Faculdade de Engenharia da Universidade do Porto

LCOM-T2G02 – "Duck Hunt"

André Reis – up201403057

João Gomes – up201403275

4 de Janeiro de 2015

ÍNDICE

Introdução

O nosso projeto, intitulado de "Duck Hunt" foi realizado, no âmbito da cadeira Laboratório de Computadores, é uma versão do jogo “Duck Hunt”, tal como o nome sugere.

O jogo em si é muito simples. O jogador tem apenas que com o rato matar os patos que vão aparecendo no ecrã. Quando não consegue matar um pato durante um certo tempo, este foge e o jogador perde uma vida. O jogo termina quando o jogador perder as suas três vidas. Quando o jogador consegue matar um pato, a pontuação aumenta conforme o tempo que o pato teve no visível no ecrã (quanto menos tempo, mais pontos) e com a velocidade do pato (quanto maior, mais pontos) visto que esta aumenta com o tempo de jogo. Existem ainda “easter eggs”.

Este projeto foi desenvolvido para ir ao encontro dos objetivos da cadeira que incluem:

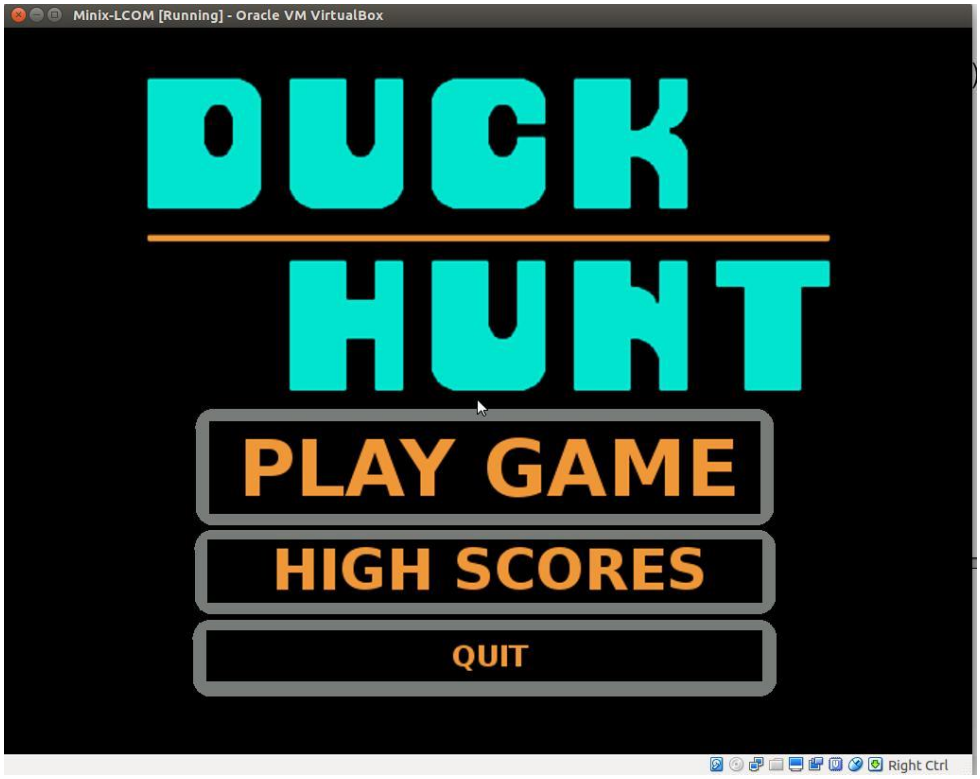
-Programar ao nível de hardware de periféricos de um computador ;

- Desenvolver programas de baixo-nível;

- Usar ferramentas de software usados em grandes projetos de programação.

Instruções de utilização

Quando inicia o programa, aparece o menu inicial:



Poderá escolher com o botão esquerdo do rato a opção que quiser.

