



**Universidade Federal  
de São João del-Rei**

Departamento do Curso de Ciência da Computação  
Programa de Graduação

## **Documentação do Primeiro Trabalho Prático de Redes de Computadores**

Alunos: João Vítor Gonçalves, Paulo Tobias  
Professor: Rafael Sachetto Oliveira

Outubro  
2018

## Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Regras do Jogo . . . . .	1
<b>2</b>	<b>Implementação</b>	<b>4</b>
2.1	Baralho . . . . .	5
2.2	Truco . . . . .	6
2.3	Protocolo . . . . .	6
2.3.1	Tipo da Mensagem . . . . .	7
2.4	Servidor . . . . .	8
2.5	Cliente . . . . .	9
2.6	Truco Servidor . . . . .	9
2.7	Truco Cliente . . . . .	9
<b>3</b>	<b>Discussões</b>	<b>10</b>

# 1 Introdução

O objetivo deste trabalho prático é implementar um jogo online de truco (com quatro jogadores). A implementação foi realizada na linguagem C e o protocolo utilizado para a troca de pacotes entre máquinas foi o TCP. A porta padrão utilizada pela aplicação é a porta 8080. As regras do jogo adotadas foram as mesmas do chamado "truco mineiro" e foram baseadas nas regras disponíveis em [1].

## 1.1 Regras do Jogo

A seguir segue o ranking de cartas (começando pela mais forte e terminando pela mais fraca):

- 4 de paus (Zap)
- 7 de copas (7copas)
- Ás de espada (Espadilha)
- 7 de ouros (7ouros)
- Coringa
- 3 (Todos)
- 2 (Todos).
- Ás (Todos exceto o ás de espadas)
- Reis (todos)
- Valetes (todos)
- Damas (todos)
- 7 (todos exceto 7 de copas e 7 de ouros)
- 6 (todos)
- 5 (todos)
- 4 (todos exceto 4 de paus).

Com o objetivo de facilitar a explicação das regras, será utilizada a seguinte notação: existem duas equipes, A e B, cada uma com um jogador (A1 e A2 contra B1 e B2). As rodadas inicialmente seguem a seguinte ordem global: A1,B1,A2,B2. O primeiro jogador de uma partida sempre é o próximo jogador da ordem global em relação ao primeiro jogador da partida anterior. Segue um exemplo abaixo.

- Partida 1, Rodada 1 - A1,B1,A2,B2.

- Partida 2, Rodada 1 - B1,A2,B2,A1.

Para vencer no truco mineiro, é necessário vencer uma queda. A queda se baseia em vencer dois jogos (isso é chamado de *melhor de três*). Os jogos são compostos de partidas e cada partida possui 3 rodadas. O jogo acaba quando uma das equipes conseguem um total de 12 pontos. Quando um jogador vence uma rodada, a próxima rodada será iniciada à partir dele. Em condições normais, o vencedor de uma partida ganha 2 pontos para o seu time, entretanto, os jogadores podem realizar o "truco", movimento em que um jogador aumenta o valor da partida em dois pontos. O jogador da outra equipe possui três opções quando é "*trucado*": aceitar o truco, recusar o truco ou aumentar a aposta novamente.

Segundo à regra oficial, um jogador só é autorizado à pedir truco para o próximo jogador em relação à ordem global (por exemplo: o jogador B2 só pode pedir truco para A1), da mesma forma, somente o jogador *trucado* poderá tomar uma decisão em relação ao truco. Entretanto, a regra adotada na implementação deste trabalho em relação à *pedir truco* seria: quando um jogador pede truco na sua vez, ambos os jogadores da equipe adversária devem responder ao pedido do truco (e ambos devem responder a mesma coisa). Se os jogadores aumentarem a aposta, então o jogador que "trucou" é identificado como o jogador que tem a vez ou o jogador à sua direita. Por exemplo, se o jogador B2 pede truco na sua vez de jogar e os jogadores do time A decidem por aumentar a aposta, então é avisado para os jogadores que o jogador A1 aumentou a aposta. E, caso os jogadores do time B aumentem novamente a aposta, então é avisado para os jogadores que B2 (o jogador original) aumentou a aposta.

A seguir estão os nomes dos aumentos de aposta e suas respectivas pontuações para uma partida:

- Truco: 4 pontos.
- Seis: 6 pontos.
- Nove: 8 pontos.
- Queda: 12 pontos.

