Trabalho Prático

Programação em C para Unix

O trabalho prático consiste num sistema de jogos do tipo *MUD – Multi-User Dungeon*[1]. A interface será em consola/modo texto, e todos os jogadores estarão na mesma máquina UNIX, ou seja, não se pretende a utilização de mecanismos de comunicação em rede – apenas mecanismos de comunicação inter-processo.

Existe um documento anexo com um exemplo acerca deste tido de jogos que se aconselha a ler, mas o enunciado propriamente dito é este documento.

1. Descrição geral do trabalho

O trabalho prático de SO consiste numa implementação de um jogo do tipo *MUD*. Este tipo de jogos consiste num labirinto no qual os jogadores navegam, encontrando objectos e enfrentando inimigos. A interface com o jogador é totalmente consola / modo texto e o jogador comunica as suas intenções escrevendo comandos de uma forma que faz lembrar (mas não é) as linhas de comandos da *Shell*. O jogo apresenta textualmente apenas aquilo que o jogador vê num determinado instante, ou seja, a sala em que ele se encontra e o seu conteúdo. Nunca é dada a visão total acerca do labirinto. O jogo apresenta a informação textualmente (descrição).

O jogo suporta vários jogadores em simultâneo, todos no mesmo labirinto. O labirinto consiste numa espécie de quadriculado em que cada posição é uma sala. Existirão portas de umas salas para as outras, permitindo um jogador passar de uma sala para a outra.

O jogo tem as seguintes características principais:

- O jogo deve suportar labirintos de dimensão 10x10 (máximo 10 salas por 10 salas). Em cada sala pode haver objectos e monstros. Existirão duas formas de gerar labirintos: aleatoriamente, ou por leitura de um ficheiro, tal como descrito mais abaixo.
- O jogo suportará a existência de "monstros" no labirinto os quais apresentam um comportamento autónomo: deslocam-se / atacam / fogem / etc. sem ser por indicação do utilizador (portanto, decididos aleatoriamente pelo computador).
- O jogo suportará a existência de objectos de vários tipos. Os objectos estarão inicialmente nas salas, podem ser apanhados e transportados pelos jogadores, e usados. Quando usados desencadeiam um determinado efeito. OS objectos são descritos mais adiante.

[1] Para mais informações acerca deste tipo de jogos: *http://pt.wikipedia.org/wiki/Multi-user_dungeon*. Importante: em caso algum o conteúdo deste artigo de wikipedia se sobrepõe ao que está escrito no enunciado.

- O jogo permite vários jogadores em simultâneo, os quais podem cooperar ou atacar-se entre si. No máximo haverá 10 jogadores. O acesso dos jogadores é validado por username/password. Estes dados existem num ficheiro de texto com um utilizador por linha, no formato "username:password". O *username* não tem nada a ver com o *username* real do utilizador na máquina e é usado para efeitos de interface com o utilizador e de identificação perante os restantes jogadores. Presume-se que esse ficheiro é criado/modificado com um editor de texto e não faz parte do trabalho estar a modificá-lo.

2. Requisitos e descrição detalhada

2.1 Arquitectura geral e lógica de funcionamento do sistema.

O jogo é obrigatoriamente constituído por vários processos a correr na mesma máquina Unix, interagindo entre si segundo a lógica cliente-servidor. Estes processos executam um de dois programas: "servidor" ou "cliente":

- **Programa "servidor"**. Este programa armazena e gere toda a informação e controla todos os aspectos centrais do jogo. Corresponde ao elemento central ("gestor") do jogo e gere toda a informação com a qual o sistema lida: labirinto, objectos, monstros, dados do jogador, etc. Só deve estar a correr um processo com este programa de cada vez. Não interage com o utilizador directamente. Não faz quaisquer perguntas via consola. Pode, quando muito, enviar mensagens para o ecrã para informação do que se passa internamente.
- **Cliente**. Este programa serve principalmente de interface com os jogadores e não valida as regras nem armazena informação local acerca do jogo (caso contrário um cliente propositadamente feito de forma viciada poderia "enganar" o servidor e fazer batota, por exemplo, fazer com que o jogador fosse imortal). Cada jogador executa um cliente. Através do "cliente" o jogador acede à funcionalidade implementada no servidor, o que inclui, por exemplo: juntar-se a um jogo, jogar, etc. O acesso de um utilizador ao jogo é controlado pelo servidor. Só são aceites os jogadores previamente identificados num ficheiro de configuração do servidor (ver mais adiante). O estilo de interacção segue a lógica consola: modo texto e comandos escritos (não é para usar o paradigma de menús). A primeira coisa que o programa faz é pedir o username ao utilizador (para identificar em termos de interface com o utilizador) e a password para verificar se tem acesso. O cliente envia o nome e password ao servidor que o validará. Só após esta validação é que os restantes comandos serão aceites.

