SSRoboime

Generated by Doxygen 1.9.8

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

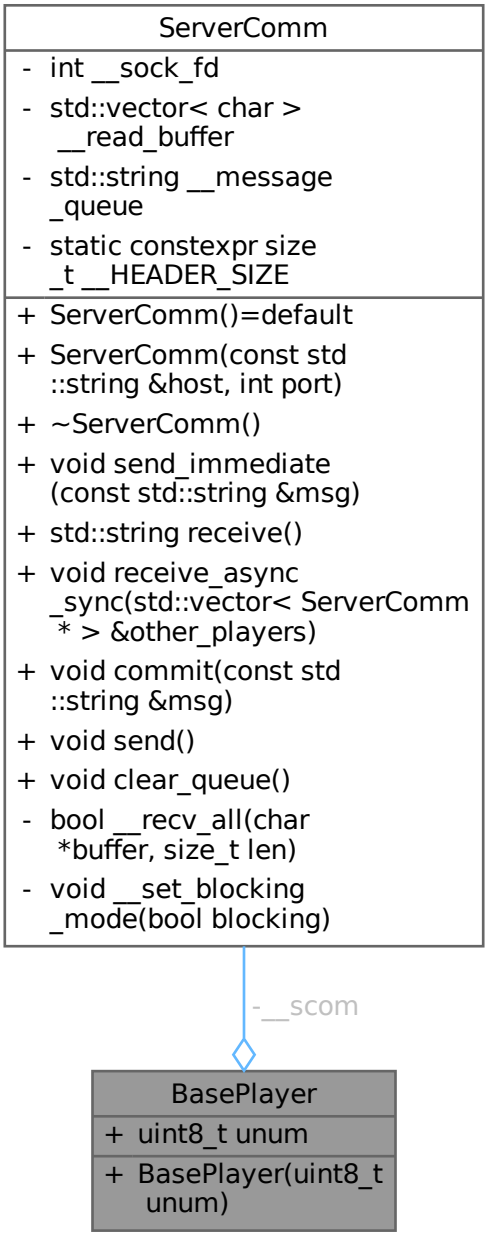
Chapter 3

Class Documentation

3.1 BasePlayer Class Reference

```
#include <BasePlayer.hpp>
```

Collaboration diagram for BasePlayer:



Public Member Functions

- BasePlayer (uint8_t unum)

Public Attributes

- uint8_t unum

  Número de Uniforme, `uint8_t` para economizar memória, dado que varia de 1 à 11.

Private Attributes

- ServerComm __scom

  Classe gerenciadora de comunicação com servidor rcssserver3d.

3.1.1 Detailed Description

Definition at line 6 of file BasePlayer.hpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BasePlayer()

```
BasePlayer::BasePlayer (
            uint8_t unum )  [inline]
```

< Realmente será útil em nosso código

Definition at line 14 of file BasePlayer.hpp.

3.1.3 Member Data Documentation

3.1.3.1 __scom

```
ServerComm BasePlayer::__scom  [private]
```

Classe gerenciadora de comunicação com servidor rcssserver3d.

Definition at line 8 of file BasePlayer.hpp.

3.1.3.2 unum

```
uint8_t BasePlayer::unum
```

Número de Uniforme, `uint8_t` para economizar memória, dado que varia de 1 à 11.

Definition at line 11 of file BasePlayer.hpp.

The documentation for this class was generated from the following file:

