PROJETO RPG - DESCRIÇÃO

Os monitores de IP ficam sempre muito cansados corrigindo listas, preparando e dando aulas e tirando dúvidas dos alunos. Depois de tanto trabalho, eles merecem relaxar, certo?! Eles adoram jogar um RPG Online (além de LOL, claro) e por isso você deve ajudá-los preparando um jogo muito divertido que eles possam jogar em conjunto, como forma de retribuição de todo o esforço deles durante este período!

Sua tarefa, então, é criar um RPG Multiplayer (com comunicação entre computadores diferentes baseado na estrutura de Socket) em C **para o sistema Linux** que permita ao jogador mover-se por um mapa utilizando teclas (w, a, s, d) e atacar os terríveis monstros que surjam em seus caminhos (e jogadores adversários, obviamente). Lembrando que todos os monitores gostam muito de inovações e, assim, quanto mais criativo seu jogo for, mais empolgados eles estarão para jogar!

➢ Mecânicas Básicas do Jogo:

Tudo se baseia em um mapa como este abaixo:

```
**************************
*******  **  *  **  *****
****    ***  *   **    **
**     ***   **   *   ***
*      ****   *   **    *
***   m  **   *   m    **
**      ***  ***      ***
***      *** **   **   **
**        *       *    **
*    ***1     * *       *
*    ***     ****     ***
*           0           *
**  ** *          **   **
*  2****         ***    *
**  ***        ****     *
*****            ***    *
* **               *     *
*     m   *       m  ****
*        **           ***
*    *****       ********
********     ************
*****          *****   **
******        ***  ***  *
** ***         ****  *3 *
**************************
HP: 200
SCORE: 0
ATK:8001
DEF:8001
```

Os jogadores são representados pelos números espalhados no mapa, onde cada cliente recebe seu número de acordo com a ordem de conexão no servidor. Vale salientar que serão fornecidas bibliotecas para o suporte da comunicação. Os heróis, podem andar pelo mapa apertando as teclas do teclado (w,a,s,d), onde cada uma destas teclas representam a direção que o jogador deseja andar.

Já os monstros, representados pelas letras 'm', andam em movimentos aleatórios que podem ser ajustados como você preferir. A quantidade deles também pode variar de mapa para mapa. Recomendamos o uso da função rand() (contida na biblioteca "stdlib.h"), que gera um número aleatório como retorno a cada chamada. Você pode usar este número da forma que você quiser, desde que, a cada número razoável de movimentações, de cada usuário, os monstros se movam.

Obs.: Quando qualquer jogador parte para cima de um monstro ou de algum outro participante, acontece a batalha(mais explicada em seu próprio tópico!), onde os dois seres duelarão até que apenas o melhor permaneça com vida no mapa. Nesse ponto, cabe a você, programador, "ajustar" os atributos dos monstros para que o jogo seja ao mesmo tempo "ganhável" e desafiador[mais detalhes sobre isso abaixo]!

De resto, onde temos que os espaços vazios ' ' são os corredores e os '*' representam árvores da floresta. Ambos, jogadores e monstros, podem andar pelos espaços vazios ' ', porém, todos os heróis do jogo têm habilidades carpinteiras, resultando, ao passarem por uma árvore, na retirada da mesma do mapa, fazendo com que o jogador que a retirou ganhe pontos/vida (a definir pelo programador do jogo), e o espaço '*' se torne vazio(' '). Os Monstros, entretanto, não possuem habilidades carpinteiras, e, por este motivo, não podem retirar as árvores do lugar. Também, por não possuir tendências suicidas, não podem caminhar "para cima" dos jogadores. Ou seja, estes dois movimentos devem ser considerados inválidos para estes misteriosos e tenebrosos seres.

Bom, vale frisar também que em toda borda do mapa há árvores que delimitam o fim do mesmo, e estas não podem ser "cortadas" pelos jogadores. Logo, elas sempre estarão lá delimitando as redondezas do cenário.