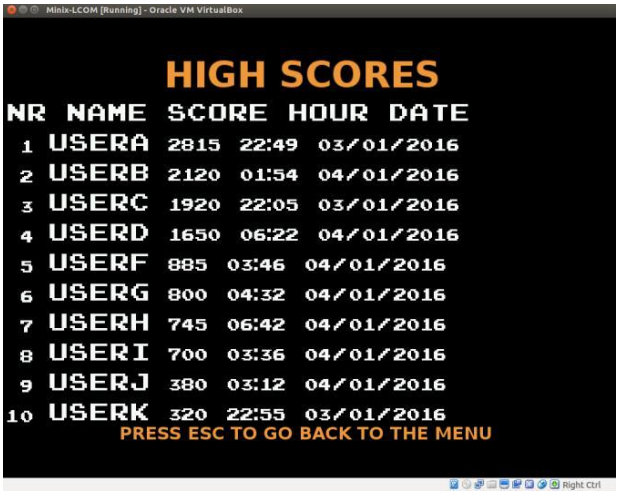
Se escolher “PLAY GAME”, o jogo começa, como mostra a imagem:

O jogador usa o rato para mover a mira e acertar no pato com o botão esquerdo do rato. Ele pode ainda introduzir o “KONAMI code” (código KONAMI – cima, cima, baixo, baixo, esquerda, direita, esquerda, direita, B, A, usando as setas do NUMPAD) e ter acesso ao primeiro “easter egg”. Ao inserir uma segunda vez este código, acede ao segundo “easter egg” e uma terceira vez, volta ao modo normal.

Ao perder o jogador é direccionado para o ecrã de fim de jogo, como mostra a imagem:

Neste ecrã o jogador deve inserir até cinco letras para se identificar seguidas da tecla ENTER para guardar a pontuação e seguir para o menu principal. Pode usar as setas do NUMPAD para navegar no nome de forma a efectuar correcções.

Se escolher “HIGHSCORES”, ira para a tabelas das dez pontuações mais altas. Como mostra a imagem:



Se não existirem dez pontuações mostra as que existem. Ao largar a tecla ESC volta ao menu principal.

Se carregar na opção “QUIT”, o programa fecha.

Funcionalidades Implementadas

Como foi referido na especificação do projeto, o periférico de porta de série era visto como um módulo opcional e não foi possível implementá-lo.

Acabámos também por não implementar um “reload” da arma visto que não nos pareceu uma mecânica interessante para o jogo.

Sendo assim, acabámos por implementar os seguintes periféricos:

Periférico	Utilização	Interrupções
Timer	Atualizar o estado do jogo	Sim
Teclado	Interação com interface (Scores e easter egg)	Sim
Rato	Interaçâo com interface (Jogo)	Sim
Placa Gráfica	Visualização do ecrã do jogo	Não
RTC	Data e hora do fim jogo (guardado nos Scores)	Não

Timer

As interrupções do timer 0 são usadas para:

-atualizar o estado do jogo, sendo o jogo atualizado 60 vezes por segundo;

-manter registo do tempo de jogo decorrido;

-limitar a duração de certas acções.

A verificação e tratamento destas interrupções está no ficheiro DuckHunt.c:

-“Handlers” – linhas 233-245 e linhas 385-402;

Teclado

As interrupções do teclado são usadas para ler as teclas que o utilizador prime. Isto é usado:

-para ter acesso aos “easter eggs” – linhas 280-333 – DuckHunt.c;

-para ler o nome/identificador do jogador no ecrã de Game Over – linhas 403-426 - DuckHunt.c;

-para sair da tabela de Highcores e do ecrã de Game Over – linhas 144-147 – Scores.c e linhas 403-430 - DuckHunt.c;

Rato

As interrupções do rato foram usadas para ler os *packets* do rato, para, com estes, actualizar a posição e estado do cursor/mira do jogo (posição do rato e botão do lado esquerdo para os *clicks*). Isto foi útil para :

-Escolher opções de menu – linhas 97-128 - DuckHunt.c;

-Apontar e disparar no jogo em si – linhas 205-232 - DuckHunt.c;

Placa gráfica

Para a utilização da placa gráfica, inspiramo-nos no código do blog do monitor Henrique Ferrolho (http://difusal.blogspot.pt/2014/07/minix-posts-index.html). A fonte-base deste código é http://forums.fedoraforum.org/archive/index.php/t-171389.html .

O modo de vídeo usado é o 0x117, resolução 1024x768p, 2 bytes por pixel, que disponibiliza na teoria 2^16 cores e o modo de cores é 5:6:5 (5 bits para a informação da cor vermelha, 6 para a informação da cor verde e 5 para a informação da cor azul – Não tem informação de transparência).

Inspirado no blog acima, usamos *double buffering* mais um buffer extra para desenhar o rato.

Para as imagens usamos o código de *bitmaps* disponibilizado. Para animar os bitmaps usamos a estrutura *Animated Sprite* disponibilizada nas aulas, alterada para maior conveniência do uso. Num nível acima, ainda temos a estrutura Duck que possuí um *array* de *Animated Sprite* para permitir animações diferentes.