O processo "servidor" interage com os processos "cliente" através de *named pipes* e de sinais. Os processos "cliente" não interagem directamente uns com os outros. Qualquer informação que seja necessária transferir de um "cliente" para o outro passará sempre pelo servidor, que mediará e reencaminhará essa informação.

Todos os programas são lançados através da linha de comandos. Cada programa pode ser lançado a partir de uma sessão/*shell* diferente sem que isso afecte a sua funcionalidade (mas serão sempre lançados a partir da mesma conta de utilizador da máquina):

- **Servidor**. Quando lançado, passa automaticamente para execução em *background*[2]. Não permite estar a correr mais do que uma vez em simultâneo (a segunda "cópia" detecta a situação e termina logo, ficando a correr apenas a primeira – há várias formas de detectar isto, proponha uma e use-a). O

[2] "automaticamente" significa que não é preciso estar a escrever "&" na linha de comandos.

servidor pode ser terminado pelo envio do sinal SIGUSR1 pela linha de comandos. Ao terminar, o servidor notifica os clientes para também terminarem.

O servidor recebe através de argumento de linha de comandos o nome do ficheiro contendo os *usernames* e *password* dos utilizadores, tal como mencionado acima. A manutenção desse ficheiro é feita externamente através de um editor de texto e não faz parte do trabalho. Este é um ficheiro de texto com um utilizador por linha, no formato "username:password". O *username* não tem nada a ver com o *username* real do utilizador na máquina e é usado para efeitos de interface com o utilizador e de identificação perante os restantes jogadores.

- **Cliente**. O cliente pode estar a correr muitas vezes em simultâneo com ele próprio, tantas vezes quantos os jogadores. O cliente, quando se executa, fica a interagir com o utilizador de forma habitual. Quando termina, avisa o servidor e mais nada de especial acontece: os restantes processos (inclusive outros clientes a correr) mantêm-se em execução. Todas as ordens que afectam o jogo são, evidentemente, efectuadas pelo servidor, que é quem tem acesso directo aos dados.

Importante:

➔ Em algumas situações pode ocorrer que os programas precisem de dar atenção a duas coisas em simultâneo, tal como ler um comando do utilizador e dar atenção a uma notificação oriunda do servidor informando que algo ocorreu. O mecanismo de sinais em Unix é muito útil para este tipo de situações, em que se deixa o cliente numa tarefa e, quando chega um sinal (enviado pelo servidor?) se deve ir dar atenção a outra coisa, regressando depois à primeira. Os exemplos nas aulas teóricas e práticas sobre sinais esclarecem este aspecto.

2.2 Elementos do jogo (labirinto, salas, jogadores, objectos, monstros)

2.2.1 Labirinto e salas

O labirinto tem sempre uma dimensão de 10 salas por 10 salas. Cada sala é uma espécie de quadrado, sempre com 4 paredes. Uma parede a sul, uma parede a norte, uma a oeste e uma a este. Sempre que for necessário dar alguma orientação ao utilizador, utilizar-se-ão sempre os pontos cardeais (norte, sul, este, oeste). As salas são interligadas através de portas. Cada sala tem no máximo 4 portas, uma em cada "parede". Quando for necessário listar as portas existentes, bastará dizer "uma porta a norte e uma porta a este", por exemplo. As salas têm uma pequena descrição "sala húmida", "corredor baixo", etc.). Estas descrições podem ser sorteadas para cada sala quando se faz o labirinto.

