# Trabalho Final – Redes de Computadores

Nomes dos alunos: Leonardo Antonietti, Rodrigo Lopes.

Matrículas: 19103405, 22100655.

1. **Objetivo/Função da Aplicação**

Nosso objetivo foi recriar um jogo do clássico Pong, com opções de criação de sala, se juntar a sala que é protegida com senha. Além disto o jogo em sua essência exige respostas em tempo real e rápida análise do jogador.

**2. Características do Protocolo Desenvolvido**

* **Arquitetura:** Cliente-Servidor.
* **Modelo de Conexão:** Sem estado.
* **Persistência:** Persistente.
* **Modo de Comunicação:** *Pull.*
* **Controle:** Na banda.
* **Segurança:** Usamos um mecanismo de criação de sala e senha restritiva**.**

**3. Tipos de Mensagens do Cliente para o Servidor**

|  |  |
| --- | --- |
| **Tipo de Mensagem** | **Descrição** |
| **CREATE\_GAME** | **Solicita para o servidor a criação de uma sala.** |
| **JOIN\_GAME** | **Solicita entrada em uma nova partida** |
| **UPDATE\_BLOCK** | **Envia a posição atual do jogador.** |
| **FINISH\_CONNECTION** | **Enviada quando o jogador sai do jogo** |

**4. Tipos de Mensagens do Servidor para o Cliente**

|  |  |
| --- | --- |
| **Tipo de Mensagem** | **Descrição** |
| **GAME\_START** | **Notificação de início da partida e distribuição dos identificadores de cada jogador** |
| **GAME\_STATE** | **Estado atualizado do jogo após cada jogada** |
| **GAME\_OVER** | **Notificação de fim de partida** |
| **GAME\_STATUS** | **Notifica o cliente se as ações do usuário são validas/ foram concluidas.** |

**5. Formato das Mensagens e Campos de Cabeçalho**

**Formato geral das mensagens (em json):**

* **type: Tipo de mensagem (ver tabela acima).**
* **command: Tipo de mensagem (ver tabela acima).**

**CREATE\_GAME:**

* **game\_id: nome da sala.**
* **password: senha da sala.**

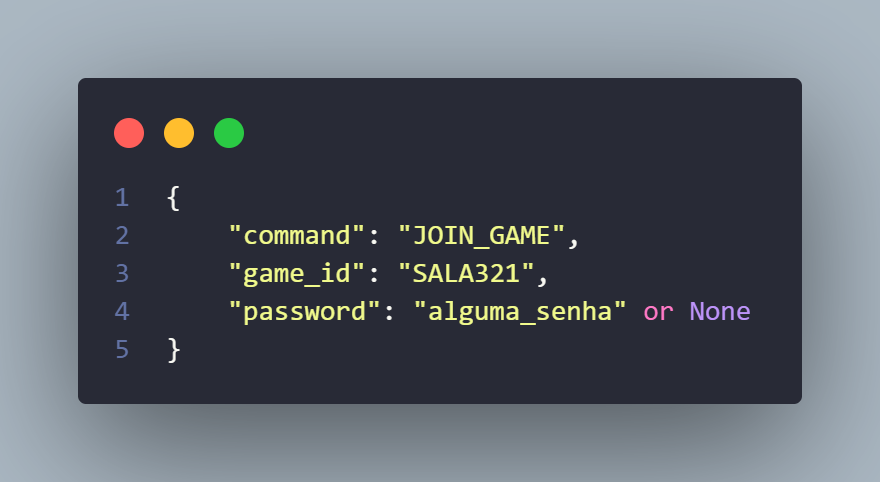
Exemplo: mensagem do tipo **GAME\_START**:

****

**JOIN\_GAME:**

* **game\_id: nome da sala.**
* **password: senha da sala.**

Exemplo: mensagem do tipo **JOIN\_GAME**:

****

**UPDATE\_BLOCK:**

* **y: posição do jogador.**

Exemplo: mensagem do tipo **UPDATE\_BLOCK**:

****

**FINISH\_CONNECTION:**

Exemplo: mensagem do tipo **FINISH\_CONNECTION**:

****

**GAME\_START:**

* **player\_id: define o id do jogador dentro da sala.**

Exemplo: mensagem do tipo **GAME\_START**:

****

**GAME\_STATE:**

* **ball: define a posição da bola.**
* **player1\_y: define a posição do jogador com ID 1.**
* **player2\_y: define a posição do jogador com ID 2.**
* **score: define o placar do jogo.**

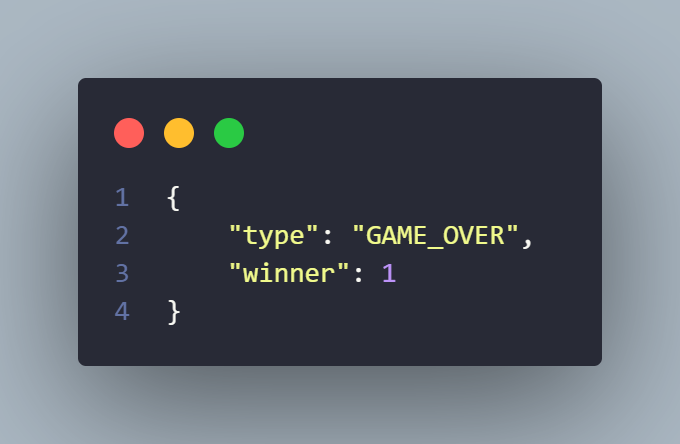
Exemplo: mensagem do tipo **GAME\_STATE**:



**GAME\_OVER:**

* **winner: define qual jogador venceu.**

Exemplo: mensagem do tipo **GAME\_OVER**:

****

**GAME\_STATUS:**

* **status: define o status da requisição.**
* **message: contém informações sobre o erro ou sucesso da requisição.**

Exemplo: mensagem do tipo **GAME\_STATUS**:

****

**6. Especificação dos Campos**

|  |  |  |
| --- | --- | --- |
| **Campo** | **Tipo** | **Valores possíveis** |
| **type** | **string** | **GAME\_START, GAME\_STATE, GAME\_OVER, GAME\_STATUS** |
| **command** | **string** | **CREATE\_GAME, JOIN\_GAME, UPDATE\_BLOCK, FINISH\_CONNECTION** |
| **game\_id** | **String** | **Identificador da sala (Ex. “sala123”)** |
| **password** | **String** | **Senha da sala (Ex. “admin”)** |
| **player\_id** | **Integer** | **0 ou 1** |
| **ball** | **Tupla[Integer, Integer]** | **Valores inteiros** |
| **player1\_y** | **Integer** | **Valores inteiros** |
| **player2\_y** | **Integer** | **Valores inteiros** |
| **score** | **Tupla[Integer, Integer]** | **Valores pertencentes a [0, 5].** |
| **winner** | **Integer** | **Jogador vencedor, 0 ou 1.** |
| **status** | **String** | **“SUCCESS” ou “ERROR”** |
| **message** | **String** | **Um texto qualquer.** |

7. Outras informações

* Casos de uso:
  + Jogador inicia a aplicação, informa os dados da partida, seja para entrar ou criar uma partida, enviando “CREATE\_GAME” OU “JOIN\_GAME”, respectivamente.
  + Após os dois jogadores entrarem na sala, o servidor inicia a partida enviando o comando “GAME\_START”.
  + A todo momento o servidor e os jogadores trocam mensagem para atualizar o estado do jogo e a posição dos jogadores.
* API’s utilizadas: Python socket, json, threading, pygame, watchdog, dotenv, subprocess, time, sys, os.
  + Um arquivo **requirements.txt** está junto aos aquivos do jogo para garantir a compatibilidade das bibliotecas.
  + Para instalar, execute dentro da pasta com os arquivos: **pip install -r requirements.txt**.
* Para executar a aplicação:
  1. **python server.py**, ou **python run\_server.py** para facilitar a re-execução do servidor em caso de alterações.
  2. **python client.py** para executar o cliente.
* Dificuldades encontradas:
  + Gerenciamento de múltiplos jogos simultâneos.
  + Sincronização de jogadas em partidas paralelas.