Snake

Relatório Final

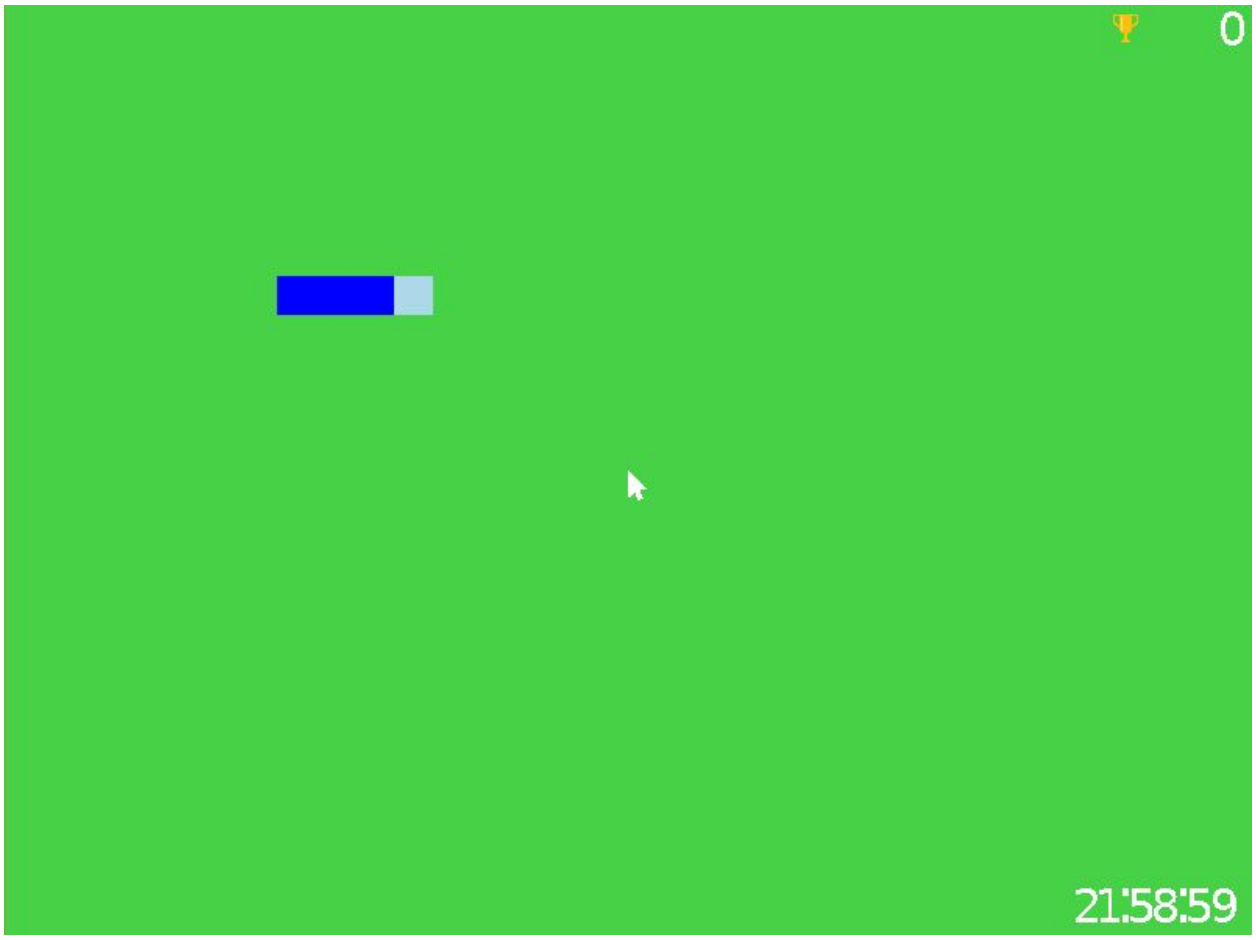
Mestrado Integrado em Engenharia Informática e Computação

Laboratório de Computadores

Grupo: T6G10
João Carlos Carreira Martins - up201605373@fe.up.pt
João Francisco de Pinho Brandão - up201705573@fe.up.pt

1. Instruções de utilização do programa

Este jogo foi feito para o **modo 115 da VBE**. Quando o utilizador compila e corre o programa, o jogo começa, como é demonstrado na imagem abaixo.



1.1. Jogabilidade

O sentido do movimento da cobra é controlado com as setas do teclado. Quando o botão esquerdo do rato é pressionado é colocada uma maçã onde se encontra o ponteiro do rato caso não haja nenhuma maçã no mapa e a cobra não esteja a passar nesse sítio.

Quando a cobra come a maçã, o seu tamanho aumenta em uma unidade, a maçã desaparece e podemos colocar novamente outra maçã no mapa clicando no botão esquerdo do rato.