- Em cada sala podem existir até 5 objectos. O jogador pode apanhar objectos até uma certa quantidade máxima (especificado adiante). Ao ser apanhado, o objecto deixa de estar na sala e passa para o inventário do jogador que o apanhou. O jogador também pode largar objectos. Nesse caso o objecto largado passa a existir na sala em que foi largado. Se a sala já tiver o seu máximo de 5 objectos, então, o objecto largado é destruído.
- Cada sala pode ter zero, um ou dois monstros. Se um terceiro monstro tentar entrar numa sala já contendo dois monstros, então o jogo impedirá isso e o monstro irá para outra sala, ou fica quieto.
- Não há limite quanto ao número de jogadores na mesma sala (é igual ao máximo de jogadores no jogo).

Sugestão de implementação: Represente cada sala (descrição, portas, conteúdo) através de uma estrutura. Representar o labirinto através de uma matriz bidimensional.

2.2.2 Jogadores

Cada jogador representa um personagem no jogo, identificado pelo *username*, e com as características:

- Saúde: inicial 20. Máximo 30 (nunca ultrapassa este máximo).
- Objectos: O jogador pode carregar objectos até um peso total máximo de 20 Kg. Inicialmente tem um canivete e uma aspirina. O resto, terá que apanhar. Pode apanhar moedas e o seu número será a sua pontuação no final do jogo.

Quando um jogador morre, os seus objectos são largados na sala em que estava. Como existe um máximo de objectos por sala, os objectos são largados pela seguinte prioridade: primeiro as moedas, depois os restantes objectos por uma ordem qualquer. Os que não couberem na sala são destruídos.

Sugestão de implementação: Represente cada jogador numa estrutura de dados independente da matriz do labirinto. A ligação jogador–sala pode ser feita de diversa formas: ou introduzir essa informação no labirinto, ou introduzir essa informação no jogador. Ambas as estratégias têm vantagens e desvantagens.

2.2.3 Objectos

Alguns objectos exercem um efeito pelo simples facto de estarem na posse do jogador; outros precisam de ser usados explicitamente para desencadear o seu efeito. Cada objecto pode ser usado explicitamente um determinado número de vezes (transportar o objecto não conta para esse número). Alguns objectos podem ser usados de forma ilimitada. Cada objecto tem um determinado peso e o jogador pode transportar até 20 kg. Cada objecto tem um factor de "raridade". Este factor vai de 0 a 100 e corresponde à probabilidade de existir inicialmente numa sala. A probabilidade 0 significa "nunca" e 100 significa sempre. Quando mais raro, mais dificilmente existirá no labirinto. O jogador pode ter mais do que um objecto do mesmo tipo.
Devem ser suportados os seguintes objectos no jogo:

Nome	Peso	Raridade	Força ataque	Força defesa	Max usos	Quando usado	Quando carregado
Sandes	0,5	10	0	0	1	Saúde += 3	---
Aspirina	0,1	20	0	0	1	Saúde += 1	---
Xarope	1	4	0	0	1	Saúde += 4	---
Faca	2	5	5	0	Sem max	p/ ataque	---
Espada	8	3	8	2	Sem max	p/ ataque	Defesa +=2
Granada	1	2	30	0	1	p/ ataque. Saúde - = 5	---
Escudo	4	4	5		Sem max	---	Defesa +=5
Moeda	0,1	5	0	0	---	---	---

Sugestão de implementação: Representar os objectos através de uma estrutura cujos campos espelham a tabela indicada acima. A ligação objecto–sala pode mais facilmente ser feita considerando os objectos como parte dos dados da sala no labirinto.

2.2.4 Monstros.

Os monstros são personagens controlados pelo jogo (leia-se "pelo servidor"). Podem exibir um comportamento aleatório, ou podem ser medianamente inteligentes, dependendo da qualidade do algoritmo de controlo implementado pelos alunos, algoritmo esse que não será directamente sujeito a avaliação nesta disciplina. Exige-se apenas que os monstros façam qualquer coisa.