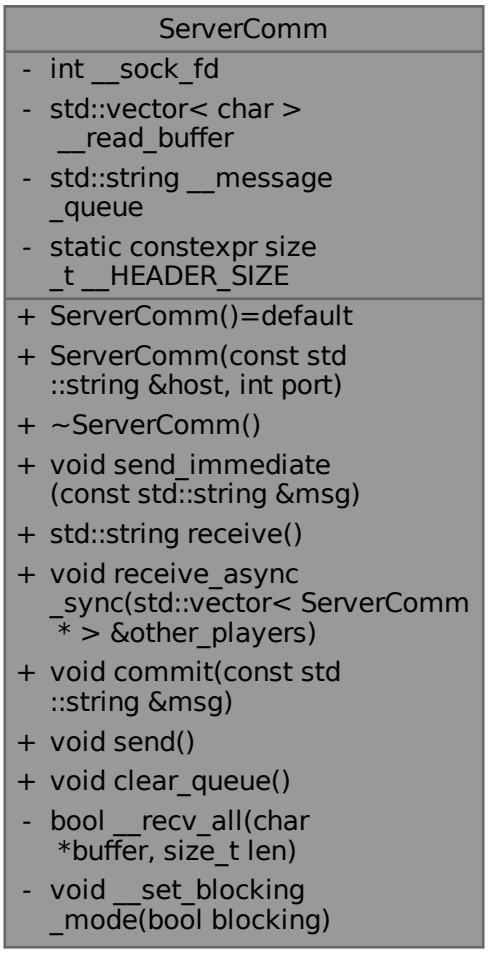
- src/Agent/BasePlayer.hpp

3.2 ServerComm Class Reference

Responsável pela implementação da comunicação com servidor rcssserver3d.

`#include <ServerComm.hpp>`

Collaboration diagram for ServerComm:



Public Member Functions

- ServerComm ()=default
- ServerComm (const std::string &host, int port)

  Construtor que inicializa o socket e buffers.
- ~ServerComm ()

  Destrutor para limpeza de recursos.
- void send_immediate (const std::string &msg)

  Envia uma mensagem instantânea usando Scatter/Gather I/O (writev).

- std::string receive ()

  Recebe dados do socket, processa o cabeçalho e extrai o payload.
- void receive_async_sync (std::vector< ServerComm * > &other_players)

  Lógica especial para sincronização inicial (handshake) com múltiplos agentes.
- void commit (const std::string &msg)

  Adiciona uma mensagem à fila de envio (bufferização).
- void send ()

  Envia todas as mensagens da fila de uma vez.
- void clear_queue ()

  Limpa a fila de mensagens sem enviar.

Private Member Functions

- bool __recv_all (char *buffer, size_t len)

  Garante o recebimento completo de N bytes (lida com fragmentação TCP).
- void __set_blocking_mode (bool blocking)

  Configura o socket para modo bloqueante ou não-bloqueante.

Private Attributes

- int __sock_fd

  File descriptor do socket.
- std::vector< char > __read_buffer

  Buffer persistente para leitura de dados (evita realocações)
- std::string __message_queue

  Fila de mensagens a serem enviadas (buffer de escrita)

Static Private Attributes

- static constexpr size_t __HEADER_SIZE = 4

  Tamanho do cabeçalho padrão do protocolo rcssserver3d.

3.2.1 Detailed Description

Responsável pela implementação da comunicação com servidor rcssserver3d.

< — Bibliotecas da Standard Library (C++) — Container dinâmico usado para buffers de leitura Manipulação de strings para filas de mensagens Entrada e saída padrão (std::cerr, std::cout) < — Bibliotecas de Sistema (POSIX/↩ Linux) —

Definition at line 24 of file ServerComm.hpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ServerComm() [1/2]

```
ServerComm::ServerComm ( )  [default]
```

3.2.2.2 ServerComm() [2/2]

```
ServerComm::ServerComm (
            const std::string & host,
            int port )  [inline]
```

Construtor que inicializa o socket e buffers.

Parameters

host	Endereço IP do servidor.
port	Porta do servidor.

Definition at line 92 of file ServerComm.hpp.

3.2.2.3 ~ServerComm()

```
ServerComm::~ServerComm ( )  [inline]
```

Destrutor para limpeza de recursos.

Definition at line 151 of file ServerComm.hpp.

3.2.3 Member Function Documentation

3.2.3.1 __recv_all()

```
bool ServerComm::__recv_all (
            char * buffer,
            size_t len )  [inline], [private]
```

Garante o recebimento completo de N bytes (lida com fragmentação TCP).

Parameters

buffer	Ponteiro para onde os dados serão escritos.
len	Quantidade exata de bytes a ler.

Returns

True se leu tudo, False se a conexão caiu ou erro.

