FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Ano Letivo 2014/15

Laboratório de Computadores

Racinix

Grupo T3G06: Edgar Duarte Ramos – up201305973@fe.up.pt

Gustavo Rocha da Silva - up201304143@fe.up.pt

Índice

1. Instruções de utilização

1.1. Menu Inicial

Primeiramente, é apresentado um menu inicial. (apresentado em baixo)

Figura 1 - Menu inicial

Este menu disponibiliza o acesso aos vários modos de jogo, assim como aos créditos e ao botão de saída. É importante referir que em todos os modos de jogo, bem como em todos os menus, como vai ser possível verificar em todas as imagens apresentadas, há o display de anúncios (chamados pelos alarmes do RTC) na parte inferior do ecrã.

Neste mesmo menu, passar o rato por cima de uma opção provoca uma mudança de cor das letras respetivas, ou seja, quando o utilizador posiciona o rato sobre a opção pretendida esta vai ficar destacada em relação às outras, de modo a ser percetível que se trata de uma opção clicável (imagem à direita).

Figura 2 - "Hovering" com o rato

1.2. Créditos

No menu inicial, temos também acesso aos créditos. Estes foram uma forma dos desenvolvedores deste projeto esclarecer o âmbito do mesmo, bem como agradecerem todo o apoio dos intervenientes neste projeto, de forma direta ou indireta. Para sair, basta um clique do rato ou pressionar a tecla "ESC".

Figura 3 - Créditos

1.3. Escolha do modo de jogo

No menu inicial podemos escolher qual o modo de jogo que queremos experienciar. Assim, o utilizador pode escolher um de três modos de jogo. Estes modos de jogo são:

- 1 Player: Neste modo de jogo o utilizador corre sozinho.
- 2 Players in the same PC: Aqui o utilizador experimenta um modo de jogo que lhe permite correr contra outro utilizador no mesmo computador.
- 2 Players via serial port: Neste modo de jogo o utilizador pode jogar com outro utilizador a partir de dois computadores diferentes (ou neste caso a partir de duas máquinas diferentes).

1.4. Modo de desenho da pista

Nos modos de jogo, após o menu inicial exposto anteriormente, é apresentado um submenu de escolha de pista. Porém, este menu só aparece caso a escolha do utilizador seja umas das primeiras 3 opções do menu inicial.

Neste submenu podemos decidir se queremos gerar uma pista de modo aleatório ("Random track") ou se queremos criar a nossa própria pista ("Design track").

Figura 4 - Context Menu

Ao passar o rato por cima de uma opção, ela sobressai ligeiramente para se perceber que se trata de um texto clicável. É importante notar também que os anúncios ocorrem ao longo de todo o ecrã e não só onde o menu se situa. (No exemplo que se segue a opção selecionada foi "2 Players in the same PC").

1.5. Track Designer

Uma pista pode ser desenhada a partir dos seus *checkpoints*. Para criar um desses pontos, basta clicar com o botão esquerdo do rato em algum sítio do ecrã.

Os pontos já criados podem ser arrastados mantendo premindo o botão esquerdo do rato ou apagados clicando com o botão direito do rato.

Para sair do "Track Designer" e começar a corrida basta clickar na tecla "Enter".

1.6. Corrida (controlos, etc.)

Na tela de jogo é mostrado o número de voltas e o número de "checkpoints", que o jogador já percorreu após a passagem pela linha de partida, no canto superior direito. Por sua vez, no canto inferior direito é apresentado um velocímetro que nos mostra a velocidade instantânea do veículo (assim como acontece na realidade). A cor dos pontos de controlo ao longo do percurso apresenta-se da cor do veículo, porém se os dois apresentarem o mesmo posto de controlo como o próximo, a cor passa a ser rosa (uma cor neutra).

Os controlos do jogo são então:

Para o "Player 1", para andar para a frente este deve pressionar a tecla W, e para andar para trás deve utilizar a tecla S. Para que a sua direção mude, deve carregar nas teclas A e D, se quiser que o veículo rode para a esquerda ou para a direita, respetivamente.

Para o "Player 2", para se mover deve utilizar as setas existentes no teclado. Se quiser mover-se para a frente este deve pressionar a seta direcionada para cima, e para andar para trás deve utilizar a tecla cuja seta aponta para baixo. Para que a sua direção mude, deve carregar nas *setas* direcionadas para a esquerda ou para a direita mediante a direção que pretende tomar. Em relação ao modo de jogo que utiliza porta de série, as teclas dos dois jogadores são as teclas referentes ao "Player 1".

