Trabalho Prático 1 - Servidor de Exploração de Labirintos

Entrega Individual

Data de Entrega: 27 de novembro de 2024, às 23:59

Introdução

Labirintos são estruturas enigmáticas que têm fascinado a humanidade por milênios. Desde os lendários labirintos da Grécia Antiga, como o famoso mito do Minotauro em Creta, até as complexas obras de arte moderna e jogos, eles continuam desafiando a mente humana. Tradicionalmente, um labirinto é um intricado caminho com múltiplas bifurcações e um único objetivo: encontrar a saída.

Neste trabalho prático, o desafio é criar uma versão de um jogo em que um cliente se conecta a um servidor e caminhe em um labirinto, buscando a saída. Dessa forma, você será responsável por criar tanto um servidor quanto um cliente para possibilitar que uma máquina explore um labirinto remotamente.

O servidor será responsável por manter e atualizar o estado do jogo, repassando as informações para o jogador. O cliente poderá enviar comandos para se mover e receber as informações sobre sua posição no labirinto.

Com essa implementação, será possível aplicar os princípios de comunicação em rede e sockets, expandindo o conhecimento de redes e programação. Vamos então nos aventurar nesse labirinto digital!

PS: Existe uma estratégia conhecida para garantir a saída de um labirinto, chamada de "regra da mão direita (ou esquerda)": ao manter uma das mãos constantemente encostada na parede do labirinto, é possível percorrê-lo até encontrar a saída sem se perder, desde que o labirinto não seja desconexo. Você vai poder testar essa regra neste trabalho.

Objetivo

O objetivo deste trabalho é criar uma versão de um jogo de *Exploração de Labirintos* que permitirá a interação entre um cliente e um servidor usando sockets em linguagem C. O servidor será projetado para acomodar a conexão de um ***único cliente***, proporcionando uma experiência de jogo remoto.

No início do jogo, o cliente envia um comando para iniciar a partida e recebe do

servidor quais movimentos o jogador pode fazer. O cliente pode então enviar para qual direção quer ir, e o servidor vai atualizar a posição do jogador (caso o movimento seja válido) e retornar quais novos movimentos ele pode fazer. O cliente pode também requisitar ver o mapa, ação que faz com que o servidor retorne o labirinto para o usuário com todas as regiões ocultas, exceto pelas que o jogador já sabe o que são (como caminhos livres ou muros).

O jogo é vencido quando o cliente encontra a saída. O servidor informa ao cliente que o jogo foi vencido, e revela o labirinto com todas as células reveladas (mesmo se o jogador não descobriu todas). A partida é encerrada com sucesso.

Contudo, o cliente pode enviar um comando de encerrar o jogo a qualquer momento, onde após enviar a mensagem vai ser desconectado, e o servidor vai resetar o estado do jogo.

Por fim, como pontuação extra, deve ser implementado um comando extra de *dica*, onde, dado a posição atual do jogador, o servidor deve retornar uma lista de comandos que resultem num caminho válido do jogador até a saída. Esse caminho não precisa necessariamente ser o mais curto, apenas um que leve o jogador até o final.

Protocolo

Labirinto
O labirinto do jogo que o servidor receberá como entrada será de 5x5 até 10x10.

Mensagem a ser trafegada
O cliente e o servidor irão se comunicar por meio de uma mensagem que será trafegada usando sockets. A estrutura da mensagem deve ser a seguinte:

```
struct action {
  int type;
  int moves[100];
  int board[10][10];
};
```

As ações possíveis são:

Tabela 1 - Conjunto de ações possíveis

Action	Type	Descrição
start	0	Enviada pelo cliente ao servidor no início do jogo
move	1	Enviada pelo cliente para se mover no labirinto
map	2	Enviada pelo cliente requisitando o mapa
hint	3	Enviada pelo cliente solicitando a dica
update	4	Enviada pelo servidor sempre que houver mudança no estado do jogo
win	5	Enviada pelo servidor ao cliente quando o jogo for ganho
reset	6	Enviada pelo cliente ao servidor para reiniciar o jogo
exit	7	Enviada pelo cliente ao servidor quando deseja se desconectar

Observação: é muito importante que a mensagem siga exatamente essa estrutura para que haja compatibilidade entre os clientes e servidores de diferentes alunos.