Abaixo deste mapa haverá um campo para "score" (pontuação), "hp" (vida), "atk" (ataque) e "def"(defesa), e cada um dos jogadores conectados receberá os seu próprios dados.

Obs.: A tela de cada jogador deve ser atualizada a cada movimento que for feito! Assim, como dica, para limpar a tela a cada mudança, pode-se usar a função **system("clear");**

Sobre os mapas, todos devem ter borda rodeada de "árvores", não importando o formato dele. Você deve oferecer aos jogadores, no mínimo, quatro mapas diferentes que podem ser escolhidos pelo usuário durante o menu de opções (Que será explicado posteriormente neste documento). Estes devem ser armazenados em arquivos ".txt" na pasta do jogo, e, obviamente, quando necessário, lido pelo seu programa para exibir-lo na tela. Nesse ".txt" deve haver, nas primeiras linhas, a altura, a largura, e a quantidade de monstros deste mapa, nesta ordem, e, só então, ter o mapa em si.

Outra informação importante é que cada mapa tem seu “pacote” de stats(ataque, defesa e vida) de todos os seus monstros, salvos, cada um, em um arquivo binário. A partir do momento que o usuário escolher seu mapa, então, não somente este deve ser lido em seu banco de dados, mas também as informações de todos os monstros que participarão da batalha.

➢ Menu Principal:

Ao iniciarmos o game, nos deparamos com o seguinte menu:

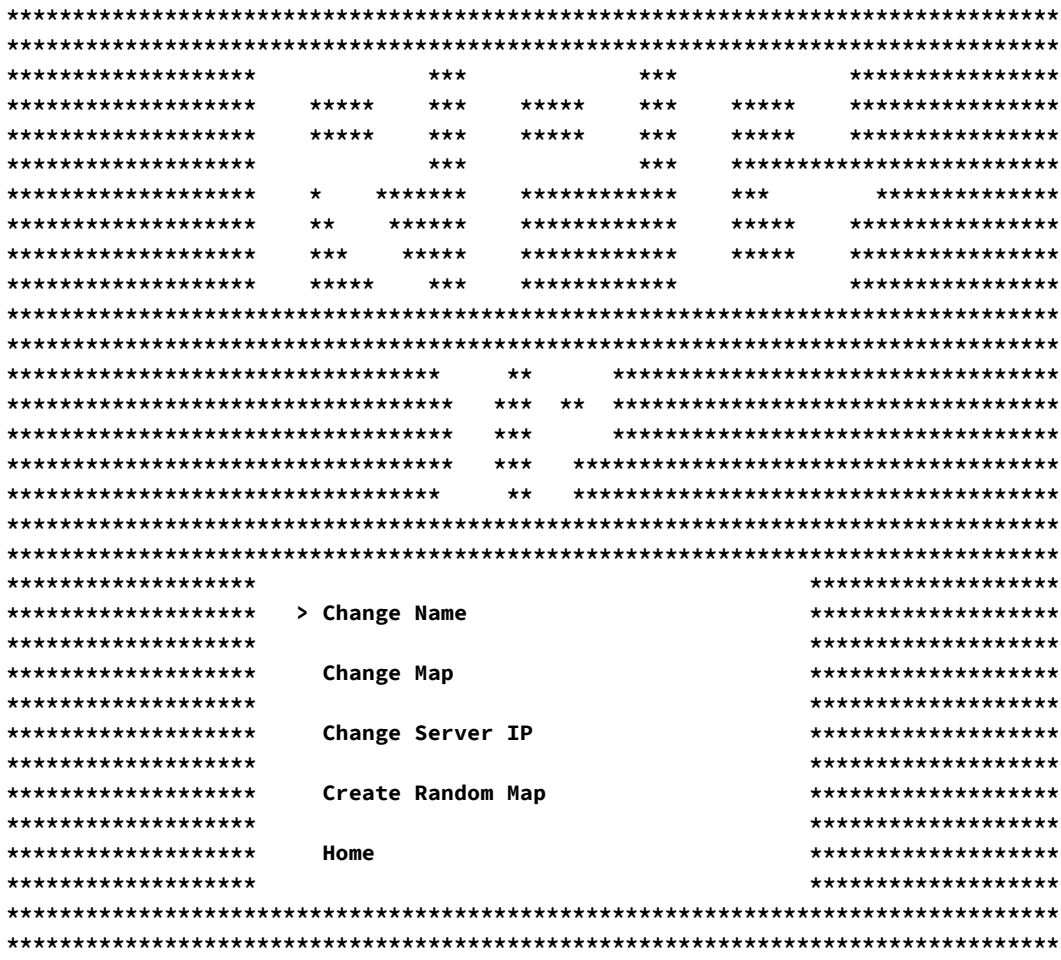
Podemos navegar entre as opções através das teclas (w,s), e escolhemos uma quando apertamos a tecla “d”.

