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

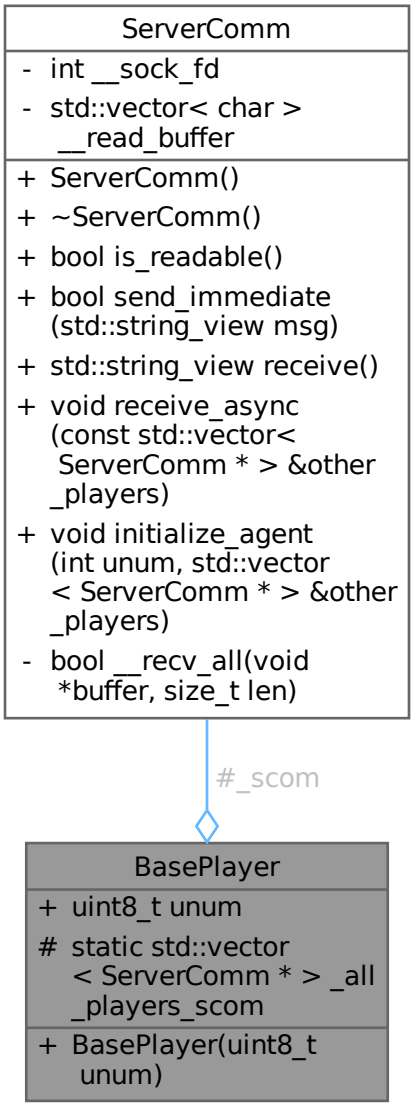
Class Documentation

3.1 BasePlayer Class Reference

Representa a entidade básica de um jogador na simulação.

`#include <BasePlayer.hpp>`

Collaboration diagram for BasePlayer:



Public Member Functions

- BasePlayer (uint8_t unum)

  Construtor: Inicializa o jogador e estabelece conexão com o servidor.

Public Attributes

- uint8_t unum

  Número do uniforme do jogador.

Protected Attributes

- ServerComm _scom

  Gerenciador de comunicação com o servidor rcssserver3d.

Static Protected Attributes

- static std::vector< ServerComm * > _all_players_scom

  Lista estática compartilhada contendo ponteiros para os comunicadores de todos os jogadores.

3.1.1 Detailed Description

Representa a entidade básica de um jogador na simulação.

Definition at line 10 of file BasePlayer.hpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BasePlayer()

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

Construtor: Inicializa o jogador e estabelece conexão com o servidor.

Realiza a reserva de memória no vetor estático, define o número do uniforme, executa o protocolo de handshake e registra o comunicador deste jogador na lista global.

Parameters

unum	Número do uniforme desejado para o agente (1 a 11).

-

Definition at line 41 of file BasePlayer.hpp.

3.1.3 Member Data Documentation

3.1.3.1 _all_players_scom

```
std::vector<ServerComm*> BasePlayer::_all_players_scom  [inline], [static], [protected]
```

Lista estática compartilhada contendo ponteiros para os comunicadores de todos os jogadores.

Usada para passar a referência dos "outros jogadores" durante a inicialização e sincronização (Keep-Alive).

Definition at line 24 of file BasePlayer.hpp.

3.1.3.2 _scom

`ServerComm BasePlayer::_scom [protected]`

Gerenciador de comunicação com o servidor rcssserver3d.

Instanciado automaticamente na criação do jogador. É responsável por enviar comandos e receber estados do jogo via TCP.

Definition at line 17 of file BasePlayer.hpp.

3.1.3.3 unum

`uint8_t BasePlayer::unum`

Número do uniforme do jogador.

Tipo `uint8_t` utilizado para otimização de memória, já que o valor varia apenas de 1 a 11.

Definition at line 31 of file BasePlayer.hpp.