Comandos

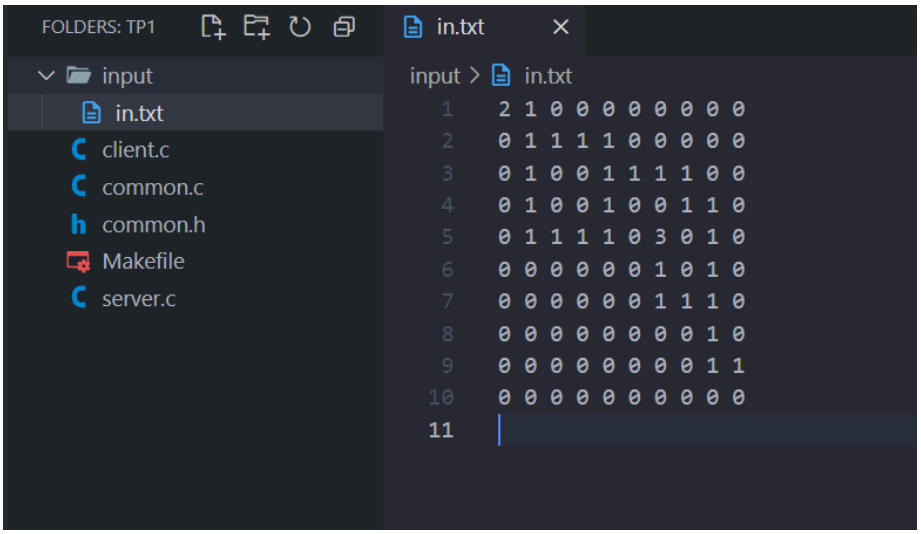
Start

No início do jogo o **servidor** lerá o labirinto completo de um arquivo. Esse arquivo deve conter todas as linhas e todos os valores separados por espaço.

Ao inicializar o servidor o caminho do arquivo deve ser passado como parâmetro da seguinte maneira:

```
./server v4 51511 -i input/in.txt
```

Abaixo um exemplo da estrutura do projeto:



Os valores possíveis do labirinto, que serão usados ao longo de todo o trabalho, estão descritos abaixo. Por convenção, neste trabalho as entradas sempre estarão nas extremidades do labirinto, mas a saída pode estar em qualquer lugar.

Tabela 2 - Representação do Labirinto

Representação em inteiros	**Representação em caracteres**	**Significado**
0	#	Muro
1	_	Caminho livre
2	>	Entrada
3	X	Saída
4	?	Não descoberto
5	+	Jogador

O cliente por sua vez se conecta da seguinte forma:

```
./client 127.0.0.1 51511
```

Ao receber a conexão, o servidor deve imprimir *client connected*. O cliente então deve mandar a mensagem *start*, onde vai receber a lista de movimentos válidos. Por fim, o servidor deve imprimir *starting new game*.

Move

Os movimentos válidos que o jogador pode fazer serão enviados como os primeiros inteiros no vetor *moves* onde cada inteiro equivale a um movimento, conforme tabela abaixo. O restante do vetor **deve ser preenchido com 0's.**

Tabela 3 - Representação dos movimentos

Representação em inteiros	**Movimento**
1	Cima
2	Direita
3	Baixo
4	Esquerda

Por exemplo, se o jogador pode ir para cima, direita e esquerda, os primeiros valores devem ser 1, 2 e 4, seguidos de 0's.

O cliente deve interpretar esses valores, e sempre imprimir no terminal "Possible moves:" seguido das direções, separadas por vírgula, com o último com ponto final. Continuando o exemplo anterior, o cliente deve imprimir

Possible moves: up, right, left.

A ordem de exibição dos movimentos na tela deve **sempre** seguir o sentido horário, partindo de "up". O jogador deve digitar qual direção deseja ir, onde o programa deve interpretar e mandar a mensagem com o inteiro equivalente.

Map

A qualquer momento do jogo, o cliente pode requisitar o mapa pelo comando *map*, que exibe uma versão parcial do labirinto conhecida pelo jogador. O labirinto completo é mantido oculto e armazenado exclusivamente no servidor, sendo responsabilidade dele rastrear a posição atual do jogador e as áreas descobertas.