- Na opção “Start Game”, iniciamos nossa comunicação com o server, preparando-se para, obviamente, começar o jogo.

- Na opção “Quit”, fechamos o jogo, ou seja, ele para a sua execução.

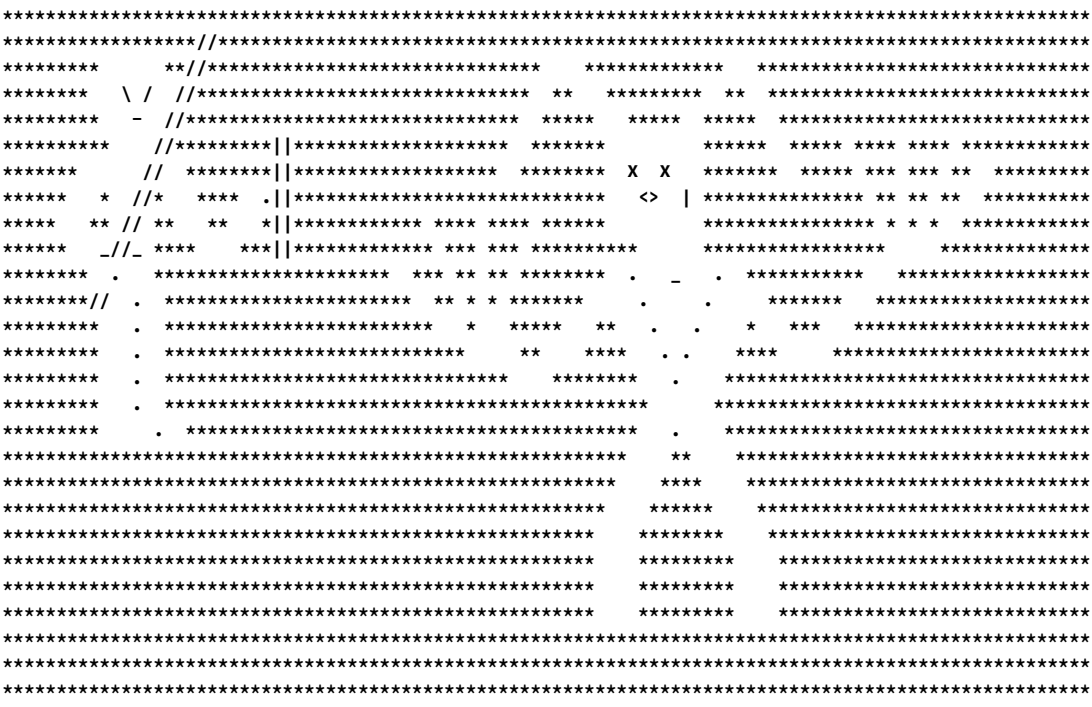
- Já a “Options”, nos leva para um novo menu (Demonstrado abaixo), onde podemos mudar o nome de nosso personagem, o mapa que será jogado por todos, o IP do server que nos comunicaremos e até criar uma mapa randômico. Nas opções de mudar valores, estes novos devem persistir enquanto o cliente não mudar novamente por este menu. Ou seja, todas essas informações devem ser guardadas em um arquivo binário para que, mesmo

que o programa seja reiniciado, as configurações de antes se mantenham. Já em relação ao criador de mapa randômico, você deve produzir um que, recebendo do usuário a altura, largura e quantidade de monstros(e, para cada um, seus respectivos stats), produza um novo mapa(.txt) e um novo arquivo binário de informações para este mapa(para que tudo funcione corretamente, estes arquivos devem ser enviados para todos que vão ser conectar ao jogo, inclusive o servidor, sem necessidade de ser “dentro do jogo”).



