Sistemas Operativos 17/18

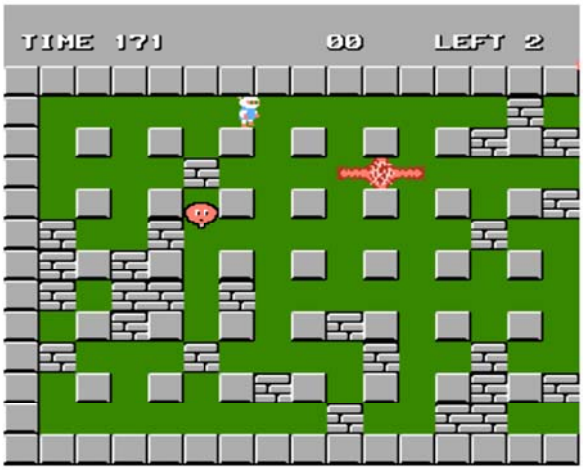
Trabalho Prático - Programação em C para UNIX

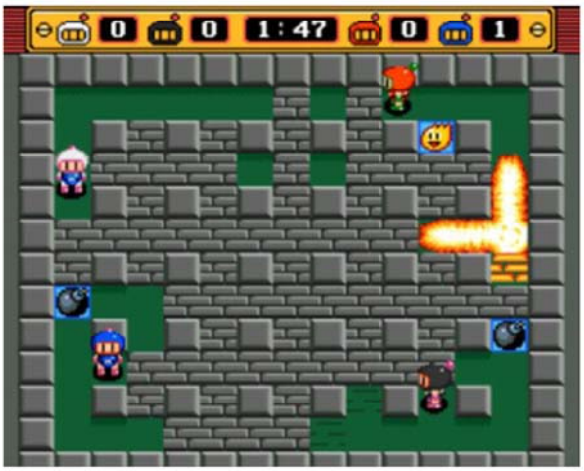
O trabalho prático consiste na implementação do jogo "*Bomberman*" para um contexto multijogador. A interface será em consola (modo-texto), e todos os jogadores estarão ligados à mesma máquina UNIX, ou seja, não se pretende a utilização de mecanismos de comunicação em rede – apenas mecanismos de comunicação inter-processo. A exploração da funcionalidade completa do trabalho pode exigir diversas ligações remotas via *putty*, mas os programas do trabalho estarão sempre numa única máquina UNIX.

O enunciado começar por descrever primeiro o funcionamento do jogo e, no fim do documento, são indicadas as regras e datas importantes.

1. Descrição geral

O trabalho prático consiste na implementação de uma versão simplificada do jogo "*Bomberman*" para ambiente de consola (modo-texto). Este jogo é extremamente simples já que, de uma forma resumida, o jogador apenas controla um personagem (o "*bomberman*") que se desloca num labirinto, tentando cumprir uma missão que consiste em apanhar objetos e sair desse mesmo labirinto. A palavra labirinto não reflete a simplicidade envolvida aqui: o labirinto é simplesmente um campo onde existem alguns obstáculos, consistindo assim num quadriculado em que algumas posições estão ocupadas por paredes e outras estão vazias. As posições vazias constituem os caminhos por onde o jogador se pode movimentar. No labirinto existem alguns inimigos que matam o jogador através do contacto. Estes inimigos possuem um movimento próprio que deve ser definido por um algoritmo muito simples (comportamento aleatório ou simplesmente perseguindo o jogador), e com este movimento devem tentar encurralar e matar o jogador. Excluindo o grafismo (que não é exigido neste trabalho), as imagens abaixo dão uma ideia do cenário descrito:





Em relação ao resultado final deste trabalho é importante notar o seguinte:

- Apesar deste jogo estar associado à ideia de gráficos, neste trabalho prático **apenas se exige uma implementação com interface em consola (modo-texto)**:
    - Em vez de imagens e ícones deverão ser utilizados caracteres e cores (de texto e de fundo) para representar cada um dos elementos do jogo (paredes, objetos, inimigos, etc.).
- As regras do jogo em si (modo de pontuação, comportamento exato dos "inimigos", etc.) são pouco relevantes para o trabalho, já que o foco deste são os mecanismos do sistema UNIX e a sua utilização. Desta forma, aceitam-se pequenas variações das regras desde que não impliquem uma redução de âmbito, dimensão ou dificuldade do trabalho.