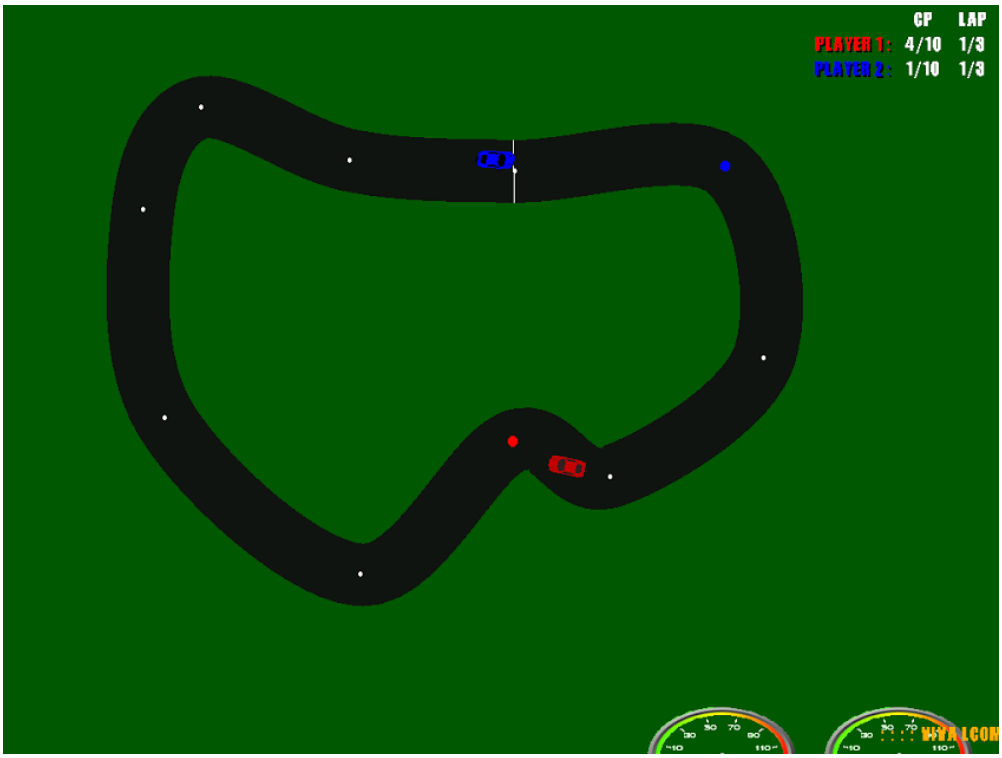

Figura 5 - Exemplo de situação de jogo

2. Estado do projeto

Dispositivo	**Função**	**Interrupções?**
Timer 0	*Frame rate*	Sim
Teclado	Controlo do veículo e saída do jogo, menus, créditos, etc.	Sim
Rato	Navegação nos menus e desenho manual da pista	Sim
Porta de série	Modo de jogo multiplayer	Sim
RTC	Semente para a geração de números aleatórios e alarme	Sim (e *polling*)
Placa gráfica	Apresentação visual dos aspetos gráficos do jogo (menus, veículos,…)	Não

Mais concretamente, os dispositivos foram utilizados dos modos apresentados de seguida.

- Timer 0 - As interrupções do timer 0 permitem controlar a *frame rate* do jogo sem ser necessário alterar a frequência por defeito. Isto é feito no ficheiro principal (proj.c).
- Teclado - É utilizado pelo módulo principal e pelo módulo "Race" para o controlo dos veículos e para saída de estados (carregando no botão "ESC").
- Rato - Permite navegar pelos menus (*event handler* do *main menu*) e desenhar pistas manualmente (*event handler* do estado *track design*).
- Porta de série - Utilizada em modo de interrupções, com uma fila que apenas recebe as strings corretamente lidas (sem erros de *overrun, parity* ou *framing*). A sua implementação encontra-se nos ficheiros proj.c e race.c.
- RTC - É utilizado em modo de polling para a obtenção da hora, útil na geração de números aleatórios (ficheiro proj.c) e em modo de interrupções para mostrar anúncios assim que ocorre um alarme.
- Placa gráfica - Funções úteis para o modo de vídeo implementadas no ficheiro video_gr.c. Por forma a garantir que o rato é atualizado o mais frequentemente possível, além da criação de um double buffer para a escrita dos gráficos, foi também usado um buffer dedicado exclusivamente ao rato, permitindo que a taxa de refrescamento do rato pudesse ser mais elevada.

