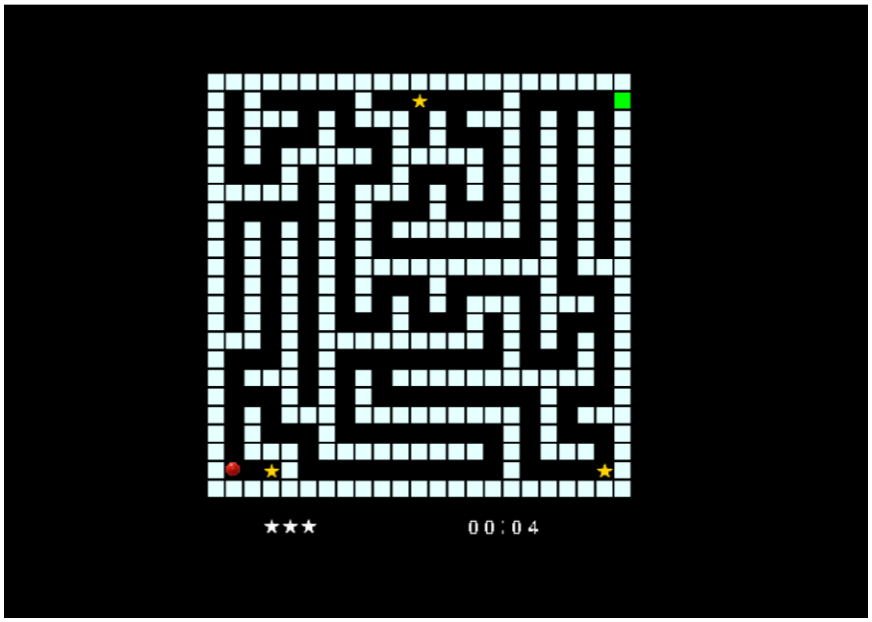
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

"GET LOST"



No âmbito da unidade curricular Laboratório de Computadores,
1º semestre 2º ano do Mestrado Integrado de Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto

Trabalho realizado por:

Mariana Almeida Truta – up201806543

Rita Matos Maranhão Peixoto – up201806257

Turma 2, Grupo 4

2019/2020

Data de entrega de 6 de janeiro de 2020

Conteúdo

1. Instruções de Utilização

1.1 Main Menu

Ao iniciar o programa, o utilizador irá deparar-se com o seguinte menu inicial:

Figura 1 Menu Inicial

No menu inicial, encontra-se um *Real Time Counter* (contador em tempo real) no rodapé que mostra a data e a hora no preciso momento em que o utilizador se encontra neste menu. Ao passar o cursor por cima dos botões, estes vão aumentar de tamanho, dando a entender que podem ser selecionados.

O utilizador deve então escolher a opção pretendida e confirmá-la, colocando o cursor em cima da opção e clicando do botão esquerdo do mesmo.

As opções correspondem a:

- ***Play*:** inicia um novo jogo;
- ***Quit*:** fecha o programa.

1.2 Play

Ao escolher a opção *play*, são dispostas no ecrã as instruções do jogo da seguinte forma:

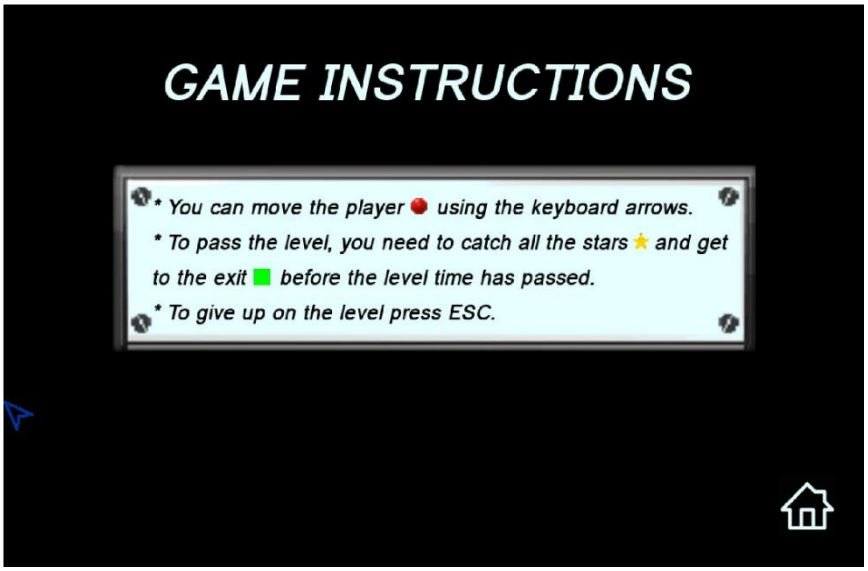


Figura 2 Instruções do jogo

Encontra-se disponível um botão para o *Main Menu* (menu inicial) representado por uma casa. Este botão cresce ligeiramente quando o cursor está em cima dele, sendo, então, apenas necessário o utilizador clicar no botão esquerdo do cursor para o selecionar.