Quando há empate na primeira rodada de uma partida, é dito que a rodada foi "*amarrada*". Nesse caso, os jogadores devem mostrar a maior carta da mão atual de cada um. Quem tiver a maior carta ganha a partida. Em caso de empate da segunda rodada, este mesmo processo é repetido novamente para a última rodada. Se por acaso todas as rodadas acabaram em empate, a partida se torna um empate e nenhuma equipe pontua.

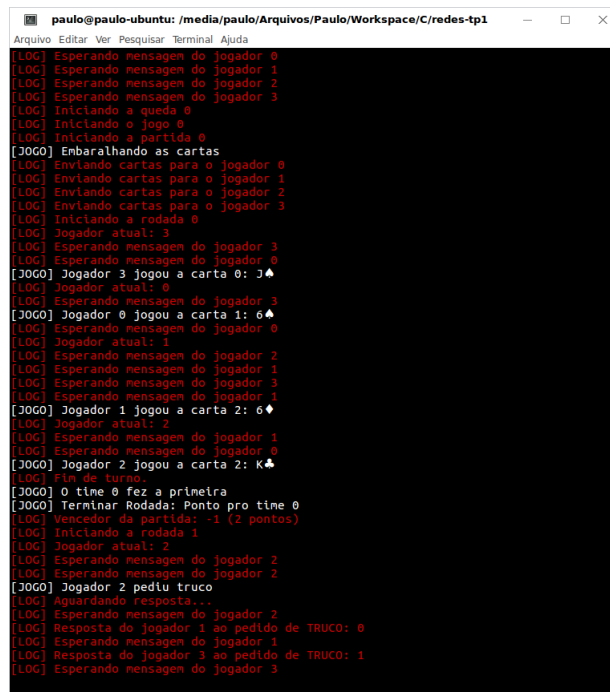
Quando apenas uma das equipe alcança 10 pontos, a queda irá para uma fase chamada "Mão de Dez", fase na qual nenhum jogador pode realizar um truco. Se a equipe com 10 pontos vencer uma partida, ela irá ganhar 2 pontos, resultando em uma vitória com 12 pontos. Entretanto, se a equipe adversária ganhar uma partida, essa equipe irá ganhar 4 pontos pela partida que venceu. Outro aspecto

dessa fase é que a equipe que está ganhando tem o poder de escolher se irá ou não jogar a partida após olharem suas próprias cartas, entretanto, eles não são autorizados à olharem a mão do seu parceiro. Se desistirem, a equipe adversária ganha apenas 2 pontos.

Quando as duas equipes possuem exatamente 10 pontos, a queda irá para uma fase chamada "Mão de Ferro". Nesta fase, os jogadores não podem ver as cartas que estão em sua mão, sendo obrigados à jogarem as cartas completamente na sorte.

## 2 Implementação

A aplicação funciona exatamente como um modelo cliente-servidor. O servidor é responsável por toda a organização da partida e toda comunicação é transmitida através dele. Os clientes enviam mensagens (quando são autorizados) para o servidor e o mesmo é responsável por fazer todo o processamento do jogo repassar o novo estado para os jogadores. Exemplo: O servidor envia uma mensagem para um dos jogadores indicando que é sua vez de jogar. Ao mesmo tempo envia mensagens aos demais jogadores informando-os de quem está jogando. Qualquer tentativa de interação por parte dos outros jogadores será ignorada. Quando o jogador autorizado fizer sua jogada, o servidor fará todas as verificações (se a carta é a mais forte da mesa; se houve empate; etc) e, após isso, enviará mensagens de atualização para todos os jogadores para que estes se mantenham atualizados.



```
paulo@paulo-ubuntu: /media/paulo/Arquivos/Paulo/Workspace/C/redes-tp1
Arquivo Editar Ver Pesquisar Terminal Ajuda
[LOG] Esperando mensagem do jogador 0
[LOG] Esperando mensagem do jogador 1
[LOG] Esperando mensagem do jogador 2
[LOG] Esperando mensagem do jogador 3
[LOG] Iniciando a queda 0
[LOG] Iniciando o jogo 0
[LOG] Iniciando a partida 0
[JOGO] Embaralhando as cartas
[LOG] Enviando cartas para o jogador 0
[LOG] Enviando cartas para o jogador 1
[LOG] Enviando cartas para o jogador 2
[LOG] Enviando cartas para o jogador 3
[LOG] Iniciando a rodada 0
[LOG] Jogador atual: 3
[LOG] Esperando mensagem do jogador 3
[LOG] Esperando mensagem do jogador 0
[JOGO] Jogador 3 jogou a carta 0: J♠
[LOG] Jogador atual: 0
[LOG] Esperando mensagem do jogador 3
[JOGO] Jogador 0 jogou a carta 1: 6♠
[LOG] Esperando mensagem do jogador 0
[LOG] Jogador atual: 1
[LOG] Esperando mensagem do jogador 2
[LOG] Esperando mensagem do jogador 1
[LOG] Esperando mensagem do jogador 3
[LOG] Esperando mensagem do jogador 1
[JOGO] Jogador 1 jogou a carta 2: 6♦
[LOG] Jogador atual: 2
[LOG] Esperando mensagem do jogador 1
[LOG] Esperando mensagem do jogador 0
[JOGO] Jogador 2 jogou a carta 2: K♠
[LOG] Fim de turno.
[JOGO] O time 0 fez a primeira
[JOGO] Terminar Rodada: Ponto pro time 0
[LOG] Vencedor da partida: -1 (2 pontos)
[LOG] Iniciando a rodada 1
[LOG] Jogador atual: 2
[LOG] Esperando mensagem do jogador 2
[LOG] Esperando mensagem do jogador 2
[JOGO] Jogador 2 pediu truco
[LOG] Aguardando resposta...
[LOG] Esperando mensagem do jogador 2
[LOG] Resposta do jogador 1 ao pedido de TRUCO: 0
[LOG] Esperando mensagem do jogador 1
[LOG] Resposta do jogador 3 ao pedido de TRUCO: 1
[LOG] Esperando mensagem do jogador 3
```