➢ Duelo:

Quando duas criaturas tentam lutar pelo mesmo espaço no mapa, deve haver um duelo para saber quem é o merecedor deste! Assim que iniciado, teremos na tela esta imagem:



```
HP:  300
ATK: 50
DEF: 18

HP:  500
ATK: 44
DEF: 0
```

Onde podemos ver os integrantes da luta posicionados, e, abaixo disso, as informações sobre os seus stats e o de seu oponente.

Ambos em suas devidas posições, inicia-se então o combate baseado em turnos, onde o herói que incitou a batalha (“chamou pro duelo”) começa atacando. Contudo, para que o ataque seja lido pelo programa, quem está na vez deve pressionar a tecla ‘a’ para que seu dano seja computado pelo servidor, passando a vez para o seu oponente. Quando seu oponente for um outro Jogador, ao passar a vez, você deve esperar a vez dele para poder atacar novamente. Já quando este for um monstro, o seu programa deverá controlá-lo, mas sempre lembrando que a batalha se baseia em turnos, onde o oponente só ataca após você atacar. Há a opção também de fugir da batalha , que será computada quando o jogador digitar a tecla ‘r’.

Você deverá montar os atributos de cada monstro e decidir como o ataque e a defesa se combinam para o dano final de todos os participantes, lembrando sempre em manter a coerência em relação aos “stats” do personagem e a dificuldade total do jogo.

Bom salientar que, quando um jogador entra numa batalha, os demais jogadores continuam jogando normalmente, porém, outros jogadores não podem duelar com seres que já estão participando de um duelo no momento. Atente que isso se aplica tanto para JogadorxMonstro ou JogadorxJogador.

➢ Conexão:

A conexão vai ser feita usando a ferramenta socket em C, com praticamente todas as funções que cuidam da comunicação implementadas nas bibliotecas “lib/client.h” , “lib/server.h” e “lib/default.h”. Com isso, alguns cuidados devem ser tomados quanto ao padrão que foi desenvolvido pela equipe de monitoria. O jogo é baseado na estrutura cliente-servidor[TCP], onde os jogadores são os clientes que enviam seus movimentos para o servidor, e o servidor faz todo o processamento do jogo e retorna para os clientes as atualizações dos estados do jogo.

Ao conectar no servidor, a primeira mensagem enviada pelo cliente deve conter suas informações iniciais (nome e mapa que foram selecionados no menu options), ela é enviada através da estrutura clientInfo(será detalhada logo abaixo). Após isso, as mensagens são trocadas através de textos de tamanho máximo BUFFER_SIZE(definido em default.h). Por simplicidade, é recomendado que o primeiro caractere defina o tipo da mensagem: se houve algum erro na conexão, ou se é uma mensagem informando que o jogo de fato vai ser iniciado.

Logo antes do jogo iniciar, algumas mensagens devem ser enviadas para todos os clientes (como nome do mapa que será utilizado, dado que eles devem carregar uma cópia do mesmo). Uma vez que o jogo é iniciado, fato que é decidido pelo primeiro cliente que se conectou, ninguém pode mais se conectar ao servidor, e as mensagem de cada jogador para o servidor são estruturas mov_msg, e do servidor para o jogador upd_msg (ambas serão detalhadas futuramente).

É importante saber que o servidor irá apenas informar as modificações feitas no mapa, hp, ataque ou defesa de seus clientes. Mas ele não deve sempre mandar o mapa inteiro toda vez que este seja modificado, somente as posições que são afetadas pelos movimentos para que a comunicação não fique sobrecarregada de informações.