Caso o botão do *Main Menu* não seja selecionado, as instruções mantêm-se no ecrã durante 10 segundos, avançando de seguida para um menu de níveis apenas com o nível 1 disponível:

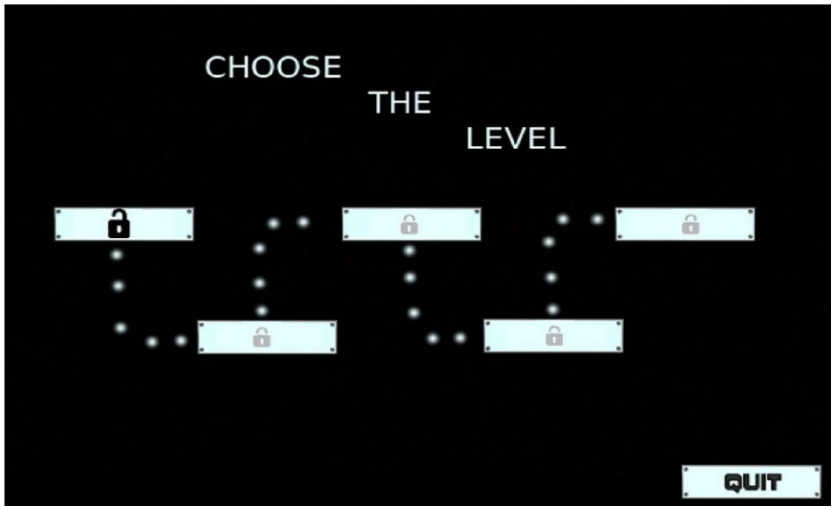


Figura 3 Menu de níveis

O utilizador pode escolher jogar o nível 1, colocando o cursor por cima do único botão desbloqueado e clicando no lado esquerdo do mesmo, ou sair do jogo fazendo de igual forma com o botão *QUIT*.

Se o utilizador clicar no botão *QUIT*, o programa termina.

Tendo escolhido continuar para o nível 1, o utilizador irá iniciar o jogo, sendo-lhe mostrado o labirinto:

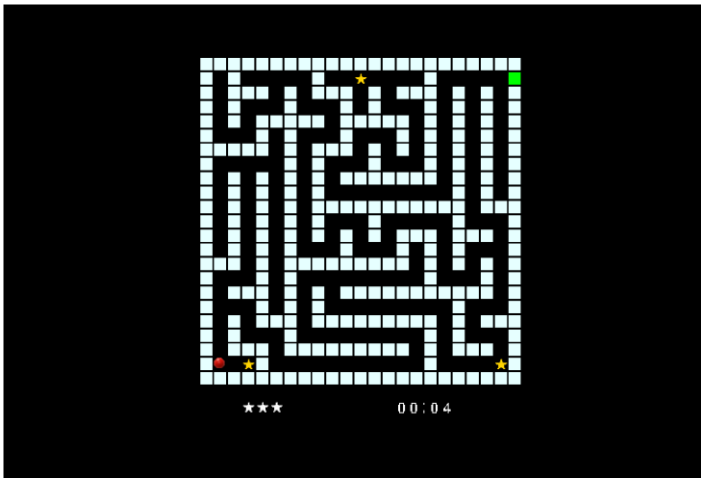


Figura 4 Labirinto exemplo do nível 1

A partir deste momento, o utilizador passa a controlar a bola vermelha com as teclas das setas do teclado, podendo utilizá-las para se movimentar ao longo do labirinto. Como indicado nas instruções, caso seja pressionada a tecla *esc*, o utilizador desiste do nível e o menu dos níveis volta a aparecer.

No jogo, chegando à saída (destacada pela cor verde), o utilizador completa o nível com sucesso caso tenham sido cumpridas as seguintes regras: apanhar todas as estrelas dispostas no labirinto e alcançar a saída num tempo inferior ao definido para este nível. Seguidamente, irá aparecer uma imagem a informar o utilizador se completou o nível (figura 5) ou não (figura 6).

Figura 5 Mensagem de nível completo

Figura 6 Mensagem de nível não completo

Se completou o nível com sucesso, será confrontado com um novo menu de níveis com o seguinte aspeto:

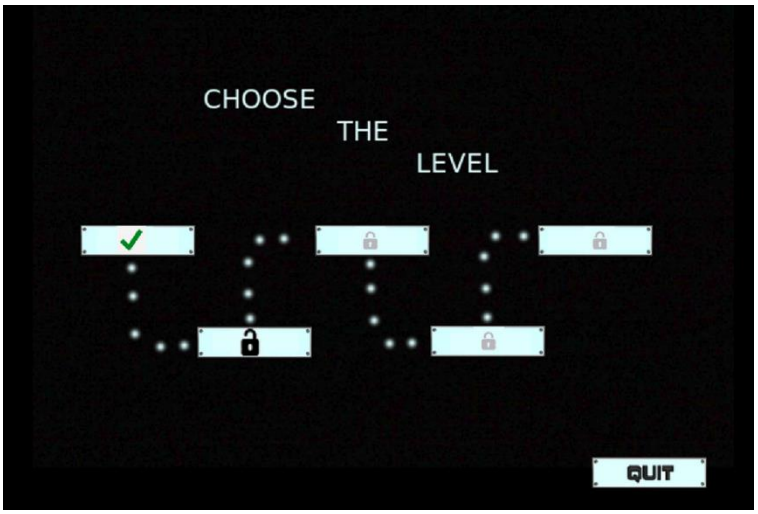


Figura 7 Menu de níveis após completar o nível 1

Neste novo menu, o utilizador já não pode voltar ao nível 1 uma vez que já o completou com sucesso e tem novamente as duas opções: escolher jogar o próximo nível ou sair do programa.

No caso de não ter completado o nível com sucesso, será confrontado novamente com o menu de níveis apenas com o nível 1 desbloqueado e pode jogar novamente este nível ou sair do programa. É de salientar o facto de que jogar novamente o nível não implica um labirinto igual, mas sim um labirinto do mesmo grau de dificuldade.

O jogo tem 5 níveis, seguindo todos estas componentes. De nível para nível, serão modificadas certas variáveis de forma a aumentar a dificuldade do nível, tais como o tamanho do labirinto, o número de estrelas a apanhar e o tempo de jogo em comparação ao tamanho.

Por fim, ao completar o 5º nível, o utilizador é avisado de que venceu (figura 8) e o programa termina.