3. Organização/Estrutura do código

O projeto foi desenvolvido tendo por base módulos bem estruturados cujas características são apresentadas de seguida, assim como o seu peso relativo no projeto.

O Gustavo ficou responsável pela realização dos módulos Racinix, Vehicle, Track, Bitmap, Race, Utilities, Queue, Font, Context Menu e Porta de Série, sendo que em todos estes módulos o Edgar tem como função ajudar em tudo o que seja necessário, para garantir um trabalho mais rápido e eficiente. Por outro lado, o Edgar ficou responsável pelo módulo vector 2D, RTC e Ad, além de ter assegurado que os periféricos trabalhados durante os labs estejam corretamente implementados, alterando o respetivo código caso seja necessária mais eficiência na execução do mesmo. É necessário ter em consideração que este é um trabalho em que os elementos do grupo, quando necessário, se entreajudaram.

3.1. Racinix (proj.h) - 15%

É o principal módulo do projeto.

Este módulo é responsável por relacionar todos os outros módulos, na medida em que serve para coordenar todas as operações que resultam de cada um dos mesmos.

3.2. Vehicle - 13%

Este é o módulo responsável pelo cálculo da posição, "steering" e velocidade de um veículo, assim como das colisões. Este é um módulo importantíssimo para o projeto e as suas dificuldades e implementação vão ser aprofundadas mais à frente neste relatório.

3.3. Track - 12%

Módulo que cria pistas de corrida de forma aleatória ou através da seleção de pontos de controlo por parte do jogador.

3.4. Vector 2D - 2%

Um vector2D é um conjunto de 2 pontos do tipo “double” que juntos representam um vetor bidimensional. O objetivo deste módulo é facilitar a aritmética com vetores.

A função isPointInPolygon() não é da nossa autoria e está protegida pelo seguinte *copyright*: *Copyright (c) 1970-2003, Wm. Randolph Franklin*

3.5. Bitmap - 4%

Responsável por ler e mostrar bitmaps (imagens do tipo “.bmp”).

3.6. Keyboard - 8%

A utilização do teclado neste projeto foi essencial para o controlo dos veículos. O módulo “keyboard” fica assim responsável por fazer a leitura dos scancodes, retornando-os e atualizando o estado da respetiva tecla numa struct de teclas, tornando possível saber, a qualquer momento, que teclas estão a ser carregadas.

3.7. Mouse - 6%

O rato foi utilizado nos menus e na criação de pontos de controlo para gerar pistas de corrida.

3.8. Timer - 4%

Módulo utilizado em modo de interrupções para controlo da taxa de refrescamento do jogo.

3.9. Ad - 2%

Módulo que é responsável pelo display de anúncios.

3.10. Race - 11%

Este módulo é responsável pela gestão de uma corrida, dos veículos, da comunicação da informação dos veículos pela porta de série, entre outros.

3.11. Porta de série - 7%

A Porta de Série foi utilizada para o modo de multiplayer, sendo que este módulo é responsável por toda a sincronização das máquinas bem como na comunicação entre as mesmas.

3.12. Font- 4%

Módulo responsável por exibir texto em várias partes do jogo.

Este módulo permite assim, o uso de várias fontes (no entanto, no projeto, apenas é utilizada uma), calcular a largura de uma determinada string em pixeis, permite também alinhar uma frase à esquerda, centro ou direita e adicionar sombra.

3.13. RTC - 4%

Módulo que permite a atualização e obtenção das horas e da data, bem como a programação de um alarme.

3.14. Utilities - 2%

Ficheiro assembly com as funções memset16() e memset32().

3.15. Queue - 3%

Biblioteca de filas genéricas, útil na implementação da porta de série.

3.16. Context Menu - 3%

Módulo que mostra um menu de contexto. Ao fazê-lo, escurece o fundo para que o foco principal seja o sub-menu. Este módulo aceita quaisquer itens selecionáveis, no entanto, para o projeto, ele apenas é utilizado na seleção do modo de geração de pista (“Random Track” ou “Design Track”).