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

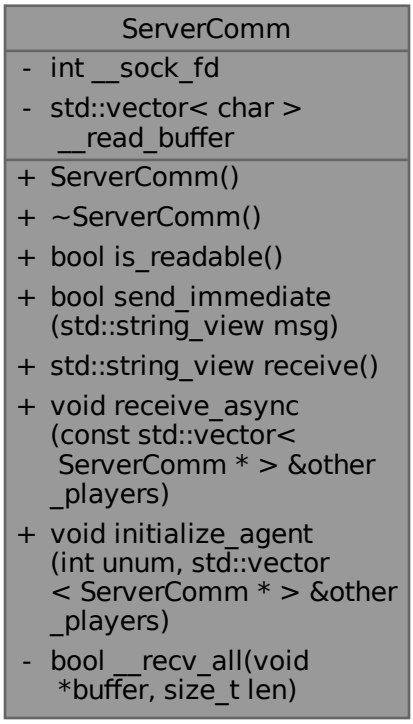
- src/Agent/BasePlayer.hpp

3.2 ServerComm Class Reference

Gerencia a comunicação TCP de baixo nível com o servidor rcssserver3d.

`#include <ServerComm.hpp>`

Collaboration diagram for ServerComm:



Public Member Functions

- ServerComm ()
  Construtor: Inicializa socket, buffers e configurações de rede.
- ~ServerComm ()
  Destrutor: Garante o fechamento correto do socket.
- bool is_readable ()
  Verifica se há dados prontos para leitura no Kernel.
- bool send_immediate (std::string_view msg)
  Envia uma mensagem imediatamente utilizando Scatter/Gather I/O.
- std::string_view receive ()
  Lê uma mensagem completa do servidor.
- void receive_async (const std::vector< ServerComm * > &other_players)
  Aguarda resposta do servidor mantendo os outros agentes vivos (Keep-Alive).
- void initialize_agent (int unum, std::vector< ServerComm * > &other_players)
  Realiza o handshake inicial do agente (Scene, Init e Sincronização).

Private Member Functions

- bool __recv_all (void *buffer, size_t len)
  Tenta ler exatamente N bytes do socket.

Private Attributes

- int __sock_fd
  Descritor de arquivo do socket.
- std::vector< char > __read_buffer
  Buffer persistente para leitura (evita realocações frequentes)

3.2.1 Detailed Description

Gerencia a comunicação TCP de baixo nível com o servidor rcssserver3d.

Implementa estratégias de buffering, leitura não-bloqueante segura (polling) e envio otimizado via writev.

Definition at line 30 of file ServerComm.hpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ServerComm()

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

Construtor: Inicializa socket, buffers e configurações de rede.

Configura TCP_NODELAY para baixa latência e SO_RCVTIMEO para evitar deadlocks.

Definition at line 82 of file ServerComm.hpp.

3.2.2.2 ~ServerComm()

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

Destrutor: Garante o fechamento correto do socket.

Definition at line 146 of file ServerComm.hpp.

3.2.3 Member Function Documentation

3.2.3.1 __recv_all()

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

Tenta ler exatamente N bytes do socket.

Parameters

buffer	Ponteiro para o destino dos dados.
len	Quantidade de bytes a serem lidos.

Returns

True se leu todos os bytes com sucesso.

False se houve erro, timeout ou fechamento da conexão (EOF).

Definition at line 46 of file ServerComm.hpp.

3.2.3.2 initialize_agent()

```
void ServerComm::initialize_agent (
            int unum,
            std::vector< ServerComm * > & other_players )  [inline]
```

Realiza o handshake inicial do agente (Scene, Init e Sincronização).

Parameters

unum	Número do uniforme do jogador.
other_players	Referência para lista de outros jogadores para sincronização.

Definition at line 325 of file ServerComm.hpp.

3.2.3.3 is_readable()

```
bool ServerComm::is_readable ( )  [inline]
```

Verifica se há dados prontos para leitura no Kernel.

Utiliza select com timeout 0 (polling) para não bloquear a thread.

Returns

True se houver bytes para ler, False caso contrário.

