# Relatório sobre elaboração do TP

## Fluxo geral do desenvolvimento

Para a elaboração do TP o primeiro passo foi dar uma olhada nos slides e nos capítulos do livro, porém eles ainda não deixaram bem claro o que deveria ser feito, para isso as vídeo aulas disponibilizadas foram extremamente úteis, pois fizeram com que todos os passos de conexao server e cliente ficassem claros, sendo suas funções utilizadas até o final do desenvolvimento.

Logo após a conexão de acordo com os vídeos ser finalizada surgiu o primeiro desafio, como fazer um “loop” de conexão, ou seja, estabelecer corretamente o fluxo de mensagens. A primeira opção era um while true tanto no server e no cliente, e a única diferença era na ordem do send e recv, o cliente começava com recv e o server com send porém com pouco tempo de desenvolvimento tal alternativa se mostrou ser falha, pois cada opção de mensagem acabava necessitando de uma quantidade de envios e recebimentos diferentes, analisando mais o enunciado notei que o servidor só recebia 2 tipos de mensagens enquanto o cliente recebia o resto, sendo então criado um switch dentro de um loop infinito para lidar propriamente com o fluxo.

```
int main(int argc, char **argv) {
    while (connection) {
        switch (mainMessage.type) {
        case 1:
            printf("%s\n", mainMessage.message);
            if (!(mainMessage.client_action <= 4 &&
                  mainMessage.client_action >= 0)) {
                printf("Erro: opção inválida de jogada.\n");
                CreateErrorMessage(clientSocket, &mainMessage
                                   "Por favor, selecione um v
                                   MSG_REQUEST);
                break;
            };

            mainMessage.result = PlayGame(&mainMessage);
            mainMessage.type = MSG_RESULT;
            EnumToString(&mainMessage);
            if (mainMessage.result == 0) {
                printf("Jogo empatado.\n");
                strcat(mainMessage.message, "Resultado: Empat
                send(clientSocket, &mainMessage, sizeof(GameM
                printf("Solicitando ao cliente mais uma escol
                StartGame(clientSocket, &mainMessage);
            } else {
                if (mainMessage.result == -1) {
                    strcat(mainMessage.message, "Resultado: V
                    mainMessage.client_wins++;
                } else {
                    strcat(mainMessage.message, "Resultado: D
                    mainMessage.server_wins++;
                }
                printf("Placar atualizado: Cliente %d x %d Se
                       mainMessage.client_wins, mainMessage.s
                send(clientSocket, &mainMessage, sizeof(GameM
                mainMessage.type = MSG_PLAY_AGAIN_REQUEST;
                printf("Perguntando se o cliente deseja jogar
                EnumToString(&mainMessage);
                send(clientSocket, &mainMessage, sizeof(GameM
                recv(clientSocket, &mainMessage, sizeof(GameM
            }
            break;

        case 4:
```

```
int main(int argc, char **argv) {
    while (gameRunning) {

        switch (mainMessage.type) {
        case 0:
            printf("%s\n", mainMessage.message);
            char action[10];
            fgets(action, 10, stdin);

            mainMessage.client_action = atoi(action);
            mainMessage.type = MSG_RESPONSE;

            send(s, &mainMessage, sizeof(mainMessage), 0);
            recv(s, &mainMessage, sizeof(mainMessage), 0);

            break;
        case 2:

            printf("%s\n", mainMessage.message);
            recv(s, &mainMessage, sizeof(mainMessage), 0);

            break;
        case 3:
            printf("%s\n", mainMessage.message);
            fgets(action, 10, stdin);

            mainMessage.client_action = atoi(action);
            mainMessage.type = 4;

            send(s, &mainMessage, sizeof(mainMessage), 0);
            recv(s, &mainMessage, sizeof(mainMessage), 0);

            break;
        case 5:
            printf("%s\n", mainMessage.message);
            recv(s, &mainMessage, sizeof(mainMessage), 0);
            break;
        case 6:
            printf("%s\n", mainMessage.message);
            close(s);
            gameRunning = 0;
            break;
        default:
            LogExit("Comunication Error");
```