Os monstros têm uma determinada capacidade de defesa e de ataque. Estas quantidades são inicialmente sorteadas entre determinados limites inferior e superior quando o jogo se inicia (dois monstros do mesmo tipo poderão ter valores diferentes, dentro desses limites). Os monstros têm uma determinada saúde, também inicialmente sorteada entre limites. Podem ser agressivos ou passivos: um monstro passivo apenas atacará o jogador que o atacar; um monstro agressivo atacará o jogador sempre que este tentar sair da sala. Os monstros podem ser do tipo quieto ou irrequieto: um monstro quieto nunca sai da sala em que está; um monstro irrequieto desloca-se aleatoriamente (ou segundo uma lógica semi-inteligente qualquer). Quando mortos, os monstros podem largar objectos. Cada monstro existirá inicialmente em determinadas quantidades no labirinto.

O jogo deve suportar os seguintes monstros:

Nome	Força Ataque	Força Defesa	Saúde	Número inicial	Agressivo / passivo	Quieto / irrequieto	Número inicial	Ao morrer larga
morcego	1 - 4	3 - 4	4 – 5	1	Agressivo	Irrequieto	10	--
escorpião	1 - 7	5 - 7	7 – 9		Agressivo	Quieto	10	Moeda
lobisomem	5 - 7	5 - 7	7 – 9	2	Agressivo	Irrequieto	4	Faca
urso	8 - 10	10 - 12	10	2	Passivo	Quieto	2	Faca + moeda
boss	10-12	15	15	2	Passivo	quieto	1	5 moedas

Sugestão de implementação: Representar os monstros através de uma estrutura cujos campos espelham a tabela indicada acima. A ligação monstro–sala pode ser feita de diversa formas, sendo a mais fácil a inclusão do monstro dentro da estrutura de dados do labirinto.

3. Decorrer do jogo

3.1 Definição inicial do labirinto

O labirinto é criado de forma aleatória pelo servidor no início do jogo. Criar um labirinto significa sortear (completamente aleatório, ou mediante uma estratégia inteligente qualquer) os seguintes itens:

- **Salas**: quais as portas existentes
- **Objectos**: quais os objectos nelas existentes (que objectos, quais as suas propriedades e quantos)
- **Monstros**: quais (propriedades) e quais as suas posições

O labirinto pode ser visto como um quadriculado de 10x10. O topo será o norte, a parte de inferior será o sul. Esquerda é oeste e direita é este. Mantendo o paradigma de quadriculado de 10x10, cada sala tem uma coordenada implícita de linha, coluna, sendo o canto superior esquerdo a posição linha 0 e coluna 0 (tal como nas matrizes). A linha e coluna da sala não são apresentadas ao jogador.

Existirá sempre uma sala considerada "início", cujas coordenadas são sorteadas pelo jogo. Os jogadores são inicialmente colocados nessa sala. O jogo avisa o jogador sempre que entra (volta à) na sala "inicio".

3.2 Leitura inicial de labirinto de um ficheiro de texto[3]

O jogo deve permitir a definição do labirinto a partir de um ficheiro previamente existente na directoria de trabalho do servidor. Para simplificar esta operação, os labirintos no ficheiro terão sempre 10 objectos e 10 monstros, e todas as salas terão automaticamente o nome de "sala". O formato do ficheiro é o seguinte:

- **Salas**: As salas são definidas nas linhas 1 a 10, 10 salas por cada linha, 4 caracteres por cada uma. Cada caracter será um 'P' ou um '.', consoante existe ounão uma porta nas direcções (por esta ordem) Norte, Sul, Este, Oeste.
- **Monstros**: cada monstro é especificado nas linhas 11 a 20. Cada linha tem a seguinte informação

  nome linha coluna

  nome é o nome do monstro, linha e coluna são dois valores que identificam as coordenadas da sala onde o objecto se encontra.
- **Objectos**: cada objecto é especificado nas linhas 21 a 30. Cada linha tem a seguinte informação

  nome linha coluna

  nome é o nome do objecto, linha e coluna são dois valores que identificam as coordenadas da sala onde o objecto se encontra.
- **Sala início**. A sala "início" é identificada na linha 31, que tem o formado

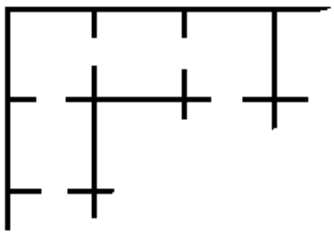
  linha coluna

  linha e coluna são os valores das coordenadas da sala "início"