Definition at line 155 of file ServerComm.hpp.

3.2.3.4 receive()

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

Lê uma mensagem completa do servidor.

Implementa estratégia de "Drenagem": Lê todas as mensagens disponíveis e retorna apenas a mais recente para evitar lag acumulado.

Returns

std::string_view apontando para o buffer interno contendo a mensagem. Vazio se erro/timeout.

Definition at line 242 of file ServerComm.hpp.

3.2.3.5 receive_async()

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

Aguarda resposta do servidor mantendo os outros agentes vivos (Keep-Alive).

Realiza polling neste socket. Se não houver dados, envia (syn) para os parceiros e drena a leitura deles para evitar buffer overflow.

Parameters

other_players	Lista de ponteiros para os comunicadores dos outros jogadores.

Definition at line 289 of file ServerComm.hpp.

3.2.3.6 send_immediate()

```
bool ServerComm::send_immediate (
            std::string_view msg )  [inline]
```

Envia uma mensagem imediatamente utilizando Scatter/Gather I/O.

Constrói o cabeçalho de 4 bytes e envia junto com o corpo em uma única syscall (ou loop de syscalls), garantindo integridade mesmo em caso de escritas parciais.

Parameters

msg	A mensagem a ser enviada (string_view evita cópias).

Returns

True se enviado com sucesso, False em caso de erro fatal.

Definition at line 180 of file ServerComm.hpp.

3.2.4 Member Data Documentation

3.2.4.1 __read_buffer

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

Buffer persistente para leitura (evita realocações frequentes)

Definition at line 35 of file ServerComm.hpp.

3.2.4.2 __sock_fd

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

Descritor de arquivo do socket.

Definition at line 33 of file ServerComm.hpp.

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

- src/Communication/ServerComm.hpp

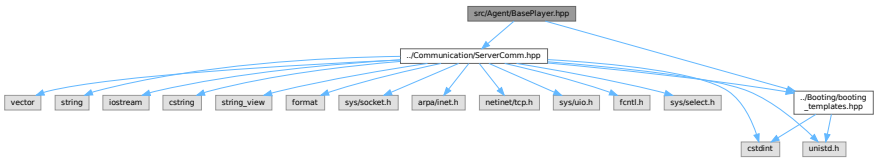
Chapter 4

File Documentation

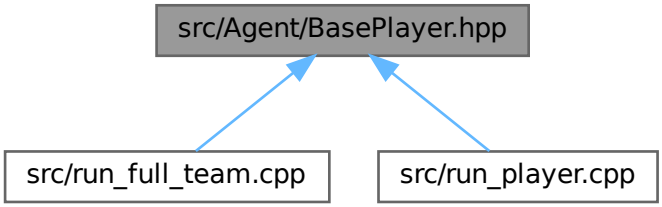
4.1 src/Agent/BasePlayer.hpp File Reference

```
#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

  Representa a entidade básica de um jogador na simulação.

4.2 BasePlayer.hpp

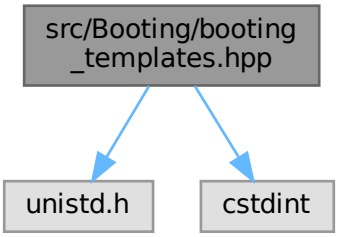
Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "../Booting/booting_templates.hpp"
00004 #include "../Communication/ServerComm.hpp"
00005
00010 class BasePlayer {
00011 protected:
00017     ServerComm _scom;
00018
00024     inline static std::vector<ServerComm*> _all_players_scom;
00025
00026 public:
00031     uint8_t unum;
00032
00033 public:
00041     BasePlayer(
00042         uint8_t unum
00043     ) {
00044         // Otimização: Evita múltiplas realocações do vetor de ponteiros
00045         if(BasePlayer::_all_players_scom.capacity() < 11){
00046             BasePlayer::_all_players_scom.reserve(11);
00047         }
00048
00049         this->unum = unum;
00050
00051         // Inicializa a conexão passando a lista atual de parceiros para sincronia
00052         this->_scom.initialize_agent(
00053             unum,
00054             BasePlayer::_all_players_scom
00055         );
00056
00057         // Registra o comunicador deste jogador na lista estática para os próximos agentes
00058         BasePlayer::_all_players_scom.emplace_back(&this->_scom);
00059     }
00060 };
```