Essas são algumas informações sobre as bibliotecas disponibilizadas pelos monitores:

- **Default.h**
  Nesta biblioteca são declaradas algumas estruturas e constantes comuns ao client e server, nela também são incluídas todas as bibliotecas utilizadas por todas as que foram dadas (provavelmente não será necessária a inclusão de mais bibliotecas, tendo em vista que praticamente todas que são necessárias para o desenvolvimento do projeto já são incluídas nesta).
  Obs: Esta biblioteca é incluída nas outras duas bibliotecas, então, ao incluir “lib/client.h” você já está incluindo tudo de default.h e client.h.

mov_msg:

Estrutura utilizada para enviar um movimento (char msg) do cliente para o servidor.

upd_msg:

Estrutura utilizada para enviar alguma modificação do jogo do server para um cliente. Os campos desta estrutura, assim como o padrão que será mostrado pela equipe pode ser modificado facilmente, mas é recomendado que ele seja seguido, ou sofra poucas modificações.

clientInfo:

Estrutura utilizada para mandar informações iniciais do cliente para o servidor.

- **Client.h:**
  - ***void connectToServer(char[]):***

    Dada uma string de entrada contendo o IP do server (Setada da forma correta, com todos os pontos em seus devidos lugares), faz com que o cliente se conecte com o servidor. Para que esta função funcione corretamente, o server deve estar ligado e aberto a novas conexões. Caso não encontre nenhum, a função mostra na tela um erro, e para o funcionamento do programa.

    int sendMovToServer(mov_msg):

    Recebe uma estrutura mov_msg, e manda-a para o server conectado.

    int readTxtFromServer(char[]):

    Recebe uma string que guardará a mensagem que o server enviou para o client, caso este envio ocorra, obviamente.

    int getch():

    Uma função presente no Windows, mas que tem que ser modificada para funcionar no linux, e serve para que seu programa consiga pegar um char do usuário sem a necessidade da tecla “Enter”. Ela não recebe nada como parâmetro, e retorna o char capturado em até um determinado tempo de espera. Caso esse tempo chegue ao limite, ela retorna “-1”.

    int sendInfoToServer(clientInfo):

    Manda para o server conectado a estrutura clientInfo que recebeu como parâmetro.

    int readUpdFromServer(upd_msg*):

    Recebe do server a mensagem de atualização (upd_msg) e coloca na estrutura presente no parâmetro desta função.

- **Server.h:**
  - **Variáveis:**

    game_status

    Variável que determina o status atual do jogo. Padrão:

    0 - Ninguém se conectou ainda.

    1 - Esperando confirmação do criador do jogo(primeiro usuário que se conectou ao servidor).

    2 - O jogo está rodando.

clients_connected

Variável que diz quantos clientes estão conectados ao servidor.

player

Estrutura que contém informações de cada cliente:

Posição no mapa, vida, ataque, defesa, nome, se está lutando, com quem está lutando, etc. (ver mais informações no arquivo lib/server.h).

clients[]

Vetor do tipo player, que salva as informações de cada cliente.

map_changes[]

Vetor com todas as mensagens de atualização do mapa que precisam ser enviadas para todos os clientes. Este vetor é utilizado na função broadcast(será detalhada futuramente).

pos_broad:

Última posição livre no vetor descrito acima, também utilizado na função broadcast deve ser modificado com cuidado.

➢ **Funções:**

int readMovFromClient(int sockid , mov_msg *);

Recebe do cliente com sockid, dado no argumento, a mensagem de movimento (mov_msg) e coloca na estrutura presente no parâmetro desta função.

int readClientInfo(int sockid, clientInfo *);

Recebe do cliente com sockid, dado no argumento, a mensagem com informações iniciais (clientInfo) e coloca na estrutura presente no parâmetro desta função.

void sendTxtToClient(int sockid, char[] msg);

Manda para o cliente com sockid, dado como argumento, a mensagem que recebeu no parâmetro (que é um texto).

void sendUpdToClient(int filedes, upd_msg change);

Manda para o cliente com sockid, dado como argumento, a mensagem que recebeu de movimento que recebeu no parâmetro.