Definition at line 44 of file ServerComm.hpp.

3.2.3.2 __set_blocking_mode()

```
void ServerComm::__set_blocking_mode (
            bool blocking )  [inline], [private]
```

Configura o socket para modo bloqueante ou não-bloqueante.

Parameters

blocking	True para bloqueante, False para não-bloqueante.

Definition at line 68 of file ServerComm.hpp.

3.2.3.3 clear_queue()

```
void ServerComm::clear_queue ( )  [inline]
```

Limpa a fila de mensagens sem enviar.

Definition at line 319 of file ServerComm.hpp.

3.2.3.4 commit()

```
void ServerComm::commit (
            const std::string & msg )  [inline]
```

Adiciona uma mensagem à fila de envio (bufferização).

Parameters

msg	Mensagem em bytes/string.

Definition at line 291 of file ServerComm.hpp.

3.2.3.5 receive()

```
std::string ServerComm::receive ( )  [inline]
```

Recebe dados do socket, processa o cabeçalho e extrai o payload.

Returns

Uma string contendo o corpo da mensagem recebida (sem os 4 bytes de tamanho).

Definition at line 194 of file ServerComm.hpp.

3.2.3.6 receive_async_sync()

```
void ServerComm::receive_async_sync (
            std::vector< ServerComm * > & other_players )  [inline]
```

Lógica especial para sincronização inicial (handshake) com múltiplos agentes.

Parameters

other_players	Ponteiro para vetor de outros agentes.

Definition at line 247 of file ServerComm.hpp.

3.2.3.7 send()

```
void ServerComm::send ( )  [inline]
```

Envia todas as mensagens da fila de uma vez.

Adiciona (syn) ao final automaticamente se o socket estiver livre para escrita.

Definition at line 299 of file ServerComm.hpp.

3.2.3.8 send_immediate()

```
void ServerComm::send_immediate (
            const std::string & msg )  [inline]
```

Envia uma mensagem instantânea usando Scatter/Gather I/O (writev).

Parameters

msg	A string de dados a ser enviada.

Usa writev para enviar Header+Body em uma única syscall sem cópia de memória.

Definition at line 162 of file ServerComm.hpp.

3.2.4 Member Data Documentation

3.2.4.1 __HEADER_SIZE

```
constexpr size_t ServerComm::__HEADER_SIZE = 4  [static], [constexpr], [private]
```

Tamanho do cabeçalho padrão do protocolo rcssserver3d.

Definition at line 36 of file ServerComm.hpp.

3.2.4.2 __message_queue

```
std::string ServerComm::__message_queue  [private]
```

Fila de mensagens a serem enviadas (buffer de escrita)

Definition at line 33 of file ServerComm.hpp.

3.2.4.3 __read_buffer

```
std::vector<char> ServerComm::__read_buffer  [private]
```

Buffer persistente para leitura de dados (evita realocações)

Definition at line 30 of file ServerComm.hpp.

3.2.4.4 __sock_fd

```
int ServerComm::__sock_fd  [private]
```

File descriptor do socket.

Definition at line 27 of file ServerComm.hpp.

The documentation for this class was generated from the following file:

- src/Communication/ServerComm.hpp

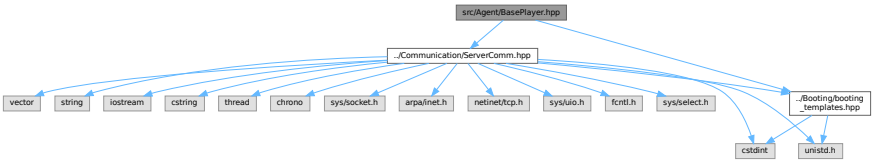
Chapter 4

File Documentation

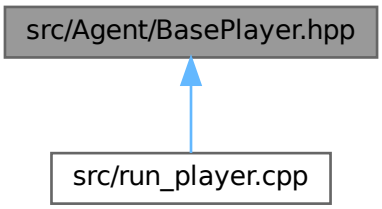
4.1 src/Agent/BasePlayer.hpp File Reference

```
#include "../Booting/booting_templates.hpp"
#include "../Communication/ServerComm.hpp"
```

