# 贪吃蛇多人对战

## nMessageHead 对应

## 1 登录请求

## 2 准备请求

## 3 申请游戏最终数据

## 4 接收玩家ID和用户名

## 5 接收蛇的位置信息

## 6 玩家死亡

## 7 游戏结束

## 8 发送蛇头方向

## RetroSnaker Client：

1. 输入IP和用户名点击登录, socket连接成功发送登录请求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMsgID | int |  |  |  |
| nMessageHead | int | 1 | 请求登录 |  |
| strPlayerName | string | Your Name | 玩家的名称 | 玩家自己输入的 |

应答

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMessageLen | int |  | 数据包大小 |  |
| nMsgID | int |  |  |  |
| nMessageHead | int | 1 | 登录成功的信号 |  |
| strPlayerName | string | Your Name | 玩家的名称 | 返回玩家名称 |
| nPlayerID | int | Num | 由服务端生成的唯一的ID，用于识别玩家。 | 用来识别玩家的 |
| nColour | int | 如250250250 | 用RBG的来表示颜色，会用这个颜色来代表该玩家蛇的颜色。 | 客户端解析成 250 250 250 |

会返回当前连接上服务端所有玩家的信息。

显示玩家信息并用一个std::unordered\_map<string(strPlayerName), int(nPlayerID)PlayerData 记录玩家信息

1. 当连接成功会切换到游戏准备界面，该界面会显示当前房间玩家的名称+ID 和玩家是否准备的状态， 还有一个准备按钮。
2. 点击准备按钮。向服务端发送

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMsgID | int |  |  | 每发送一个包会nMsgID++ |
| nMessageHead | int | 2 | 玩家请求开始的头消息 |  |
| nPlayerId | int | Num | 用来识别玩家 | 由系统一开始生成的。 |

应答

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMessageLen | int |  | 数据包大小 |  |
| nMsgID | int |  |  |  |
| nMessageHead | int | 2 | 返回玩家状态 | 向所有连接上客户端发送该玩家状态 |
| nPlayerId | int | Num | 用来识别玩家 | 由系统一开始生成的。 |

客服端等待游戏开始命令或者接收其他玩家状态的消息。

1. 当服务端发来游戏开始信息（这里开始这样判断可以么， 直接由服务端发起开始？）

推送：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMessageLen | int | len | 表示当前数据包的大小 |  |
| nMsgID | int | 0 |  | 每发送一个包会nMsgID++ |
| nMessageHead | int | 5 | 游戏开始消息 | 由服务端判断人数够了直接发起游戏开始 |
| nDotX | int | XX | 由系统随机生成的点的X轴坐标 |  |
| nDotY | int | YY | 由系统随机生成的点的Y轴坐标 |  |
| nSnakeNumber | int | Num | 当前对局中存活的蛇的个数 |  |
| nPlayerOneID | int | Num | 用来识别玩家 | 由系统一开始生成的。 |
| nDirectionOne | int | Num | 代表蛇头的方向 |  |
| nSnakerOneLen | int | Num | 代表蛇的大小 |  |
| nSnakeX | int | XX | 代表蛇一节的X轴坐标 |  |
| nSnakeY | int | YY | 代表蛇一节的Y轴坐标 |  |
| ....... |  |  |  |  |
| nPlayerTweID | int | Num | 用来识别玩家 | 由系统一开始生成的。 |
| nDirectionTwe | int | Num | 代表蛇头的方向 |  |
| nSnakerTweLen | int | Num | 代表蛇的大小 |  |
| nSnakeX | int | XX | 代表蛇一节的X轴坐标 |  |
| nSnakeY | int | YY | 代表蛇一节的Y轴坐标 |  |
| ...... | ...... | ...... | ...... | ...... |

1. 经过处理后得到

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nDotX | int | XX | 由系统随机生成的点的X轴坐标 |  |
| nDotY | int | YY | 由系统随机生成的点的Y轴坐标 |  |
| nSnakeNumber | int | Num | 当前对局中存活的蛇的个数 |  |
| nPlayerOneID | int | Num | 用来识别玩家 | 由系统一开始生成的。 |
| nDirectionOne | int | Num | 代表蛇头的方向 |  |
| nSnakerOneLen | int | Num | 代表蛇的大小 |  |
| SnakePos | std::vector<Vec2> |  | 储存蛇各个节点的(x, y)坐标 |  |
| nPlayerTweID | int | Num | 用来识别玩家 | 由系统一开始生成的。 |
| nDirectionTwe | int | Num | 代表蛇头的方向 |  |
| nSnakerTweLen | int | Num | 代表蛇的大小 |  |
| SnakePos | std::vector<Vec2> |  | 储存蛇各个节点的(x, y)坐标 |  |
| ....... |  |  |  |  |