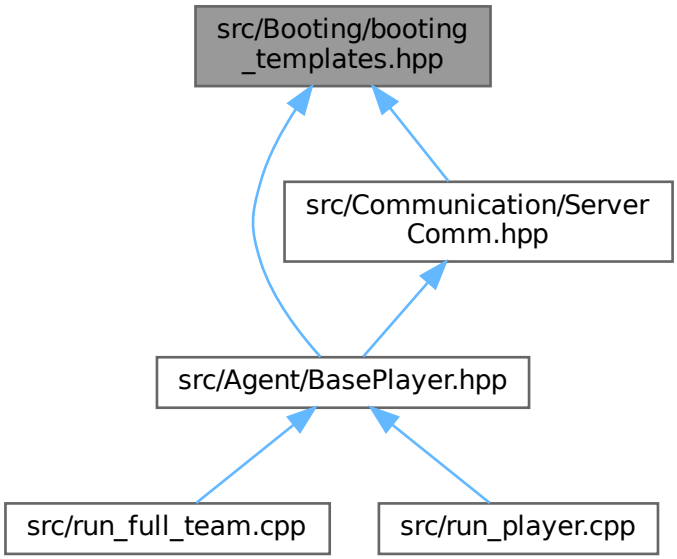
4.3 src/Booting/booting_templates.hpp File Reference

```
#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

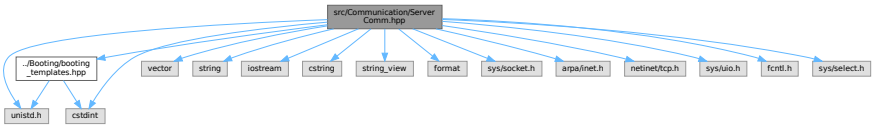
Go to the documentation of this file.

```
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
```

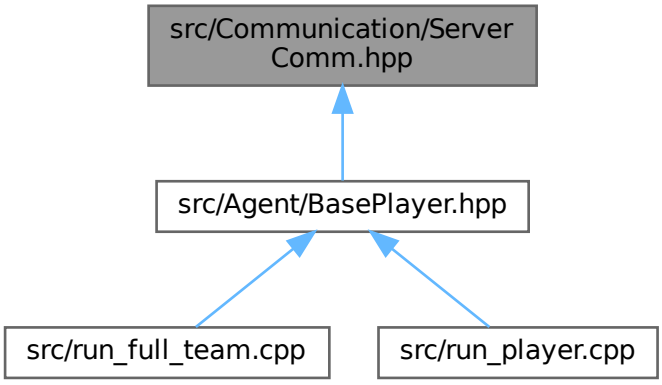
4.5 src/Communication/ServerComm.hpp File Reference

```
#include "../Booting/booting_templates.hpp"
#include <vector>
#include <string>
#include <iostream>
#include <cstring>
#include <cstdint>
#include <string_view>
#include <format>
#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

  Gerencia a comunicação TCP de baixo nível com o servidor rcssserver3d.