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 ajudou em diversos momentos a deixar o código limpo.

```c
#include "../headers/gameFunctions.h"

void EnumToString(GameMessage *currentMessage) {
    const char *gameArray[] = {"Nuclear Attack", "Intercept Attack",
                               "Cyber Attack", "Drone Strike", "Bio Attack"};
    switch (currentMessage->type) {
    case 0:
        strcpy(currentMessage->message,
               "Escolha sua jogada:\n0 - Nuclear Attack\n1 - Intercept "
               "Attack\n2 - Cyber Attack\n3 - Drone Strike\n4 - Bio Attack");
        break;
    case 1:
        char intToString[10];
        char temp[256];

        sprintf(intToString, "%d", currentMessage->client_action);
        strcpy(temp, "Client Escolheu ");
        strcat(temp, intToString);
        strcpy(currentMessage->message, temp);

        break;
    case 2:
        sprintf(currentMessage->message,
                "Você escolheu: %s\nServidor escolheu: %s\n",
                gameArray[currentMessage->client_action],
                gameArray[currentMessage->server_action]);
        break;
    case 3:
        strcpy(currentMessage->message,
               "Deseja jogar novamente?\n1- Sim\n0-Nao");
        break;
    case 4:
        if (currentMessage->client_action == 0) {
            strcpy(currentMessage->message,
                   "Cliente não deseja jogar novamente");
        } else {
            strcpy(currentMessage->message, "Cliente deseja jogar novamente.");
        }
        break;
    case 6:
        char clientWinsToString[10];
        char serverWinsToString[10];
        sprintf(clientWinsToString, "%d", currentMessage->client_wins);
        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

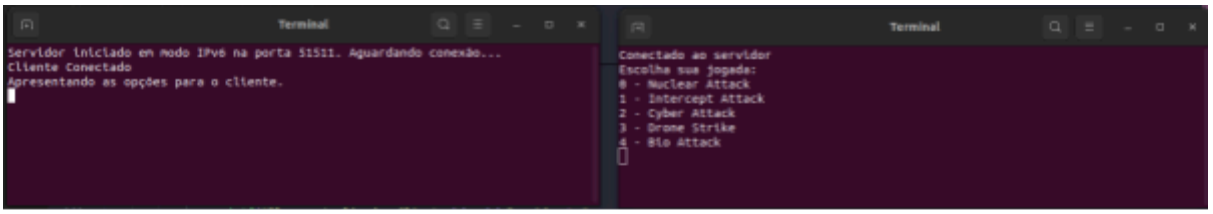
um If extra que primeiro checaria o empate e tanto a derrota quanto a vitória estariam dentro do else dessa condição, dessa forma apenas uma pequena mudança no código fez com que todas as condições fossem satisfeitas.

## Principais dificuldades encontradas

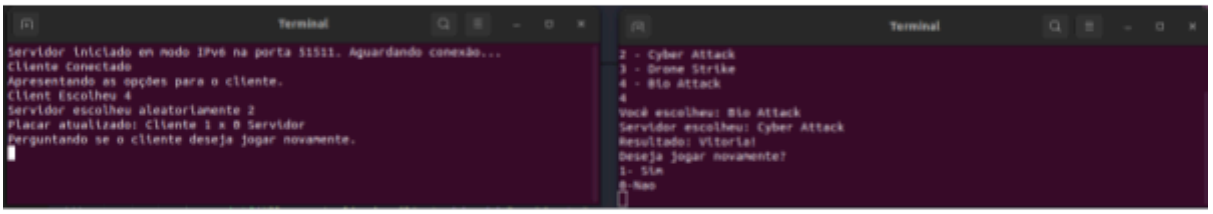
As maiores dificuldades de produção do código ocorreram em seu início, sendo uma delas a dificuldade em entender a programação de sockets, apesar de parecer ser simples no papel até entender de fato tudo o que era pedido no enunciado foi necessário um certo esforço, como por exemplo a parte da necessidade de usar a struct e enums daquela forma só forma compreendidos quando todas as video aulas foram finalizadas. Alem disso, como já citado anteriormente o compreendimento 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




Conexão e apresentação de escolhas




Cálculo dos ataques


Mensagens de erro


Placar final e finalização da conexão