Quanto a colisões, as únicas existentes no projecto são: as com os limites do ecrã (não muito relevantes) e a do rato com o pato. O método de detectar colisão usado foi: Verificar se a mira do rato está dentro das coordenadas do rectângulo do bitmap do pato e depois verificar se, nesse bitmap, a cor não é a cor de fundo/transparência (se não for, é porque acertou no pato).

Para escrever no ecrã, na maioria dos casos (que não requeriam uma estilização especial) usámos uma *font* "criada" por nós – isto é – aproveitámos a *font* da **NES** e aumentámos e reajustámos as letras, os dígitos e os caracteres ':' e '/' face à necessidade.

A maioria do código deste dispositivo está nos ficheiros: vbe.c videoGraphics.c bitmap.c AnimSprite.c Duck.c e Font.c – Embora o código seja transversal a quase todos os ficheiros.

RTC

O RTC é usado para obter a data actual quando é guardado um *highscore*. O código referente a isto está situado no ficheiro RTC.c e na linha 76 do ficheiro Scores.

Estrutura de código

Módulos

-AnimSprite.c

Este módulo implementa as funções de criar, actualizar e desenhar AnimSprites (implementa a estrutura AnimSprite, também), para poder ser usado depois no Duck.c.

O responsável deste módulo foi o João e teve um peso de 5%.

-bitmap.c

Este módulo implementa as funções de criar e desenhar bitmaps (implementa a estrutura bitmap) para ser usado nas AnimSprites e no desenho de imagens em geral.

O responsável deste módulo foi o João e teve um peso de 5%.

-Duck.c

Neste módulo implementámos a estrutura Duck, que contém a informação necessária para o pato e as funções de criar, inicializar posições e actualizar o estado, em geral.

O responsável foi o André e teve um peso de 20%.

-DuckHunt.c

Este módulo implementa as funções principais do projeto, que têm interacção com o utilizador. É, basicamente, a camada mais superficial do projecto.

O responsável foi o André e teve um peso de 25 %.

-Font.c

Este módulo implementa as funções de usar a font criada para escrever no ecrã.

O responsável foi o João e teve um peso de 5%.

-Keyboard.c

É o módulo de implementações relativas ao teclado.

O responsável foi o João e teve um peso de 5%.

-Mouse.c

É o módulo de implementações relativas ao rato.

O responsável foi o João e teve um peso de 5%.

-RTC.c

É o módulo de implementações relativas ao RTC.

O responsável foi o João e teve um peso de 5%. -

Scores.c

Neste módulo implementámos as funções que escrevem e lêem do ficheiro dos Highscores.

O responsável foi o André e teve um peso de 10%.

-Timer.c

É o módulo de implementações relativas ao timer.

O responsável foi o João e teve um peso de 5%.

-utilities.c

É o módulo de funções utilitárias que não se adaptam em específico a nenhum dos outros módulos mas que são necessárias para estes.

O responsável foi o João e teve um peso de 5%.

-vbe.c

É o módulo trabalhado nas aulas práticas que contém a função que lê a informação do modo de vídeo.

