Instituto Superior de Engenharia de Coimbra
Departamento de Engenharia Informática e de Sistemas

Sistemas Operativos 23/24

Trabalho Prático - Programação em C para UNIX

O trabalho prático de Sistemas operativos consiste num jogo de labirinto multijogador, com vários níveis. O objetivo dos jogadores é mover-se de um ponto inicial para um ponto final num labirinto em que poderão surgir obstáculos em posições aleatórias. Quando o jogador completa um labirinto, avança de nível no jogo e o novo mapa será mais complexo. Os labirintos são previamente definidos em ficheiro e não fazem parte do trabalho algoritmos de geração de labirintos. Também não haverá inimigos a perseguir os jogadores não está envolvida nenhuma lógica de comportamento automático inteligente. O trabalho foca, acima de tudo, gestão de processos, comunicação entre processos e execução de tarefas em paralelo.

O trabalho prático aborda acima de tudo os conhecimentos do sistema Unix, e deve ser concretizado em linguagem C, para plataforma Unix (Linux), usando os mecanismos deste sistema operativo abordados nas aulas teóricas e práticas. O trabalho foca-se no uso correto dos mecanismos e recursos do sistema. Assuntos de carácter mais relacionado com o tema (forma de pontuação, etc.) são considerados secundários e passíveis de escolhas pelos alunos desde que não removam complexidade quanto a assuntos centrais à disciplina. Não é dada particular preferência a escolhas na implementação de aspetos de caráter periférico à disciplina, tais como, por exemplo, escolhas quanto a listas ligadas *vs. arrays* dinâmico, e serão como os alunos preferirem. O enunciado fornece algumas simplificações de forma a tornar o trabalho mais focado em sistemas operativos e menos em programação (por exemplo: são indicados números máximos para diversos elementos – jogadores, obstáculos, etc. – de forma a evitar a necessidade de estruturas de dados complexas.

No que respeita à manipulação de recursos do sistema, deve ser dada prioridade ao uso de chamadas ao sistema operativo[1] face ao uso de funções biblioteca[2] (por exemplo, devem ser usadas *read()* e *write()* em vez de *fread()* e *fwrite()*). Excetua-se aqui a leitura de ficheiros de configuração que podem ser lidos com o habitual *fread* e *fscanf* (já no uso de *named pipes* devem mesmo ser usadas as funções sistema).

Não é permitido o recurso a bibliotecas de terceiros que façam parte do trabalho, ou que ocultem o uso dos recursos e mecanismos estudados na disciplina. O uso de API do sistema Unix que não tenha sido abordado nas aulas é permitido desde que dentro da mesma categoria de recursos estudados, e terá que ser devidamente explicado, em particular durante a defesa.

Não é permitida uma abordagem baseada na mera colagem de excertos de outros programas ou de exemplos. Todo o código apresentado terá que ser entendido e explicado por quem o apresenta, caso contrário não será contabilizado.

1. Descrição geral, conceitos principais e elementos envolvidos

O trabalho consiste numa plataforma de jogo para vários jogadores em simultâneo. Antes do início do jogo há um período durante o qual os jogadores podem associar-se ao jogo. Findo esse período, o jogo inicia e outros jogadores que surjam podem ver, mas não participar. A plataforma apenas terá a decorrer um jogo em simultâneo.

Os jogadores competem entre si num labirinto, deslocando-se de um ponto de partida até um ponto de chegada. O labirinto é gerido centralmente, mas é dado a conhecer a todos os jogadores e é representado visualmente como uma grelha de

[1] Documentadas na secção 2 das *manpages*
[2] Documentadas na secção 3 das *manpages*

caracteres. O ponto de partida é diferente para cada jogador (sorteado entre 5 posições possíveis na última linha do labirinto). O ponto de chegada é comum a todos os jogadores e encontra-se localizado sensivelmente no centro da primeira linha.

Cada nível tem um tempo máximo para jogar. O primeiro jogador a chegar ao ponto de chegada é o vencedor. Se nenhum jogador atingir esse ponto dentro do tempo especificado para esse nível, todos os jogadores perdem, terminando o jogo, sendo acionadas as ações associadas ao fim de jogo (descritas mais adiante).