O jogo tem as seguintes características principais:

- As dimensões do labirinto devem permitir que este caiba na sua totalidade no ecrã. No entanto, este tem obrigatoriamente de ter, pelo menos, 20x30 posições (poderá ter, opcionalmente, mais).
- O labirinto pode ser visto como um quadriculado definido pela ocupação das suas quadrículas. Posições vazias constituem as passagens. Existem também blocos (obstáculos) que podem ser destruídos, passando assim as suas posições a ficarem "vazias" (dando origem a novas passagens), e por blocos não destrutíveis. Existirá também uma posição que será a "saída" do labirinto: quando o jogador atinge essa posição o labirinto é considerado "ganho", terminando o jogo ou passando para um labirinto mais difícil (por exemplo, com mais inimigos). A "saída" só tem efeito se o jogador tiver apanhado todos os objetos (ver a seguir).
- Algumas passagens (posições "vazias") deverão conter inicialmente objetos que devem ser apanhados pelo jogador. Para apanhar um objeto o jogador passará pela posição que o contém. Os inimigos não interagem com estes objetos. Apanhar um objeto acrescenta um novo ponto à pontuação do jogador. O número e a posição inicial dos objetos são características do labirinto e estas características podem ser usadas para definir a sua dificuldade.
- As posições podem ser atravessadas livremente pelo "*bomberman*" (jogador) e pelos inimigos. Os blocos destrutíveis podem ser destruídos pela detonação de uma bomba nas imediações, dando origem a uma posição "vazia". As bombas são colocadas pelo jogador (mais sobre esta questão é dito a seguir). Os blocos "não destrutíveis" (também chamados de "paredes") não são afetados por bombas.
- O número de inimigos e o número de objetos a apanhar em cada labirinto são definidos da seguinte forma:
    - Para o labirinto inicial (ou seja, o 1º do jogo), se existirem as variáveis de ambiente NOBJECT e NENEMY com valores inteiros válidos, o jogo usará o valor da primeira para o número de objetos e o valor da segunda para o número de inimigos. De referir que este labirinto não deve ser demasiado "básico" para que o jogo não se torne aborrecido. Fica a cargo dos alunos tomarem a melhor decisão para esta questão.

- Ainda para o labirinto inicial, se as variáveis de ambiente referidas no ponto anterior não existirem, o jogo deverá definir estes valores de forma aleatória, dentro de limites razoáveis.
- Em ambos os casos anteriores, sempre que o jogador completa um labirinto o jogo define um novo labirinto com mais um inimigo e mais um objeto. Estes valores só são incrementados enquanto existir espaço no quadriculado. A partir do momento que deixar de existir espaço o labirinto anterior deve ser considerado como o último e o jogo deve terminar.

