ASTERINIX

Relatório

Daniel Alexandre Pimenta Lopes Fernandes - up201406329@fe.up.pt
Tiago Lascasas dos Santos - up201503616@fe.up.pt

Índice

Utilização do programa

Menu

Ao iniciar o programa é apresentado um menu inicial com várias opções. Pode-se optar por jogar um jogo, ver a tabela de pontuações ou consultar um manual que explica o funcionamento do jogo.

Ao passar com o rato sobre um botão, este muda de cor de modo a mostrar que é clicável.

Jogo

Caso se opte por iniciar o jogo, dever-se-á ter um ecrã semelhante a este:

Este é o ecrã de jogo, no qual se pode ver a nave do jogador, os tiros que a sua nave dispara e os asteroides.

Para mover a nave, deve-se utilizar as teclas WASD ou então as setas. Para disparar deve-se utilizar o botão esquerdo do rato. Cada asteroide destruído incrementa em 1 ponto o valor da pontuação.

O jogador tem 3 vidas. Quando perde 1 vida tem uma “spawn protection”, visto que um asteróide pode estar a passar pelo spawn point no momento em que o jogador ressuscita, o que o mataria de novo.

A “spawn protection” dura dois segundos e enquanto está ativa uma mensagem é mostrada no topo do ecrã, como evidenciado na figura acima.

Caso o jogador perca as 3 vidas, perderá o jogo, apresentando-se o seguinte ecrã:

Se a pontuação do jogador estiver entre as nove melhores, um ecrã diferente é apresentado, mostrando que a sua pontuação ficou registada na tabela de pontuações:

Pontuações

Podem também ser vistas as pontuações:

É possível apagar todas as pontuações clicando no botão Reset Scores. Para voltar para trás, basta clicar no botão Back.

Ecrã How To Play

É ainda possível consultar um pequeno manual de utilização do jogo:

Por fim, clicar na tecla esc em qualquer altura do jogo causa a saída imediata do mesmo.

Estado do projeto

Descrição geral

A única funcionalidade não implementada é o modo multijogador, que teria utilizado a porta de série. O código de interface da porta em série encontra-se, contudo, no projeto.

Dispositivo	**Função**	**Interrupções?**
Timer	Controlar a frame rate e definir o tempo entre balas e de spawn.	Sim
Teclado	Controlo da nave no jogo e de alguns menus.	Sim
Rato	Apontar e disparar balas da nave e clicar em menus.	Sim
Placa gráfica	Display do menu e do jogo.	Não
RTC	Criação de uma seed para a geração de números aleatórios e providenciar uma timestamp com a data e a hora para a tabela de pontuações.	Não

Timer

O timer é utilizado neste projeto para ter controlar a frame rate do jogo. A cada interrupção gerada pelo timer, é criado e desenhado um novo frame. A função que utiliza o timer é **updateInput(Game* game, Asterinix* asterinix),** presente no ficheiro Game.c. Esta função gere as interrupções de vários dispositivos, entre os quais o timer, processando a interrupção ao ativar a flag game->tick no estado do jogo. Esta flag, quando ativa, permite a actualização do estado do jogo e o desenho de um frame.

Teclado

O teclado é usado um pouco por todo o programa. A sua principal funcionalidade é no jogo em si, no qual é usado para controlar a nave. No entanto, também pode ser usado nalguns menus. A função que utiliza o teclado é também **updateInput(Game* game, Asterinix* asterinix).** Uma vez gerada uma interrupção no teclado, é primeiramente feita uma leitura de um único byte do teclado, seguida de um update de um objeto do tipo **key_status_t**, que é uma estrutura com o estado das teclas (premidas/libertadas), chamando a função **updateKeyStatus(key_status_t* key_status, scancode sc)**, que atualiza esse objecto. É realizado o devido processamento de scancodes com dois bytes:

se o byte lido for 0xE0, que é o byte mais significativo de um scancode com dois bytes, a estrutura não atualiza o estado de nenhuma tecla mas ativa uma flag a indicar que o scancode é de dois bytes, esperando pela próxima interrupção de modo a obter o byte menos significativo do scancode e só então atualizar o estado da tecla correspondente. O objeto do tipo key_status_t encontra-se referenciado numa struct do tipo Asterinix, que contém o estado dos dispositivos.