Exemplo parcial de ficheiro de labirinto

```
.PP.  .P.P  ..PP   etc
P.P.  etc
etc
faca 0 0
moeda 0 0
espada 5 6
etc
lobisomem 2 3
boss 7 8
etc
4 5
```

O ficheiro à esquerda representa um mapa com uma faca e uma moeda na sala (0,0), uma espada na sala ((5,6), um lobisomem na sala (2,3) e um boss na sala (7,8). A sala de início é a (4,5).
O canto norte-oeste do labirinto, descrito no exemplo (o resto não foi apresentado), seria:



3.3 Criação de um jogo

O primeiro jogador a ligar-se ao servidor adquire poderes especiais. Nomeadamente, pode criar e terminar o jogo. Este jogador é referido aqui como "primeiro jogador". Se o "primeiro jogador" sair, então o servidor elege automaticamente o jogador seguinte como "primeiro jogador", notificando esse jogador/cliente que adquiriu esse estatuto especial.

[3] Recordar que, em Unix, não existe diferença real entre ficheiros de texto e ficheiros binários quanto às operações de abertura, leitura, escrita, etc. Ao Dizer-se que o ficheiro é de texto, está-se a dizer que o conteúdo do ficheiro é texto (em vez de ser, por exemplo, uma imagem, um trecho sonoro, etc.).

A criação de um novo jogo é feita mediante um comando enviado pelo primeiro jogador, ficando esse jogador logo a participar no jogo. Após a sua criação, o servidor aguarda um intervalo de tempo durante o qual aceita novos jogadores. O número de segundos a aguardar é especificado na criação do jogo.

O comando de criação de jogo tem duas sintaxes possíveis:

- ***novo valor-timeout nome-ficheiro***
  Diz ao servidor para cria um novo jogo, lendo o labirinto a partir do ficheiro cujo nome foi especificado. *Timeout* é usado para estabelecer o período, em segundos, durante o qual serão aceites mais jogadores.

- ***novo valor-timeout valor-dificuldade***
  Diz ao servidor para criar um novo jogo, criando o labirinto, monstros e objectos aleatoriamente. O valor especificado estará entre 1 e 10 e indica o nível de dificuldade. O valor poderá ou não ser levado em conta pelo servidor. O nível de dificuldade não é uma característica obrigatória (se não for implementada, o valor é ignorado, mas deve ser especificado na mesma). O valor de *timeout* é usado para estabelecer o período, em segundos, durante o qual serão aceites mais jogadores.

O comando de criação de novo jogo é ignorado se já estiver um jogo a decorrer.
Após a criação de um jogo, são aceites novos jogadores até que seja atingido o limite de jogadores (10 jogadores) ou até que o período de *timeout* se esgote.

Quando um jogo é criado, todos os utilizadores ligados ao servidor são avisados ("avisados" significa que o utilizador recebe uma mensagem no ecrã).

Qualquer jogador pode associar-se ao jogo escrevendo o comando

- ***jogar***
  O comando será aceite se ainda não tiver sido atingido o número máximo de jogadores, nem o *timeout* se tiver esgotado. Nota: o servidor informa sempre o cliente/jogador acerca do resultado de qualquer comando enviado pelo utilizador.

3.4 Terminação de um jogo

O jogo termina assim que um jogador sai do labirinto. Todos os jogadores são avisados, aparecendo a identificação de qual o jogador que sai e quantas moedas tem. Para sair, o jogador tem que estar na sala inicial e usar o comando

- ***sair***
  Sai do labirinto. Só tem efeito se estiver na sala "início". Tendo saído, o jogo termina para todos, mas ligação dos clientes ao servidor continua activa e podem participar noutro jogo. Todos os jogadores são notificados com a informação de quem saiu e quantas moedas tinha. Para voltar a jogar será necessário que o primeiro jogador crie um novo jogo.

O jogo também pode ser terminado explicitamente através de um comando enviado pelo primeiro jogador:

- ***terminar***
  (Só para o primeiro jogador) Termina o jogo, independentemente do seu estado. Todos os jogadores ligados são avisados, sendo informados de qual o jogador que tinha mais moedas.