1.2. Sair do jogo

Para sair do jogo o utilizador pode escolher uma destas opções:

- Pressionar a tecla **ESC**: o jogo fecha-se instantaneamente.
- Perder o jogo (fazer com que a cobra colida com si mesma): é definida a hora do alarme do RTC que é 4 segundos após perder o jogo. Após estes 4 segundos o jogo fecha-se.

2. Estado do projeto

2.1. Dispositivos utilizados

Na especificação do nosso projeto mencionamos usar todos os dispositivos possíveis. No entanto, acabamos por não usar a porta-série por falta de tempo. Os restantes dispositivos foram usados da seguinte forma:

Dispositivos	Uso	Interrupções
Timer	Controlar a frame rate da cena e a velocidade da cobra	Sim
Keyboard	Mudar o sentido do movimento da cobra e sair do jogo pressionando a tecla ESC	Sim
Mouse	Colocar a maçã no mapa	Sim
Video Card	Desenhar a cena	Não
Real-Time clock	Ler a hora atual e usar alarme para fechar o jogo 4 segundos após a cobra colidir com si mesma	Sim (na leitura da hora e no alarme)

2.1.1. Timer

O timer foi usado para controlar a frame rate da cena e a velocidade da cobra. A cada 1 interrupção, ou seja, uma frame rate igual a 60 fps, a cena é desenhada chamando a função **draw_scene** e a cada 3 interrupções a cobra move-se chamando a função **move_snake**.

2.1.2. Keyboard

O keyboard foi usado para mudar o sentido do movimento da cobra, chamando a função **change_snake_direction** que muda o sentido da cobra conforme a seta do teclado pressionada, e para sair do jogo pressionando a tecla ESC (quando o byte do keyboard for igual ao makecode da tecla ESC, 0x81, o ciclo do jogo termina).

2.1.3. Mouse

O mouse foi usado para colocar a maçã no mapa carregando no botão esquerdo. Cada evento do rato gera uma interrupção que chama a função **mouse_process_event**. Quando este evento é um clique do botão esquerdo é colocada uma maçã no mapa na posição do ponteiro do rato caso não haja nenhuma maçã já no mapa e a cobra não esteja a passar naquela posição. Quando este evento é um movimento do rato são atualizados os valores **x** e **y** da **struct Mouse *mouse** que guarda a posição do rato. O ecrã ao ser atualizado vai desenhar o rato na nova posição.

2.1.4. Video Card

O jogo foi feito para o **modo 115** (resolução 800x600, Direct Color, 24 bits per pixel r:g:b(8:8:8), aproximadamente 16.8 milhões de cores).
A video card foi usada para desenhar toda a cena, invocando a função **draw_scene**. Esta função, usando **double buffering** e chamando outras funções auxiliares, desenha o mapa, a cobra, a maçã (caso haja), o resultado, a hora e o ponteiro do rato.

2.1.5. Real-Time Clock

O RTC foi usado para ler a hora atual e usar alarme para fechar o jogo 4 segundos após a cobra colidir com si mesma. Para isso é chamada a função **rtc_ih** quando há uma interrupção do RTC. Esta função verifica a origem da interrupção (update ou alarm). Caso seja Update vai atualizar a hora, minutos e segundos da **struct Time *rtc_time** que será mostrada no ecrã quando este for atualizado. Caso seja Alarm vai tornar a variável do tipo booleano **alarm_interrupt** (função **game_loop**, ficheiro **proj.c**) verdadeira que fará com que o ciclo do jogo termine e assim o jogo se feche.

3. Organização e Estrutura do código

O projeto está organizado em três pastas: **res**, onde tem as fotos, **src**, onde tem o código e **doc**, onde tem a documentação doxygen, o relatório e um vídeo demo do nosso jogo.

3.1. proj.c

- **proj_main_loop** - função main, após correr o programa esta função é chamada mudando o modo da placa gráfica para o passado como argumento (no nosso caso, passamos o **115**). Depois inicializa o rato e a cobra, carrega todos os ficheiros xpm precisos e chama a função **game_loop**.

- **game_loop** - é nesta função que decorre o jogo. São subscritas todas as interrupções necessárias e depois entra no ciclo do jogo. Este ciclo, que continua enquanto o utilizador não perde o jogo e não carrega na tecla ESC, consiste em receber sinais de interrupção e tomar uma ação conforme a sua origem. Estas ações foram descritas na secção acima 2.1 Dispositivos Utilizados.

3.2. graphics.c

Neste módulo encontram-se todas as funções usadas para desenhar cor ou imagens no ecrã. A função que é chamada no ciclo do jogo para desenhar toda a cena é **draw_scene** que usando **double buffering** e chamando funções auxiliares desenha o mapa, a cobra, a maçã (caso exista), o resultado, a hora e o ponteiro do rato.