O labirinto é constituído por obstáculos fixos ("paredes"), e caminhos entre essas paredes, sendo definido no ficheiro que é lido. Não existem inimigos no labirinto, mas à medida que o tempo decorre podem aparecer alterações dinâmicas no labirinto que complicam a vida aos jogadores, sendo vistas por estes imediatamente. Existem dois tipos de alterações:

- obstáculos temporários ("pedras") em localizações aleatórias e que permanecem por um período de tempo limitado, findo o qual voltam a desaparecer. O aparecimento das "pedras" é controlado por um programa especializado ("bot") que é quem decide onde, quando e durante quanto tempo aparecem as pedras.
- "entupimentos móveis": são bloqueios que surgem de forma aleatória e se vão deslocando no labirinto, fechando passagens de forma imprevisível e trocando as voltas aos jogadores. Os "entupimentos móveis" são controlados pelo programa que gere o jogo ("motor") e cada entupimento consiste numa tarefa que decorre em paralelo com as outras tarefas desse programa.

O jogo encontra-se organizado em três níveis: o nível avança sempre que um dos jogadores completa o nível. Os mapas são mais complexos à medida que o nível sobe (por exemplo: mais obstáculos temporários, mais vezes e durante mais tempo).

Durante o decurso do jogo, os jogadores envolvidos podem comunicar entre si através de mensagens de texto. Estas mensagens são enviadas diretamente de um utilizador para o outro e são visíveis apenas para os dois jogadores envolvidos. A mensagem é enviada diretamente de um jogador a outro sem passar por um ponto central. No entanto, os jogadores necessitam de conhecer a existência um do outro e a obtenção dessa informação pode envolver o motor central do jogo.

Programas envolvidos

O sistema será composto por três programas distintos: o programa **jogoUI**, o programa **motor** e o programa **bot.**

- O programa **jogoUI** é usado pelo utilizador para ver o labirinto e mensagens do motor, jogar (indicar ordens de jogo) e comunicar com outros jogadores, e estas ações devem poder decorrer em simultâneo. A funcionalidade do programa **jogoUI** é, essencialmente, a de interface de utilizador, sendo toda a lógica (gestão de regras, labirinto e controlo do jogo) feita centralmente no programa **motor**. O utilizador apenas conseguirá utilizar a funcionalidade da plataforma depois de se identificar. A identificação consiste num nome, sem qualquer password, sendo sempre aceite desde que não haja já outro jogador com esse nome, sendo esta validação feita pelo **motor**. Existirá um processo **jogoUI** em execução para cada utilizador ligado à plataforma.

- O programa **motor** gere o jogo, funcionando como um servidor e interagindo com os vários jogadores. Este programa carrega os mapas dos labirintos de acordo com os níveis, inicia e termina os jogos, insere obstáculos (fixos e móveis) nos mapas durante o jogo e pode também enviar informações aos jogadores sem que estes as tenham solicitado. Além disso, o programa **motor** interage com o(s) programa(s) **bot**. Pode interagir diretamente com um utilizador (administrador da plataforma), o qual especifica comandos (recebidos pelo **motor** no seu *stdin*) que permitem, por exemplo, terminar o jogo, mostrar jogadores ativos e expulsar um jogador. Em cada momento existirá no máximo um único processo **motor** em execução. Este programa **motor** também deverá interagir com os diversos programas **bot**.

- O programa **bot** envia ao programa **motor** informação contendo as coordenadas e a duração dos obstáculos temporários fixos (as "pedras") a serem introduzidos no mapa durante os jogos. A informação é dada como texto (uma linha de texto) pelo ***stdout*** do **bot** e tem o seguinte formato:

  ```
  lin col duração
  ```

  Exemplo

  ```
  3 5 10
  ```

  Significa: Deve aparecer um obstáculo na posição 3,5 (linha,coluna), se estiver livre, e permanece durante 10 segundos. Mais adiante são dados detalhes adicionais acerca destes programas.