Qualquer jogador poderá desistir do jogo, através do comando

- ***desistir***

  Este comando faz o jogador abandonar o jogo. Se o jogador não estivesse a jogar, o comando é ignorado. Os restantes jogadores são avisados (exemplo: "o jogador xyz abandonou o jogo"). Quando o último jogador abandona o jogo, este é encerrado. Quando um jogador abandona o jogo, o seu processo cliente continua a correr e ligado ao servidor. Sair do jogo não significa "encerrar o cliente".

Antes do jogo ter início apenas são aceites pelo servidor os comandos:

- ***novo*** (pelo primeiro jogador). Fica automaticamente associado a esse jogo
- ***jogar*** (por qualquer outro utilizador que queira participar no jogo)

Durante o jogo são aceites:

- Os comandos de jogo, propriamente ditos, a especificar mais adiante
- ***desistir*** (para abandonar o jogo)
- ***terminar*** (pelo primeiro jogador, para encerrar o jogo)

Após o jogo terminar, está-se como de antes de jogo começar, ou seja, antes do **próximo** jogo começar.

Qualquer jogador pode abandonar o jogo. Para tal, usa-se o comando

- ***logout***

  Este comando encerra o cliente. O servidor continua a executar mesmo que não haja nenhum cliente. O comando só tem efeito se o jogador não estiver a meio de um jogo. Se estiver, deve abandonar o jogo primeiro (comando "desistir"). Se o primeiro jogador sair, o servidor deve eleger o jogador seguinte como "primeiro jogador" informando esse jogador/cliente que adquiriu esse estatuto.

3.5 Decorrer do jogo - comandos

Os comandos de jogo são aqueles destinados a serem usados durante o jogo e que permitem, mover, apanhar e usar objectos, ver o que rodeia o jogador, e interagir com inimigos e jogadores – ou seja, jogar.

Ver

- Comandos ***ver*** , ***ver nome-monstro*** , ***ver nome-objecto*** , ***ver nome-jogador***

  Obtém a descrição do objecto, monstro ou jogador indicado (desde que estejam na sala). Se não se indicar objecto/monstro/jogador, então dá a descrição da sala. No caso de:

  - Sala: indica informação igual à obtida quando se entra na sala (incluindo obrigatoriamente a indicação das saídas, dos jogadores, dos monstros e dos objectos existentes na sala).
  - Monstro: indica pelo menos a saúde, força de ataque e de defesa.
  - Objecto: indica pelo menos a descrição e o peso.
  - Jogador: descrição do jogador incluindo a saúde, força de ataque e de defesa. Não indica os objectos que transporta.
- ***info***

  Fornece dados acerca do próprio jogador, incluindo o seu inventário e a sua saúde.

Movimento

- Comandos ***norte*** , ***este*** , ***sul*** , ***oeste***
  Movimenta o jogador para a sala na direcção indicadapelo nome do comando. Se não houver porta nessa direcção, o comando é ignorado. Se existir algum inimigo agressivo na sala actual, o inimigo ataca o jogador, e o movimento fica sem efeito.

Interacção com o jogo

- ***apanha nome-objecto***
  Apanha o objecto indicado, desde que exista na sala e ainda caiba no inventário do jogador.
- ***usa nome-objecto***
  Usa o objecto indicado, desde que se encontre no inventário. Os efeitos do objecto são accionados.
- ***larga nome-objecto***
  Larga o objecto indicado, passando do inventário para a sala. Se não couber na sala, é destruído.
- ***ataca nome-monstro objecto-arma***
  Ataca o monstro indicado, desde que este esteja na sala, usando o objecto (do inventário) como arma. Se houver dois monstros com esse nome (que corresponde ao seu tipo) na sala, ataca um deles. Tente evitar que haja dois monstros do mesmo tipo na mesma sala (não é obrigatório).
- ***ataca nome-jogador objecto-arma***
  Ataca o jogador indicado (*username*), desde que este esteja na sala, usando o objecto (do inventário) como arma. O jogador atacado contra-ataca automaticamente da mesma forma como se fosse um monstro. O jogador atacado é notificado que está a ser atacado e a contra-atacar.
- ***diz etc-etc***
  O jogador "diz" qualquer coisa. Aquilo que disse é enviado para todos os jogadores que estejam na mesma sala que ele. A mensagem é tudo aquilo que tiver sido escrito À frente do comando "diz".
- ***grita etc-etc***
  O jogador "grita" qualquer coisa. Aquilo que gritou é enviado para todos os jogadores, mesmo que estejam noutra sala. A mensagem é tudo aquilo que tiver sido escrito à frente do comando "grita". Gritar é muito cansativo e consome 3 pontos de saúde.