3.3. keyboard.c

Este módulo tem funções relacionadas com o teclado, nomeadamente a função **kbc_ih** que lê um byte do output buffer. No ciclo de jogo quando há uma interrupção do teclado esta função é chamada e conforme o byte lido é tomada uma ação: em caso de seta o sentido da cobra é mudado (a menos que o novo sentido seja oposto ao atual, por exemplo se a cobra estiver-se a mover para cima não se pode mudar o seu sentido para baixo diretamente), em caso de ESC o jogo termina, em caso de outra tecla qualquer não se faz nada.

3.4. mouse.c

Este módulo tem todas as funções relacionadas com o rato como ler o packet enviado pelo rato e fazer o seu parsing. Para além disso tem uma variável global **struct Mouse *mouse** que contém as coordenadas x e y da posição do rato, que é depois usada para desenhar o ponteiro do rato no ecrã e colocar uma maçã no mapa.

3.5. rtc.c

Este módulo contém todas as funções relacionadas com o RTC e uma variável global **struct Time *rtc_time** que contém as horas, minutos e segundos lidos do RTC. Esta variável é alterada a cada interrupção Update do RTC e é usada para desenhar a hora no ecrã.

3.6. snake.c

Este módulo contém funções relacionadas com a cobra como verificar colisões, mudar o sentido do movimento e mover-se. Contém também as variáveis globais **struct Snake *snake** e **struct Apple *apple**. A variável ***snake** contém uma variável do tipo **struct Node** com informações sobre a cabeça da cobra, o resultado (número de maçãs comidas) e o sentido atual da cobra. A estrutura do tipo **Node** contém o valor das coordenadas x e y da cabeça da cobra e um apontador do tipo **struct Node** para o próximo nó da cobra. Basicamente uma lista ligada.