4.6 ServerComm.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include "../Booting/booting_templates.hpp"
00004
00005 // --- Bibliotecas da Standard Library ---
00006 #include <vector>
00007 #include <string>
00008 #include <iostream>
00009 #include <cstring>
00010 #include <cstdint>
00011 #include <string_view>
00012 #include <format>
00013
00014 // --- Bibliotecas de Sistema (POSIX) ---
00015 #include <sys/socket.h>
00016 #include <arpa/inet.h>
00017 #include <netinet/tcp.h>
00018 #include <unistd.h>
00019 #include <sys/uio.h>
00020 #include <fcntl.h>
00021 #include <sys/select.h>
00022
00023
00030 class ServerComm {
00031 private:
00033     int __sock_fd;
00035     std::vector<char> __read_buffer;
00036
00037     // Pode ser que precisemos implementar um buffer de envio.
00038
00046     bool __recv_all(
00047         void* buffer,
00048         size_t len
00049     ) {
00050         size_t total_read = 0;
00051         char* ptr = static_cast<char*>(buffer);
00052
00053         while(total_read < len){
00054             ssize_t bytes = ::recv(
00055                 this->__sock_fd,
00056                 ptr + total_read,
00057                 len - total_read,
00058                 0
00059             );
00060
00061             if(bytes > 0){
00062                 total_read += bytes;
00063             }
00064             else if(bytes == 0){
00065                 return False; // EOF (Servidor fechou)
00066             }
00067             else {
00068                 if(errno == EINTR){ continue; }
00069                 // Timeout do socket (SO_RCVTIMEO) configurado no construtor
00070                 if(errno == EAGAIN || errno == EWOULDBLOCK){ return False; }
00071                 return False; // Erro fatal
00072             }
00073         }
00074         return True;
00075     }
00076
00077 public:
00082     ServerComm() {
00083         // Ajuste para 64KB (mensagens de visão podem ser grandes)
00084         this->__read_buffer.resize(65536);
00085
00086         this->__sock_fd = socket(
00087             AF_INET,
00088             SOCK_STREAM,
00089             0
00090         );
00091
00092         if(this->__sock_fd < 0) {
00093             std::cerr « "Erro fatal: Socket falhou." « std::endl;
00094             exit(1);
00095         }
00096
00097         // 1. TCP_NODELAY (Performance: envia pacotes pequenos imediatamente)
00098         int flag = 1;
00099         setsockopt(
00100             this->__sock_fd,
00101             IPPROTO_TCP,
```