Genéricos

- ***sair***
  Se estiver na sala "início", sai do labirinto. Para este jogador o jogo terminou, tal como descrito acima. Se não estiver na sala "início" o comando não tem efeito.
- ***desistir***
  Abandona o jogo, mas mantém a ligação do cliente ao servidor activa.
- ***terminar***
  Apenas para o primeiro jogador. Encerra o jogo.
- ***quem***
  Indica quem está "logado" no jogo, e destes, quem é que está actualmente a jogar.

3.6 Feedback dado ao utilizador

Sempre que algo acontece, o jogo informa o utilizador. Isto implica duas coisas:

1. O servidor, que é quem controla tudo, informa o cliente o(s) cliente(s) que algo ocorreu.
2. O cliente, recebendo o sinal/mensagem do servidor, informa o utilizador.

Aqui poderá ser necessário em simultâneo o mecanismo de sinais Unix e o mecanismo de *named pipes* Unix.
Sempre que um jogador desencadeia uma operação (comando), naturalmente receberá feedback acerca do que fez. Isto pode ser, por exemplo, entrar numa sala e ter como resposta a descrição da sala onde entrou. É de notar que uma acção feita por um jogador poder influenciar outro jogador, devendo esse jogador ser notificado (ver ponto seguinte).

3.7 Passagem de tempo

O tempo decorre apenas e sempre que um jogador faz algo. Se nenhum jogador introduzir um comando, nada acontece no labirinto. Havendo vários jogadores, pode ser que aconteça algo em função de um jogador ter desencadeado uma acção, mesmo os restantes não tendo feito nada. Isto pode levar a algumas situações tais como os seguintes exemplos:

- O jogador A entre numa sala onde já se encontrava o jogador B. Naturalmente, o jogador A apercebe-se disso, pois a descrição da sala inclui a existência do jogador B que já lá estava. No entanto, o jogador B, mesmo sem ter feito nada, deverá receber um aviso que o jogador A entrou na sala. Isto não implica nenhum trabalho adicional: o jogo notifica sempre todos os jogadores do que se passa, e cada cliente vê se a informação lhe interessa. Este aspecto não deverá ficar esquecido.
- Há dois jogadores na sala. Existe uma moeda e ambos a viram. O jogador A apanha a moeda. Quando o jogador B tentar apanhar a moeda, o servidor diz que a moeda não existe, pois já foi apanhada. O servidor valida sempre os comandos e, portanto, este aspecto não acrescenta nada de especial.

3.8 Mecânica dos combates

Os combates são do tipo duelo e sempre até à morte de um dos intervenientes. Cada combatente desfere um golpe à vez na lógica do ataque - contra-ataque. Esto repete-se até que a saúde de um dos combatentes chegue a 0. A forma calcular o dano e defesa é a seguinte:

- O dano infligido ao inimigo com um golpe é um valor aleatório entre 0 e força de ataque (atacante) – valor aleatório entre 0 e 0,5 x força de defesa (do atacado). Não há danos negativos.
- A força de ataque é calculada como força de ataque da arma usada. A força de defesa é calculada com base nos objectos de protecção que estão em uso por quem recebe o golpe.

O combate é processado pelo servidor. Deve haver feedback ao jogador acerca de todos os golpes dados.

3.9 Valores usados no jogo

Os valores usados no jogo especificados aqui (pesos, forças de ataque e defesa, raridade dos objectos, etc.) **podem** ser modificados desde que não alterem a mecânica do jogo e não simplifiquem os mecanismos a usar. No entanto, se alterados, devem ser acompanhados de uma explicação legítima acerca da razão pela qual foram alterados.