Rato

O rato é utilizado para disparar balas com a nave, bem como para navegar nos menus. Para ambas estas funcionalidades são usados a posição e os botões deste dispositivo. Mais uma vez, a parte referente ao rato começa em **updateInput(Game* game, Asterinix* asterinix).** Após a geração de uma interrupção, é lido um único byte do rato, e de seguida é chamada a função **updateMouse(Mouse* mouse, char packet)**, que atualiza um objecto do tipo Mouse (que é uma estrutura com um sprite do rato, informação sobre o estado dos botões, o deslocamento e o packet atual), que processa o byte lido, inferindo qual a sua posição no packet. Quando esse objeto tem os três bytes do packet operacionais, realiza o seu processamento, atualizando o estado dos botões e a posição do rato com base no deslocamento relativo. Estas duas atualizações são feitas chamando, respectivamente, as funções **updateMouseButtons(Mouse* mouse)** e **updateMouseMoves(Mouse* mouse)**. O objecto do tipo Mouse encontra-se, tal como o key_status_t, referenciado na única instanciação de uma estrutura do tipo Asterinix.

Placa Gráfica

A placa gráfica é utilizada para desenhar no ecrã o estado atual do jogo. A resolução utilizada é 1024x768 a 16-bit (modo 0x117). O modo de vídeo é inicializado partindo da função **vg_init(unsigned short mode),** função essa que faz uso da biblioteca liblm.a e que para além de alocar espaço para o buffer de vídeo habitual, aloca também espaço para um buffer auxiliar, de modo a tornar possível o double buffering, implementado recorrendo a funções escritas em Assembly (e cujos detalhes estão mais abaixo, na parte da implementação). A nível visual, existem objetos em movimento (asteroides, por exemplo), bem como animações (explosões). É também utilizada uma fonte de modo a desenhar pontuações e timestamps, e todas as imagens são carregadas a partir de bitmaps, usando a biblioteca do Henrique Ferrolho (creditada mais abaixo), e à qual foi adicionada a funcionalidade de permitir desenhar com transparência (também detalhada na parte da implementação). A função que mais uso faz da placa gráfica é **draw(Game* game, Asterinix* asterinix)**, função esta que desenha o estado do jogo no ecrã: primeiro, procura saber qual a fase/estado do jogo (menu, jogo, ecrã de pontuações, etc) e desenha os vários elementos de acordo com essa mesma fase. Independentemente da fase, o rato é sempre desenhado, e no final, quando estão todos os elementos desenhados, segue-se a implementação do double buffering.

No final do programa, é reativado o modo de texto recorrendo à função **vg_exit()**.

RTC

O RTC é usado para criar uma seed para a geração de números aleatórios e para a criação de *timestamps*. A leitura da data e da hora é feita a partir da função **rtc_get_date_and_time(char* date)**, que configura o RTC para o modo binário e de 24 horas e que efetua a leitura dos registos de data e horas, preenchendo o array *date que lhe é passada como argumento. A escrita para o RTC é feita usando a função **rtc_write(int address, char info)** e a leitura é feita usando a função em Assembly **rtc_read_asm**. O processo referente à transformação da informação do RTC em timestamps UNIX encontra-se mais abaixo, na secção dos detalhes de implementação.

Estrutura e organização do código

Descrição dos módulos

Aqui estão listados, alfabeticamente, todos os módulos que compõem o projeto, com as suas respectivas funcionalidades, pesos e contribuições.

Módulos em C:

Asterinix (.c e .h) - 5% do projeto, desenvolvido pelo Tiago Santos

Este módulo contém a estrutura Asterinix, que contém o estado dos dispositivos, bem como as suas subscrições de interrupções. Contém também duas estruturas adicionais, BitmapCollection e Font, que carregam e guardam apontadores para bitmaps e para uma fonte, respectivamente.