Para simplificar a implementação, vamos considerar que o jogador descobre todas as células em um raio de uma célula de cada posição que ele visita. Dessa forma, o jogador pode ver apenas a célula imediatamente à frente, permitindo que ele visualize obstáculos, caminhos livres e até a saída, mesmo sem ainda tê-los

visitado. No entanto, isso se limita a apenas a próxima célula: mesmo que existam muitos caminhos livres à frente, o jogador verá apenas o próximo passo possível.

O tráfego do mapa entre servidor e cliente deve ser dado conforme descrito na Tabela 2, na representação em inteiros. Para exibir para o usuário, o cliente deve converter os valores para os caracteres, para melhor visualização. Como exemplo, considere o jogo abaixo, com um labirinto 5x5. O jogador começou na posição (0,0), e andou um movimento para direita e um para baixo, antes de solicitar o mapa.

Tabela 4 - Exemplo de labirinto trafegando pela rede

Labirinto Completo	Labirinto enviado pelo servidor	Labirinto exibido no terminal do cliente
2 1 0 0 0 0 1 1 1 0 0 1 0 1 1 0 1 1 1 3 0 0 0 0 0	2 1 0 4 4 0 5 1 4 4 0 1 0 4 4 4 4 4 4 4 4 4 4 4 4	> _ # ? ? # + _ ? ? # _ # ? ? ? ? ? ? ? ? ? ? ? ?

Hint

Como pontuação extra, o comando de dica deve retornar uma lista de movimentos válidos onde, se o jogador seguir, vai chegar à saída, partindo da sua posição **atual** (e não inicial do labirinto). Essa lista deve vir no vetor *moves*, seguida de 0's quando acabar. Conforme a exibição dos movimentos possíveis, o código do cliente deve converter os inteiros para os movimentos, para exibir na tela.

Continuando o exemplo da seção anterior, se o jogador solicitar a dica na posição (1,1), pode receber um vetor como [2, 2, 3, 2, 3, 0…], mas também pode receber [3, 3, 2, 2, 2, 0…], dentre outros. Isso porque deve ser uma sequência válida, não precisando ser necessariamente o menor caminho ou o com menos mudanças de direções. Contudo, atente-se que o tamanho do vetor *moves* é 100, então cuidado para que o caminho não passe por muitos locais redundantes.

Os comandos devem ser exibidos na tela da seguinte forma:

Hint: right, right, down, right, down.

Update

As mensagens do tipo *update* são as respostas do servidor para os comandos do cliente. Portanto, todas as respostas dos comandos acima (descritas nos comandos) devem vir com o tipo *update*.

Win

Quando o jogador se move para saída, o servidor deve enviar uma mensagem do tipo *win*, com o labirinto totalmente revelado (mesmo em locais onde o jogador não foi). O cliente deve exibir a mensagem *You escaped!*, com o labirinto totalmente revelado nas próximas linhas (conforme Tabela 2). Após vencer o jogo, os únicos comandos que devem ser aceitos são os *reset* e o *exit*.

Reset

O cliente poderá a qualquer momento resetar o jogo, e nesse caso o jogador volta para o início do labirinto, resetando o estado do jogo. Assim como no comando *start*, o servidor deve imprimir *starting new game*.

Exit

O cliente poderá a qualquer momento sair do jogo enviando o comando *exit*. Nesse caso, a conexão do cliente com o servidor se encerrará e o cliente terminará a execução. O servidor imprime *client disconnected*.

Erros

O usuário pode vir a digitar um comando inválido, ou fazer algum movimento que não seja possível no momento. Esses erros **devem** ser tratados do lado do cliente. Abaixo uma tabela com o mapeamento de **todos os possíveis** erros, e as mensagens que devem ser exibidas. **Os demais casos não mapeados na tabela não necessitam de uma mensagem de erro.**

Tabela 5 - Erros possíveis e respectivas mensagens

movimento não permitido no momento	error: you cannot go this way
comando inexistente	error: command not found
comandos antes de iniciar o jogo com o comando *start*	error: start the game first

Exemplos de jogo

Movimentos e erros

Terminal do cliente	Terminal do Servidor
	Labirinto do exemplo: 2 1 0 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 3 0 0 0 0 0
	> ./server v4 51511 -i input/in.txt
> ./client 127.0.0.1 51511	client connected
> right error: start the game first	
> start Possible moves: right, down.	
> right Possible moves: down, left.	
> right error: you cannot go this way	
> down Possible moves: up, right, down, left.	
> direita error: command not found	
> right Possible moves: right, left.	