然后储存到任务队列里面

1. 主线程设置一个定时器定时刷新任务队列是否为空，有则拉取任务处理。
2. 控制方向4个控件，点击发送改变方向请求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMsgID | int | Num |  | 每发送一个包会nMsgID++ |
| nMessageHead | int | 3 | 请求改变方向 |  |
| nPlayerId | int | Num | 用来识别玩家 | 由系统一开始生成的。 |
| nDirection | int |  | 请求改变方向 |  |

服务端会在下一次发送数据的时改变蛇头方向（具体应答见4）

1. 如果玩家死亡。客户端接收

推送

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMsgID | int | 0 | 服务端主动发出命令显示0 | 每发送一个包会nMsgID++ |
| nMessageHead | int | 6 | 该玩家已经阵亡。 |  |
| nPlayerID | int | Num | 用来识别玩家 | 由系统一开始生成的。 |

该玩家则进入观战状态。

1. 当只剩下一条蛇的时候，游戏结束。

推送

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMsgID | int | 0 | 服务端主动发出命令显示0 | 每发送一个包会nMsgID++ |
| nMessageHead | int | 7 | 跳转游戏结束界面的命令 |  |
| nPlayerId | int | Num | 用来识别玩家 | 由系统一开始生成的。 |

1. 客户端请求游戏得分

请求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMsgID | int | 0 | 服务端主动发出命令显示0 | 每发送一个包会nMsgID++ |
| nMessageHead | int | 3 | 游戏申请游戏数据 |  |

应答

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMessageLen | int |  | 数据包长度 |  |
| nMsgID | int |  |  | 每发送一个包会nMsgID++ |
| nMessageHead | int | 6 | 显示玩家数据 |  |
| nPlayerId | int | Num | 用来识别玩家 | 由系统一开始生成的。 |
| nScore | int |  | 玩家得分 |  |
| nPlayerID | int |  | 玩家二ID |  |
| nScore | int |  | 玩家二得分 |  |
| ... |  |  |  |  |
|  |  |  |  |  |

界面下面有再来一局

发送一个准备请求（具体应答见3.）。跳转准备界面。

1. 还有一个退出按钮。点击发送退出数据包并退出。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMsgID | int | 0 | 服务端主动发出命令显示0 | 每发送一个包会nMsgID++ |
| nMessageHead | int | 0 | 退出命令 |  |
| nPlayerId | int | Num | 用来识别玩家 | 由系统一开始生成的。 |

服务端向其他客服发出应答

推送

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| nMessageLen | int |  | 数据包大小 |  |
| nMsgID | int | 0 | 服务端主动发出命令显示0 | 每发送一个包会nMsgID++ |
| nMessageHead | int | 5 | 跳转游戏结束界面的命令 |  |
| nPlayerId | int | Num | 用来识别玩家 | 由系统一开始生成的。 |

不显示退出玩家ID和名字

## RetroSnakerServer:

1. 开启一个线程处于循环接收消息状态。
2. 接收到玩家登录请求时。

服务器创建玩家唯一ID和玩家colour.

并记录到结构体里面

结构体：struct PlayerData:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 值 | 描述 | 备注 |
| strPlayerName | str |  | 玩家定义的自己的名称 |  |
| nColour | int |  | 玩家颜色 |  |
| nScore | int |  | 玩家的得分 |  |
| sock | SOCKET |  | 连接对应的套接字 |  |

创建一个std::nordered\_map<int(nPlayerId), PlayerData)> m\_PlayerData;储存玩家信息。

1. 玩家准备人数超过两个人，如果房间没有其他人的时候则开始。

如果两人准备，房间还有其他人的话。则等待30秒，为准备次玩家则进入观战状态。

1. 主线程有一个定时器1秒刷新蛇的坐标然后迭代m\_PlayerData发送给客户端。
2. 检测玩家是否死亡。
3. 有些结束发送跳转游戏界面命令。