Asteroid (.c e .h) - 5% do projeto, desenvolvido 80% pelo Daniel Fernandes, 20% pelo Tiago Santos

Este módulo contém a estrutura Asteorid, que compõe um asteróide. Possui toda a lógica, atributos e estados para que o asteroide seja gerado, percorra o seu trajeto e desapareça ou seja abatido, incluindo para tal efeito funções que definem o movimento do asteróide e que geram posições aleatórias nas bordas do ecrã para o seu respectivo *spawn*.

Bitmap (.c e .h) - 2% do projeto, desenvolvido por outrem. Mudanças feitas pelo Tiago Santos

Este módulo contém funções para carregar, desenhar e apagar bitmaps, e foi originalmente criada por Henrique Ferrolho. Foi adicionada a possibilidade de desenhar com transparência à função **drawBitmap**. A fonte encontra-se no fim do relatório.

Bullet (.c e .h) - 3% do projeto, desenvolvido pelo Daniel Fernandes

Este módulo contém a estrutura Bullet, que é utilizada pelo Player para a manutenção da lógica de uma bala. É mantido o estado da bala, bem como todos os aspetos essenciais a que esta seja criada, percorra o seu trajeto e desapareça (por sair do mapa ou atingir um alvo).

Game (.c e .h) - 25% do projeto, desenvolvido equitativamente por ambos

Este módulo contém a estrutura Game, que gere a lógica de jogo. Esta inclui o jogador (Player), a fase atual (Stage), bem como outros elementos essenciais a que o jogo funcione corretamente. Contém a função que gere as interrupções, que atualiza o estado do jogo e que desenha o frame atual no ecrã. É servido ainda do ficheiro Positions.h, com constantes que contém coordenadas dos sprites e/ou bitmaps a desenhar.

Graphics (.c e .h) - 3% do projeto, desenvolvido equitativamente por ambos

Este módulo contém as funções que inicializam/inibem o modo gráfico, assim como funções que pintam um pixel com uma determinada cor ou preenchem o ecrã, mas estas últimas foram apenas utilizadas na fase inicial do projeto. Contém ainda uma estrutura Coord, que representa uma coordenada (x,y) e que é usada na geração aleatória do *spawn* dos asteroides. Utiliza o ficheiro **video_defs.h**, que contém macros e constantes necessários à configuração do modo de vídeo, assim como alguma manipulação de cor.

Highscores (.c e .h) - 4% do projeto, desenvolvido pelo Tiago Santos

Este módulo contém duas estruturas: a estrutura Highscore guarda informação sobre uma pontuação, incluindo o seu valor e a sua timestamp. A estrutura HighscoreTable possui um array com nove apontadores para Highscores, assim como funções que permitem a inserção de novos highscores na tabela. Contém funções de desenho para a tabela e respectivas pontuações e possui ainda funções de leitura e escrita para um ficheiro de texto, onde são guardadas as pontuações.

Keyboard (.c e .h) - 5% do projeto, desenvolvido equitativamente por ambos

Este módulo contém uma interface para trabalhar com o teclado, incluindo funções de subscrição de interrupções e respectivo cancelamento e funções de leitura e escrita. Possui ainda uma estrutura key_status_t, que guarda o estado (premido/libertado) de todas as teclas necessárias ao programa, e funções que permitem a sua instanciação e atualização. Utiliza o ficheiro **i8042.h**, que contém comandos e registos necessários para interagir com o teclado.

main (.c) - 1% do projeto, desenvolvido pelo Tiago Santos

O ficheiro main serve de entrada do programa. Inicializa o serviço, permite I/O em Assembly, ativa e configura o rato, entra no modo de vídeo predefinido, inicializa o gerador de números aleatórios, cria as estruturas necessárias ao jogo (do tipo Game e Asterinix.) e, por fim, inicializa o jogo em si. Quando termina, cancela as subscrições dos periféricos, desativa o rato e volta ao modo de texto.

Mouse (.c e .h) - 5% do projeto, desenvolvido equitativamente por ambos

Este módulo contém uma interface para trabalhar com o rato, incluindo funções de subscrição de interrupções e respectivo cancelamento e funções de leitura e escrita. Possui também uma estrutura Mouse, que contém um sprite para o apontador no ecrã, informação sobre os botões e o movimento/deslocamento e ainda o packet atual, assim como funções para instanciar e atualizar objetos desse tipo. Utiliza o ficheiro **mouse_defs.h**, que contém comandos e registos necessários à interação com o rato.