Utilizadores

Existem dois tipos de utilizador neste sistema:

- **Jogador**. Este é aquele que, após se identificar, participa no jogo. Utiliza apenas o programa **jogoUI**, sendo que cada jogador executa a sua própria instância do programa **jogoUI**. Toda a interface com o utilizador jogador é feita em ambiente de consola (terminal em modo de texto). O movimento do jogador é realizado através das teclas de direção (deve ser usada a **biblioteca *ncurses*** para o efeito); a funcionalidade de troca de mensagens entre jogadores é usada através de comandos escritos. Toda a informação disponibilizada é apenas texto. Para acrescentar um novo jogador será simplesmente aberto um novo terminal através do qual esse novo utilizador do jogo interage com o **jogoUI**.
- **Administrador**. Controla o **motor** do jogo e é responsável por lançá-lo. Interage com a plataforma através do programa **motor**, seguindo a lógica de comandos escritos que são processados pelo **motor** (estes comandos serão descritos em detalhe mais adiante). É importante salientar que o "administrador" não tem qualquer relação com o administrador (*root*) do sistema operativo.

Todos os utilizadores do jogo correspondem ao mesmo utilizador do sistema operativo onde os vários programas do jogo são executados.

Ficheiros de dados envolvidos

Existirão ficheiros de texto que descrevem os diversos mapas que serão utilizados no jogo. O programa **jogoUI** não precisa nem deve aceder a estes ficheiros. A forma de gestão/carregamento dos mapas é da responsabilidade do programa **motor**, mas sugere-se que utilize o formato de uma grelha de caracteres para cada mapa. Pode utilizar um ficheiro de mapas por nível.

2. Utilização da plataforma (funcionalidade vista pelos utilizadores)

Do ponto de vista do utilizador *jogador* (programa **jogoUI**)

- O utilizador jogador executa o programa **jogoUI** indicando as suas credenciais (nome), através da linha de comandos. Exemplo:

```
$ ./jogo manuel
```

- O **jogoUI** envia ao ***motor*** o nome do jogador. Caso a validação seja bem-sucedida (não exista outro jogador com o mesmo nome, esteja no período de inscrições e existam vagas), o programa **jogoUI** fica a aguardar pelo início do jogo. Caso contrário, o **motor** informa o jogador do sucedido e ignora todos os pedidos desse jogador, mas ainda permite a este jogador ver o desenrolar do jogo. Em ambos os casos, o **jogoUI** é informado do resultado.
- O utilizador interage com o **jogoUI** através das **teclas de direção** e **comandos escritos**, os quais despoletam o envio de mensagens para dois programas no decorrer do jogo: a) para o ***motor,*** quando se trata de jogar; ou b) diretamente para outra instância do **jogoUI** (ou seja, outro jogador), quando pretende comunicar com ele.
- A comunicação de mensagens entre jogadores é feita diretamente entre os dois processos jogo envolvidos. No entanto, os processos necessitam de saber acerca da existência um do outro e isso deve envolver o **motor**.

Durante o decorrer do jogo, o programa **jogoUI** estará no modo "leitura de teclas de direção", ignorando a introdução de outro texto. Para escrever comandos terá que se indicar ao **jogoUI** essa intenção carregando na **tecla espaço**. Ao detetar essa tecla, o **jogoUI** passa para o modo leitura de comandos, durante a qual aceita a escrita de texto "normal" (comandos) e ignora as teclas de direção. O **jogoUI** voltará automaticamente ao modo "teclas de direção" quando o jogador concluir a introdução do comando. Naturalmente, convém que o jogador se despache a escrever o comando pois durante essa escrita não conseguirá controlar o seu personagem no labirinto. É necessário usar a biblioteca ***ncurses*** para lidar com a deteção de teclas de direção.