Figura 1: Logs do Servidor.

Além disso, também foi implementado um chat, onde os jogadores podem trocar mensagens de texto entre si a qualquer momento. Por conta disso, o uso de paralelismo no código se fez necessário, visto que um jogador não precisa esperar sua vez de jogar para enviar uma mensagem no chat.

Seguem abaixo as descrições das funções de implementação mais relevantes do trabalho prático.

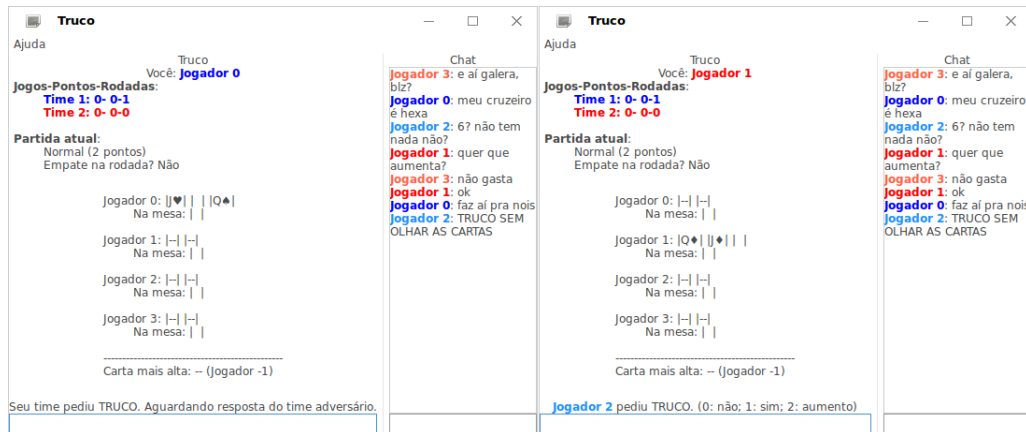


Figura 2: Interface do Jogo de Truco. Perspectiva dos Jogadores 0 e 1.