Player (.c e .h) - 6% do projeto, desenvolvido pelo Daniel Fernandes

Este módulo contém a estrutura Player, que gere a lógica do jogador. Possui todos os atributos necessários para que o jogador possa ter todos os estados cruciais ao funcionamento do jogo, incluindo o seu sprite, a informação sobre o seu movimento, as suas vidas e as suas balas atuais.

RTC (.c e .h) - 4% do projeto, desenvolvido pelo Tiago Santos

Este módulo contém a interface para aceder ao RTC. Contém funções de read/write, funções inline de inibição/reabilitação de interrupções e a função que carrega um array com a data e hora atual. Utiliza o ficheiro **RTC_defs.h**, que contém constantes com os registos do RTC.

Score (.c e .h) - 2% do projeto, desenvolvido pelo Tiago Santos

Este módulo contém a estrutura Score, que contém uma pontuação e a sua representação recorrendo a uma fonte, assim como funções de incremento e de reset.

Sprite (.c e .h) - 5% do projeto, desenvolvido 70% pelo Tiago Santos, 30% pelo Daniel Fernandes

Este módulo contém duas estruturas: a estrutura Sprite contém um bitmap, uma posição (x e y) e uma velocidade (x e y). A estrutura MutableSprite é em tudo semelhante à primeira mas contém um array de bitmaps em vez de um único bitmap, permitindo a animação do mesmo. Ambas as estruturas possuem funções que as manipulam, desenham e, no caso da MutableSprite, muda qual o sprite que está ativo num determinado momento.

Stage (.c e .h) - 15% do projeto, desenvolvido equitativamente por ambos

Este módulo contém a estrutura Stage, que é a estrutura responsável pela manutenção da fase atual do jogo, seja ela o mapa de jogo ou os menus.Possui funções que efetuam a atualização de todos os objetos pertencentes às várias fases do jogo, como por exemplo (e dependendo do tipo de fase) dos asteróides, do jogador, das balas ou dos botões dos menus. Verifica ainda a colisão de asteroides com o jogador e de balas com os asteroides.

Timer (.c e .h) - 2% do projeto, desenvolvido equitativamente por ambos

Este módulo contém funções de subscrição (e respectivo cancelamento) das interrupções do timer. É servido do ficheiro **i8254.h**, que contém constantes necessárias à subscrição de interrupções.

UART (.c e .h.) - 0% do projeto, desenvolvido pelo Tiago Santos

Este módulo contém a interface para interagir com a porta em série, incluindo funções de subscrição e cancelamento de interrupções e receção e transmissão de bytes em modo polled e em modo de interrupções. Não possui peso no projeto visto o modo multijogador não ter sido implementado. Utiliza ainda o ficheiro **UART_defs.h**, que contém constantes com os valores dos registos da porta em série.

Utilities (.c e .h) - **3% do projeto, desenvolvido pelo Tiago Santos**

Este módulo contém funções diversas, como uma que cancela todas as subscrições dos periféricos, uma que cria caminhos absolutos para ficheiros a partir de um relativo e ainda duas funções que interagem com o RTC, retornando timestamps.

VBE (.c e .h) **- 1% do projeto, desenvolvido equitativamente por ambos**

Este módulo contém a estrutura vbe_mode_info_t, que após inicializada contém informação sobre o modo de vídeo atual.

Módulos em Assembly:

Asm_utils.S **- 2% do projeto, desenvolvido pelo Tiago Santos**

Este módulo contém duas funções relacionadas com a aplicação do double buffering e com funcionalidades semelhantes a memset e memcpy.

kbd_read_asm.S **- 1% do projeto, desenvolvido equitativamente por ambos**

Este módulo contém uma função que efetua a leitura de um byte do teclado, verificando erros de leitura ou de timeout.

rtc_read_asm.S **- 1% do projeto, desenvolvido pelo Tiago Santos**

Este módulo contém uma função que efetua a leitura de um byte de um registo do RTC que lhe é passado por argumento.