Include dependency graph for BasePlayer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BasePlayer

4.2 BasePlayer.hpp

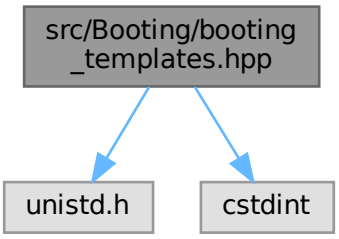
Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "../Booting/booting_templates.hpp"
00004 #include "../Communication/ServerComm.hpp"
00005
00006 class BasePlayer {
00007 private:
00008     ServerComm __scom;
00009
00010 public:
00011     uint8_t unum;
00012
00013 public:
00014     BasePlayer(
00015         uint8_t unum
00016     ) {
00017
00019         this->unum = unum;
00020         this->__scom = ServerComm(AGENT_HOST, AGENT_PORT);
00021
00022
00023
00024     }
00025
00026
00027
00028
00029
00030
00031
00032 };
```

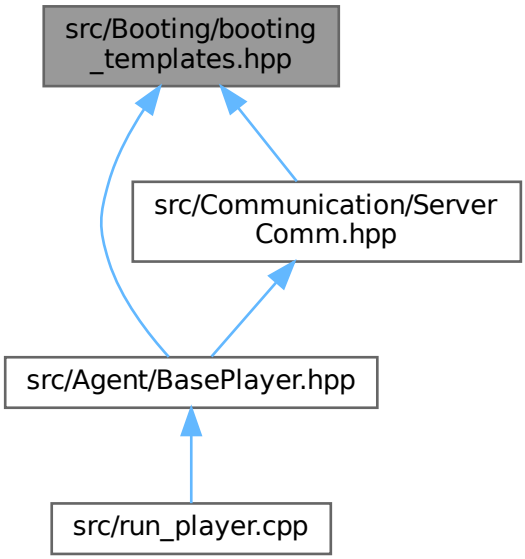
4.3 src/Booting/booting_templates.hpp File Reference

```
#include <unistd.h>
#include <cstdint>
```

Include dependency graph for booting_templates.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define True true
- #define False false

Variables

- constexpr const char * AGENT_HOST = "localhost"
- constexpr int AGENT_PORT = 3100
- constexpr const char * TEAM_NAME = "RoboIME"
- constexpr bool DEBUG_MODE = False

4.3.1 Macro Definition Documentation

4.3.1.1 False

```
#define False false
```

Definition at line 7 of file booting_templates.hpp.

4.3.1.2 True

```
#define True true
```

Definition at line 6 of file booting_templates.hpp.

4.3.2 Variable Documentation

4.3.2.1 AGENT_HOST

```
constexpr const char* AGENT_HOST = "localhost"  [inline], [constexpr]
```

Definition at line 9 of file booting_templates.hpp.

4.3.2.2 AGENT_PORT

```
constexpr int AGENT_PORT = 3100  [inline], [constexpr]
```

Definition at line 10 of file booting_templates.hpp.

4.3.2.3 DEBUG_MODE

```
constexpr bool DEBUG_MODE = False  [inline], [constexpr]
```

Definition at line 12 of file booting_templates.hpp.

4.3.2.4 TEAM_NAME

```
constexpr const char* TEAM_NAME = "RoboIME"  [inline], [constexpr]
```

Definition at line 11 of file booting_templates.hpp.

4.4 booting_templates.hpp

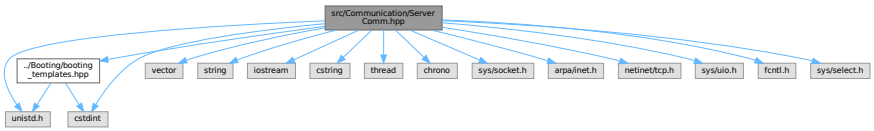
Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <unistd.h>
00004 #include <cstdint>
00005
00006 #define True true
00007 #define False false
00008
00009 inline constexpr const char* AGENT_HOST = "localhost";
00010 inline constexpr int AGENT_PORT = 3100;
00011 inline constexpr const char* TEAM_NAME = "RoboIME";
00012 inline constexpr bool DEBUG_MODE = False;
00013
```