Figura 8 Mensagem após todos os níveis serem

2. Estado do projeto

2.1 Dispositivos utilizados

Dispositivo	Funcionalidade	Interrupções
TIMER	Atualizar o ecrã com os menus, os labirintos, movimentos do cursor e do jogador; contabilizar o tempo que o jogador demora a concluir o nível	Sim
KEYBOARD	Navegar ao longo do labirinto; sair do nível, voltando ao menu anterior	Sim
MOUSE	Navegar e selecionar botões nos menus	Sim
VIDEO CARD	Visualizar os menus, o jogo e objetos	-
RTC	Obter a data e hora atuais	Não

Figura 9 Tabela dos dispositivos utilizados e respetivas funcionalidades

2.2 Timer

O *timer* é utilizado para atualizar o ecrã (*frame rate*) ao longo do programa bem como contar o tempo que o jogador demora a completar um determinado nível. Para tal, são utilizadas as interrupções do *timer*, usando a sua frequência base: 60 interrupções por segundo. A subscrição e o cancelamento da subscrição destas interrupções são realizados através das funções *timer_subscribe_int() e timer_unsubscribe_int()*, respetivamente.

Para além disso, o *timer_counter* é atualizado a cada interrupção com a função *timer_int_handler()*.

Para a primeira parte, o timer utiliza as funções *clean_mouse()* e *draw_mouse()*, para que, sempre que haja movimento do cursor nos menus, este seja atualizado consoante a sua posição final.

É também no *timer* que se realiza o *double buffering*, através da função *updateBuffer()*.

2.3 Keyboard

O *keyboard* é utilizado no jogo para navegação do jogador (bola vermelha) no labirinto. O botão da seta esquerda ("→") provoca o movimento para a esquerda, o botão da seta direita ("←") provoca o movimento para a direita, o botão da seta para baixo provoca o movimento para baixo e o botão da seta para cima provoca o movimento para cima.

Para tal, são utilizadas as interrupções do *keyboard* que, há semelhança do *timer*, têm de ser subscritas e, no fim, a sua subscrição cancelada através das funções *kbd_susbscribe_int()* e *kbd_unsubscribe_int()*, respetivamente. A sua interrupção é inicialmente tratada na função *kbd_ih()* e, posteriormente, é interpretada em várias funções ao longo do programa.

A informação da tecla utilizada é transmitida para a função *check_movement()* que verifica se o movimento do jodador é possível sem colidir com as paredes do labirinto e, também, é esta função que lida com tudo o que este movimento provoca no jogo.

Se, durante o nível, o utilizador pressionar a tecla *ESC*, significa que pretende desistir do jogo e voltará ao menu de níveis anterior.

2.4 Mouse

O rato é utilizado no jogo para escolher as opções dos diferentes menus.

Para tal acontecer, mais uma vez, as suas interrupções têm de ser subscritas utilizando a função *mouse_subscribe_int()* e , no final, a sua subscrição tem de ser cancelada através da função *mouse_unsubscribe_int()*. No caso do rato, é necessário também habilitar inicialmente a receção de informação do mesmo através da função *write_command()* tendo como argumento a macro *MOUSE_ENABLE* e, no final do programa, procede-se ao cancelamento da habilitação referida com a mesma função mas desta vez tendo como argumento a macro *MOUSE_DISABLE*.

A cada interrupção do rato é atualizada a posição do cursor tanto na *struct mouse* como no ecrã, sendo a informação inicialmente tratada na função *mouse_ih()*.

Ao atualizar a posição do cursor, são tidos em conta os limites do ecrã, nunca permitindo que o cursor "desapareça" nem que realize movimentos que o utilizador não consiga acompanhar. Esta funcionalidade só é possível devido à função *check_possible_mov()*.

Existe três funções baseadas no mesmo de conjuntos de estados que interpretam as ações realizadas com o rato nos diferentes níveis:

- *mouse_gestures ()* – estando no menu inicial, esta função permite que, quando o cursor é colocado por cima de uma das opções (*PLAY* ou *QUIT*), o botão respetivo seja ligeiramente aumentado e que se o utilizador carregar no botão esquerdo do rato por cima de uma das opções, esta seja selecionada e interpretada.
- *mouse_button ()* – no caso do menu dos níveis, esta função informa o resto do programa quando o utilizador seleciona o nível desbloqueado com o botão esquerdo do rato ou quando carrega no botão *QUIT*.
- *mouse_instructions ()* – quando estão a ser mostradas as instruções, esta função possibilita que o utilizador tenha a certeza que está a passar o cursor por cima do botão de voltar ao menu inicial, uma vez que este aumenta de tamanho, e que essa opção possa ser selecionada carregando no botão esquerdo do rato.

2.5 Video Card

A placa gráfica é utilizada para visualizar o jogo, isto é, para criar a interface gráfica do projeto. Esta encontra-se a operar no modo *0x14C* (32 *bits* por *pixel*, 8:8:8:8), cuja resolução é 1152x864.

São utilizados *XPMs* para as imagens.

Foi implementada a técnica de *double buffering*, isto é, a cada interrupção do *timer* é chamada a função *updateBuffer()* que copia para a *VRAM* o conteúdo *double_buff*, passando de seguida para a frame seguinte. A alocação da memória para os dois *buffers* é efetuada na função *vg_init()*. Esta técnica é utilizada de forma a que todas as mudanças que aconteçam relacionadas com imagens e fundos sejam realizadas de forma suave e

fluida. Assim. para dar a ideia de movimento, sempre que são atualizadas as novas posições dos objetos no ecrã é desenhada uma nova *frame*.

Implementou-se duas funções para detetar colisões através da cor dos pixéis: a *check_colisions()* que impede que o jogador ultrapasse os limites do labirinto e as paredes do mesmo; e a *check_catch_star()* que informa o programa caso o jogador tenha apanhado uma estrela o que fará com que seja pintada uma estrela a amarelo na barra inferior de estrelas apanhadas.