- O jogador pode colocar bombas nas posições vazias do labirinto. Após uma bomba ser colocada esta explode, eliminando inimigos, jogador(es) e blocos destrutíveis que estejam ao seu alcance. Existem dois tipos de bombas: a "bombinha", com alcance de 2 quadrículas em redor da posição onde foi colocada, e a "mega bomba", com alcance de 4 quadrículas em redor da sua posição (este alcance aplica-se a todas as direções). Em ambos os casos, a bomba explode sempre 2 segundos depois de ter sido colocada e a explosão "dura" outros dois segundos. Durante a explosão, as quadrículas ao alcance da bomba são letais para os jogadores e para os inimigos. Desta forma, convém fugir para longe assim que se coloca uma bomba e aguardar que a sua explosão termine antes de se voltar ao local. O jogador tem inicialmente 2 "mega bombas" e 3 "bombinhas". O jogador pode colocar novas bombas mesmo que as que colocou anteriormente ainda não tenham explodido.
- Um inimigo ao morrer pode eventualmente largar um item que os jogadores podem apanhar. Detalhando o que o inimigo pode largar:
  - Uma "mega bomba" (acontece raramente).
  - Uma "bombinha" (acontece raramente, mas não tanto como uma "mega bomba").
  - Um coletor automático de objetos que faz com que os primeiros 5 objetos encontrados pelo programa sejam apanhados pelo jogador, o que irá permitir que este termine o labirinto mais depressa (é invulgar de acontecer).
  - Um extra que adicionada 10 pontos à pontuação do jogador (é vulgar de acontecer).
  - Um extra que adiciona uma vida ao jogador (é extremamente raro e é opcional, já que só faz sentido se os alunos estiverem a implementar o conceito de número de vidas).
  - São aceites outros extras, com diferentes efeitos, propostos pelos alunos. Sejam criativos e divirtam-se, mas convêm que grupos diferentes inventem coisas diferentes.
- O jogo suporta vários jogadores em simultâneo. Neste caso os jogadores colaboram entre si, sendo que se considera a missão cumprida mesmo que cada jogador tenha apenas apanhado uma parte dos objetos (o que interessa é que todos os objetos tenham sido apanhados). Assim sendo, o labirinto é ganho assim que um dos jogadores ativa a saída. A existência de mais do que um jogador em simultâneo obriga ao uso de uma máquina para cada um (para se terem vários teclados). Isto apenas exige uma ligação remota via *putty* e não acrescenta nada de especial ao trabalho.
- Cada jogador é identificado por um par *username*/*password* e o acesso ao jogo é validado. A informação *username*/*password* de cada jogador é armazenada num ficheiro de texto.
- Não podem existir dois inimigos na mesma posição nem dois jogadores na mesma posição. Um jogador e um inimigo na mesma posição só acontece em teoria porque o jogador morrerá logo.

2. Arquitetura geral

O jogo é obrigatoriamente constituído por vários processos a correr na mesma máquina UNIX, processos esses que interagem entre si através de uma arquitetura cliente/servidor. Estes processos executam um de dois programas: "servidor" ou "cliente".

Servidor

Este programa armazena e gere toda a informação relativa ao jogo, controlando todos os aspetos deste. Só pode estar a correr um processo com este programa de cada vez.
Este programa não é destinado ao jogador, mas sim a um utilizador que configura/controla o jogo, permitindo executar ações administrativas como a gestão de utilizadores e do jogo que está a decorrer. A interação com o utilizador é feita através do paradigma de linha de comandos.
O servidor controla todos os inimigos, sendo que cada inimigo corresponderá a uma *thread*. Para além disso, o servidor é também responsável por controlar as bombas depois destas serem colocadas por um jogador (ou seja, fazer decorrer o seu tempo para explodir, causar a explosão e gerir os seus efeitos).

Como já foi referido, o servidor pode ser controlado diretamente por um administrador através de uma interface semelhante a uma linha de comandos. Os comandos que o servidor reconhece são os seguintes:

- **add username password**
  Cria uma conta nova com o *username* e a *password* especificados, a não ser que já exista uma conta com o mesmo *username*. A informação relativa ao novo jogador deve ser acrescentada ao ficheiro de texto que armazena a informações de todos os jogadores.
- **users**
  Mostra a lista dos jogadores ativos (os que estão atualmente a jogar).
- **kick username**
  Termina a ligação do jogador identificado. O utilizador não é eliminado do ficheiro de texto, mas é removido do jogo (se estivesse a jogar), a sua ligação ao servidor é encerrada, e o seu cliente deve encerrar de forma controlada, avisando o jogador do sucedido.
- **game**
  Mostra informação acerca do jogo que está a decorrer: quais os jogadores e as suas pontuações, quantos objetos ainda existem por apanhar no labirinto atual, etc.
- **shutdown**
  Desliga o servidor. O servidor deve terminar o jogo e informar os clientes desse facto. Toda a informação relevante deve ser armazenada em disco e os recursos temporários que estão em utilização devem ser eliminados.
- **map *nome-ficheiro***
  Carrega a definição de um labirinto a partir de um ficheiro em disco. O novo labirinto será usado na vez imediatamente seguinte em que o jogo gerar um labirinto (ou seja, a próxima vez que o jogo

precisar de gerar um labirinto, usará o que carregou do ficheiro. Este comando é opcional e pode ser considerado como uma funcionalidade extra. A sua implementação pode ser usada para recuperar pontuação perdida noutra parte.

