

```

7 int main(int argc, char **argv) {
8     while (gameRunning) {
9
10        switch (mainMessage.type) {
11        case 0:
12            printf("%s\n", mainMessage.message);
13            char action[10];
14            fgets(action, 10, stdin);
15
16            mainMessage.client_action = atoi(action);
17            mainMessage.type = MSG_RESPONSE;
18
19            send(s, &mainMessage, sizeof(mainMessage), 0);
20            recv(s, &mainMessage, sizeof(mainMessage), 0);
21
22            break;
23        case 2:
24
25            printf("%s\n", mainMessage.message);
26            recv(s, &mainMessage, sizeof(mainMessage), 0);
27
28            break;
29        case 3:
30            printf("%s\n", mainMessage.message);
31            fgets(action, 10, stdin);
32
33            mainMessage.client_action = atoi(action);
34            mainMessage.type = 4;
35
36            send(s, &mainMessage, sizeof(mainMessage), 0);
37            recv(s, &mainMessage, sizeof(mainMessage), 0);
38
39            break;
40        case 5:
41            printf("%s\n", mainMessage.message);
42            recv(s, &mainMessage, sizeof(mainMessage), 0);
43            break;
44        case 6:
45            printf("%s\n", mainMessage.message);
46            close(s);
47            gameRunning = 0;
48            break;
49        default:
50            fprintf(stderr, "Communication Error");
51        }
52    }
53    return 0;
54 }

```

A imagem da esquerda representa o loop de comunicação do servidor e a da direita o do cliente.

Depois da ideia do switch a prioridade de tarefas a serem desenvolvidas foram : fluxo do jogo -> condições de erro -> condições de empate.

Para isso a próxima parte do código a ser tratada foi a modularização do gameFunctions, com a criação da “EnumToString” que no momento inicial teria papel de gerar as mensagens padrões e evitar repetição de código, tal função apresenta alguns problemas no futuro ainda mas ajudou em diversos momentos a deixar o código limpo.

```
1 #include "../headers/gameFunctions.h"
2
3 void EnumToString(GameMessage *currentMessage) {
4     const char *gameArray[] = {"Nuclear Attack", "Intercept Attack",
5                                 "Cyber Attack", "Drone Strike", "Bio Attack"};
6     switch (currentMessage->type) {
7     case 0:
8         strcpy(currentMessage->message,
9                 "Escolha sua jogada:\n0 - Nuclear Attack\n1 - Intercept "
10                "Attack\n2 - Cyber Attack\n3 - Drone Strike\n4 - Bio Attack");
11         break;
12     case 1:
13         char intToString[10];
14         char temp[256];
15
16         sprintf(intToString, "%d", currentMessage->client_action);
17         strcpy(temp, "Client Escolheu ");
18         strcat(temp, intToString);
19         strcpy(currentMessage->message, temp);
20         break;
21     case 2:
22         sprintf(currentMessage->message,
23                 "Você escolheu: %s\nServidor escolheu: %s\n",
24                 gameArray[currentMessage->client_action],
25                 gameArray[currentMessage->server_action]);
26         break;
27     case 3:
28         strcpy(currentMessage->message,
29                 "Deseja jogar novamente?\n1- Sim\n0-Nao");
30         break;
31     case 4:
32         if (currentMessage->client_action == 0) {
33             strcpy(currentMessage->message,
34                     "Cliente não deseja jogar novamente");
35         } else {
36             strcpy(currentMessage->message, "Cliente deseja jogar novamente.");
37         }
38         break;
39     case 6:
40         char clientWinsToString[10];
41         char serverWinsToString[10];
42         sprintf(clientWinsToString, "%d", currentMessage->client_wins);
43         sprintf(serverWinsToString, "%d", currentMessage->server_wins);
```

A parte de criação do fluxo geral do jogo, cliente digitar server rodar aleatório e mostrar resultado ao cliente ocorreu sem nenhum erro, nessa parte do desenvolvimento a parte de enviar as escolhas ao cliente acabou sendo transformada em uma função “StartGame” pois percebi que dessa forma ela poderia ser mais bem utilizada.

A segunda parte, seria a parte de criação das condições de erro, onde deparei com mais um problema, existiam duas condições de erro, na hora de enviar o ataque e na hora de jogar novamente, porém só uma MSG\_ERROR no enum, dificultando a utilização da função EnumToString para gerar a mensagem. Foi então decidido criar uma outra função específica para lidar com os erros, sendo ela a CreateErrorMessage, que gera a mensagem para as duas opções, apesar de não ser totalmente otimizada para o envio da mensagem de erro para seleção de ataque.