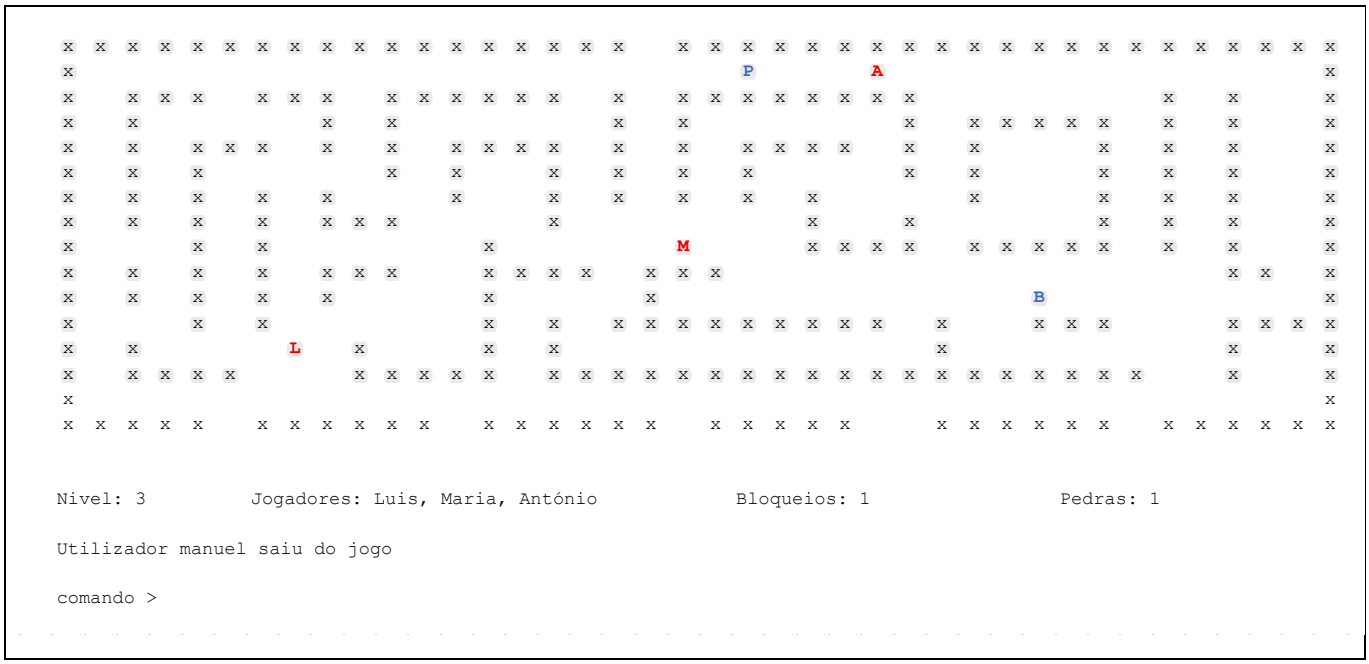
O utilizador ***jogador*** pode fazer:

- Jogar no mapa: teclas ←, →, ↓ e ↑
- Mudar para modo de escrita de comando: tecla **espaço**
- Obter a listagem de todos os jogadores ativos no sistema: comando **players**
- Enviar mensagem a um utilizador especifico: comando **msg** <nome_utilizador> <mensagem>
- Sair, encerrando o processo **jogoUI**: comando **exit** (o ***motor*** deve ser avisado)

Todos os comandos devem ter feedback visível ao jogador que indique o sucesso/insucesso da operação e os dados relevantes, conforme o comando escrito. Durante o decorrer do jogo, o labirinto será sempre atualizado.

A informação apresentada deve ser organizada em duas zonas: uma onde é apresentado o labirinto, e outra onde são escritos os comandos e onde aparece o feedback/output dos comandos e demais avisos do **motor**. A figura abaixo mostra um exemplo.

A biblioteca **ncurses** é essencial para se conseguir posicionar o cursor em coordenadas específicas e alcançar a forma de apresentação desejada. Serão dadas algumas indicações sobre esta biblioteca nas aulas.



Outras funcionalidades associadas com a visualização

- O utilizador recebe uma notificação/mensagem cada vez que um jogador sai do jogo;
- O utilizador é informado quando um obstáculo temporário de posições ("pedra") fixas é colocado de forma automática no mapa.
- No labirinto, os jogadores são identificados pela sua inicial. As pedras têm a letra 'P' e os bloqueios móveis tem a letra 'B' (estes elementos já foram descritos, mas haverá mais informações mais adiante).