2.6 RTC (*Real Time Clock*)

O *RTC* é utilizado para calcular e mostrar a data e a hora atuais quando o utilizador se encontra no menu principal. Isto é possível através da função *getRealTime()* e todas as funções auxiliares que esta chama no ficheiro *rtc.c*.

3. Organização e estrutura do código

3.1 Proj

É o ficheiro base do projeto, fornecido pelos docentes e responsável por inicializar a placa vídeo em modo gráfico (*vg_init()* → *video_gr.c*) e retornar ao modo *default* do MINIX 3 (*text mode*, *vg_exit()* → *video_gr.c*) após a chamada da função que inicializa o programa (*menu()* → *menu.c*) estiver concluida.

Peso: 3%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.2 Timer

Neste módulo, encontram-se todas as funções para manipulação do *timer*, tendo sido importado do *Lab2*.

Todas as constantes simbólicas necessárias para a utilização do *timer* encontram-se definidas no ficheiro'*i8254.h*', tendo sido fornecido no *lab2* pelos docentes.

Peso: 5%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.3 Keyboard

Neste módulo, encontram-se todas as funções para manipulação do *keyboard*, tendo sido importado do *Lab3*.

Todas as constantes simbólicas necessárias para a utilização do *keyboard* encontram-se definidas no ficheiro'*i8042.h*'.

Peso: 5%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.4 Mouse

Neste módulo, encontram-se todas as funções para manipulação do rato, tendo sido importado do *Lab4*.

No entanto, foram acrescentadas as seguintes funções: *mouse_gestures()*, *mouse_button()* e *mouse_instructions()* que já foram mencionadas e explicadas anteriormente. Todas estas funções fazem as alterações necessárias caso algum botão tenha sido selecionado.

Neste ficheiro, são utilizados três estruturas de dados que facilitam todo o armazenamento de diversas informações importantes para o progresso do jogo: a *enum state_t* que representa diferentes eventos do rato, a *enum type* que representa o menu em que se encontra o utilizador e *a struct mouse_ev* providenciada no *Lab4*.

Todas as constantes simbólicas necessárias para a utilização do *mouse* encontram-se definidas no ficheiro'*i8042.h*'.

Peso: 10%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.5 Video Gr

Neste módulo, encontram-se todas as funções para manipulação da *VBE*, tendo sido importado do *Lab5*, com pequenas alterações para correção de erros e o com o acréscimo do *double buffering*. Assim sendo, foi necessário criar a função *updateBuffer()* que copia para a *VRAM* o conteúdo do *buffer* auxiliar. Todas as funções passaram a copiar os pixéis para o buffer auxiliar ao invés da *VRAM*.

Todas as constantes simbólicas necessárias para a utilização do *video gr* encontram-se definidas no ficheiro *'video_gr_macros.h*'.

Peso: 5%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.6 RTC

Este módulo é responsável pela implementação do *RTC*, tendo sido utilizadas as seguintes funções:

- *bcd_decimal_converter()***:** função que converte um valor de BCD (*binary coded decimal*) para decimal;
- *read_rtc_value()***:** função que lê um determinado registo do *RTC*;
- *getRealTime()*: função que lê os valores atuais do *RTC* e converte-os para decimal, atualizando depois a *struct realtime* e limpando o atual mostrador do *RTC*, se não for a primeira vez que é mostrado (1 milissegundo), antes de mostrar o novo;
- *draw_real_time()*: função que exibe no ecrã o tempo atual;
- *clean_time()***:** função que limpa o tempo anterior do ecrã.

Neste módulo, são usadas algumas estruturas de dados, tais como *RealTime*, *Numbers* e *Menu*.

Peso: 3%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.7 Menu

É, neste módulo, que, de acordo com o estado do programa, se gere todo o processo de menus, níveis, interrupções.

- *menu()*: esta função começa por alocar memória e inicializar algumas *structs* necessárias ao longo do programa, sendo elas a *struct mouse*, a *struct menu* (fazendo *load* de todos os menus necessários), a *struct* que contem números do *RTC* e *struct* que contem os números do *counter*. Coloca também a variável *state* no estado inicial do programa (*state_menu*) e faz *load* dos menus de níveis. Para além disso, calcula o tempo real e trata da subscrição de todas as interrupções e cancelamento das mesmas. Por fim, no meio da função, existe um ciclo *while* que lida com os diferentes estados do programa, chamando as funções que lidam com as interrupções em cada um destes estados e agindo de acordo com o retorno destas funções.

.