A terceira parte foi a elaboração do código de empate, para isso foi necessário modificar um pouco o código do case 1 do switch do server, colocando

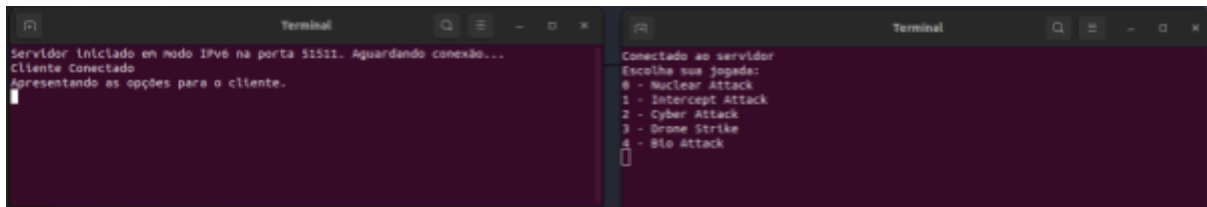
um `if` extra que primeiro checaria o empate e tanto a derrota quanto a vitória estariam dentro do `else` dessa condição, dessa forma apenas uma pequena mudança no código fez com que todas as condições fossem satisfeitas.

## Principais dificuldades encontradas

As maiores dificuldades de produção do código ocorreram em seu início, sendo uma delas a dificuldade em entender a programação de sockets, apesar de parecer ser simples no papel até entender de fato tudo o que era pedido no enunciado foi necessário um certo esforço, como por exemplo a parte da necessidade de usar a `struct` e `enums` daquela forma só forma compreendidos quando todas as video aulas foram finalizadas. Além disso, como já citado anteriormente o entendimento de como deveria ser o fluxo de mensagens do cliente e servidor acabou gerando alguns problemas, antes de chegar na solução do `switch` constantemente acaba por cair em loops infinitos ou em enviar mensagens que não chegavam a lugar algum.

Outra problema decorrido foi o mal planejamento inicial do código, como a utilização de algumas funções que não cumpriram 100% seu papel e sendo necessárias algumas refatorações conforme o código progredia. Com o conhecimento adquirido *agr*, e a visão melhor sobre o escopo total do projeto seria possível fazer um planejamento melhor e evitar grande parte desses erros e repetições ao se realizar um planejamento mais coerente com o esperado.

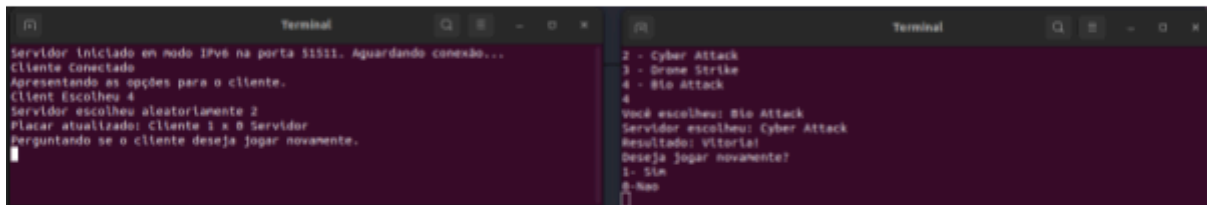
# Avaliações



```
Terminal
Servidor iniciado em modo IPv6 na porta 51511. Aguardando conexão...
Cliente Conectado
Apresentando as opções para o cliente.

Terminal
Conectado ao servidor
Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack
```

Conexão e apresentação de escolhas



```
Terminal
Servidor iniciado em modo IPv6 na porta 51511. Aguardando conexão...
Cliente Conectado
Apresentando as opções para o cliente.
Client Escolheu 4
Servidor escolheu aleatoriamente 2
Placar atualizado: Cliente 1 x 0 Servidor
Perguntando se o cliente deseja jogar novamente.

Terminal
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack
4
Você escolheu: Bio Attack
Servidor escolheu: Cyber Attack
Resultado: Vitória!
Deseja jogar novamente?
1- Sim
0-Nao
```

Cálculo dos ataques



```
Deseja jogar novamente?
1- Sim
0-Nao
1
Por favor, digite 1 para jogar novamente ou 0 para encerrar.
Deseja jogar novamente?
1- Sim
0-Nao
0
Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack
4
Por favor, selecione um valor de 0 a 4
Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack
```

Mensagens de erro



```
Servidor escolheu aleatoriamente 2
Placar atualizado: Cliente 2 x 2 Servidor
Perguntando se o cliente deseja jogar novamente.
Cliente não deseja jogar novamente
Enviando placar final.
Encerrando conexão.
Cliente desconectado.

Deseja jogar novamente?
1- Sim
0-Nao
0
Fim de jogo!
Placar final: Você 2 x 2 Servidor
Obrigado por jogar!
```

Placar final e finalização da conexão

Link do github: <https://github.com/Zel-ucas/TP-Redes-1>