4. Gráfico de chamada de funções

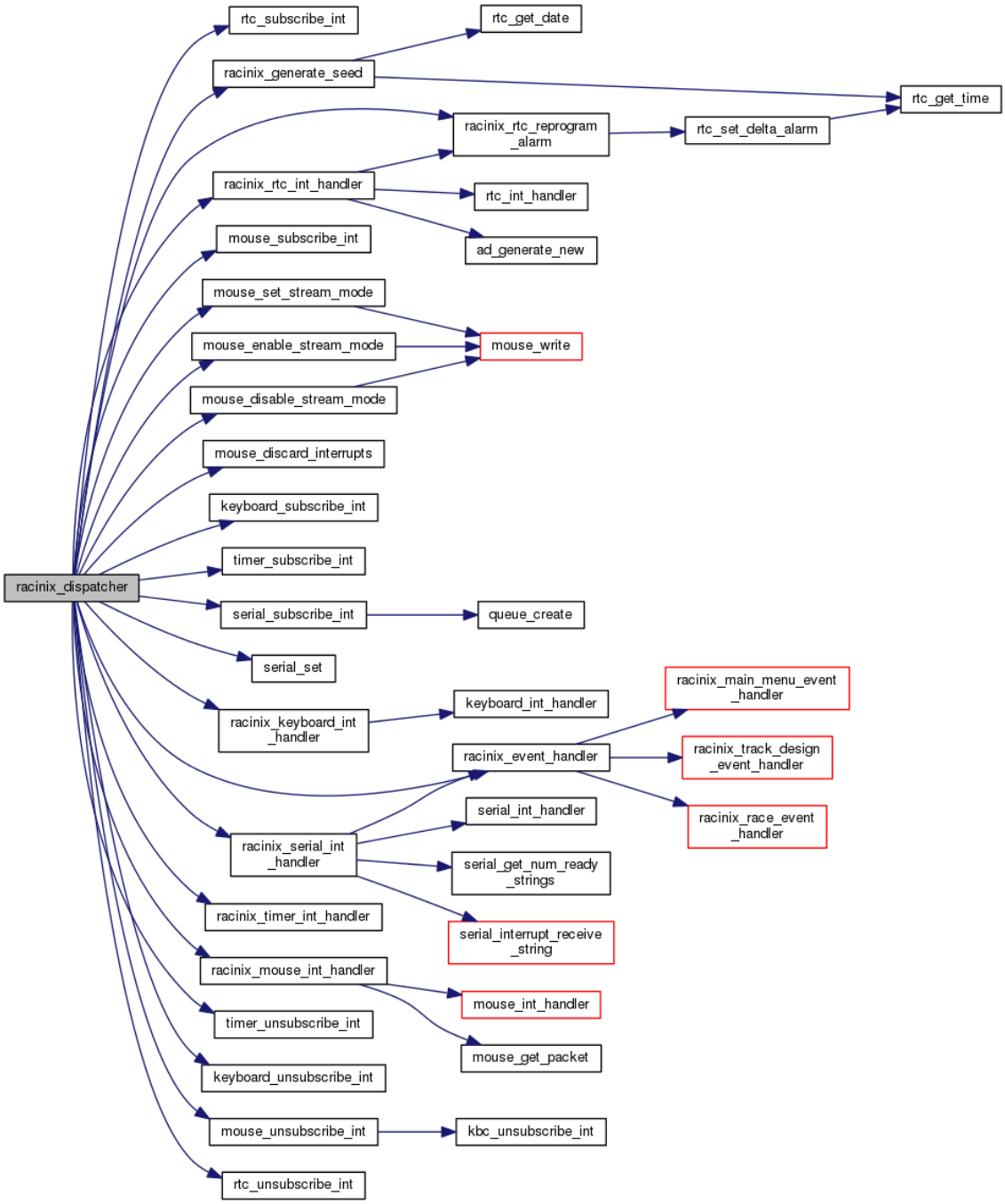

Figura 6 - Gráfico de chamada de funções

O *dispatcher* é chamado na função main() depois do racinix_start() e antes do racinix_end(). Ele é responsável por encaminhar os eventos dos periféricos para o *event handler*.

5. Detalhes de implementação

O projeto Racinix foi desenvolvido tendo por base os *labs* realizados nas aulas práticas, porém foram realizadas pequenas alterações aos mesmos de modo a poderem ser utilizados no projeto.

5.1. Desafios de implementação do módulo veículo

O projeto foi exigente em vários pormenores, como por exemplo o veículo e toda a sua mecânica.

5.1.1. Rotação

Quanto maior o tempo que o utilizador pressiona a tecla de viragem, maior será a rotação das rodas frontais, de modo a ser possível tanto pequenas viragens como grandes viragens. No ficheiro *vehicle.h* é possível definir o limite do ângulo da viragem, de maneira a que se o aumentarmos o carro passará a virar mais.