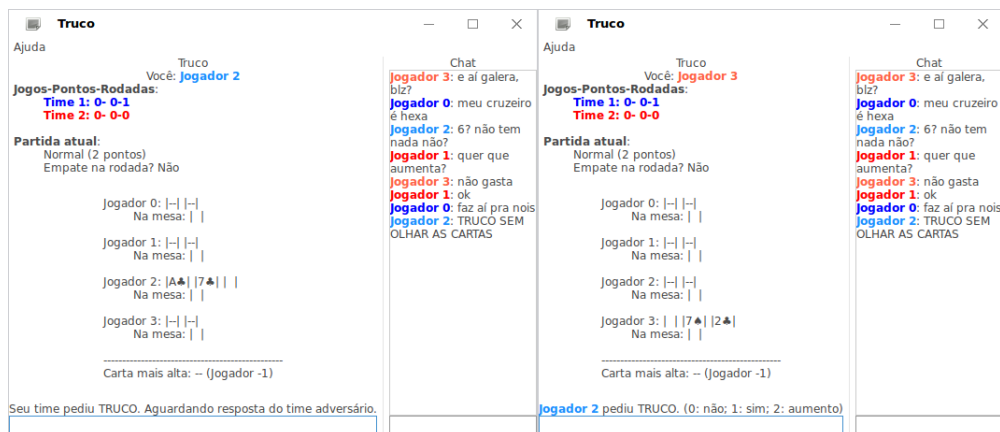


Figura 3: Perspectiva dos Jogadores 2 e 3.

## 2.1 Baralho

As implementações neste módulo são relacionadas à funções básicas de um baralho de cartas. Entre elas estão: embaralhar cartas, exibir o naipe da carta, virar uma carta e limpar atributos de carta. A *struct Carta* guarda informações do número, naipe e poder de uma carta. O poder é usado para comparar duas cartas. O poder do zap, por exemplo, é o mais alto de todos, seguido pelo do 7 de copas. O poder da 'carta virada' (uma carta que não é visível para os jogadores) é 0. Sendo assim, ela sempre será derrotada.

## 2.2 Truco

Módulo responsável por armazenar as informações sobre o jogo de truco. Possui funções para início/fim da rodada, partida e do jogo. As informações que precisam ser compartilhadas com os jogadores estão divididas nas *structs* *EstadoJogo* e *EstadoJogador*. Estas estruturas serão mais detalhadas na seção 2.3

- **FASE\_JOGO**: Enumera as possíveis fases do jogo, como *enviando cartas* ou *pediu truco*.
- **RESPOSTAS**: Enumera as possíveis respostas que um jogador pode dar, seja quando ele for trucoado ou se ele deseja jogar a mão de 10, etc.
- **gbaralho[NUM\_CARTAS]**: Vetor com todas as cartas presentes no truco.
- **gvencedor\_primeira\_rodada**, **gvencedor\_partida**, **gvencedor\_jogo** e **gvencedor\_queda**: Armazenam, respectivamente, o id do time vencedor da rodada, partida, jogo e queda.
- **gjogadores\_ativos**: Máscara de bits que armazena quais jogadores podem mandar mensagem.
- void **iniciar\_rodada()**: Reseta os valores para dar início a uma nova rodada.
- int8\_t **terminar\_rodada(int8\_t vencedor\_partida)**: Aumenta o número de rodadas vencidas pelo time do jogador que jogou a carta com maior poder. Caso um time tenha vencido duas rodadas, então este time será apontado como o vencedor da partida e ganhará os pontos. Também verifica se o time vencedor da partida entrou na Mão de 10.
- void **iniciar\_partida()**: Reseta os valores para dar início a uma nova partida.
- int8\_t **terminar\_partida()**: Chamada ao final de cada partida, reseta o número de rodadas vencidas por cada time e, caso algum deles tenha feito 12 ou mais pontos, os declara como vencedores do jogo.
- int8\_t **terminar\_jogo()**: Chamada ao final de cada jogo, reseta os pontos feitos pelos dois times e, caso algum time tenha vencido dois jogos, os declara como vencedores da queda.

## 2.3 Protocolo

Define a estrutura das mensagens que são trocadas entre servidor e cliente. Optamos por mensagens binárias, ao contrário de protocolos em texto plano, como é o caso do HTTP, pois a decodificação do conteúdo da mensagem é mais simples. Uma Mensagem possui as seguintes informações:



- **tipo:** Tipo da mensagem. Os tipos serão explicados a seguir.
- **atualizar\_estado\_jogo:** Valor lógico que indica se a mensagem contém dados do estado do jogo.
- **atualizar\_estado\_jogadores:** Valor lógico que indica se a mensagem contém dados do estado dos jogadores.
- **tamanho\_dados:** Tamanho em bytes de quaisquer dados extras que a mensagem possua.
- **estado\_jogo:** Dados do estado do jogo, como pontuação, se houve empate, qual foi o último time a pedir truco, qual time está na mão de 10, etc.
- **estado\_jogadores:** Informações necessárias sobre os jogadores, como a quantidade de cartas na mão e a última carta jogada.
- **dados:** Até 256 bytes de dados extras podem ser enviados, como, por exemplo, as mensagens do chat.