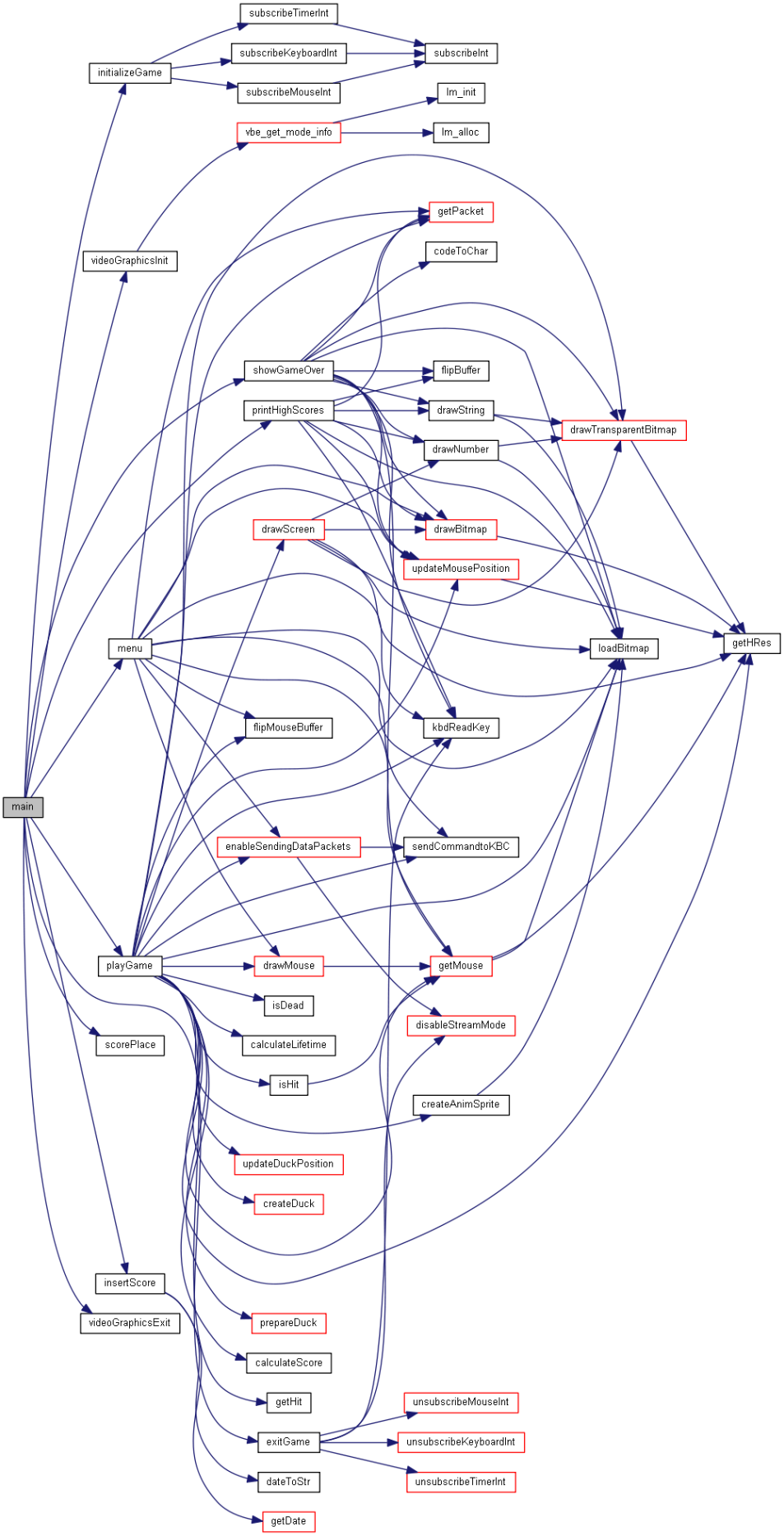
Teve um peso de 0% porque já estava implementado.

-videoGraphics.c

Neste módulo implementamos as funções de interacção com a memoria grafica e os buffers de memória.

O responsável foi o João e teve um peso de 5%.

Function Call Graph



Detalhes de Implementação

Quanto aos detalhes de implementação, fazemos notar que organizámos o nosso trabalho por camadas, desde a implementação base de periféricos, a camada superior que usa esses periféricos e a camada superficial que utiliza as funções que usam os periféricos.

Ainda quanto à organização de código, implementámos as nossas estruturas no estilo de *object oriented* C.

Quanto a implementação de máquinas de estado e código “event-driven” , usámos com a enumeração do estado do pato na estrutura Duck, uma espécie de máquina de estados na função principal playGame(), que a interrupção do Timer agia de maneira diferente conforme o estado do pato.

Quantos às dificuldades que não foram cobertas nas aulas e tivemos que aprender por nós, sentimos algumas na gestão de memória e apontadores e nas funções utilizadas para ficheiros e *strings* em C. Em suma, aspectos básicos de C.

Conclusão

Pensámos que é bastante trabalhosa para os créditos que tem, é difícil adaptar à programação a baixo-nível e em C e o modo em que são dados os *labs* (os alunos têm que aprender por si, em grande parte) tornam a cadeira desinteressante até ao projecto. No entanto, nesta última torna-se interessante e cativante de vermos os resultados do nosso trabalho de uma maneira prática.

A contribuição para o relatório final foi de 50% para cada um.

Apêndice

Baseamo-nos nos scripts disponibilizados pelo monitor Henrique Ferrolho e alterámos de modo a ser compatível com o nosso programa.

Para os ficheiros, não usámos caminhos relativos, mas aos instalar o programa criamos uma pasta no directório *home* com os recursos necessários para o programa de forma a ser seguro aceder com caminho absoluto.

Para correr o programa deve, dentro do directório *proj*, executar os seguintes comandos, por esta ordem:

sh install.sh

sh compile.sh

sh run.sh