Do ponto de vista do utilizador *administrador* (programa *motor*)

O administrador executa uma instância do programa ***motor***, sendo que apenas uma instância (processo) deste programa pode estar a correr a um dado instante. A validação deste aspeto fica a cargo do programa e não do utilizador. Exemplo de execução:

```
$ ./motor
```

O único utilizador que interage com o ***motor*** é o **administrador** da plataforma, através de comandos escritos (não são menus) para efetuar ações de controlo. Não existe nenhum procedimento de *login*: o **administrador** é simplesmente quem executa o programa ***motor***, ficando a interagir com ele. As ações (comandos) disponíveis ao **administrador** são:

- Listar jogadores atualmente a usar a plataforma: comando **users**
- Banir um jogador atualmente registado (porque sim): comando **kick** *<nome do jogador>*
  Tem o mesmo efeito que o comando **exit** feito pelo jogador (pode voltar a entrar). O utilizador em questão é informado, e o seu programa **jogoUI** deve terminar automaticamente.
  Exemplo: *kick rogerio66*
- Listar os **bots** atualmente ativos: comando **bots**
  São apresentados os **bots** e respetiva configuração.
- Insere um **bloqueio móvel**. Um bloqueio móvel é um pedaço de parede que se vai movimentando a cada segundo para uma posição adjacente. Apenas se move para posições livres, não atropelando jogadores nem outros obstáculos que haja no labirinto. Serve para complicar a vida aos jogadores. A posição inicial é sorteada. Comando **bmov**
- Elimina um **bloqueio móvel**. Caso haja vários é removido o que foi criado em primeiro lugar. Comando **rbm**
- Inicia manualmente o jogo, mesmo que não exista um número mínimo de jogadores: comando **begin**
- Encerrar a jogo: comando **end**
  Encerra o jogo (se estiver a decorrer) e termina o **motor**. Os processos a correr o **jogo** são notificados, devendo também terminar. Os processos **bots** a correr devem também terminar. Os recursos do sistema em uso são libertados.

3. Funcionamento da plataforma

Motor

- Só aceita executar se ainda não estiver a correr nenhum outro motor.
- Interage com o ***administrador*** e os processos **jogoUI** e **bot**, sem que a interação com um atrase a interação com os outros.
- Ao encerrar avisa todos os intervenientes para encerrarem também. O encerramento deve ser ordeiro.
- Envia informações aos jogadores (i.e., aos **jogoUI**). Estas informações podem ser respostas a pedidos feitos por estes, mas também informações enviadas por iniciativa própria (sem qualquer pedido inicial), e já anteriormente descritas.
- Lança e gere a execução dos programas **bots**, interagindo com estes para obter as coordenadas dos obstáculos. O número de **bots**, deverá aumentar à medida que a dificuldade (níveis) de jogo aumenta.
- Em caso de problemas, o programa deve sair de forma ordeira, libertando os recursos ocupados.
- Na implementação **pode assumir que existem máximos** para as seguintes entidades
  - **Utilizadores**: máximo **5**
  - **Processos bot**: máximo **10**
  - **Pedras** no labirinto (definidas pelos **bots**) **em simultâneo**: máximo **50** (mais do que os **bots** porque as pedras podem permanecer por algum tempo e acumular com outras que vão surgindo).
  - **Número de níveis:** máximo **3**
  - **Bloqueios móveis**: máximo **5**
- **A dimensão do mapa é sempre 16 linhas e 40 colunas**
- A duração do período de inscrições é dada pela **variável de ambiente INSCRICAO**, em segundos. O jogo inicia automaticamente findo esse período se tiver o número de jogadores mínimo, estipulado na **variável de ambiente NPLAYERS**. Caso não haja esse número de jogadores, o jogo só terá início quando estiverem reunidos os jogadores necessários, ou então, por ordem do administrador.
- O primeiro nível demora uma quantidade de segundos indicada na **variável de ambiente DURACAO**. Cada nível seguinte demora menos **DECREMENTO** (outra variável de ambiente) segundos que o anterior.

