsg ctl_msg;
void send_ctl();
void send_ctl() {
    if (ctl_msg.ctl.dirx || ctl_msg.ctl.diry)
        send(sockfd, (void *)&ctl_msg, sizeof(ctl_msg), 0);
    ctl_msg.ctl.dirx = ctl_msg.ctl.diry = 0;
}
```