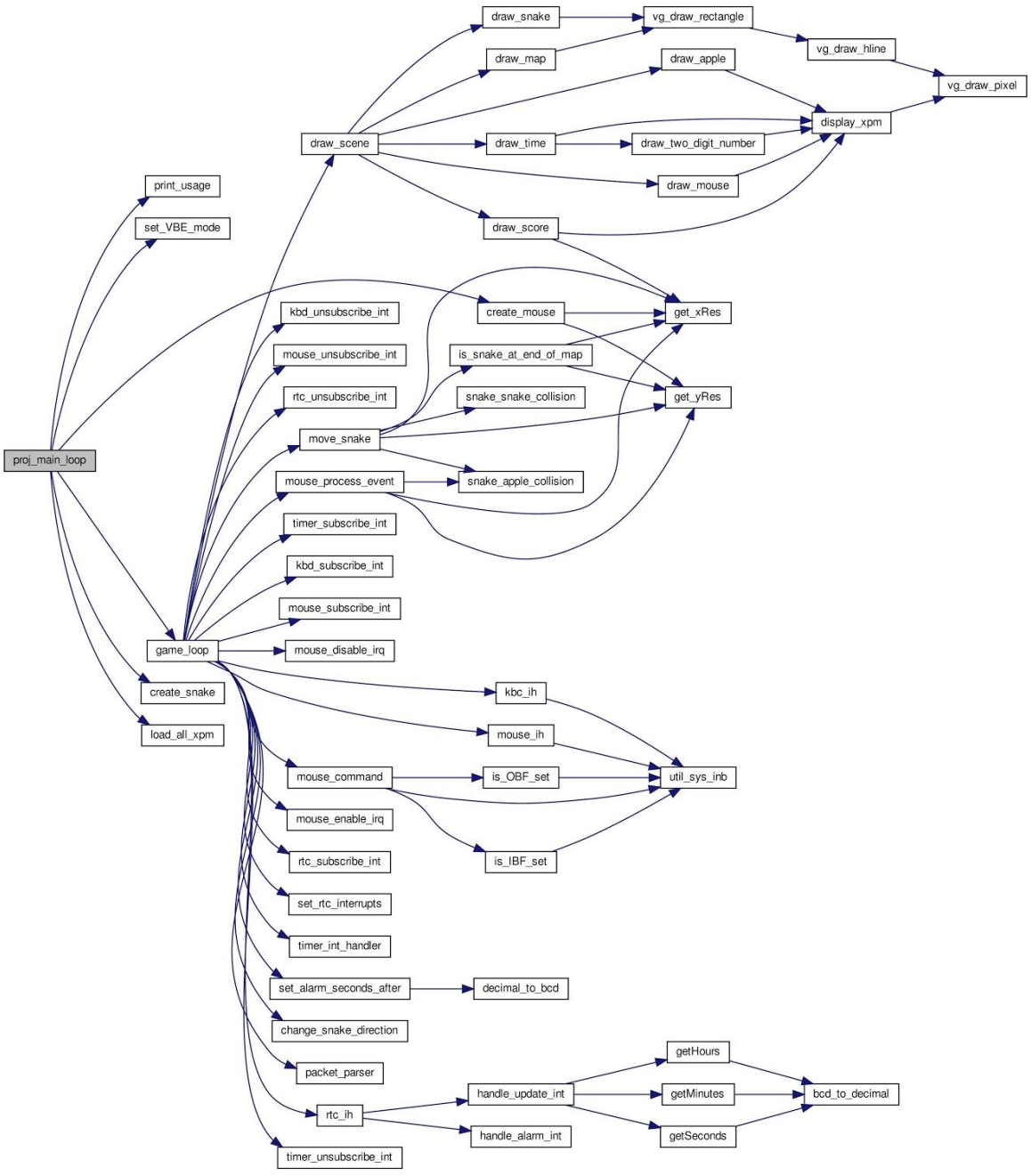
3.7. timer.c

Este módulo contém funções relacionadas com o timer.

Peso Relativo

Módulo	Peso relativo (%)	Feito por
proj.c	10	Ambos
graphics.c	30	Ambos
keyboard.c	3	Ambos
mouse.c	5	Ambos
rtc.c	15	Ambos
snake.c	35	Ambos
timer.c	2	Ambos

Caller Graph



Implementação

Para tornar a manipulação dos sprites mais fácil, todos os ficheiros xpm são carregados no início do jogo com a função **xpm_load** e a sua informação é guardada numa variável global do tipo **xpm_image_t**.

A posição (coordenadas x e y) do ponteiro do rato é guardada numa **struct Mouse**. Os seus valores são alterados cada vez que é detectado um movimento do rato. Esta estrutura é usada para desenhar o ponteiro no ecrã e para saber a posição onde plantar a maçã depois de ser pressionado o botão esquerdo do rato.

As informações da cobra foram guardadas numa **struct Snake** que tem o resultado (número de maçãs comidas pela cobra), o sentido atual da cobra e um apontador do tipo **struct Node** que guarda as coordenadas x e y da cabeça da cobra e outro apontador do tipo **struct Node** com o próximo nó da cobra, ou seja, foi implementada uma lista ligada para guardar informação da cobra.

A implementação do RTC foi fácil mas ainda deu um pouco de trabalho. Começamos por implementar, através de polling, a leitura da hora do RTC e depois desenhá-la no ecrã. Acabamos depois por implementar os alarmes do RTC e passamos a usar o RTC através de interrupções: quando é detectada uma interrupção do RTC é chamada a função **rtc_ih** que lê o byte do **Register C** e vê quais as flags que estão ativas (no nosso caso, **Alarm Flag** e/ou **Update-Ended Flag**) e no caso de **Alarm Flag**, altera o booleano passado por referência **alarm_interrupt** para verdadeiro o que faz terminar o ciclo do jogo, fechando assim o programa (nós fizemos com que o jogo se fechasse sozinho 4 segundos após o utilizador perder o jogo através do uso do alarme do RTC), e no caso de **Update-Ended Flag** lê a hora, os minutos e os segundos do RTC (isto é feito enviando para o registro **0x70** o registro do qual queremos ler que é **0x04** para horas, **0x02** para minutos e **0x00** para segundos e posteriormente ler do registro **0x71** o respetivo valor).

Conclusões

LCOM foi uma cadeira com muitas horas de trabalho mas também bastante interessante. Aprendemos várias coisas como SVN, máquinas virtuais, C e dispositivos I/O.

Quanto aos pontos negativos, alguns guiões (especialmente o da placa gráfica pela nossa experiência) não estão muito bem explicados o que torna difícil tentar fazer as labs sozinhos mas este ponto é compensado com o auxílio prestado pelo professor e monitor. O facto de estarmos a fazer esta cadeira pela segunda vez ajudou-nos bastante pois não é a primeira vez que vemos esta matéria. No entanto, para alguém que está a fazer esta cadeira pela primeira vez pode ser um pouco intimidante e levar os alunos a desistir (exatamente o que aconteceu connosco no ano passado). Uma sugestão seria mudar o plano de estudos do 1º/2º ano pois temos muitas cadeiras de matemática e física. Mudar o plano para cadeiras com trabalhos de grupo e mais programação ajudaria os alunos a ter os conhecimentos de programação e “work ethic” necessários para fazer LCOM.

Quanto aos pontos positivos, o acompanhamento dos professores e monitores nas aulas é muito bom. Aqui temos que agradecer ao professor Pedro Souto e ao monitor Rui Alves pela ajuda prestada nas aulas práticas. Perceber os guiões sozinhos ia ser muito mais difícil. As labs a cada duas semanas também são boas para que haja uma certa rotina e assim torna a cadeira mais fácil.