Mapa

Terminal do cliente	Terminal do Servidor
	Labirinto do exemplo: 2 1 0 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 3 0 0 0 0 0
	> ./server v4 51511 -i input/in.txt
> ./client 127.0.0.1 51511	client connected
> start Possible moves: right, down.	starting new game
> right Possible moves: down, left.	
> down Possible moves: up, right, down, left.	
> map > _ # ? ? _ + _ ? ? # _ # ? ? ? ? ? ? ? ? ? ? ? ?	
> down Possible moves: up, down.	
> map > _ # ? ? _ _ _ ? ? # + # ? ? # _ _ ? ? ? ? ? ? ?	

Dica

Terminal do cliente	Terminal do Servidor
	Labirinto do exemplo: 2 1 0 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 3 0 0 0 0 0
	> ./server v4 51511 -i input/in.txt
> ./client 127.0.0.1 51511	client connected
> start Possible moves: right, down.	
> right Possible moves: down, left.	
> down Possible moves: up, right, down, left.	
> hint Hint: down, down, right, right, right.	
> down Possible moves: up, down.	
> down Possible moves: up, right.	
> hint Hint: right, right, right.	

Vitória

Terminal do cliente	Terminal do Servidor
	Labirinto do exemplo: 2 1 0 0 0 1 1 3 1 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0
	> ./server v4 51511 -i input/in.txt
> ./client 127.0.0.1 51511	client connected
> start Possible moves: right, down.	
> right Possible moves: down, left.	
> down Possible moves: up, right, down, left.	
> right You escaped! > _ # # # _ _ X _ # # _ # _ # # _ _ _ # # # # # #	
> down	
> reset Possible moves: right, down.	starting new game

Reset e Exit

Terminal do cliente	Terminal do Servidor
	Labirinto do exemplo: 2 1 0 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 3 0 0 0 0 0
	> ./server v4 51511 -i input/in.txt
> ./client 127.0.0.1 51511	client connected
> start Possible moves: right, down.	
> right Possible moves: down, left.	
> down Possible moves: up, right, down, left.	
> map > _ # ? ? _ + _ ? ? # _ # ? ? ? ? ? ? ? ? ? ? ? ?	
> reset Possible moves: right, down.	starting new game
> map + _ ? ? ? _ _ ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?	
> exit	client disconnected

Tabela de comandos

Comando	Cliente	Servidor
conexão do cliente	Não imprime nada no cliente	imprime *client connected*
start	Imprime *Possible moves:* {os movimentos possíveis}	Inicializa o estado do jogo; Envia movimentos para o cliente; Imprime *starting new game*;
move	Imprime *Possible moves:* {os próximos movimentos possíveis}	Atualiza o estado do jogo; Envia movimentos para o cliente; Não imprime nada*;*
map	Imprime o mapa convertendo número para caracteres	Envia mapa para o cliente; Não imprime nada*;*
hint	Imprime *Hint:* {movimentos até a saída}	Envia lista de movimentos; Não imprime nada*;*
reset	Imprime *Possible moves:* {os movimentos (iniciais) possíveis}	Reseta o estado do jogo; Imprime *starting new game*;
exit	Desconecta do servidor	Reseta o estado do jogo; Imprime *client disconnected*

Detalhes de implementação

- Note que o servidor irá guardar tanto a matriz contendo a resposta do labirinto quanto a matriz contendo o estado atual do jogo. Sempre que uma ação é feita a matriz de estado é atualizada

- O cliente deve imprimir o labirinto com a representação em caracteres
    - O caractere do jogador sobrepõe a entrada, se o mapa for solicitado na entrada (veja exemplo do “Reset e Exit”)
    - Entretanto, quando o usuário chegar à saída, não deve ser enviado e nem exibido o caractere dele (ele já não está mais no labirinto)
    - Esses dois casos são responsabilidade do servidor de enviar corretamente o estado do jogo

- **As posições na matriz devem ser imprimidas uma *tab* de separação: "\t"**

Desenvolvimento

O projeto requer a implementação de dois componentes essenciais: um servidor e um cliente, ambos baseados no protocolo **TCP**. É crucial garantir que ambos os componentes sejam compatíveis **tanto com o IPv4 quanto com o IPv6**, proporcionando flexibilidade na escolha do endereço IP.