```
00102             TCP_NODELAY,
00103             (char*)&flag,
00104             sizeof(int)
00105         );
00106
00107         // 2. Timeout de Recebimento (Segurança: evita travamento eterno na leitura)
00108         struct timeval tv = {2, 0}; // 2 segundos
00109         setsockopt(
00110             this->__sock_fd,
00111             SOL_SOCKET,
00112             SO_RCVTIMEO,
00113             (const char*)&tv,
00114             sizeof(tv)
00115         );
00116
00117         struct sockaddr_in serv_addr;
00118         std::memset(
00119             &serv_addr,
00120             0,
00121             sizeof(serv_addr)
00122         );
00123         serv_addr.sin_family = AF_INET;
00124         serv_addr.sin_port = htons(AGENT_PORT);
00125         inet_pton(
00126             AF_INET,
00127             AGENT_HOST,
00128             &serv_addr.sin_addr
00129         );
00130
00131         // Tentativa de conexão com espera ativa simples
00132         while(
00133             connect(
00134                 this->__sock_fd,
00135                 (struct sockaddr*)&serv_addr,
00136                 sizeof(serv_addr)
00137             ) != 0
00138         ){
00139             usleep(500000); // 0.5s wait
00140         }
00141     }
00142
00146     ~ServerComm() {
00147         if(this->__sock_fd >= 0) close(this->__sock_fd);
00148     }
00149
00155     bool is_readable() {
00156         fd_set readfds;
00157         FD_ZERO(&readfds);
00158         FD_SET(
00159             this->__sock_fd,
00160             &readfds
00161         );
00162         struct timeval tv = {0, 0}; // Retorno imediato
00163
00164         return select(
00165             this->__sock_fd + 1,
00166             &readfds,
00167             NULL,
00168             NULL,
00169             &tv
00170         ) > 0;
00171     }
00172
00180     bool send_immediate(
00181         std::string_view msg
00182     ) {
00183         if(msg.empty()){ return True; }
00184
00185         uint32_t msg_len_host = static_cast<uint32_t>(msg.size());
00186         uint32_t msg_len_net = htonl(msg_len_host);
00187
00188         struct iovec iov[2];
00189         size_t total_to_send = 4 + msg_len_host;
00190         size_t total_sent = 0;
00191
00192         char* header_ptr = reinterpret_cast<char*>(&msg_len_net);
00193         const char* body_ptr = msg.data();
00194
00195         while(total_sent < total_to_send){
00196             int iov_cnt = 0;
00197
00198             if(total_sent < 4){
00199                 // Parte 1: Cabeçalho ainda não foi totalmente enviado
00200                 iov[iov_cnt].iov_base = header_ptr + total_sent;
00201                 iov[iov_cnt].iov_len  = 4 - total_sent;
00202                 iov_cnt++;
00203
```

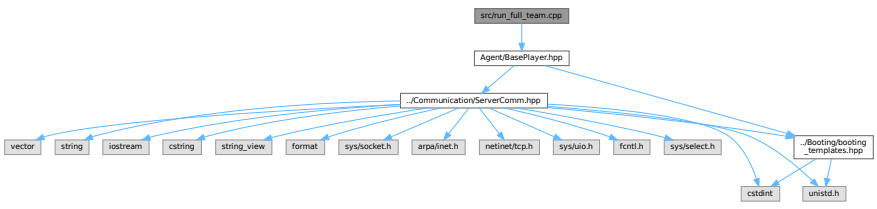
```
00204                 // Parte 2: Corpo inteiro ainda precisa ir
00205                 iov[iov_cnt].iov_base = (void*)body_ptr;
00206                 iov[iov_cnt].iov_len  = msg_len_host;
00207                 iov_cnt++;
00208             }
00209             else{
00210                 // Parte 1 já foi, enviando apenas o restante do corpo
00211                 size_t body_offset = total_sent - 4;
00212                 iov[iov_cnt].iov_base = (void*)(body_ptr + body_offset);
00213                 iov[iov_cnt].iov_len  = msg_len_host - body_offset;
00214                 iov_cnt++;
00215             }
00216 
00217             ssize_t res = ::writev(
00218                 this->__sock_fd,
00219                 iov,
00220                 iov_cnt
00221             );
00222 
00223             if(res > 0){ total_sent += res; }
00224             else if(res < 0){
00225                 if(errno == EINTR){ continue; }
00226                 if(errno == EAGAIN || errno == EWOULDBLOCK) {
00227                     usleep(1000); // Backoff curto para não fritar CPU
00228                     continue;
00229                 }
00230                 return False; // Erro real
00231             }
00232         }
00233         return True;
00234     }
00235 
00242     std::string_view receive() {
00243         uint32_t last_msg_size = 0;
00244 
00245         while(True) {
00246             uint32_t net_len = 0;
00247 
00248             // Tenta ler o cabeçalho (4 bytes)
00249             if(
00250                 !this->__recv_all(
00251                     &net_len,
00252                     4
00253                 )
00254             ){ break; }
00255 
00256             uint32_t msg_len = ntohl(net_len);
00257 
00258             // Tenta ler o corpo da mensagem
00259             if(
00260                 !this->__recv_all(
00261                     this->__read_buffer.data(),
00262                     msg_len
00263                 )
00264             ){ break; }
00265 
00266             last_msg_size = msg_len;
00267 
00268             // Estratégia de Drenagem: Se não há mais dados pendentes no Kernel,
00269             // paramos aqui e retornamos o que temos.
00270             if(!this->is_readable()){ break; }
00271         }
00272 
00273         if(last_msg_size > 0){
00274             this->__read_buffer[last_msg_size] = '\0'; // Null-terminate por segurança
00275             return std::string_view(
00276                 this->__read_buffer.data(),
00277                 last_msg_size
00278             );
00279         }
00280         return {};
00281     }
00282 
00289     void receive_async(
00290         const std::vector<ServerComm*>& other_players
00291     ) {
00292         // Se não houver ninguém, apenas lê (pode bloquear por até 2s no timeout configurado)
00293         if(other_players.empty()){
00294             this->receive();
00295             return;
00296         }
00297 
00298         while(True){
00299             // 1. Se EU tenho dados, leio e saio imediatamente.
00300             if(this->is_readable()){
00301                 this->receive();
00302                 break;
```

```
00303             }
00304
00305             // 2. Mantenho os outros vivos enquanto espero
00306             for(auto* p : other_players){
00307                 p->send_immediate("(syn)");
00308
00309                 // Drena buffer dos outros SE houver dados
00310                 if(p->is_readable()) {
00311                     p->receive();
00312                 }
00313             }
00314
00315             // Yield para a CPU (1ms) para evitar uso de 100% em busy wait
00316             usleep(1000);
00317         }
00318     }
00319
00325     void initialize_agent(
00326         int unum,
00327         std::vector<ServerComm*>& other_players
00328     ) {
00329         // Scene: Define o modelo do corpo do robô
00330         this->send_immediate(
00331             std::format(
00332                 "(scene rsg/agent/nao/nao_hetero.rsg {})",
00333                 (unum <= 1) ? 0 :
00334                 (unum <= 4) ? 1 :
00335                 (unum == 5) ? 2 :
00336                 (unum <= 8) ? 3 : 4
00337             )
00338         );
00339         this->receive_async(other_players);
00340
00341         // Init: Define time e número
00342         this->send_immediate(
00343             std::format(
00344                 "(init (unum {}) (teamname {}))",
00345                 unum,
00346                 TEAM_NAME
00347             )
00348         );
00349         this->receive_async(other_players);
00350
00351         // Sync Loop: Garante que todos entrem no ciclo de simulação juntos
00352         for(int i = 0; i < 3; ++i){
00353             this->send_immediate("(syn)");
00354
00355             for(auto* p : other_players){
00356                 p->send_immediate("(syn)");
00357             }
00358
00359             // Drena outros sem travar
00360             for(auto* p : other_players) {
00361                 if(p->is_readable()){ p->receive(); }
00362             }
00363
00364             if(this->is_readable()){ this->receive(); }
00365         }
00366     }
00367 };
```

4.7 src/run_full_team.cpp File Reference

`#include "Agent/BasePlayer.hpp"`

Include dependency graph for run_full_team.cpp:



Functions

- int main ()

4.7.1 Function Documentation

4.7.1.1 main()

```
int main ( )
```

Definition at line 3 of file run_full_team.cpp.

4.8 run_full_team.cpp

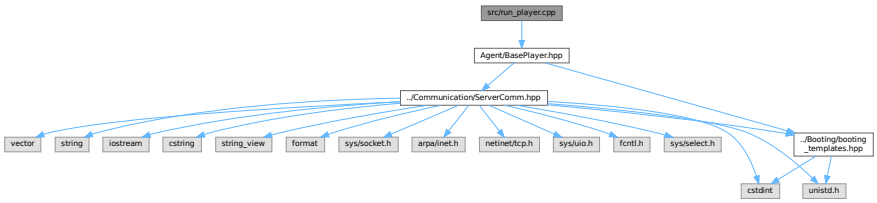
Go to the documentation of this file.

```
00001 #include "Agent/BasePlayer.hpp"
00002
00003 int main() {
00004
00005     std::vector<BasePlayer> players;
00006     players.reserve(11);
00007     for(
00008         int i = 1;
00009         i <= 11;
00010         i++
00011     ){
00012         players.emplace_back(i);
00013     }
00014
00015     usleep(50000000);
00016
00017
00018
00019     return 0;
00020 }
```

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     usleep(50000000);
00008
00009     return 0;
00010 }
```

Index