4. Considerações finais

- Podem existir diversas situações de erros potenciais que não são explicitamente mencionadas no enunciado. Estas situações devem ser identificadas e o programa deve lidar com elas de uma forma controlada e estável. O terminar abruptamente o programa não é considerado uma forma adequada.
- Caso haja *bugs* no enunciado, serão corrigidos e anunciada a sua correcção. Poderá haver lugar a documentos adicionais no *moodle*, nomeadamente um *Frequently Asked Questions* com as perguntas mais frequentes e respectivas respostas.
- Vai ser dada uma **aula (teórica) suplementar de apoio** à elaboração do trabalho: dia **6 de Maio, 16h30** A.1.1, para o diurno e pós laboral em simultâneo (a seguir à aula extra das 14h30 para a o diurno).

5. Regras gerais

Aplicam-se as seguintes regras (descritas na primeira aula e na ficha de unidade curricular):

- O trabalho pode ser realizado em grupos de um ou de dois alunos (sem excepções, não vale a pena enviar e-mails com pedidos nesse sentido). No caso de ser realizado individualmente, o trabalho é o mesmo e não será simplificado (o trabalho já foi dimensionado a pensar nesse caso).
- O trabalho vale cinco valores na nota final da disciplina. Tem mínimos de 25% que serão dispensados para os alunos que tenham 80% de presenças nas aulas práticas.
- **Prazo de entrega**: final do dia de **6 de Junho (Sábado), 23:00h** via moodle.
    - Atraso máximo sem descontos: **8 de Junho, 9h00 (Segunda de manhã)**.
    - Após o dia 8, há desconto de 25% por cada dia a mais.
- A entrega do trabalho é feita sob a forma de um arquivo **zip** (leia-se: "**zip**". Não é arj, rar, tar, ou outros) contendo os ficheiros do projecto (código fonte) e o PDF do relatório. O arquivo tem obrigatoriamente o nome

  so_1415_tp_nome1_numero1_nome2_numero2.zip

  em que ***nome1*** e ***numero1*** são o nome e número de um dos alunos do grupo e ***nome2*** e ***número2*** dizem respeito ao segundo aluno do grupo (se existir).

O formato e nome do ficheiro é importante para agilizar as defesas e para perceber a quem diz respeito cada trabalho. A não conformidade com este formato pode originar a seguinte penalização:

- Outro formato que não o zip – 5% de penalização
- Não conformidade no nome do ficheiro – 5% de penalização

Evite problemas e respeite este requisito.

5.1 Avaliação do trabalho

Para a avaliação do trabalho serão tomados em conta os seguintes elementos:

- **Arquitectura do sistema** – Embora não haja muita margem para inovação dado a simplicidade estrutural do problema, há aspectos relativos à interacção dos vários processos que devem ser cuidadosamente planeados de forma a se ter uma solução elegante, leve e simples. A arquitectura deve ser bem explicada no relatório para não haver mal entendidos.
- **Implementação** – Deve ser racional, não desperdiçar recursos do sistema. As soluções encontradas para cada problema no trabalho devem ser claras. O estilo de programação deve seguir as boas práticas de programação. O código deve ter comentários relevantes. Os recursos do sistema devem ser usados de acordo com a sua natureza.
- **Relatório** – Deverá ser escrito e entregue um relatório completo descrevendo a estratégia e modelos seguidos, a estrutura da implementação e as opções tomadas (máximo de 10 páginas). Este relatório será enviado juntamente com o código no arquivo submetido via *moodle* e também será entregue em mão (i.e., impresso) na defesa.
- **Defesa** – Os trabalhos são sujeitos a defesa individual onde será testada a autenticidade dos seus autores. Pode haver mais do que uma defesa caso subsistam dúvidas quanto à autoria dos trabalhos. A nota final do trabalho é directamente proporcional à qualidade da defesa. Elementos do mesmo grupo podem ter notas diferentes consoante o empenho individual na sua realização.

  Nota: apesar da defesa ser individual, ambos os elementos do grupo devem comparecer à defesa ao mesmo tempo. A falta à defesa implica automaticamente a perda da totalidade da nota do trabalho.
- Os trabalhos que não funcionam são fortemente penalizados independentemente da qualidade do código fonte apresentado. Trabalhos que nem sequer compilam terão uma nota extremamente baixa.
- A identificação dos elementos de grupo deve ser clara e inequívoca (tanto no arquivo como no relatório). Trabalhos anónimos não são corrigidos.