Os veículos são definidos por 2 eixos e 4 rodas e o cálculo da nova posição é feita com base nos 2 eixos, em que apenas o eixo frontal sofre rotação, tal como acontece na vida real.

5.1.2. Colisões entre veículos

A deteção das colisões é feita através de uma função que verifica alguma das rodas de um veículo A está em contacto com o retângulo pelo qual um veículo B é representado. No entanto, a verificação da colisão entre veículos usa uma *bounding box* para tornar o processo mais eficiente, pelo que apenas aplica esse método se os veículos A e B se encontrarem muito próximos.

Após esta deteção, surge o maior desafio: o tratamento da colisão. O veículo responde apropriadamente tendo em conta a velocidade dos veículos em colisão e o momento provocado pela força de embate. Assim, é possibilitada a alteração da intensidade das forças de rotação em caso de colisão entre veículos (momento).

Optámos por retirar a deteção e tratamento de colisões do modo *multiplayer* via porta de série, uma vez que, por não ser nossa intenção implementar as colisões no momento em que propusemos o projeto, teríamos que reestruturar o modo de funcionamento da porta de série para evitar *lag* que impossibilitasse o tratamento das colisões entre veículos.

5.1.3. Colisões entre um veículo e os limites do ecrã

É verificada e devidamente tratada a colisão de um veículo com os limites do ecrã, para garantir que o utilizador não sai do terreno de jogo. No entanto, a implementação destas colisões foi significativamente mais simples que a implementação das colisões entre veículos.

5.1.4. Outros aspetos

Ainda em relação aos veículos e a tudo o que estes envolvem, existe opção para mostrar o vetor da velocidade, que é ativada descomentando um define no ficheiro *vehicle.h*.

Alguns elementos da struct do veículo são guardados por motivos de eficiência, para não terem de ser recalculados, o que levaria a um gasto desnecessário de recursos computacionais.

É possível definir a aceleração, para a frente ou para a retaguarda, que queremos que o veículo apresente assim como a constante referente às várias forças de resistência que atuam sobre o veículo, e que limitam assintoteticamente a sua velocidade.

5.2. Desafios de implementação do módulo Track

A pista foi também um dos módulos que necessitaram de bastante tempo e revisão.

5.2.1. Desenho de pistas

A geração aleatória das pistas foi um aspeto desafiante.

Um dos problemas foi a nível de eficiência no desenho da mesma. O primeiro algoritmo, pouco eficiente, em que era verificado *pixel* a *pixel* se o mesmo fazia parte ou não da pista, foi revisto, para que, depois de uma segunda implementação bastante mais eficiente se fizessem mais algumas alterações para obter um algoritmo de desenho que funciona em cerca de meio segundo.

5.2.2. Geração de pistas

Cada pista é representada por um conjunto de pontos de controlo e por uma curva que passa nesses pontos de controlo. Além disso, de forma a dar largura à pista, a sua representação também inclui uma linha interior e uma linha exterior, que são os seus limites.

A ideia principal para o algoritmo de geração de pistas automaticamente foi obtida no *blog* do programador Gustavo Maciel, que pode ser consultado em http://bordeen.blogspot.com.

Esse processo usa os seguintes passos:

- Criação aleatória de alguns pontos com uma *seed* passada por parâmetro.
- Escolha do melhor ponto para a linha de partida (de acordo com o ângulo formado entre esse ponto, o anterior e o seguinte), para que a linha de partida seja aí colocada, imitando, de certa forma, aquilo que acontece nas pistas de corridas reais.
- Verificar se a pista não se interceta a si própria em nenhum ponto e, caso se intercete, tentar novamente do início.

No outro modo de geração de pistas, manualmente no “Track Designer”, quando se cria um ponto ele procura a linha mais próxima para o criar. Assim, se for selecionar um ponto muito perto de outros dois pontos, é selecionado o ponto mais próximo do rato. Para ser possível adicionar e remover pontos da *array* de pontos, foi implementado um algoritmo de rotação de *arrays*.

De seguida, para ambos os casos, interpola-se os pontos aplicando a fórmula de geração da curva *Catmull-Rom spline* e atualiza-se a array *track_points* com *false* nos *pixéis* a preencher com relva e com *true* nos *pixéis* a preencher com a pista (para, posteriormente, o processo de desenho ser mais rápido).

5.3. Utilização de bitmaps

De modo a ser possível mostrar imagens no projeto, sem as limitações do tipo utilizado nas aulas (XPM), foi criada uma biblioteca de bitmaps.