Detalhes de implementação

Destacam-se os seguintes detalhes de implementação:

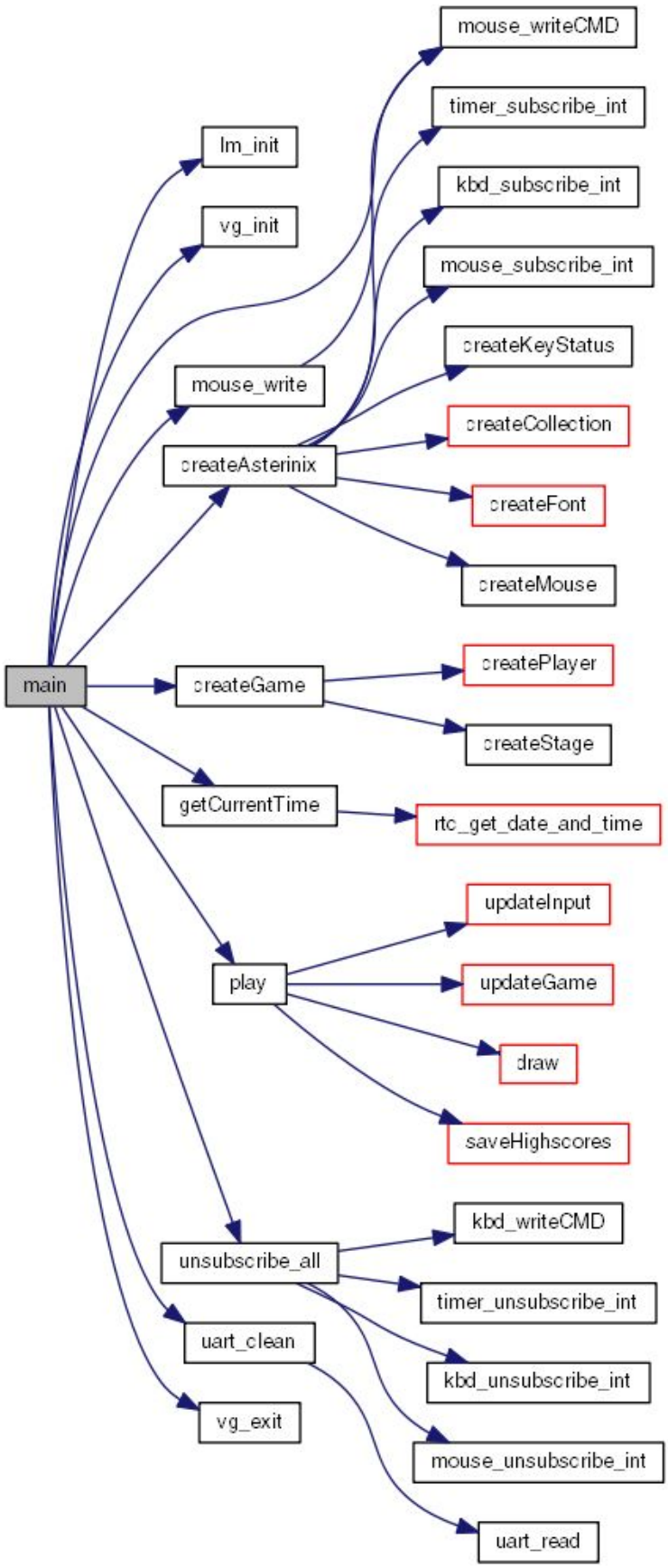
- **Double Buffering**: Sempre que um elemento é desenhado, é sempre usado um buffer auxiliar. Quando todos os elementos estão desenhados, o conteúdo desse buffer é copiado para a memória de vídeo habitual recorrendo à função em Assembly **copyBuffer** e depois é limpo recorrendo à função em Assembly **cleanBuffer**.