Cliente

Este programa serve de interface com os jogadores, sendo responsável pela interação com estes e por apresentar o labirinto atual do jogo. Sendo a sua principalmente missão interagir com o jogador, este programa **não valida quaisquer regras nem armazena informação localmente** acerca do jogo. Cada jogador executa um e só um cliente. A quando da execução, o jogador insere o seu *username* e a sua *password* e, se estes dados forem aceites pelo servidor, o jogador junta-se automaticamente ao jogo que estiver a decorrer.

Todos os clientes apresentam uma visão atualizada do que se passa no jogo, independentemente de o jogador estar a usar furiosamente o teclado ou estar simplesmente a apreciar o que se passa enquanto toma tranquilamente um chá (por outras palavras, a interação entre cliente e servidor não é gerida diretamente pela interação entre o jogador e o cliente).

Interface com o utilizador

Os requisitos da interface com o utilizador são, de uma forma geral, os seguintes: deve ser fácil de utilizar, deve fazer sentido e deve permitir usar todas as funcionalidades do jogo. As seguintes perguntas e respostas (*Q&A*) permitem clarificar este aspeto:

- ***Q.*** *Que teclas devem ser usadas para controlar o "bomberman"?* ***A.*** *Sei lá. Tanto faz. Desde que dê para mexer o "bomberman" em condições, que tenha uma tecla para cada direção, uma para largar uma bombinha, outra para largar uma mega bomba, etc., está bom.*

- ***Q.*** *Como é que represento o labirinto no ecrã?* ***A.*** *Tanto faz desde que dê para ver o que se passa: onde anda(m) o(s) jogador(es), onde estão os inimigos, quais as passagens que ainda têm objetos, quais são os blocos destrutíveis e quais são os blocos indestrutíveis, onde está a saída (ativada ou desativada), olha está ali uma vida extra que um inimigo deixou cair, olha está ali uma explosão a decorrer, etc. Use caracteres e cores (tanto de texto como de fundo) para conseguir isto (se achar que a sua interface é feia ou difícil de usar, se calhar o professor também vai achar).*

O processo "servidor" interage com os processos "cliente" apenas através de *named pipes* e *sinais*. Os processos "cliente" não interagem diretamente uns com os outros. Qualquer informação que se queira transferir de um "cliente" para outro terá sempre de passar pelo servidor, sendo que este mediará e reencaminhará essa informação para o "cliente" destino.

Todos os programas são lançados através da linha de comandos. Cada programa deverá ser lançado a partir de uma sessão/*shell* diferente (localmente ou através do *putty* para jogadores remotos), mas utilizando a mesma conta de utilizador na máquina.

3. Requisitos e descrição adicionais

Os requisitos aqui apresentados definem aspetos que deve obrigatoriamente cumprir. Servem também para entender a lógica geral e deduzir algum aspeto omisso. Alguma informação aqui apresentada já foi indicada antes, mas pode usar esta repetição para incorporar melhor os requisitos.

Servidor

- Apenas deve existir um processo "servidor" em execução. Caso seja executada uma nova instância do "servidor", esta deverá detetar a existência da primeira e terminar de imediato.
- Quando lançado, o "servidor" recebe através da linha de comandos o nome de um ficheiro de texto que contém os *usernames* e as *passwords* dos utilizadores. Este ficheiro é lido e gerido pelo "servidor", sendo também da sua responsabilidade acrescentar-lhe os novos utilizadores. Inicialmente, o ficheiro pode estar vazio.
- Quando termina, o "servidor" deve informar todos os "clientes" do sucedido.
- O "servidor" deverá terminar caso receba o sinal SIGUSR1.
- Os inimigos são controlados por *threads*. É obrigatório o uso deste mecanismo para este efeito. O movimento dos inimigos deve ser minimamente racional: não pode ser demasiado lento nem demasiado rápido e não pode ficar parado num canto do labirinto.
- O acesso das várias *threads* aos dados deve ser acautelado através de mecanismo de sincronização (*mutexes* ou semáforos).
- O "servidor" pode ser controlado por um utilizador (administrador) através dos comandos escritos que estão identificados na arquitetura geral.
- O "servidor" poderá ter que dar atenção a mais do que uma entrada de dados em simultâneo (por exemplo, *pipes* e teclado). Esta situação deve ser resolvida de forma elegante usando os mecanismos estudados (*threads*, *select*, etc.).