Um problema na criação da biblioteca, deveu-se ao facto de, uma vez que as imagens tinham 16 *bits per pixel*, sobrarem dois *bytes* no fim de cada linha com "lixo", porque é feito *padding* a 4 *bytes* em cada linha.

5.4. Desafios de implementação da porta de série

Para possibilitar a implementação da porta de série de uma forma robusta, foi criada uma biblioteca de filas genéricas.

Sempre que se pretende enviar informação pela porta de série, a *string* a enviar é colocada numa fila de transmissão, que é esvaziada sempre que o *buffer* de transmissão da UART se encontra vazio.

Para a receção de informação são utilizadas duas filas: uma para receber cada um dos caracteres e outra responsável por guardar as strings lidas sem erros. Desta forma, garante-se que sempre que ocorra um erro de paridade, *framing* ou *overrun*, a string é corretamente descartada.

5.5. Frame rate

A partir das interrupções do timer 0, temos a possibilidade de ter um qualquer número de *fps,* de 0 a 60 (valor por defeito do minix), sem este ser necessariamente múltiplo de 60.

5.6. Código em assembly

No projeto, foram implementadas funções em assembly

5.6.1. vg_set_pixel()

Verifica se o *pixel* passado como argumento está dentro do ecrã e desenha-o. Para tornar esta verificação mais rápida faz *cast* das coordenadas para *unsigned*, para que seja apenas necessário verificar se estas excedem o limite superior, visto que se excederem o limite inferior, o *cast* fará com que o seu valor seja superior aos limites superiores do ecrã. Desta forma reduz-se o número de comparações de 4 para 2 (x e y).

Porém, a versão assembly da função é mais lenta que a obtida com a otimização máxima do compilador, pelo que optámos por manter a versão em C.

5.6.2. memset16() e memset32()

As funções memset16() e memset32() são programadas em assembly de modo a serem mais eficientes. A função memset16() é utilizada no projeto e chamada sucessivas vezes, para muitos *pixéis*, em funções como a vg_fill() e a vg_draw_rectangle(). A função memset() do C não serve porque só opera com números do tipo *char* e, no projeto, são utilizadas cores de 16 bits. Tal como a função memset() do C, a função que implementámos retorna um apontador

para o primeiro endereço de destino. Foi utilizada a instrução REP STOS para garantir a máxima eficiência.

5.7. Problemas de *debugging*

Descobrimos que variáveis não inicializadas geram comportamento indefinido sempre que são utilizadas, porque a versão compilada não corresponde ao que se espera do código em C. Por exemplo:

```
bool x;
int y = x ? (y = 0) : (y = 1);
```

Seria de esperar que o valor de y fosse 0 ou 1, no entanto o que verificamos foi que o valor dessa variável era de 53, o que nos dificultou no apuramento da causa do problema (a não inicialização de uma variável).

6. Avaliação da Unidade Curricular

Pela positiva, destacamos o facto de termos aumentado os nossos conhecimentos nos seguintes aspetos:

- Programação em C.
- "C orientado a objetos".
- Programação mista C e Assembly.

Pela negativa, destacamos os seguintes aspetos:

- Grau de dificuldade da unidade curricular superior para as turmas cuja aula prática é à segunda-feira, antes da aula teórica;
- Por duas vezes, parte da matéria lecionada na aula teórica foi referente ao *lab* da semana anterior, para as turmas de 2ª feira;
- Relativamente ao *lab* do rato, nada foi dito relativamente ao facto de alguns bits não serem corretamente emulados e de alguns bits não serem usados (os bits de overflow), o que nos obrigou a perder bastante tempo a tentar resolver um problema insolucionável;
- Foi enviado um e-mail ao prof. Pedro Souto pedindo o adiamento do prazo de entrega do 5º lab devido a alterações que foram feitas na biblioteca fornecida pelo professor, visto que o novo prazo que tinha sido definido para essa compensação (17:30h) não era suficiente, tendo em conta que ambos os membros do grupo tinham aulas de tarde nesse dia. Esse e-mail ficou sem resposta;
- Foi enviado um e-mail ao prof. José Pinto a questionar o motivo de desconto de 5 pontos numa função do lab 2. Após cerca de uma semana sem resposta, foi necessário conversa pessoal para ter a cotação devidamente revista;
- Trabalho necessário para a unidade curricular superior a 6 créditos.

7. Instruções de instalação

O ficheiro conf/proj deve ser colocado na pasta /etc/system.conf.d. Isto pode ser feito com o comando "cp conf/proj /etc/system.conf.d" (necessário permissões *root*).