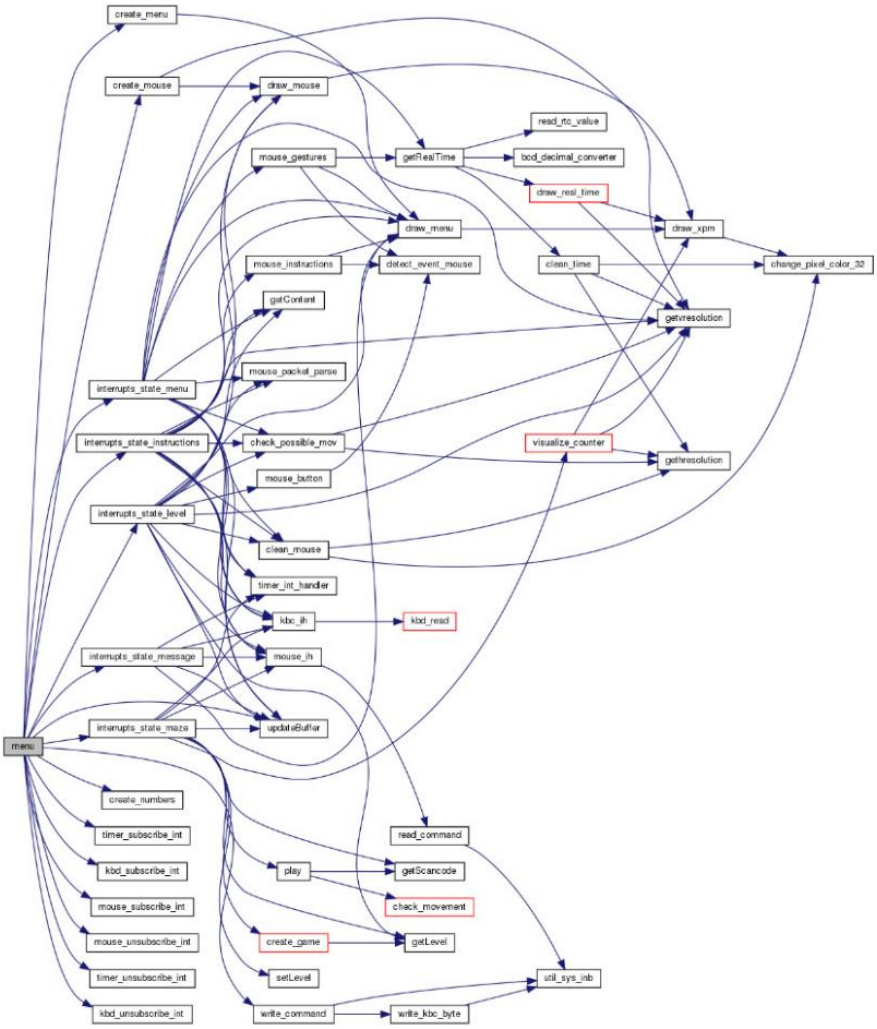


Figura 9 *Call graph* da função *menu()*

Neste módulo, são utilizadas algumas estruturas de dados, tais como *RealTime*, *Numbers*, *Menu* e *Mouse*.

Peso: 15%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.8 Interrupts

Este módulo trata das interrupções geradas pelos periféricos nos diferentes estados do programa, através das funções:

- *interrupts_state_menu()*: função que lida com as interrupções quando o programa se encontra no menu principal. Faz o *parse* do *packets* recebidos do *mouse* e chama a função *check_possible_mov()* → *auxiliar_functions.c*, para verificar se o movimento do rato é possível. De seguida, chama a função *mouse_gestures()* → *mouse.c*, que verifica se o utilizador selecionou algum botão (*play* ou *quit*). Se algum botão for escolhido, executa as ações necessárias. Utiliza o *timer* para atualizar as *frames*.

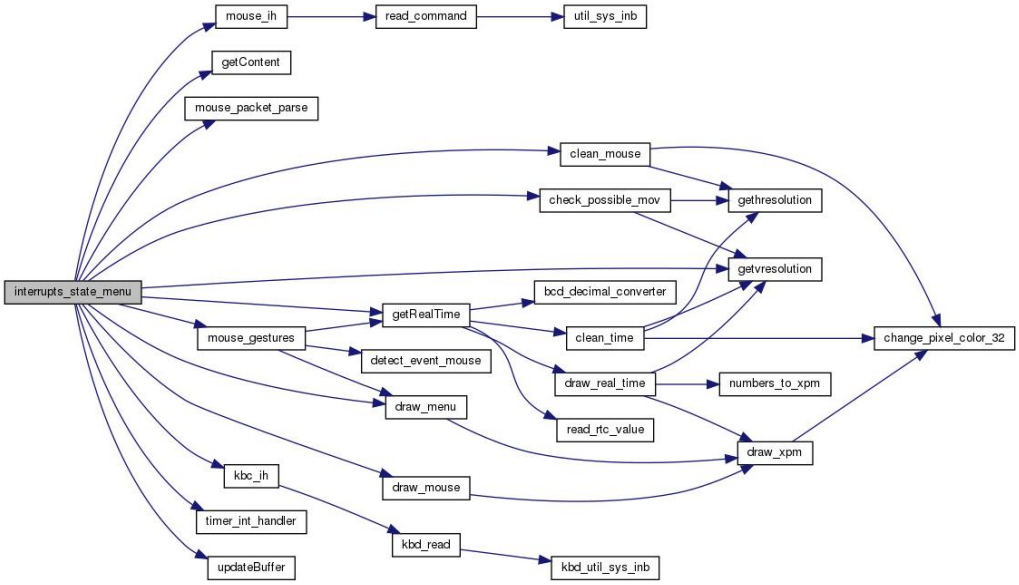


Figura 10 *Call graph* da função *interrupts_state_menu()*

- *interrupts_state_instructions()*: função que lida com as interrupções quando o programa se encontra nas instruções. Faz o *parse* dos *packets* recebidos do *mouse* e chama a função *check_possible_mov()* → *auxiliar_functions.c*, para verificar se o movimento do rato é possível. De seguida, chama a função *mouse_instructions()* → *mouse.c*, que verifica se o utilizador selecionou o botão de *main menu*. Se este botão for escolhido, executa as ações necessárias. Utiliza o *timer* para atualizar as

frames e para contabilizar os 10 segundos que estas instruções se mantém no ecrã se não for pressionado o botão de *main menu*.