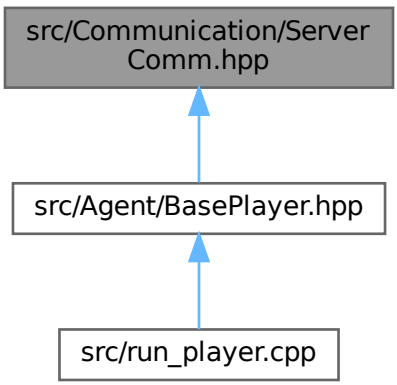
4.5 src/Communication/ServerComm.hpp File Reference

```
#include "../Booting/booting_templates.hpp"
#include <vector>
#include <string>
#include <iostream>
#include <cstring>
#include <cstdint>
#include <thread>
#include <chrono>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <sys/uio.h>
#include <fcntl.h>
#include <sys/select.h>
```

Include dependency graph for ServerComm.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class ServerComm

  Responsável pela implementação da comunicação com servidor rcssserver3d.

4.6 ServerComm.hpp

Go to the documentation of this file.

```
00001 #pragma once
00003
00004 #include "../Booting/booting_templates.hpp"
00005
00007 #include <vector>
00008 #include <string>
00009 #include <iostream>
00010 #include <cstring>      // Manipulação de memória bruta (std::memcpy, std::memset)
00011 #include <cstdint>      // Tipos de inteiros com tamanho fixo (uint32_t)
00012 #include <thread>       // (Opcional) Para sleep_for se desejar substituir usleep
00013 #include <chrono>       // (Opcional) Para unidades de tempo
00014
00016 #include <sys/socket.h> // API principal de Sockets (socket, connect, recv, send)
00017 #include <arpa/inet.h>  // Conversão de endereços IP (sockaddr_in, inet_pton, htons)
00018 #include <netinet/tcp.h>// Definições específicas do protocolo TCP (TCP_NODELAY)
00019 #include <unistd.h>     // Chamadas de sistema Unix padrão (close, writev, usleep)
00020 #include <sys/uio.h>    // Estruturas para I/O vetorial (struct iovec para writev)
00021 #include <fcntl.h>      // Controle de descritores de arquivo (bloqueante/não-bloqueante)
00022 #include <sys/select.h> // Multiplexação de I/O síncrono (select)
00023
00024 class ServerComm {
00025 private:
00027     int __sock_fd;
00028
00030     std::vector<char> __read_buffer;
00031
00033     std::string __message_queue;
00034
00036     static constexpr size_t __HEADER_SIZE = 4;
00037
00044     bool __recv_all(char* buffer, size_t len) {
00045         size_t total_read = 0;
00046
00047         while(total_read < len) {
00048             // Calcula onde escrever no buffer e quanto falta ler
00049             char* write_ptr = buffer + total_read;
00050             size_t bytes_needed = len - total_read;
00051
00052             ssize_t bytes = ::recv(this->__sock_fd, write_ptr, bytes_needed, 0);
00053
00054             if (bytes <= 0) {
00055                 return False;
00056             }
00057
00058             total_read += bytes;
00059         }
00060
00061         return True;
00062     }
00063
00068     void __set_blocking_mode(bool blocking) {
00069         int flags = fcntl(this->__sock_fd, F_GETFL, 0);
00070
00071         if (flags == -1) {
00072             return;
00073         }
00074
00075         if (blocking) {
00076             flags &= ~O_NONBLOCK;
00077         } else {
00078             flags |= O_NONBLOCK;
00079         }
00080
00081         fcntl(this->__sock_fd, F_SETFL, flags);
00082     }
00083
00084 public:
00085     ServerComm() = default;
00086
00092     ServerComm(const std::string& host, int port) {
00093         // Inicialização de variáveis membro
00094         this->__sock_fd = -1;
00095         this->__read_buffer.resize(4096); // Pré-aloca 4KB
00096         this->__message_queue.reserve(4096); // Reserva espaço
00097
00098         // 1. Criação do Socket
00099         this->__sock_fd = socket(AF_INET, SOCK_STREAM, 0);
00100
00101         if (this->__sock_fd < 0) {
00102             std::cerr « "Erro fatal: Falha ao criar socket." « std::endl;
00103             exit(1);
00104         }
```

```
00105
00106         // 2. Otimização de Performance: TCP_NODELAY
00107         int flag = 1;
00108         int result_opt = setsockopt(
00109             this->__sock_fd,
00110             IPPROTO_TCP,
00111             TCP_NODELAY,
00112             (char*)&flag,
00113             sizeof(int)
00114         );
00115
00116         if (result_opt < 0) {
00117              std::cerr << "Aviso: Falha ao definir TCP_NODELAY." << std::endl;
00118         }
00119
00120         // 3. Configuração do Endereço
00121         struct sockaddr_in serv_addr;
00122         std::memset(&serv_addr, 0, sizeof(serv_addr));
00123
00124         serv_addr.sin_family = AF_INET;
00125         serv_addr.sin_port = htons(port);
00126         inet_pton(AF_INET, host.c_str(), &serv_addr.sin_addr);
00127
00128         // 4. Tentativa de Conexão (Verticalizado)
00129         while (true) {
00130             int connection_result = connect(
00131                 this->__sock_fd,
00132                 (struct sockaddr*)&serv_addr,
00133                 sizeof(serv_addr)
00134             );
00135
00136             if (connection_result == 0) {
00137                 // Conexão bem sucedida
00138                 break;
00139             }
00140
00141             // Em produção, usar usleep para não travar CPU
00142             usleep(500000); // 0.5s
00143         }
00144
00145         printf("Consegui");
00146     }
00147