O cliente tem a responsabilidade de receber mensagens diretamente do teclado e exibi-las na tela. O servidor será responsável por fazer todo o processamento do jogo.

É importante observar que, para este trabalho, **não é necessário que o servidor suporte múltiplos clientes simultaneamente**. Ele será projetado para lidar com conexões de um cliente por vez.

Para configurar o servidor corretamente, este deve ser capaz de receber, seguindo rigorosamente essa **ordem, o tipo de endereço desejado (v4 para IPv4 ou v6 para IPv6) e um número de porta espe**cificado na linha de comando. Recomenda-se a utilização da porta 51511 para manter a padronização do projeto.

Da mesma forma, o cliente precisa receber, também rigorosamente nessa ordem, o endereço IP do servidor e o número de porta para estabelecer a conexão com sucesso. Certifique-se de que ambos os componentes sejam configurados de acordo com essas diretrizes para uma integração eficaz e a comunicação adequada entre servidor e cliente.

A seguir, um exemplo de execução dos programas em dois terminais distintos:

IPv4:

```
no terminal 1: ./server v4 51511 -i input/lab1.txt
no terminal 2: ./client 127.0.0.1 51511
```

IPv6:

```
no terminal 1: ./server v6 51511 -i input/lab1.txt
no terminal 2: ./client ::1 51511
```

Materiais para Consulta:

- Capítulo 2 e 3 do livro sobre programação com sockets disponibilizado no Moodle.
- Playlist de programação com sockets do professor Ítalo Cunha

Avaliação

O trabalho deve ser realizado individualmente e **deve ser implementado com a linguagem de programação C (não sendo permitida a utilização de C++)** utilizando somente a biblioteca padrão (interface POSIX de sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve utilizar bibliotecas Windows, como o winsock**. Seu programa deve interoperar com qualquer outro programa implementando o mesmo protocolo **(você pode testar com as implementações dos seus colegas)**. Procure escrever seu código de maneira clara, com comentários pontuais e bem identados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Correção Semi-automática

Seu servidor será corrigido de forma semi-automática por uma bateria de testes. Cada teste verifica uma funcionalidade específica do servidor. O seu servidor será testado por um cliente implementado pelo professor com funcionalidades adicionais para realização dos testes. Os testes avaliam a aderência do seu servidor ao protocolo de comunicação inteiramente através dos dados trocados através da rede (a saída do seu servidor na tela, e.g., para depuração, não impacta os resultados dos testes).

Para a correção os seguintes testes serão realizados **(com IPv4 e IPv6)**:

- Iniciar o jogo - **5%**
- Repassar corretamente os movimentos possíveis - **10%**
- Avançar no labirinto adequadamente - **10%**
- Tratar todos os erros corretamente - **5%**
- Exibir o Mapa corretamente - **20%**
- Jogo terminar em vitória se o jogador sair do labirinto e exibição correta - **10%**
- Resetar o jogo - **10%**
- Fechar conexão com o servidor corretamente - **10%**
- Documentação **20%**
- Funcionalidade da “dica” - **extra**

Informações importantes

Cada aluno deve entregar documentação em PDF de até 4 páginas (duas folhas), sem capa, utilizando fonte Arial tamanho 12, e figuras de tamanho adequado ao tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos do projeto, bem como as soluções adotadas para os problemas.

Atenção: Será dado grande importância a parte da *discussão dos desafios encontrados*, pois as dificuldades ao longo do projeto sempre existem.

A documentação corresponde a 20% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cola de trabalhos.

Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.

Cada aluno deve entregar, além da documentação, o código fonte em C e um Makefile para compilação do programa.

Instruções para submissão e compatibilidade com o sistema de correção semi-automática:

- O Makefile deve compilar o cliente em um binário chamado “**client**” e o servidor em um binário chamado “**server**” dentro de uma pasta chamada **"bin"** na raiz do projeto

- Seu programa deve ser compilado ao se executar apenas o comando “make”, ou seja, sem a necessidade de parâmetros adicionais
- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: TP1_MATRICULA.zip

Desconto de Nota por Atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:01 do dia seguinte à entrega no relógio do Moodle, os trabalhos já estarão sujeitos a penalidades.

A fórmula para desconto por atraso na entrega do trabalho prático é:

$$desconto = 2^{d} - 1$$

onde *d* representa o número de dias úteis de atraso.

Obs.: Note que após 3 dias, o trabalho não pode ser mais entregue.