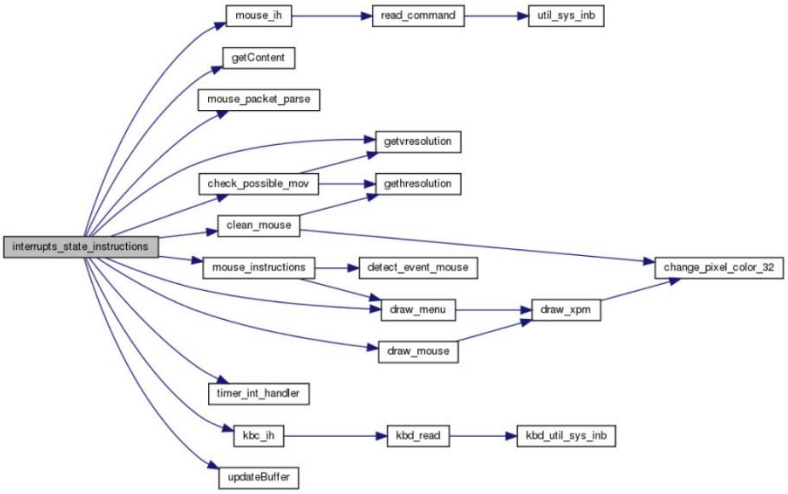


Figura 11 *Call graph* da função *interrupts_state_instructions()*

- *interrupts_state_level()*: função que lida com as interrupções quando o programa se encontra no menu de níveis. Faz o *parse* do *packets* recebidos do *mouse* e chama a função *check_possible_mov()* → *auxiliar_functions.c*, para verificar se o movimento do rato é possível. De seguida, chama a função *mouse_button()* → *mouse.c*, que verifica se o utilizador selecionou algum botão (nível disponível ou *quit*). Se algum botão for escolhido, executa as ações necessárias. Utiliza o timer para atualizar as *frames*, de acordo com o nível em que se encontra.

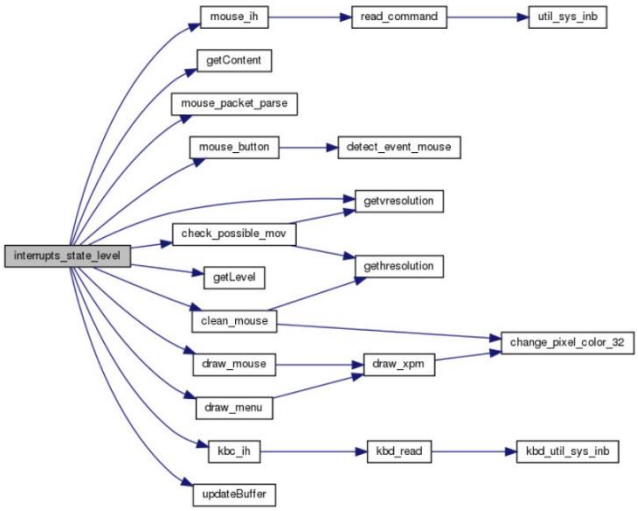


Figura 12 *Call graph* da função *interrupts_state_level()*

- *interrupts_state_maze()*: função que lida com as interrupções quando o programa se encontra no labirinto (jogo). Chama a função *play()*, que irá verificar o que provoca a informação transmitida pela tecla premida. Se o utilizador terminar o nível, age em conformidade com o caso de ter ou não completado o nível, fazendo as alterações necessárias. Realiza as alterações provocadas pressão da tecla *esc*. Utiliza o *timer* para atualizar as *frames* e o *counter* do tempo que passou desde que o utilizador iniciou o nível.

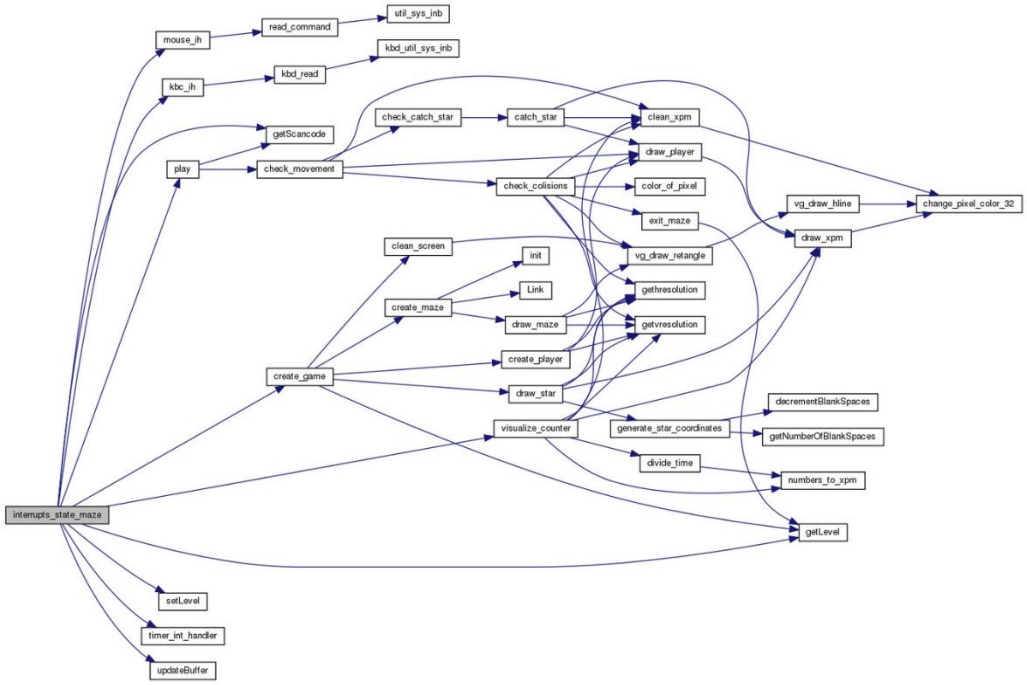


Figura 13 *Call graph* da função *interrupts_state_maze()*

- *interrupts_state_message()*: função que lida com as interrupções quando o programa se encontra a mostrar mensagens ao utilizador. Ao fim de cada nível, mostra mensagem uma correspondente ao que se passou durante 3 segundos.

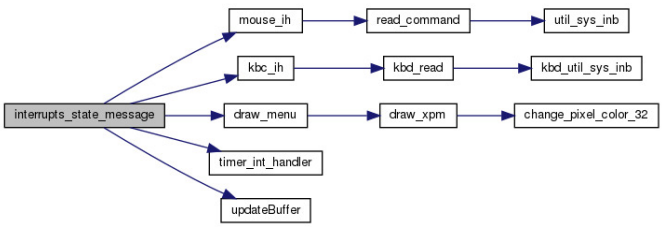


Figura 14 *Call graph* da função *interrupts_state_message()*

Neste módulo, são utilizadas algumas estruturas de dados, tais como *RealTime*, *Numbers*, *Menu* e *Mouse*.