00151     ~ServerComm() {
00152         if (this->__sock_fd >= 0) {
00153             close(this->__sock_fd);
00154         }
00155     }
00156
00162     void send_immediate(const std::string& msg) {
00163         if (msg.empty()) {
00164             return;
00165         }
00166
00167         // Prepara o tamanho em Network Byte Order (Big Endian)
00168         uint32_t msg_len = static_cast<uint32_t>(msg.size());
00169         uint32_t net_len = htonl(msg_len);
00170
00171         // Estrutura para envio vetorial (Zero-Copy concatenation)
00172         struct iovec iov[2];
00173
00174         // Parte 1: Cabeçalho (4 bytes)
00175         iov[0].iov_base = &net_len;
00176         iov[0].iov_len = sizeof(net_len);
00177
00178         // Parte 2: Corpo da mensagem
00179         iov[1].iov_base = (void*)msg.data();
00180         iov[1].iov_len = msg_len;
00181
00182         // Envia ambos os buffers atomicamente
00183         ssize_t bytes_written = writev(this->__sock_fd, iov, 2);
00184
00185         if (bytes_written < 0) {
00186             std::cerr << "Erro ao enviar dados via writev." << std::endl;
00187         }
00188     }
00189
00194     std::string receive() {
00195         // Verifica se há dados disponíveis (non-blocking check via select)
00196         fd_set readfds;
00197         FD_ZERO(&readfds);
00198         FD_SET(this->__sock_fd, &readfds);
00199
00200         // Timeout zero = retorno imediato se não houver dados
00201         struct timeval tv = {0, 0};
00202
00203         int activity = select(
```

```
00204             this->__sock_fd + 1,
00205             &readfds,
00206             NULL,
00207             NULL,
00208             &tv
00209         );
00210
00211         if (activity <= 0) {
00212             return ""; // Sem dados
00213         }
00214
00215         // 1. Ler Cabeçalho (4 bytes)
00216         char header[this->__HEADER_SIZE];
00217         bool header_ok = this->__recv_all(header, this->__HEADER_SIZE);
00218
00219         if (!header_ok) {
00220             return ""; // Erro ou desconexão
00221         }
00222
00223         // 2. Converter tamanho
00224         uint32_t net_len;
00225         std::memcpy(&net_len, header, this->__HEADER_SIZE);
00226         uint32_t msg_len = ntohl(net_len);
00227
00228         // 3. Ajustar buffer se necessário (evita alocação se msg for pequena)
00229         if (this->__read_buffer.size() < msg_len) {
00230             this->__read_buffer.resize(msg_len);
00231         }
00232
00233         // 4. Ler Corpo
00234         bool body_ok = this->__recv_all(this->__read_buffer.data(), msg_len);
00235
00236         if (!body_ok) {
00237             return "";
00238         }
00239
00240         return std::string(this->__read_buffer.data(), msg_len);
00241     }
00242
00247     void receive_async_sync(std::vector<ServerComm*>& other_players) {
00248         if (other_players.empty()) {
00249             this->receive();
00250             return;
00251         }
00252
00253         this->__set_blocking_mode(false); // Ativa modo não-bloqueante
00254
00255         while (true) {
00256             // Tenta 'espiar' o socket para ver se tem dados
00257             char peek_buf;
00258             ssize_t ret = ::recv(this->__sock_fd, &peek_buf, 1, MSG_PEEK);
00259
00260             if (ret > 0) {
00261                 // Dados encontrados! Volta para bloqueante e lê normal
00262                 this->__set_blocking_mode(true);
00263                 this->receive();
00264                 break;
00265             }
00266             else if (ret == -1 && (errno == EAGAIN || errno == EWOULDBLOCK)) {
00267                 // Socket vazio: manda (syn) para os outros agentes para mantê-los vivos
00268                 for (auto* scom : other_players) {
00269                     scom->send_immediate("(syn)");
00270                 }
00271
00272                 for (auto* scom : other_players) {
00273                     scom->receive(); // Drena o buffer dos outros
00274                 }
00275
00276                 usleep(10000); // 10ms sleep
00277             }
00278             else {
00279                 // Erro real
00280                 break;
00281             }
00282         }
00283
00284         this->__set_blocking_mode(true); // Restaura
00285     }
00286
00291     void commit(const std::string& msg) {
00292         this->__message_queue += msg;
00293     }
00294
00299     void send() {
00300         if (this->__message_queue.empty()) {
00301             // Se vazio, apenas envia sync
00302             this->send_immediate("(syn)");
```

```
00303             return;
00304         }
00305
00306         // Adiciona syn
00307         this->__message_queue += "(syn)";
00308
00309         // Envia tudo de uma vez
00310         this->send_immediate(this->__message_queue);
00311
00312         // Limpa buffer mantendo capacidade reservada (rápido)
00313         this->__message_queue.clear();
00314     }
00315
00319     void clear_queue() {
00320         this->__message_queue.clear();
00321     }
00322 };
```