Bot

Um **bot** é um programa que imprime periodicamente no seu *stdout* duas coordenadas e respetiva duração para a inserção de pedras dentro de um labirinto por parte do programa motor. A informação é impressa no formato "`<Coordenada_X> <Coordenada_Y> <Duração>`", tal como descrito anteriormente. O programa **motor** deverá capturar esta informação para determinar em que posição e durante quanto tempo a pedra fica no labirinto. O **motor** é responsável por garantir que a pedra é colocada nas coordenadas indicadas pelo período de tempo estipulado. A única coisa que o **bot** faz é enviar a informação para o seu *stdout* e o **motor** faz o resto.

As coordenadas e a duração especificadas pelo **bot** são geradas aleatoriamente e é garantido que estão sempre entre os valores 0-39 para a coluna ("x") e 0-15 para a linha ("y"). A duração durante a qual a pedra deve permanecer e a frequência com que o **bot** envia informação para colocar novas pedras é controlada por **parâmetros de linha de comandos** do **bot**:

- O primeiro parâmetro é o número de segundos entre cada envio.
- O segundo parâmetro é o número de segundos que a pedra permanece.

Esta descrição interessa acima de tudo para o **motor** saber como se usa o **bot**, dado que o programa **bot** é fornecido já feito pelos professores.

Para aumentar a complexidade conforme o nível, o programa **motor** deve lançar/encerrar processos **bot** de acordo com a seguinte distribuição de níveis:

- Nível 1 – Existem dois **bots** com a seguinte configuração:
  - Bot 1 - Envios de coordenadas a cada 30 segundos e duração de permanência de pedras de 10 segundos;
  - Bot 2 - Envios de coordenadas a cada 25 segundos e duração de permanência de pedras de 5 segundos.
- Nível 2 – Existem três **bots** com a seguinte configuração:
  - Bot 1 - Envios de coordenadas a cada 30 segundos e duração de permanência de pedras de 15 segundos;
  - Bot 2 - Envios de coordenadas a cada 25 segundos e duração de permanência de pedras de 10 segundos;
  - Bot 3 - Envios de coordenadas a cada 20 segundos e duração de permanência de pedras de 5 segundos.
- Nível 3 – Existem quatro **bots**, com a seguinte configuração:
  - Bot 1 - Envios de coordenadas a cada 30 segundos e duração de permanência de pedras de 20 segundos;
  - Bot 2 - Envios de coordenadas a cada 25 segundos e duração de permanência de pedras de 15 segundos;
  - Bot 3 - Envios de coordenadas a cada 20 segundos e duração de permanência de pedras de 10 segundos;
  - Bot 4 - Envios de coordenadas a cada 15 segundos e duração de permanência de pedras de 5 segundos.

No final de cada nível, os **bots** devem ser terminados pelo motor, e novos **bots** devem ser lançados, seguindo a distribuição previamente mencionada.

JogoUI

Faz a interação entre o utilizador **jogador** e o resto da plataforma. As suas características gerais são:

- Só aceita executar se o **motor** estiver em funcionamento.
- Recebe o nome do jogador por argumentos da linha de comando.
- Está apto a interagir com o **motor** e com o utilizador em paralelo sem que uma interação impeça ou atrase a outra.
- Deve permitir comunicar com outros jogadores (processos **jogoUI**), em paralelo com o ato de jogar.
- Não deve permanecer em execução se o **motor** terminar, ou se for informado que o seu utilizador foi excluído (comando *kick* do administrador).