void init();

Seta as informações iniciais como zero. Funciona como um inicializador de variáveis

void sleepServer();

Faz com que o server “durma”(pare o funcionamento) até que receba uma conexão, ou mensagem.

void checkConn();

Seta as variáveis necessárias para aceitar a conexão de um novo client. Caso o máximo de clientes já tiver sido atingido ou o jogo já estiver ativo, manda para o client em questão uma mensagem de erro de comunicação. Caso contrário, a conexão é ativada, chamando a função clientConnected(Mais detalhes abaixo).

void wasClient();

Verifica se algum dos client's deseja enviar uma mensagem para o server. Caso o jogo não tenha sido iniciado ainda e a mensagem foi recebida do primeiro client, significa que o jogo deve começar, chamando então a função clientConfirmed()(Mais detalhes abaixo). Caso esteja no jogo, significa que esta mensagem é uma atualização de movimento, chamando a função clientMoved(Mais detalhes abaixo).

void broadcast();

Verifica se o jogo ainda está de pé(há clients ainda jogando). Se sim, enviamos todas as atualizações do map_changes[] para os clients.

void broadcastTxt(const char msg[], int s);

Manda para todos os clientes (menos o com id = s[segundo parâmetro]) conectados a mensagem passada como primeiro parâmetro. Para mandar para todos os clientes(s = -1).

void desconnectCleint(int id):

Fecha a conexão com o cliente de id dado como parâmetro.

Variáveis de ponteiros para funções: elas vão guardar os ponteiros para funções, com assinatura descrita abaixo. Você deve criar as funções que tratam cada evento em seu código, com os mesmos parâmetros, e atribuir os endereços destas funções às respectivas variáveis!!!

(*clientMoved)(int, mov_msg)

Esta função será chamada toda vez que o cliente enviar uma "mensagem de movimento". O primeiro parâmetro é o id do cliente, e o segundo a mensagem recebida.

(*clientConnected)(int, clientInfo)

Esta função será chamada toda vez que o cliente se conectar ao servidor, mandando as suas informações iniciais. O primeiro parâmetro é o id do cliente, e o segundo a mensagem recebida.

(*clientDesconnected)(int)

Esta função será chamada toda vez que o cliente se desconectar do servidor. O único parâmetro é o id de tal cliente. Esta é a única função que não é necessária a inicialização(supondo que os clientes não irão se descontar subitamente).

(*clientConfirmed)(void)

Função sem parâmetros, que é executada toda vez que o cliente que criou o jogo (primeiro a se conectar) confirmar o incio.

➢ Resumo:

1. Servidor on;
2. Clientes se conectam;
3. Carregam mapas e dados;
4. Começa o pau;
5. Apenas o mais forte sobrevive;
6. Temos um vencedor;
7. Fim de jogo :D.

➢ Considerações finais:

Ufa, após toda essa explicação, vamos falar agora sobre como tudo isso deve ser feito por vocês.

Este projeto deve ser produzido em grupos de, no máximo, 5 pessoas (apenas alguns grupos deverão ter 6 pessoas, para que não sobre ninguém, então, veja a disponibilidade destes grupos antes de combinar com os amiguinhos :D). Cada grupo terá seu próprio monitor responsável (Verificar tabela no grupo do facebook) de tirar a maior quantidade de dúvidas possíveis, logo, caso necessário, contate-o! Este, assim que o prazo acabar, marcará com o grupo um encontro para a apresentação do projeto.

É bom frisar que toda a parte de comunicação foi feita baseada na plataforma **Linux Ubuntu**, então, seu jogo **TEM** que ser feito para **Linux**!! Consequentemente, deixando claro, toda a parte de correção e avaliação de seu projeto será feita em máquinas **Linux** do próprio Cin!

Lembrem-se de que este arquivo contém apenas as diretrizes bases do jogo, vocês têm um leque imenso de possibilidades para adentrar. Quanto mais criativo seu jogo for, mais bem avaliado ele será pelo seu monitor, e maior sua nota!!

Boa Sorte :D!

Duelo: