U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

RUN ROCKY RUN

Projeto de Laboratório de Computadores – MIEIC – 2º Ano, 2º Semestre

JANEIRO DE 2019

TURMA 4 – GRUPO 08
Pedro Esteves - up201705160@fe.up.pt
Diogo Mendes – up201605360@fe.up.pt

Agradecimentos

Serve a presente secção deste relatório para agradecermos a todo o *staff* da unidade curricular, que para além de todo o conhecimento que nos deram, também nos ajudaram a realizar este projeto.

Um especial agradecimento ao professor e monitor da turma 4 que durante as aulas práticas nos foram ajudando a perceber e realizar os laboratórios e também inicializar o projeto.

Índice

Lista de imagens 4
1. Introdução 5
2. Instruções de utilização 6
   2.1. Menu 6
   2.2. Jogo 8
3. Estado do Projeto 9
   3.1. Timer 9
   3.2. Keyboard 10
   3.3. Mouse 10
   3.4. Video Card 11
   3.5. RTC 12
   3.6. UART 12
4. Organização e Estrutura do Código 13
   4.1. Módulo 8042 (i8042 no Doxygen) 13
   4.2. Módulo AnimSprite 13
   4.3. Módulo Assembly 14
   4.4. Módulo Game 14
   4.5. Módulo Highscores 14
   4.6. Módulo i8254 15
   4.7. Módulo Keyboard 15
   4.8. Módulo Proj 15
   4.9. Módulo XPM 15
   4.10. Módulo Menu 16
   4.11. Módulo Mouse 16
   4.12. Módulo RTC 17
   4.13. Módulo Sprite 17
   4.14. Módulo Timer 18
   4.15. Módulo Video Card 18
   4.16. Peso Relativo de cada Módulo 19
   4.17. Gráfico de Chamada de Funções 20
5. Detalhes da Implementação 21
   5.1. Camadas (*Layering*) 21
   5.2. *Event Driven Code* e *State Machines* 21
   5.3. Orientação a Objetos 22

5.4. Geração de *frames* 22
5.5. Código em Assembly 22
5.6. Detalhes da Implementação do RTC 23
5.7. Highscores e Ficheiros 23
5.8. Deteção de Colisões 24
5.9. Pixmaps (xpms) 24
5.10. Documentação e gráficos de chamadas de funções 25
6. Conclusões 26
Referências Bibliográficas 27

Lista de imagens

Figura 1 – Imagem que mostra o menu principal do jogo, quando este é jogado durante o dia. É também possível ver a interação entre o cursor e o mesmo menu.

Figura 2 – Imagem que mostra o menu das instruções, que apenas permite visualizar uma pequena imagem e regressar ao menu principal.

Figura 3 – Imagem que mostra o menu dos recordes, que apenas permite visualizar as 10 melhores pontuações e quem as fez ou regressar ao menu principal.

Figura 4 – Imagem que mostra o menu dos créditos, que apenas permite visualizar um pequena imagem com os nomes de quem realizou este projeto ou regressar ao menu.

Figura 5 – Esta imagem mostra o momento que precede o jogo, no qual é pedido ao jogador que introduza o seu nome.

Figura 6 – Figura igual à da capa, no qual se mostra o jogo no seu modo diurno.

Figura 7 – Imagem que aparece quando o guaxinim morre.

Figura 8 – Mostra o jogo no seu modo noturno.

Figura 9 – Mostra o menu no seu modo noturno.

Figura 10 – Relação entre as “classes” AnimSprite e Sprite: AnimSprite é como uma classe derivada de Sprite.

Figura 11 – Gráfico com as funções chamadas pela função game().

Figura 12 – Gráfico com as funções chamadas pela função menu().

Figura 13 – Gráfico com todas as funções chamadas através da função proj_main_loop(), função principal do projeto.

1. Introdução

De acordo com os objetivos da unidade curricular **Laboratório de Computadores**, escolhemos implementar um "Endless Runner" que tem como personagem principal um guaxinim que representa a mascote do MINIX, o Rocky. O objetivo do jogo é conseguir chegar o mais longe possível evitando os vários obstáculos que aparecem no caminho. O jogo termina quando o jogador colidir com algum dos obstáculos.

O projeto tem como objetivo o uso abundante de vários periféricos do computador.

2. Instruções de utilização

O nosso projeto, como é um jogo, é dividido em 2 partes principais: o menu e o jogo.

2.1. Menu

Ao entrar no jogo, é apresentado um menu com as seguintes opções:

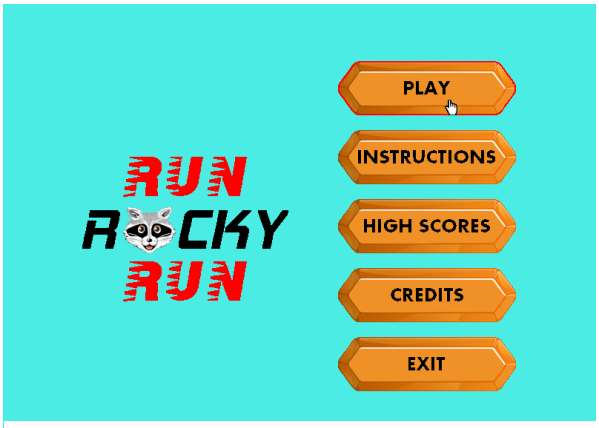

Fig. 1: Menu

- **Play**: Inicia o jogo
- **Instructions**: Mostra uma breve abordagem do jogo e como o jogador pode interagir com o mesmo
- **High Scores**: Mostra 10 recordes obtidos anteriormente no mesmo computador
- **Credits**: Mostra um ecrã de créditos
- **Exit**: Sai do jogo

O jogador pode selecionar qualquer uma das opções utilizando para tal o rato ou o teclado. Ao utilizar o rato, sempre que uma opção está a ser selecionada, a opção é rodeada por uma linha vermelha e o cursor do rato transforma-se numa mão. Ao utilizar o teclado é possível jogar através do P (de Play), consultar as instruções ou os recordes através do I (de Instructions) e do H (de High Scores), respetivamente, visualizar os créditos através do C (de Credits) e sair com a tecla ESC (tipicamente usada com este fim) ou com o E (de Exit).

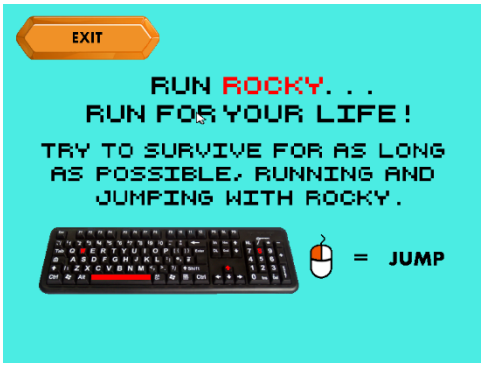

Fig. 2: Instruções

Fig. 3: HighScores

Fig. 4: Créditos

Na **opção das Instruções** é apenas possível interagir com uma caixa de regresso ao menu, utilizando para tal o rato ou o teclado com a mesma lógica do menu. O mesmo se aplica **às opções dos High Scores e Créditos**.

2.2. Jogo

O jogo é também dividido em duas partes: numa primeira parte é pedido o nome ao jogador, de modo a criar um high score associado a ele e na segunda está o verdadeiro jogo.

Na primeira parte, a seleção do nome é realizada recorrendo ao teclado e apenas é possível sair desta opção quando for introduzido um nome válido e premido a tecla ENTER (entenda-se por nome válido, um nome com comprimento superior a 0).

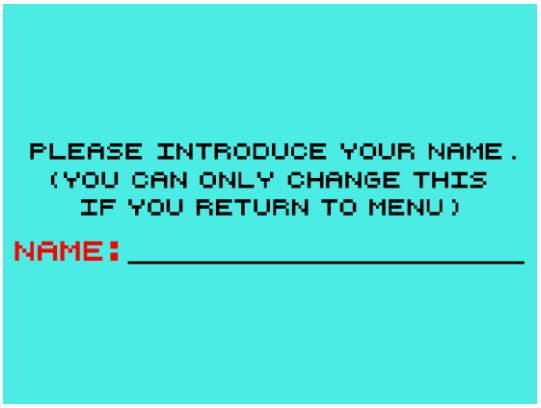

Fig. 5: Introduza o Nome

Na segunda parte, está o verdadeiro jogo. O jogo, como já foi referido, consiste num guaxinim que corre pela sua vida evitando diversos obstáculos que aparecem no seu caminho. Na imagem ao lado é possível ver, para além do guaxinim e de um dos obstáculos:

Fig. 6: Jogo

- Um **sol**, que apenas aparece durante o dia. Num horário noturno dá origem a uma **lua**. Ambos são estáticos, ou seja, a sua posição não varia no decorrer do jogo
- **Score**: é incrementado ao longo do tempo e sempre desenhado na mesma posição do ecrã. Pode ter até 6 dígitos, o que torna possível grandes pontuações.

Fig. 7: Morte da Personagem

Sempre que o jogador colide com um obstáculo é desencadeado um novo menu onde é necessário escolher através do teclado se quer continuar a jogar ou se quer regressar ao menu.

3. Estado do Projeto

Todas as funcionalidades referidas em secções anteriores foram implementadas por nós, excetuando algumas funções fornecidas pela lcf (xpm_load() e todas as outras funções que não eram necessário implementar no decorrer dos labs, como vg_exit()).

Os periféricos utilizados e a sua utilização podem ser consultados na seguinte tabela:

Periférico	**Funcionalidade**	**Interrupções**
Timer	Controlar a "*frame rate*"	Sim
Keyboard	Navegação no menu Inserção do nome Controlo da personagem durante o jogo Escolha se quer continuar ou não a jogar	Sim
Mouse	Navegação no menu Controlo da personagem	Sim
Video Card	Desenhos no ecrã	Não
RTC	Modo diurno/noturno	Não
UART	Não utilizada no projeto	Não

3.1. Timer

O timer é utilizado no projeto como controlo da *"frame rate"*. Para tal são utilizadas várias funções desenvolvidas no Lab2, como **timer_subscribe_int()** e **timer_unsubscribe_int()** definidas no ficheiro timer.c.

De modo a forçar o uso das 60 "*frames*"/interrupções por segundo é utilizada também uma função desenvolvida durante o mesmo laboratório designada por **timer_set_frequency()**, que altera a frequência de um dos timer, no caso do nosso projeto, altera a frequência do timer 0 para 60 Hz (caso a sua frequência não seja esta); isto previne que o jogo corra muito devagar ou muito rápido.

As interrupções do Timer 0 eram previamente tratadas com a linguagem C, mas numa fase mais tardia do desenvolvimento do projeto foi adicionada uma função de tratamento das interrupções do timer em assembly. Esta função, **timer_asm_ih()** está declarada no ficheiro **assembly.h** e definida no ficheiro **timer_asm.S**.

3.2. Keyboard

No nosso projeto, o teclado é utilizado na navegação do menu através das teclas P (de Play), I (de Instructions), H (de High Scores), C (de Credits) e ESC (tecla utilizada normalmente para fechar programas) e E (de Exit), que correspondem a cada caixa que aparece no ecrã.

Para além do menu, o teclado é usado para inserir o nome, podendo ser utilizada qualquer tecla alfabética, *space*, *backspace* e o *enter* que permite selecionar o nome, no jogo, sendo possível controlar a personagem com a seta para cima, o W, o 8 do *numpad* e *space.* No final do jogo é possível escolher se quer continuar a jogar ou regressar ao menu utilizando para tal as duas setas horizontais e o *enter.* Para além destas funcionalidades, é possível utilizar a tecla T de modo a fazer com que o guaxinim coloque a língua dentro da boca (esta funcionalidade surgiu porque os sprites que encontramos na internet continham a língua de fora (como se pode ver também no jogo) e achamos que o jogador deveria poder escolher se queria manter a sua personagem assim ou não).

Para tal são utilizadas várias funções desenvolvidas no Lab3, como **keyboard_subscribe_int()** e **keyboard_unsubscribe_int()** definidas no ficheiro keyboard.c. Tal como no timer o seu *interrupt handler* foi desenvolvido em linguagem C e em assembly numa fase posterior, sendo por isso usada uma função em assembly para tratar das interrupções. A função que trata as interrupções do teclado está definida no ficheiro **assembly.h** e implementada no ficheiro **keyboard_asm.S**, sendo esta função chamada **kbd_asm_ih()**.

3.3. Mouse

No nosso projeto, o rato é utilizado na navegação do menu podendo ser movido por todo o ecrã e, utilizando o botão do lado esquerdo pode-se interagir com as caixas que são apresentadas no ecrã; o mesmo se aplica aos ecrã das instruções, recordes e créditos. De notar que sempre que o cursor do rato passa por cima das caixas é transformado numa mão e a caixa é rodeada por uma linha vermelha.

Para além do menu, o rato é utilizado para controlar a personagem no jogo utilizando o botão esquerdo para fazer a personagem saltar.

São utilizadas funções desenvolvidas no Lab4, como **mouse_subscribe_int()** e **mouse_unsubscribe_int()** definidas no ficheiro mouse.c. O *interrupt handler* foi também desenvolvido em linguagem C e em assembly numa fase posterior estando esta função, chamada **mouse_asm_ih()**, definida no ficheiro **assembly.h** e implementada no ficheiro **mouse_asm.S**.

Para além destas funções de tratamento de interrupções são utilizadas funções, como **issueMouseCommand()** e **mouseIrqSet()** para enviar comandos para o rato e **parsePacket()** que auxilia no tratamento dos pacotes de dados vindos do rato.

3.4. Video Card

A carta gráfica constitui a parte principal do nosso projeto. Todo o jogo é possível recorrendo a este periférico, que usamos para desenhar os menus no ecrã, o cursor do rato, a personagem e as suas animações, os obstáculos, etc.

As funções que interagem diretamente com a carta gráfica chamam-se **drawPixel()** que coloca uma cor num certo pixel e **copyToVRAM()** que copia o segundo buffer para a memória gráfica. Todas as outras funções dos ficheiros videoCard foram desenvolvidas no Laboratório 5 ou são apenas funções auxiliares.

As "classes" Sprite e AnimSprite são classes que de forma indireta interagem com a carta gráfica, uma vez que contêm todas as imagens a serem desenhadas no ecrã.

O **modo utilizado** no projeto é o modo VBE **0x115**, que contém 24 bits por pixel (3 bytes), 800 pixeis de largura, 600 de altura e cerca de 16.8 milhões de cores (16777216 cores, para ser exato).

É utilizado ***double buffering*** que permite uma melhor eficiência do jogo.

As **colisões dos objetos** são realizadas através de caixas. No decorrer do projeto tentamos desenvolver colisões pixel a pixel, mas sem sucesso. Apesar de serem através de caixas, funcionam perfeitamente, pois num salto da personagem, em que os cantos vazios da imagem (entenda-se por canto vazio um canto que não é desenhado no ecrã, mas pertence à imagem original) podem tocar-se, é introduzida uma tolerância que permite o guaxinim saltar e nunca colidir com os obstáculos quando os cantos vazios das imagens se tocam.

Os **objetos são movidos** de acordo com funções declaradas nos ficheiros Sprite e AnimSprite, sendo as animações da personagem feitas através de um objeto do tipo AnimSprite.

Para o desenvolvimento do projeto apenas foram utilizadas as **funções VBE 0x01**, que retorna a informação de uma dado modo, e **0x02**, que altera o modo gráfico.

3.5. RTC

Fig. 8: Jogo noturno

O RTC (Real Time Clock) é utilizado no nosso projeto para definir um modo diurno e outro noturno. A diferença entre estes modos está na cor do fundo do ecrã e no astro que se situa no canto superior direito do jogo.

As funções utilizadas para isto estão implementadas no ficheiro rtc.c e chamam-se **updating_registers()** que verifica se os registos estão/vão ser atualizados, **get_register_value()** que retorna o valor de um certo registo especificado como parâmetro, **is_BCD()** e **convert_to_binary()** que verificam se os valores obtidos estão em BCD e, se isto acontecer, os converte para binário e, por fim, **get_night_hour()** que retorna a hora da noite conforme o mês em que se joga.

Fig. 9: Menu Noturno

3.6. UART

A *serial port* não foi implementada no nosso projeto.

4. Organização e Estrutura do Código

O código desenvolvido no projeto foi desenvolvido em vários módulos, de modo a este ficar melhor estruturado. Apesar de os diagramas de chamadas de funções terem sido gerados pela ferramenta Doxygen e estando, portanto, incorporados na documentação, serão apresentados alguns a seguir.

4.1. Módulo 8042 (i8042 no Doxygen)

Módulo que contém todas as constantes e variáveis globais utilizadas na programação do rato e do teclado. Foi essencialmente desenvolvido durante os Laboratórios 3 e 4, tendo sido apenas complementado durante o projeto.

No projeto este módulo contém constantes usadas para detetar os scancodes provenientes do teclado e os pacotes de dados do rato e também para subscrever e cancelar a subscrição das interrupções dos periféricos.

A complementação deste ficheiro durante o projeto foi feita pelo Pedro Esteves (percentagens da realização do módulo foram entregues no formulário dos Labs 3 e 4).

4.2. Módulo AnimSprite

Este módulo contém todas as funções que permitem animar a personagem do jogo, o guaxinim, tendo sido totalmente desenvolvido durante o projeto.

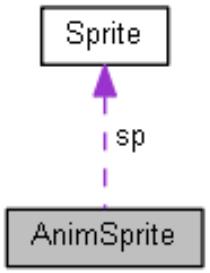

Fig. 10: Relação entre as "classes" AnimSprite e Sprite

O módulo AnimSprite contém uma struct com o mesmo nome, cujos parâmetros são: um apontador para uma struct Sprite (módulo que será abordado no ponto 4.12), que contém o atual Sprite da animação, quatros valores inteiros (int), chamados aspeed, cur_aspeed, num_fig e cur_fig, que correspondem, respetivamente, ao número de frames por Sprite, número de frames até mudar de Sprite, número de Sprites e um identificador do Sprite atual, e também contém um array de apontadores para pixmaps, chamado map.

Uma das funções que permite a troca entre animações com e sem língua de fora do guaxinim está implementada no ficheiro AnimSprite.c e chama-se **swap_animSprites()** que permite a troca dos arrays de pixmaps de dois objetos do tipo AnimSprite.

Neste módulo é chamada uma função, **xpm_load()**, que não foi desenvolvida por nós, tendo sido dada pelo *staff* de LCOM.

Estes ficheiros foram totalmente desenvolvidos durante o projeto pelo Pedro Esteves.

4.3. Módulo Assembly

Este módulo foi desenvolvido numa fase mais tardia do projeto e contém as declarações de todas as funções escritas em assembly (*interrupt handlers*), **timer_asm_ih()**, **kbd_asm_ih()** e **mouse_asm_ih()**, cujas implementações estão nos ficheiros **timer_asm.S**, **keyboard_asm.S** e **mouse_asm.S**.

Foi também implementado pelo Pedro Esteves.

4.4. Módulo Game

O módulo Game contém um ciclo com a função **driver_receive()** todas as funções relacionadas diretamente com o jogo, como a função **game()**, que apresenta o jogo, **drawBackground()** que desenha o fundo do jogo, **game_background_alloc()** que aloca espaço para o fundo do jogo e **free_game_background()** que o liberta, e também **draw_score()** que desenhar os pontos do jogador no ecrã.

As quatro últimas funções mencionadas acima foram desenvolvidas pelo Pedro Esteves, enquanto que a função **game()** foi desenvolvida pelos dois membros do grupo (Diogo Mendes e Pedro Esteves).

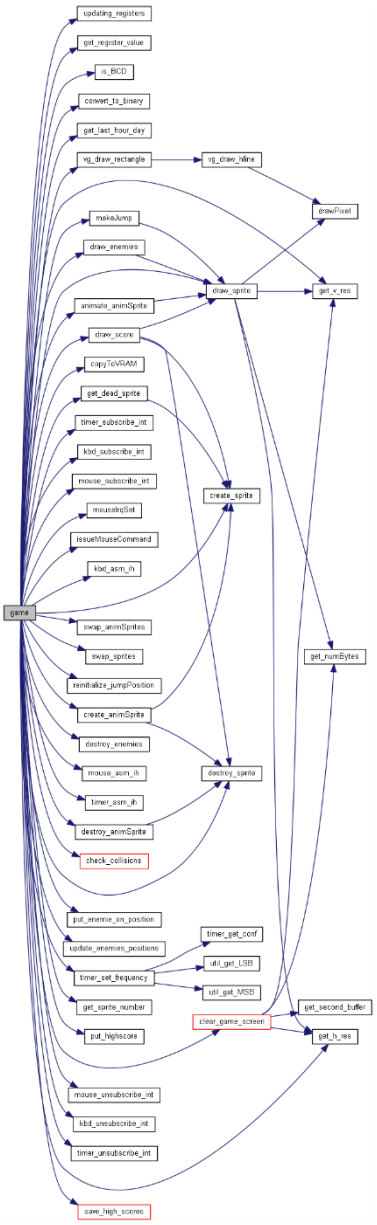

Fig. 11: Gráfico com as funções chamadas pela função game()

4.5. Módulo Highscores

O módulo Highscores contém todas as funções relacionadas com os recordes e os ficheiros onde estes são guardados.

As funções relacionadas com highscores são **put_highscore()**, que coloca um recorde num array de recordes do jogador, **save_high_scores()**, que guarda os recordes do jogador num ficheiro de texto **high_scores.txt**, comparando cada

recorde do jogador com aqueles já contidos no ficheiro e colocando apenas os recordes mais altos, **draw_high_scores()** que desenha no ecrã todos os recordes obtidos até ao momento e **free_high_scores()** que liberta a memória alocada para o array que contém todo o desenho apresentado na opção High Scores do menu.

Este módulo foi totalmente desenvolvido pelo Pedro Esteves.

4.6. Módulo i8254

Módulo que contém todas as constantes e variáveis globais utilizadas na programação do timer. Foi totalmente desenvolvido durante o Laboratório 2.

As percentagens da realização do módulo foram entregues no formulário do Lab 2.

4.7. Módulo Keyboard

Módulo que contém todas as funções que interagem diretamente com o teclado. Foi totalmente desenvolvido durante o Laboratório 3.

Contém as funções **kbd_subscribe_int()** e **kbd_unsubscribe_int()** que subscrevem e cancelam a subscrição das interrupções deste periférico.

As percentagens de participação neste módulo também foram entregues no formulário de avaliação do Lab 3.

4.8. Módulo Proj

O módulo Proj contém algumas macros utilizadas em todo o projeto bem como a função main do projeto.

Este módulo foi implementado pelo Pedro Esteves

4.9. Módulo XPM

Módulo que contém os *include* de todos os ficheiros xpm, ou seja, que inclui todas as imagens no jogo.

Foi desenvolvido durante o projeto por ambos os membros do grupo, mas principalmente pelo Diogo Mendes.

4.10. Módulo Menu

O módulo Menu contém um ciclo com a função **driver_receive()** todas as funções relacionadas diretamente com o menu e seus derivados, como a função **menu()**, que apresenta todos os menus (Menu principal, Instruções, High Scores, Créditos), **get_menu_background()** que retorna uma apontador para o fundo do menu, **menu_background_alloc()** e **free_menu_background()**, que alocam e libertam espaço para o fundo do menu, e também **get_character ()** que retorna um apontador para um Sprite com a letra ou número utilizado nos highscores (também é utilizada pela primeira parte do jogo para escrever o nome do jogador).

As quatro últimas funções mencionadas acima foram desenvolvidas pelo Pedro Esteves, enquanto que a função **menu()** foi desenvolvida pelos dois membros do grupo (Diogo Mendes e Pedro Esteves).

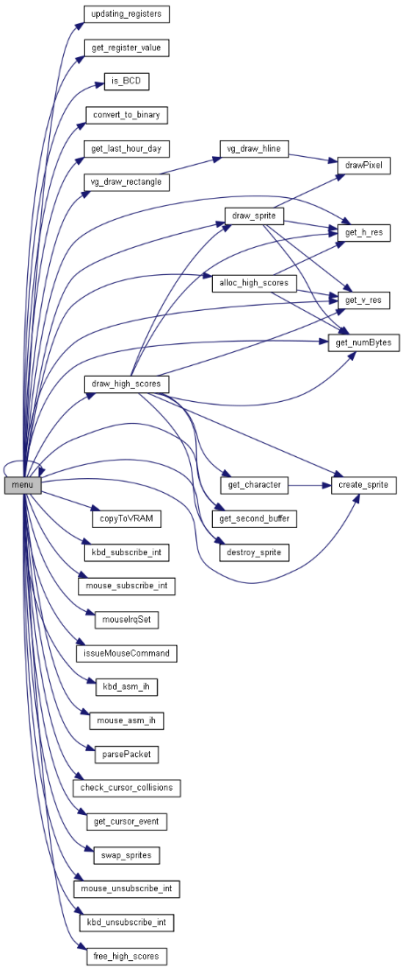

Fig. 12: Gráfico com as funções chamadas pela função game()

4.11. Módulo Mouse

Módulo que contém todas as funções que interagem diretamente com o rato. Foi totalmente desenvolvido durante o Laboratório 4.

Contém as funções **mouse_subscribe_int()** e **mouse_unsubscribe_int()** que subscrevem e cancelam a subscrição das interrupções deste periférico. Para além destas funções relativas às interrupções, contém também funções auxiliares como **parsePacket()** que contrói uma struct do tipo packet atra´ves de um pacote de dados do rato, e também **mouseIrqSet()** e **issueMouseCommand()**, que ativa ou desativa as interrupções do rato e envia um comando para o rato, respetivamente.

As percentagens de participação neste módulo também foram entregues no formulário de avaliação do Lab 4.

4.12. Módulo RTC

Módulo que contém todas as funções que interagem diretamente com o RTC (Real Time Clock).

Contém as funções **updating_registers()**, que verifica se os registos do RTC estão a ser ou vão ser atualizados, **get_register_value()** que retorna o valor de um certo registo especificado como parâmetro, **is_BCD()** que verifica se os valores dos registos lidos estão em BCD ou binary, **convert_to_binary()**, que converte valores em BCD para valores em binary e, por fim, **get_night_hour()** que devolve a última hora do dia conforme o mês em que o jogador se encontra

Este módulo foi totalmente desenvolvido pelo Pedro Esteves.

4.13. Módulo Sprite

Este módulo contém todas as funções que permitem criar imagens do jogo (obstáculos, carateres alfanuméricos, a própria personagem…), tendo começado a ser desenvolvido durante o Laboratório 5.

O módulo Sprite contém uma struct com o mesmo nome, cujos parâmetros são: dois inteiros (int), x e y, que correspondem às coordenadas iniciais da imagem, dois int, width e height, que correspondem ao comprimento e altura da imagem, dois int, xspeed e yspeed, que correspondem à velocidade em x e y da imagem (assim se fazem os obstáculos percorre o ecrã) e também contém um pixmap, chamado map.

Uma das funções que permite a troca entre animações com e sem língua de fora do guaxinim está implementada no ficheiro Sprite.c e chama-se swap_sprites() que permite a troca dos pixmaps de dois objetos do tipo Sprite.

Neste módulo é também chamada uma função, xpm_load(), que não foi desenvolvida por nós, tendo sido dada pelo staff de LCOM.

É também neste módulo que são verificadas as colisões dos objetos através das funções check_collisions(), check_collision() e check_cursor_collisions().

Os obstáculos/inimigos e as suas funções também estão presentes neste módulo atravé das funções **put_enemie_on_position()**, que coloca um inimigo numa posição do array enemies, **update_enemies_positions()**, que atualiza a posição de todos os inimigos criados, **draw_enemies()**, que desenha todos os inimigos e **destroy_enemies()** que os destrói (destruindo também o array).

Para além das funções já mencionadas tem também funções desenvolvidas durante o Lab5, como **create_sprite()**, **destroy_sprite()** e **draw_sprite()**, e também outras funções implementadas apenas para o nosso

projeto, como **makeJump()**, que faz a animação do salto da personagem, **clear_game_screen()**, que apaga o ecrã do jogo, desenhando o seu fundo, **get_dead_sprite()**, que retorna o sprite morto correspondente à animação atual da personagem (estes sprites caracterizam-se por uma pupila dilatada) e **reinitialize_jumpPosition()** que coloca a variável global **jumpPosition** a 0 (variável que mantém o estado do salto).

Estes ficheiros foram complementados pelo Pedro Esteves.

4.14. Módulo Timer

Módulo que contém todas as funções que interagem diretamente com o timer. Foi totalmente desenvolvido durante o Laboratório 2.

Contém as funções **timer_subscribe_int()** e **timer_unsubscribe_int()** que subscrevem e cancelam a subscrição das interrupções deste periférico.

As percentagens de participação neste módulo foram entregues no formulário de avaliação do Lab 2.

4.15. Módulo Video Card

Módulo que contém todas as funções que interagem diretamente com a carta gráfica. O seu desenvolvimento começou durante o Laboratório 5, mas sofreu alguns melhoramentos durante o projeto.

Contém as funções **vbeGetModeInfo()** e **copyToVRAM()** que interagem diretamente com este periférico. Para além disto, contém funções relativas ao desenho de pixéis (**drawPixel()**), ao segundo buffer(**get_second_buffer()** e **freeBuffers()**) e ainda outras funções auxiliares que retornam, por exemplo, a resolução horizontal e vertical do ecrã.

Para além destas funções contém também algumas macros que ajudam na manipulação deste periférico

As percentagens de participação neste módulo foram entregues no formulário de avaliação do Lab 5. O seu melhoramento foi feito pelo Pedro Esteves.

4.16. Peso Relativo de cada Módulo

Módulo	Peso no Projeto
8042	1%
AnimSprite	5%
Assembly	4%
Game	13%
Highscores	5%
I8254	1%
Keyboard	10%
Proj	4%
XPM	3%
Menu	13%
Mouse	10%
RTC	5%
Sprite	7%
Timer	9%
VideoCard	10%

4.17. Gráfico de Chamada de Funções

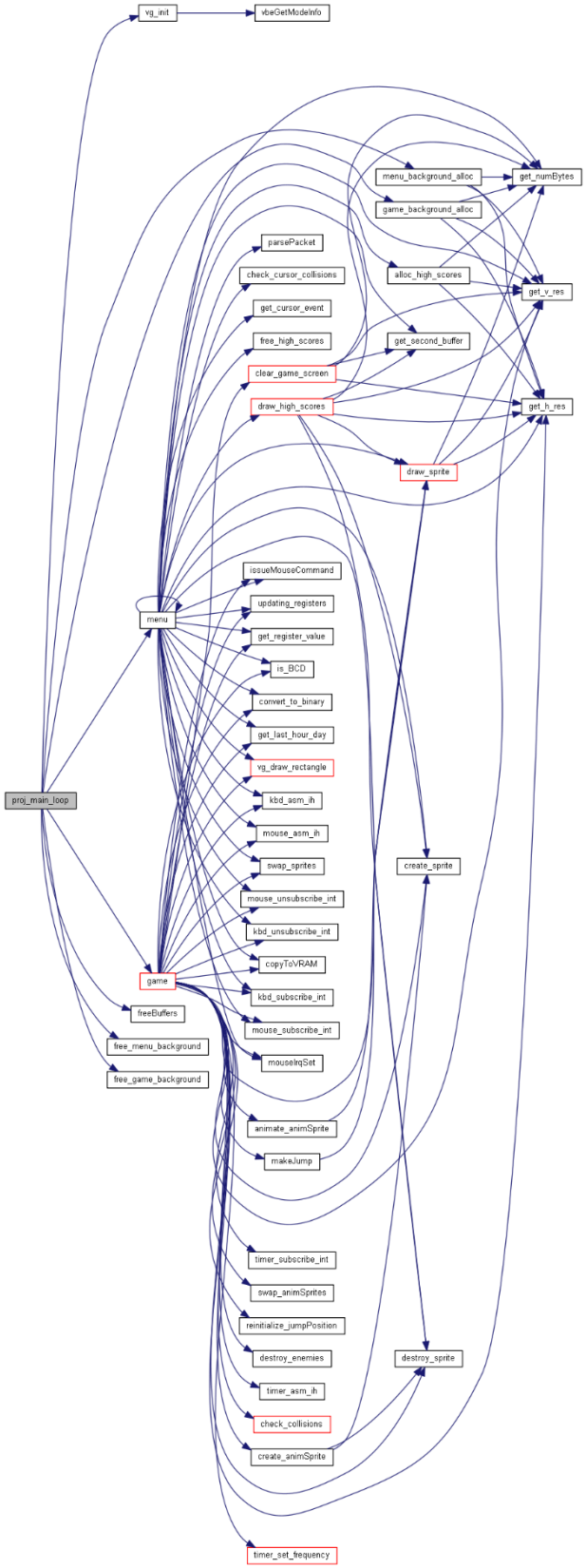

Fig. 13: Gráfico de Chamada de Funções (Pode ser também consultado na Documentação do Projeto: /proj/doc/html)

5. Detalhes da Implementação

Quase todos os tópicos abordados nas aulas teóricas foram usados no nosso projeto à exceção dos tópicos relacionados com a UART. Para além disso tentamos ainda utilizar funcionalidades pouco ou nada abordadas nestas aulas. Todas estas funcionalidades utilizadas estão explicadas a seguir.

5.1. Camadas (*Layering*)

Todo o nosso projeto passou por um processo de **diferenciação** em **várias camadas** de modo a organizar melhor o código: camadas relativas aos periféricos diferenciam-se de camadas de mais alto nível.

5.2. *Event Driven Code* e *State Machines*

O nosso projeto contém muitas **situações originadas por eventos**, para além das interrupções de cada periférico.

Para além desta **máquina de estados**, contêm também **máquinas de estados**:

- O próprio projeto é uma máquina de estados com dois estados, ambos finais: o menu e o jogo (só entra no jogo se a opção do menu for 1).
- O menu também contém máquinas de estados: cada secção do menu corresponde a um estado, sendo os estados dos créditos, highscores e intruções apenas acedidos através do estado menu.
- O cursor do rato no menu tem dois estados: cursor normal ou uma mão quando este colide com uma das caixas.
- As caixas do menu também têm os mesmos dois estados: quando o cursor está por cima da caixa esta ganha uma linha vermelha à volta para mostrar a sua seleção.
- O rato do menu contém ainda outra máquina de estados. Esta máquina contém 3 estados: o inicial, um estado em que o cursor se situa por cima de uma das caixas e o botão esquerdo do rato ainda não foi premido e o final que representa a aceitação do clique no botão e entrada na respetiva secção.
- O jogo (função game()) contém três estados: o estado inicial pede ao jogador o seu nome, o segundo estado a personagem corre e salta e, se colidir, morre e aparece um ecrã diferente. Estes

últimos dois estados alternam entre si até que o jogador selecione não voltar a jogar e regressar ao menu.

- O salto da personagem representa também uma máquina com 6 estados. Os 4 estados do meio têm o mesmo efeito (aumentar ou diminuir a posição em y), enquanto que o 1º e o último, para além disto alteram a velocidade em y para o seu negativo (primeiro sobe e depois desce com a mesma velocidade).
- A imagem do guaxinim morto é também ela dada por uma máquina de estados que retorna um Sprite diferente conforme o estado da animação em que está.

5.3. Orientação a Objetos

No nosso projeto também foi usada uma implementação orientada a objetos.

Para tal foram declaradas e implementadas 2 "classes": Sprite e AnimSprite. Ambas as classes são definidas usando structs e já foram referenciadas nos pontos 4.2 e 4.12. São utilizadas para animação da personagem e desenhos no ecrã.

5.4. Geração de *frames*

Todos os **eventos** são **gerados através das interrupções** dos vários periféricos, principalmente do Timer.

Durante o jogo, a cada 4 interrupções do Timer, se tiver sido começado um salto, é continuado esse salto. A cada 10 interrupções do Timer é aumentado o score do jogador. A partir das 500 interrupções do Timer com um intervalo que vai diminuindo, é criado um inimigo no ecrã e a cada 5000 interrupções do mesmo periférico, a velocidade do jogo é aumentada.

***Nota**: O aumento da velocidade não é feito durante todo o jogo. Após chegar a uma certa velocidade para cada inimigo, animação e tempo de criação de inimigo, é parado este aumento de velocidade, de modo a não criar erros no jogo (como a animação parar).*

5.5. Código em Assembly

Numa fase tardia do projeto foram implementados todos os ***interrupt handlers*** em linguagem **assembly**, o que torna o projeto mais eficiente.

A sintaxe utilizada foi a da Intel, aprendida na unidade curricular Microprocessadores e Computadores Pessoais. Toda a documentação da linguagem foi consultada nos ficheiros de MPCP e alguns diapositivos das aulas teóricas de LCOM.

5.6. Detalhes da Implementação do RTC

O **RTC** foi implementado após o término das aulas prático-laboratoriais. Foi implementado usando os diapositivos das aulas teóricas e a documentação do último laboratório realizado.

A **abordagem utilizada** foi a seguinte: verificar se os registos estão a ser atualizados, ler os registos ou esperar 244 microssegundos, verificar os valores estão em BCD e converter em binário caso estejam nesta configuração.

No projeto é utilizado este periférico para definir os **modos diurno e noturno**, caracterizados por uma cor de fundo do ecrã diferente e também um astro diferente no canto superior direito do ecrã, durante o jogo.

5.7. Highscores e Ficheiros

Embora este tópico não tenha sido abordado nas aulas teóricas, foi algo que teve na nossa mente desde o início do projeto.

Qualquer bom jogo tem sempre uma **tabela de recordes** (ou algo do género) que desafia o jogador a querer sempre melhorar os seus resultados ou resultados mundiais. O nosso não foge à regra!

No nosso projeto implementamos um módulo dedicado apenas aos highscores. Este módulo utiliza bastante **ficheiro de C** (**FILE ***), que não foram abordados em nenhuma das aulas práticas. Para tal, dividimos este módulo em algumas partes: criar um array com 10 highscores do próprio jogador; ler e mostrar os recordes obtidos anteriormente no mesmo computador; ler os recordes anteriores e, se possível, guardar os novos num ficheiro de texto **high_scores.txt** (a consulta deste ficheiro pode ser realizada no diretório **../proj/src/high_scores**).

Aprendemos a utilizar ficheiros através de vários sites na internet:

- *https://www.programiz.com/c-programming/c-file-input-output*
- *https://www.tutorialspoint.com/cprogramming/c_file_io.htm*
- *http://www.tutorialspoint.com/ansi_c/c_working_with_files.htm*

E também alguma documentação da linguagem.

Inicialmente também tinha sido pensada uma utilização da UART com a transmissão dos recordes de vários jogadores em diferentes computadores, mas essa implementação acabou por não ser concretizada.

5.8. Deteção de Colisões

A deteção de colisões foi introduzida no projeto num fase inicial (antes do término das aulas prático-laboratoriais). Desde aí se tentou realizar implementar **colisões pixel a pixel** (ou seja, verificar se em cada pixel existe colisão), mas sem sucesso.

Uma vez que este tipo de colisões não conseguiram ser implementadas, optamos por uma abordagem diferente, considerando cada obstáculo e a personagem como **retângulos** e obtendo assim as colisões. No início, algumas destas colisões não funcionavam, uma vez que tanto o guaxinim como os obstáculos têm alguns cantos vazios (sem nada desenhado), o que fazia com que o jogador perdesse sem sequer tocar em nenhum dos obstáculos. Assim, foi introduzida uma **tolerância** para o salto da personagem, que permite que estas "colisões" não sejam detetadas, mas apenas aquelas em que o guaxinim colide mesmo com o obstáculo.

5.9. Pixmaps (xpms)

Os **xpms** incluídos no projeto foram, alguns retirados da internet, outros realizados principalmente pelo Diogo Mendes.

No que diz respeito aos obstáculos e aos astros (lua ou sol), todos foram retirados da internet. Também os cursores tiveram a mesma fonte.

Todos os xpms com carateres alfanuméricos, incluindo as imagens do menu, os seus retângulos, os créditos, as instruções e os highscores (excetuando a imagem que aparece quando a personagem morre, que foi também retirada da internet) foram realizados pelo Diogo Mendes, utilizando a ferramenta GIMP.

As animações da personagem principal foram também retiradas da internet, à exceção de todas as que não contêm a língua e todas as que mostram a personagem morta (com pupilas dilatadas). Estas últimas foram desenhadas pelo Pedro Esteves, utilizando as outras imagens, com a ajuda do Diogo Mendes nas imagens sem língua.

O aperfeiçoamento de todas as imagens (remoção de pixéis, redimensionamento, …) foi realizado por ambos.

A consulta das imagens utilizadas pode ser realizada na pasta xpms, contida na pasta src.

5.10. Documentação e gráficos de chamadas de funções

A documentação do projeto e todos os seus gráficos foram realizados, utilizando para tal a ferramenta Doxygen, pelo Pedro Esteves.

Esta ferramenta de geração de documentação foi abordada e também utilizada na unidade curricular Algoritmos e Estruturas de Dados, tendo sido também consultada mais alguma documentação desta ferramenta, bem como a documentação de ficheiros dados pelo *staff* de LCOM (por exemplo, do ficheiro timer.h).

Esta documentação é possível ser visualizada na pasta doc.

6. Conclusões

Apesar de ser uma das unidades curriculares mais trabalhosas do semestre, ambos gostamos daquilo que se aprendeu, principalmente de trabalhar no projeto.

Algo que achamos que deveria ser melhorado são os guiões dos Laboratórios e a documentação da LCF e dos vários periféricos, uma vez que nas primeiras semanas nos sentimos um bocado perdidos em relação àquilo que tínhamos que fazer nas aulas práticas.

Para além disto, achamos que deveriam ser abordados outros tópicos como os **ficheiros** que certamente muita gente utilizou no seu projeto, em vez de, por exemplo, serem abordados aspetos de **debugging** numa das últimas aulas do semestre, pois estes são muito relativos a cada situação e pessoa.

Referências Bibliográficas

Implementações dos Periféricos:

- https://web.fe.up.pt/~pfs/aulas/lcom2018/
- https://web.fe.up.pt/~pfs/aulas/lcom2013/labs/lab6/lab6.html

Documentação da Linguagem Assembly:

- Documentação da UC Microprocessadores e Computadores Pessoais (https://sigarra.up.pt/feup/pt/conteudos_geral.ver?pct_pag_id=249640&pct_parametros=pv_ocorrencia_id=399884)

Ficheiros:

- https://www.programiz.com/c-programming/c-file-input-output
- https://www.tutorialspoint.com/cprogramming/c_file_io.htm
- http://www.tutorialspoint.com/ansi_c/c_working_with_files.htm

Doxygen:

- http://www.doxygen.nl/manual/index.html
- UC Algoritmos e Estruturas de Dados

Sprites:

- https://dribbble.com/shots/1649484-Raccoon-Animation-Demo-Sprite-Sheets
- https://opengameart.org/