Peso: 24%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.9 Play

Este módulo gere o jogo em si.

- *create_game()*: função que limpa do ecrã o menu, cria o labirinto e desenha-o no ecrã, bem como as estrelas e o jogador.

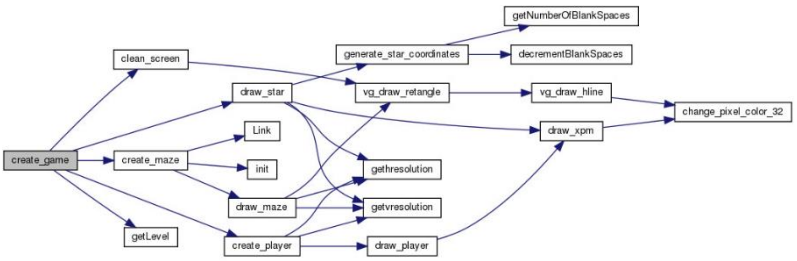


Figura 15 *Call graph* da função *create_game()*

- *play()*: função que lê a informação transmitida pela tecla e chama a função *check_movement()* → *player.h* que lida com o que o movimento transmitido pela tecla provoca. Verifica ainda se o nível acabou e se foi com ou sem sucesso.

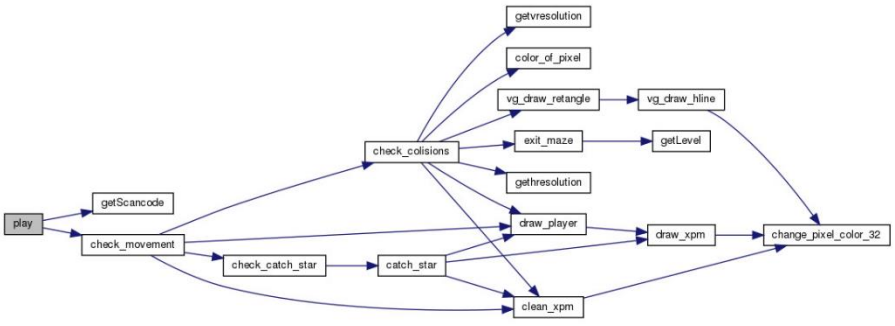


Figura 16 *Call graph* da função *play()*

Peso: 5%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.10 Maze

Neste modulo, é gerado o labirinto de acordo com um algoritmo que é uma versão randomizada da busca em profundidade. Este algoritmo foi escolhido uma vez que, além de ser o mais simples de ser implementado, é também o que tem um baixo fator de ramificação e "corredores" mais longos. Este método foi retirado de um website (*https://en.wikipedia.org/wiki/Maze_generation_algorithm*), tendo sido realizadas algumas alterações, tal como a forma do labirinto ser mostrado no ecrã, já que foram utilizados quadrados para representar as paredes e não carateres.

O algoritmo consiste no seguinte: começando com uma grelha quadrada de células, cada uma destas contem quatro paredes inicialmente. Escolhendo uma célula aleatoriamente, é selecionado de forma aleatória um vizinho da mesma, sendo removida a "parede" que os separava e este vizinho é assinalado como visitado. O processo continua da mesma forma até que cada célula tenha sido visitada e tenha sido garantido que existe, pelo menos, um caminho que ligue uma célula a outra qualquer escolhida.

- *Init()*: função que inicializa o labirinto com todas as células constituídas por 4 paredes, começando por alocar a memória para o labirinto.
- *Link()*: função que tenta fazer a conexão do *node* com um vizinho aleatório
- *draw_maze()*: função que desenha o labirinto gerado. Começa para alocar memória para as variáveis *blank_spaces* e *walls*. Desenha a saída a verde e todas as restantes paredes a azul.
- *create_maze()*: função que cria o labirinto, começando por criar os nós principais, depois conecta os nós a partir de um *node* com a função *link()* até que se volte ao *node* inicial. Por fim, desenha o labirinto.

Neste módulo, são utilizadas algumas estruturas de dados, tais como Walls, Node e Blank_space.

Peso: 5%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.11 Player

É o módulo responsável pela criação de todos os objetos do jogo, deteção de colisões, movimentos e da saída do labirinto.

- *create_player()*: função que, no início do jogo, cria o jogador (bola vermelha) e o desenha nas suas coordenadas iniciais.

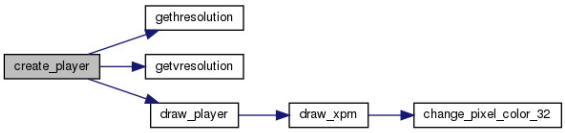


Figura 17 *Call graph* da função *create_player()*

- *draw_player()*: função que desenha o jogador (bola vermelha) no início do jogo e sempre que é atualizada a sua posição.

Figura 18 *Call graph* da função *draw_player()*

- *draw_star()*: função chamada no início de cada nível que aloca para espaço para o *array stars* (estrelas do nível) e grey_*stars* (estrelas no "contador" de estrelas apanhadas), fazendo também o *load* destes *xpms*, e desenha as estrelas no labirinto, sendo o número de estrelas a ser desenhado definido em cada nível. Para além disso, desenha no rodapé do labirinto a contagem de estrelas apanhadas, isto é, inicialmente são desenhadas o mesmo número de estrelas cinzentas que o jogador tem de apanhar, no nível no rodapé do labirinto.