4.7 src/run_full_team.cpp File Reference

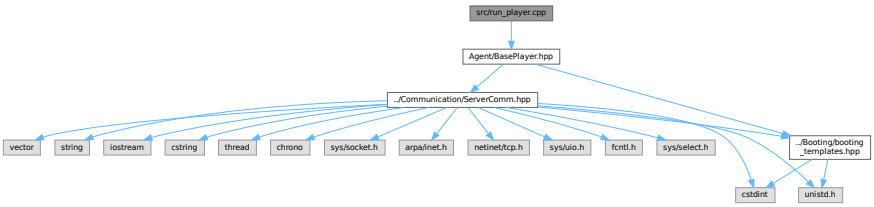
4.8 run_full_team.cpp

Go to the documentation of this file.

4.9 src/run_player.cpp File Reference

```
#include "Agent/BasePlayer.hpp"
```

Include dependency graph for run_player.cpp:



Functions

- int main ()

4.9.1 Function Documentation

4.9.1.1 main()

```
int main ( )
```

Definition at line 3 of file run_player.cpp.

4.10 run_player.cpp

Go to the documentation of this file.

```
00001 #include "Agent/BasePlayer.hpp"
00002
00003 int main() {
00004
00005     BasePlayer p = BasePlayer(1);
00006
00007
00008
00009     return 0;
00010 }
```

Index