- **Geração de timestamps**: após o preenchimento do array com a informação de data e hora do RTC, pode-se efetuar dois procedimentos distintos: para a criação de uma seed é invocada a função **getCurrentTime()**, em que os valores carregados para o array são usados para preencher uma struct do tipo tm da biblioteca standard de C, que de seguida é usada para criar um objeto do tipo time_t, obtendo-se assim uma timestamp UNIX equivalente ao que se teria se se invocasse time(NULL). Para a criação de uma timestamp legível é invocada a função **getCurrentTimeStr()**, que é em tudo semelhante à anterior mas que em vez de retornar um objeto do tipo time_t, retorna uma string contendo uma timestamp obtida através da função **asctime(tm* time),** também da biblioteca standard de C.

- **Implementação da transparência**: à biblioteca de bitmaps utilizada foi adicionada a possibilidade de desenhar usando uma cor passada por argumento como transparência. Caso a função seja invocada com essa possibilidade ativada, ao copiar o conteúdo do bitmap para o buffer de memória, em vez de ser realizado um memcpy é usado um ciclo específico, em que todas as palavras de 16-bit iguais à cor passada como argumento são ignoradas, o que permite manter a informação que está por “detrás” da imagem naquele ponto.

- **Caminho relativo**: o caminho dos dos ficheiros .txt e .bmp a serem lidos é sempre relativo, sendo que o caminho absoluto do executável é passado como argumento da função main. O caminho absoluto dos ficheiros é criado juntando o caminho do executável e o caminho relativo usando a função **makePath(char* path, char* file)**. Uma correção que foi necessária fazer foi eliminar a parte final do caminho do executável, “/asterinix”, que é um resíduo obtido por usar `pwd` como argumento.

Gráfico de chamada de funções condensado



Avaliação da unidade curricular

Ao longo de toda a unidade curricular foi possível efetuar uma ligação mais próxima aos diversos dispositivos do computador. Sendo uma disciplina onde se trabalha em níveis de abstração bastante reduzidos, torna-se bastante fácil cair numa diversidade enorme de erros, e torna-se ainda mais complicado (em comparação com disciplinas nas quais se trabalha a níveis mais altos de abstração) identificá-los, bem como corrigi-los.

No geral, a forma como a disciplina funciona nas aulas práticas é boa, estando, do nosso ponto de vista, muita informação à nossa disposição que nos pode ser útil. Os guiões dos trabalhos são suficientemente detalhados e contêm importantes dicas relacionadas com problemas recorrentes. No entanto, continua a haver bastantes problemas aos quais se torna difícil fugir. Referimo-nos em boa parte a problemas relacionados com a Virtual Box, o Eclipse e a ligação com o Minix. É bastante recorrente a presença de erros no nosso código que podem danificar o estado da Virtual Box. Nesta disciplina, e estando a trabalhar em C e Assembly, é bastante fácil depararmo-nos com erros relacionados com memória mal alocada, tentativa de acesso a locais indevidos de memória, entre outros problemas deste tipo. Estes são problemas que por vezes suscitam um reboot na Virtual Box, ou outras eventuais medidas que nos retiram substancialmente tempo. Queremos com isto dizer que por vezes acabamos por perder mais tempo a resolver problemas relacionados com a Virtual Box ou a ligação ao Minix do que verdadeiramente a produzir código. Acabamos por perder mais tempo a configurar todo o setup outra vez do que realmente a procurar e corrigir o problema no código.

Em relação ainda ao funcionamento das aulas, achamos que a extensão do prazo de entrega em alguns labs por 24 horas foi de louvar, visto que retirou bastante da pressão causada sobre a data de entrega e permitiu obter um resultado mais polido.

Por fim, julgamos que pode fazer alguma falta uma prévia introdução à linguagem C, que apesar de ser relativamente intuitiva, principalmente para quem programou em C++ (nosso caso), tem alguns aspetos relacionados com gestão de memória que podem suscitar algumas dúvidas e que retiram porventura algum tempo na realização dos diversos laboratórios.

Instruções de instalação

Para correr o programa após ter compilado usando make e colocado o ficheiro asterinix da pasta conf em /etc/system.conf.d/, basta executar o script sh run.sh, script este que contém o comando service run `pwd`/asterinix -args `pwd`.