4. Requisitos e restrições

Implementação

- Ficheiros regulares são repositórios de informação - não são mecanismos de comunicação.
- A comunicação direta entre os dois clientes é feita com ***named pipes***.
- O **bot** faz apenas aquilo que foi referido na descrição dada atrás.
- O mecanismo de comunicação entre **motor** e **jogoUi** é o *named pipe*. O número de *named pipes* envolvidos, quem os cria, e a forma como são usados devem seguir os princípios exemplificado nas aulas.
- Só podem ser usados os mecanismos de comunicação que foram abordados nas aulas.
- Se forem fornecidas bibliotecas pelos professores, então estas serão de uso obrigatório.
- A API do sistema deve ser aquela que foi estudada. Uma função pouco abordada, mas relacionada com as estudadas, é aceite, mantendo-se dentro do contexto do que foi abordado (exemplo: *dup2* em vez de *dup* é aceite - mas uso de memória partilhada não).
- O uso de bibliotecas de terceiros (exceto as fornecidas ou indicadas pelos professores) não é aceite.
- As questões de sincronização que possam existir devem ser acauteladas e tratadas da forma adequada.
- Situações que obriguem os programas a lidar com ações que possam ocorrer em simultâneo não podem ser resolvidas com soluções que atrasem ou impeçam essa simultaneidade. Essas situações, se ocorrerem, têm formas adequadas de solução que foram estudadas nas aulas e devem ser observadas.
- Excerto de código de "stackoverflow / github / etc" não poderá ser extenso nem abordar as questões centrais da matéria, e terá sempre que ser explicado na defesa, por mais pequeno que seja.
- Todo o código terá que ser explicado na defesa, tenha ou não sido retirado de exemplos - se não for explicado, será entendido como "o conhecimento não está lá" e, por conseguinte, a parte não explicada não é contabilizada.

Terminação dos programas

- Quer seja feita a pedido do utilizador, ou por não estarem reunidos os pressupostos para o programa executar, ou por situação de erro em *runtime*, os programas devem terminar de forma ordeira, libertando os recursos usados, avisando tanto quanto possível o utilizador e os programas com que interajam.

5. 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 e deve ser realizado em grupos de dois alunos (grupos de três não são aceites).
- Existe defesa obrigatória. Os moldes exatos serão definidos e anunciados através dos canais habituais na altura em que tal for relevante. A defesa será presencia e individual, mas os alunos do grupo comparecem em conjunto.
- Existem a meta intermédia e a meta final, tal como descrito na FUC. As datas e requisitos das metas são indicados mais abaixo. Em todas as metas a entrega é feita via nónio (inforestudante) através da submissão de um único **arquivo *zip*[3]** cujo **nome** respeita o seguinte padrão[4]:

  so_2324_tp_metaN_*nome1_numero1_nome2_numero2*.zip

  (metaN, nome e número serão adaptados à meta, nomes e números dos elementos do grupo)

[3] 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.

[4] O não cumprimento do formato do nome causa atrasos na gestão dos trabalhos recebidos e será **penalizado**.

- A não adesão ao formato de compressão indicado (.zip) ou ao padrão do nome do ficheiro será penalizada, *podendo levar a que o trabalho nem sequer seja visto*.
- Cada grupo submete o trabalho uma vez, sendo indiferente qual dos dois alunos o faz.
- **É obrigatório** que o aluno que faz a submissão **associe no nónio a entrega também ao outro aluno do grupo**.
- **É necessário que ambos estejam inscritos em turmas práticas (mesmo que seja em turmas diferentes)**

Metas: requisitos e datas de entrega

Meta 1: 18 de Novembro

Requisitos:

- Planear e definir as **estruturas de dados** responsáveis por gerir as definições de funcionamento no **jogoUI** e no **motor**. Definir os *header files* com constantes simbólicas e declarações associadas às estruturas de dados.
- **JogoUI**:
  - implementar a parte da leitura de comandos e respetiva validação. Todos os comandos devem ter a sua sintaxe validada. Os comandos não farão ainda nada, mas será reconhecido como válido ou inválido, incluindo parâmetros.
  - Deve implementar a receção das credenciais do utilizador dentro do **jogoUI**.
- **Motor**:
  - Implementar a leitura de comandos do administrador, validando a sintaxe de todos. Os comandos não farão ainda nada, mas será reconhecido como válido ou inválido, incluindo parâmetros.
  - Implementar a parte de lançamento dos **bots** e recepção das coordenadas e duração. Será apresentado no ecrã, pelo motor, o conteúdo das mensagens que os **bots** produzem. Nesta meta, apenas será lançado o primeiro **bot**. Para testar e verificar esta funcionalidade na meta 1, deve ser acrescentado um comando extra ao **motor** – comando "**test_bot**. Este comando lança um **bot**, recebe as mensagens enviadas por este e mostra-as no monitor (precedidas de "RECEBI: "). Exemplo: "RECEBI: 3 5 10".
  - Leitura do ficheiro com os mapas. Nesta fase deve apenas apresentar o mapa no ecrã do programa motor.
- **JogoUI** / **motor**: outros aspetos necessários às funcionalidades referidas atrás.
  - *makefile* que possua os *targets* de compilação "all" (compilação de todos os programas), "jogoUI" (compilação do programa **jogoUI**), "motor" (compilação do programa **motor**) e "clean" (eliminação de todos os ficheiros temporários de apoio à compilação e dos executáveis).
- **ncurses**: apresentação de um uso inicial de **ncurses** numa funcionalidade qualquer que faça sentido para o trabalho (ou seja, já tem que ter experimentado **ncurses** para o uso que vá no sentido do necessário ao **jogoUI**)

Data de entrega: Sábado, 18 de Novembro, 2023 . Sem possibilidade de entrega atrasada.

Na **meta 1** deverá ser entregue um breve documento (pdf, incluído no ficheiro zip) com duas páginas descrevendo os pormenores da implementação e principais opções tomadas. A capa de título e o índice não contam como páginas.

Meta 2: 30 de Dezembro

Requisitos:

- Todos os requisitos expostos no enunciado.

Data de entrega: Sábado, 30 de Dezembro, 2023. Sujeito a ajustes caso haja alterações no calendário escolar.

Na **meta final** deverá ser entregue um **relatório** (pdf, incluído no zip). O relatório compreenderá o conteúdo que for relevante para justificar o trabalho feito, deverá ser da exclusiva autoria dos membros do grupo. Poderão eventualmente ser dadas indicações adicionais quanto ao relatório.

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 planeados de forma a apresentar-se uma solução elegante, leve e simples. A arquitetura deve ser bem explicada no relatório.
- **Implementação** – Deve ser racional e não desperdiçar recursos do sistema. As soluções encontradas devem ser claras e bem documentadas no relatório. 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** – O relatório deve descrever convenientemente o trabalho. Descrições meramente superficiais ou genéricas de pouco servirão. De forma geral, o relatório descreve a estratégia e os modelos seguidos, a estrutura da implementação e as opções tomadas. Podem vir a ser dadas indicações adicionais sobre a sua elaboração. O relatório deve ser entregue juntamente com o código no arquivo submetido na meta em questão.
- **Defesa** – Os trabalhos são sujeitos a defesa individual, durante a qual será verificada a autoria e conhecimentos, podendo haver mais do que uma defesa caso subsistam dúvidas. A nota final do trabalho é diretamente proporcional à qualidade da defesa. Elementos do mesmo grupo podem ter notas diferentes consoante o desempenho e grau de participação individuais que demonstraram na defesa.
  A falta à defesa implica automaticamente a perda da totalidade da nota do trabalho.
  Plágios e trabalhos feitos por terceiros: o regulamento da escola descreve o que acontece nas situações de fraude.
- Os trabalhos que não funcionem serão fortemente penalizados independentemente da qualidade do código-fonte ou arquitetura apresentados. Trabalhos que nem sequer compilam terão uma nota extremamente baixa.
- A identificação dos elementos de grupo deve ser clara e inequívoca (tanto no arquivo zip como no relatório). Trabalhos anónimos não são corrigidos.
- Qualquer desvio quanto ao formato e forma nas submissões (exemplo, tipo de ficheiro) dará lugar a penalizações.

Importante: O trabalho deve ser realizado por ambos os elementos do grupo**. Não são aceites divisões herméticas em que um elemento faz uma parte e apenas essa, nada sabendo do restante**. Se num grupo existir uma participação desigual, deverá esse grupo informar o professor que lhe faz a defesa. Se não o fizer e for detetada essa desigualdade, ambos os alunos ficarão prejudicados em vez de apenas aquele que trabalhou menos.