### 2.3.1 Tipo da Mensagem

O tipo das mensagens é representado por um número inteiro não-negativo. Cada tipo tem uma mensagem de texto associado que pode ser usado pelo cliente para exibí-la na tela, mas esta mensagem de texto não faz parte do conteúdo da mensagem. Portanto, alguns tipos de mensagem não possuem dados associados e servem somente como forma de informar melhor o jogador sobre o que está acontecendo. As mensagens podem ter diferentes significados quando lidas pelo cliente ou pelo servidor. Os diferentes tipos de mensagem são:

- **ERRO:** erro na aplicação.
- **BEM\_VINDO:** mensagem de boas-vindas ao cliente. Contém o id de jogador para que possa se identificar no jogo.
- **PROCESSANDO:** Normalmente usada para forçar uma atualização no estado ou para enviar alguma mensagem qualquer aos clientes.
- **ENVIANDO\_CARTAS:** A mensagem contém as cartas que o jogador usará na partida. Caso o jogo esteja na fase Mão de Ferro, as cartas já serão enviadas 'viradas' e o cliente não terá acesso ao conteúdo delas.
- **SEU\_TURN0:**
  - **Servidor - Cliente:** Informa ao jogador que é sua vez de realizar sua jogada. Não contém nenhum dado.
  - **Cliente - Servidor:** Informa ao servidor o índice da carta jogada. Se o índice for 0, então o jogador "trucou". Se for negativo, então a carta foi jogada no monte.

- **AGUARDANDO\_TURNO:** Informa o id do jogador que está jogando.
- **JOGADA\_ACEITA:** Informa ao cliente que a jogada foi válida e que ele pode atualizar remover a carta no índice indicado.
- **TRUCO:**
  - **Servidor - Cliente:** Enviada ao time adversário. Contém o id do jogador que pediu o truco para que o cliente mostre uma mensagem informativa na tela.
  - **Cliente - Servidor:** Enviada ao servidor quando um jogador adversário pede truco. Contém a resposta do jogador (correr, aceitar ou aumentar).
- **EMPATE:** Enviada quando a primeira rodada termina em empate. Não contém dados.
- **MAO\_DE\_10:**
  - **Servidor - Cliente:** Enviada para a equipe que está na Mão de Dez. Os jogadores podem escolher se irão jogar essa partida ou não após observarem suas cartas. Não contém dados.
  - **Cliente - Servidor:** Enviada ao servidor contendo a resposta do jogador sobre jogar a Mão de Dez ou não.
- **FIM\_QUEDA:**
  - **Servidor - Cliente:** Enviada quando o time vence uma queda. Contém o id do time vencedor e prepara o servidor para esperar a resposta dos jogadores.
  - **Cliente - Servidor:** Enviada ao servidor contendo a resposta do time sobre continuar jogando.
- **CHAT:**
  - **Servidor - Cliente:** Contém o texto – já formatado – que algum jogador enviou pelo chat. O texto é então enviado a todos os jogadores.
  - **Cliente - Servidor:** Envia para o servidor o texto –já formatado – que o jogador digitou no campo de texto *chat* para que o servidor o envie para todos os jogadores.

## 2.4 Servidor

Módulo responsável por gerenciar toda a comunicação com os jogadores. Contém informações sobre os descritores de arquivos dos *sockets* criados para cada um deles, as *threads* para recebimento das mensagens, além de outros dados de controle de concorrência para o paralelismo.

- struct **Jogador**: Armazena o id, o descritor do *socket* e a thread que recebe e processa as mensagens.
- void **jogador\_init(Jogador \*jogador, uint8\_t id, int sfd)**: Inicializa um novo Jogador e cria sua *thread* para recebimento das mensagens.
- void **\*t\_leitura(void \*args)**: Recebe e processa as mensagens enviadas pelo cliente.
- void **enviar\_mensagem(const Mensagem \*mensagem, uint8\_t new\_msg)**: Envia a mensagem para os destinatários corretos.
- int **avisar\_truco(int8\_t jogador\_id)**: Responsável por notificar todos os jogadores de que alguém truco, assim como lidar com as devidos procedimentos de um truco.

## 2.5 Cliente

Módulo responsável por receber, processar e enviar mensagens para o servidor.

- void **\*t\_receive(void \*)**: Responsável por receber mensagens que estão sendo enviadas pelo servidor. As informações na tela são atualizadas de acordo com o conteúdo da mensagem.
- void **t\_send(GtkEntry \*entry, gpointer user\_data)**: Converte o comando em texto digitado pelo usuário na interface gráfica em uma mensagem esperada pelo servidor. Tem uma certa autonomia para recusar o envio de mensagens consideradas inválidas, como tentar enviar a carta de número 4.