Cliente

- Podem estar a correr vários processos do "cliente" em simultâneo (um por jogador).
- Existe um número máximo de jogadores que podem estar ativos em simultâneo. Este número é definido pela variável de ambiente NMAXPLAY. Pode assumir que esta variável terá sempre um valor entre 1 e 20.
- A qualquer instante o jogador poderá abandonar o jogo ou mesmo abandonar o programa de todo. O servidor deverá ser informado desta situação e agir em conformidade.
- A interface do cliente é totalmente em "modo-texto". A apresentação do labirinto, dos inimigos, do "*bomberman*", das explosões e dos demais elementos exige um controlo que ultrapassa a funcionalidade das bibliotecas C *standard*. **A construção da interface do cliente será feita com recurso à biblioteca *ncurses*.** Será fornecida alguma informação adicional sobre esta biblioteca.

- O uso da biblioteca gráfica *Qt* será recompensado com cotação extra que poderá atingir no máximo 15%. Não será dito praticamente nada acerca desta biblioteca nas aulas, e por essa razão é que a sua utilização é considerada e recompensada como extra.

- O cliente manterá uma visão do jogo permanentemente atualizada: mesmo que o jogador desse cliente não faça nada os outros jogadores ou os inimigos poderão estar a fazer algo, e o cliente manterá o jogador sempre atualizado. Este aspeto não é complexo, mas exige algum planeamento inicial quanto à lógica de comunicação (ver também o próximo ponto).
- O cliente poderá ter que dar atenção a mais do que um assunto (em particular, a várias entradas de dados) em simultâneo (por exemplo, informações do utilizador, teclado, etc.). Esta situação deve ser resolvida de forma elegante usando os mecanismos estudados (*threads*, *select*, etc.).

4. Extras

O trabalho contempla um conjunto de extras que podem elevar a nota final do mesmo até um máximo de 115%. Estes extras poderão também compensar partes do trabalho menos bem conseguidas. Os extras são:

- "Inteligência artificial" nos inimigos controlados pelo servidor. Entende-se por "inteligência artificial" a capacidade de perseguir e cercar o "*bomberman*" (10%).
- Diferentes labirintos e capacidade de os carregar em *runtime* – comando *map* (5%).
- Labirintos maiores do que o que se consegue ver no ecrã, fazendo com que cada cliente veja apenas a parte do mapa à volta da posição do seu personagem, e deslocando essa zona de visibilidade à medida que se desloca para outras partes do labirinto (5%).
- Uso da biblioteca *Qt* (15%)

Os alunos podem implementar quantos extras entenderem, mas a cotação para a totalidade destes extras será sempre no máximo 15%. Os extras são opcionais e, portanto, não terão o mesmo nível de apoio por parte dos professores que o resto dos requisitos.

5. Considerações adicionais

Os alunos devem ter em conta as seguintes considerações finais:

- Podem existir diversas situações e potenciais erros que não são explicitamente mencionados no enunciado. Estas situações devem ser identificadas e os programas devem lidar com elas de uma forma controlada e estável. Um programa terminar abruptamente perante uma destas situações não é considerada uma forma adequada.
- Caso existam *bugs* no enunciado estes serão corrigidos e a sua correção será anunciada. Poderá haver lugar a documentos adicionais no *moodle*, nomeadamente um *Frequently Asked Questions* com as perguntas mais frequentes e respetivas respostas.