## 2.6 Truco Servidor

Código principal que será executado pelo servidor da aplicação. Ele é responsável por fazer as conexões com os clientes e gerenciar toda a lógica e sequência de uma queda de truco. Ele irá exibir o estado atual do jogo para todos os clientes, sendo assim, todos estão cientes do que está acontecendo no jogo e poderão realizar seus movimentos na rodada atual ou aguardar o jogador da vez fazer sua jogada.

## 2.7 Truco Cliente

Responsável por se conectar ao servidor. Também é responsável pela instância da interface gráfica do jogo para que o cliente possa utilizar o programa corretamente.

### 3 Discussões

A aplicação apresenta algumas formas de manter a segurança e evitar a trapaça de clientes mal-intencionados. São elas:

- **Não-uso de identificadores por parte do cliente:** Os jogadores são identificados por números inteiros que variam de 0 a 3. Uma forma de comprometer o estado do jogo é fazer com que um jogador use o *id* de outro para fazer jogadas indevidas. Porém, o servidor mantém uma *thread* exclusiva para o recebimento das mensagens para cada jogador. Portanto, esse tipo de ação não pode ser feito de maneira trivial.
- **Uso de variáveis de controle:** O servidor possui duas variáveis de controle para poder saber quem pode enviar que tipo de mensagem a cada momento do jogo: *fase* e *jogadores\_ativos*. A primeira indica qual o momento atual do jogo. A segunda indica quais jogadores podem enviar mensagens nesta fase. Portanto, se um jogador mal-intencionado envia uma mensagem para o servidor do tipo *pedir truco* mas o servidor está na fase *distribuir cartas*, a mensagem do jogador é ignorada. Se a fase do jogo for *esperando jogada* mas o turno é de outro jogador, a variável *jogadores\_ativos* será encarregada de negar a mensagem do jogador mal-intencionado. Vale ressaltar que, caso a mensagem seja do tipo *CHAT*, ela será aceita e imediatamente transmitida a todos os jogadores.
- **Não enviar informações sensíveis:** O servidor só envia para os jogadores informações absolutamente necessárias. Por exemplo, quando ocorre a mão de ferro, em que os jogadores não podem ver suas cartas, o servidor "vira" as cartas antes de enviá-las aos jogadores, ou seja, eles não podem trapacear para tentar olhar suas cartas.

Estas soluções implementadas, apesar de, de certa forma, evitarem de maneira efetiva algumas tentativas fraude, não impedem que, por exemplo, os pacotes enviados aos clientes sejam capturados e analisados. Nenhuma mensagem enviada entre servidor e cliente é criptografada. Sendo assim, qualquer mensagem capturada pode ser facilmente convertida em dados úteis em posse da biblioteca *protocolo.h*, que documenta como as mensagens são estruturadas e é disponibilizada para a implementação do cliente.

Outro dilema encontrado foi sincronizar o *loop* da interface gráfica com *loop* da *thread* que recebe as mensagens do servidor e faz as atualizações na interface pois, de acordo com a documentação do *gtk+ 3.0*, ele só deve ser usado na *thread* em que foram usadas *gtk\_init* e *gtk\_main*[3]. A saída encontrada foi utilizar uma função do GTK chamada *g\_main\_context\_iteration*[2], que realiza apenas uma iteração do loop da interface gráfica. Esta função, que está dentro de um *loop* infinito, é protegida por um *mutex* para que a interface possa ser atualizada pela *thread* que recebe as mensagens do servidor sem que ocorra condições de corrida. A *thread* também utiliza o mesmo *mutex*.

## Referências

- [1] Hastatus.com.br. *Truco Mineiro*. 2016. URL: <https://trucomineiro.com.br/regras.php> (acedido em 12/10/2018).
- [2] The GNOME Project. *The Main Event Loop: GLib Reference Manual*. 2014. URL: <https://developer.gnome.org/glib/stable/glib-The-Main-Event-Loop.html#g-main-context-iteration> (acedido em 24/10/2018).
- [3] The GNOME Project. *Threads: GDK 3 Reference Manual*. 2014. URL: <https://developer.gnome.org/gdk3/stable/gdk3-Threads.html#gdk3-Threads.description> (acedido em 24/10/2018).