6. Regras gerais do trabalho, METAS e DATAS IMPORTANTES

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

- O trabalho pode ser realizado em grupos de um ou de dois alunos (sem exceções, e por isso não vale a pena os alunos enviarem e-mails com pedidos nesse sentido). No caso de ser realizado individualmente, o trabalho (e a valorização) é o mesmo.
- A valorização do trabalho, a sua obrigatoriedade e os mínimos a cumprir (bem como a respetiva consequência da não obtenção dos mínimos) corresponde ao que está descrito na FUC.
- Existe uma defesa obrigatória, também descrita na FUC. A defesa será efetuada em moldes a definir via *moodle* na altura em que tal for relevante.
- Existem três metas ao todo, tal como descrito na FUC. As datas e requisitos das metas são indicados mais abaixo. Em todas as metas a entrega é feita via *moodle* através da submissão de um único **arquivo *zip***[1] cujo **nome** segue a seguinte lógica[2]:

  so_1718_tp_*nome1_numero1_nome2_numero2*.zip

  (evidentemente, nome e número serão os nomes e números dos elementos do grupo)

  A não aderência ao formato *zip* e ao formato do nome do ficheiro será penalizada.

Metas: requisitos e datas de entrega

Meta 1:

- **Requisitos**: estruturas de dados para representar o labirinto e o seu conteúdo; toda a informação dos clientes (aquela que for relevante para jogadores e processos clientes – tem que ser algo com sentido e não há uma solução única); funcionalidade de armazenamento dos pares *username*/*password*; leitura e validação (sintaxe) dos comandos do servidor.
- **Data de entrega**: Domingo, 12 de Novembro. Sem possibilidade de entrega atrasada.

Meta 2:

- **Requisitos**: todos os requisitos da meta 1; comunicação com *named pipes* a funcionar; gestão dos clientes (detetar que existem, avisar que o servidor encerrou, etc.) a funcionar; *login* dos jogadores a funcionar.
- **Data de entrega**: Domingo, 10 de Dezembro. Sem possibilidade de entrega atrasada.

Meta 3:

- **Requisitos**: todos os requisitos expostos no enunciado.
- **Data de entrega**: Domingo, 14 de Janeiro, 2018. Sujeito a ajustes caso haja alterações no calendário escolar. Entrega atrasada possível, mas condicionada e fortemente penalizada.

[1] Leia-se "***zip***" - não é *arj*, *rar*, *tar*, ou outros. O uso de outro formato será **penalizado**. Há muitos utilitários da linha de comando UNIX para lidar com estes ficheiros (zip, gzip, etc.). Use um.
[2] O não cumprimento do formato do nome será **penalizado**.

Em todas as metas deverá ser entregue um relatório. O relatório compreenderá o conteúdo que for relevante para justificar o trabalho feito, deverá ser da total autoria dos alunos, e deverá ainda compreender um conteúdo obrigatório que será especificado no *moodle* atempadamente.

7. Avaliação do trabalho

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

- **Arquitetura do sistema** – Há aspetos relativos à interação dos vários processos que devem ser cuidadosamente planeados de forma a se ter uma solução elegante, leve e simples. A arquitetura deve ser bem explicada no relatório para não existirem mal-entendidos.
- **Implementação** – Deve ser racional e não deve desperdiçar recursos do sistema. As soluções encontradas para cada problema do trabalho devem ser claras. O estilo de programação deve seguir as boas práticas. O código deve ter comentários relevantes. Os recursos do sistema devem ser usados de acordo com a sua natureza.
- **Relatório** – Em todas as metas existirá um relatório. O relatório seguirá indicações a fornecer no *moodle* atempadamente. De forma geral, o relatório descreve a estratégia e os modelos seguidos, a estrutura da implementação e as opções tomadas. Este relatório será enviado juntamente com o código no arquivo submetido via *moodle* e o da entrega final será também 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 diretamente proporcional à qualidade da defesa. Elementos do mesmo grupo podem ter notas diferentes consoante o empenho individual que demonstraram na realização do trabalho.

  Apesar de a defesa ser individual, ambos os elementos do grupo devem comparecer ao mesmo tempo. A falta à defesa implica automaticamente a perda da totalidade da nota do trabalho.

  Na FUC está descrito o que acontece nas situações de fraude (ex., plágio) no 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.