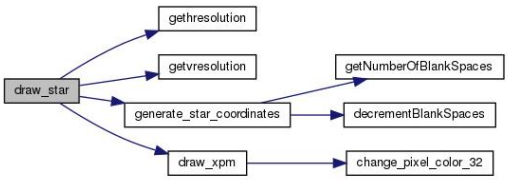


Figura 19 *Call graph* da função *draw_star()*

- *generate_star_coordinates()*: função que gera de forma aleatória as coordenadas onde a nova estrela pode ser criada, utilizando o algoritmo *rand* da *standard library*, procurando quais os locais ainda vazios e atualizando no final estes locais.

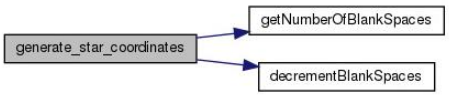


Figura 20 *Call graph* da função *generate_star_coordinates()*

- *check_movement()*: função chamada sempre que é lida informação do teclado que verifica que tipo de movimento o utilizador quer fazer (*up*,*down*, *left*,*right*), verifica se este movimento não provoca nenhuma colisão com paredes, chamando a função *check_colisions()*; caso o movimento seja possível atualiza as coordenadas do jogador. Verifica ainda se alguma estrela está a ser apanhada chamando a função *check_catch_star()* e se o jogador se encontra na saída do labirinto, através da função *check_colisions()*. Se o jogador estiver na saída, a função examina se foram cumpridas as condições para passar para o nível seguinte.

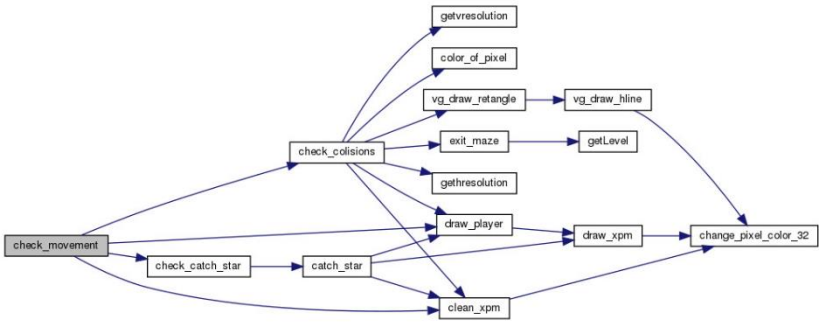


Figura 21 *Call graph* da função *check_moviment()*

- *check_colisions()*: função que verifica se o movimento não provoca colisão com paredes e se o jogador com este movimento alcança a saída, atualizando variáveis necessárias para o restante decorrer do jogo.

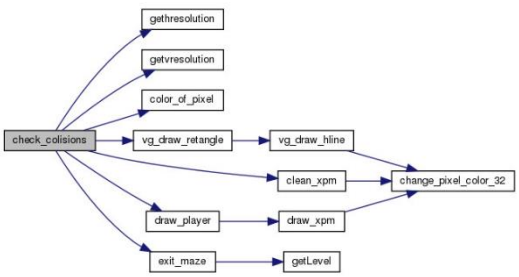


Figura 22 *Call graph* da função *check_colisions()*

- *catch_star()*: função que "apanha" uma estrela, isto é, coloca o parâmetro *catch* da estrela apanhada a *true*, elimina a estrela do labirinto desenhando a bola no seu local e atualiza o contador de estrelas apanhadas.

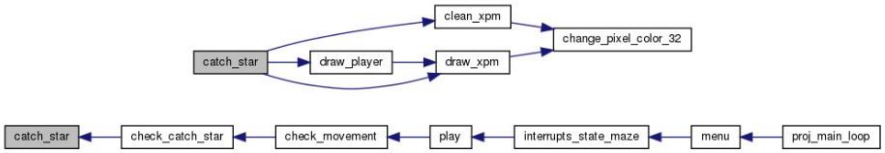


Figura 23 *Call graph* da função *catch_star()*

- *check_catch_star()*: função chamada a cada movimento do jogador. Verifica se o movimento faz com que apanhe uma estrela e, se tal for o caso, chama a função *catch_star()* para tratar desse caso.

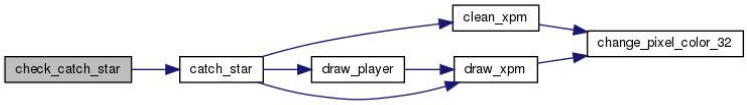


Figura 24 *Call graph* da função *check_catch_star()*

- *exit_maze ()*: função chamada quando o jogador se encontra na saída e que serve para verificar se todas as condições para completar o nível foram cumpridas.

Neste módulo, são utilizadas algumas estruturas de dados, tais como *Star*, *Ball* e *Blank_space*.

Peso: 15%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.12 Auxiliar Functions

Neste modulo encontram-se funções auxiliares ao bom funcionamento do programa.

- *create_mouse()*: cria uma *struct Mouse* com a imagem do cursor e na posição inicial que pretendemos e, por fim, desenha-o no ecrã .

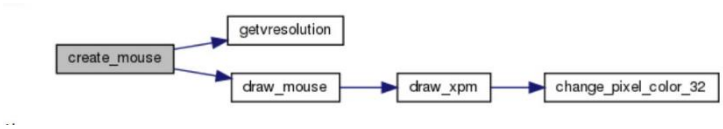


Figura 25 *Call graph* da função *create_mouse()*

- *draw_mouse()*: desenha o *mouse* no ecrã.

Figura 26 *Call graph* da função *draw_mouse()*

- *check_possible_mov()*: função que calcula as novas coordenadas do rato com o movimento imposto pelo utilizador, verificando se não excede os limites do ecrã e lida com o caso de o movimento provocar a saída do ecrã.

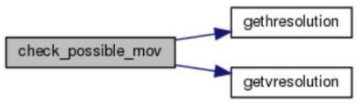


Figura 27 *Call graph* da função *check_possible_mov()*

- *clean_mouse()*: esta função apaga o rato do ecrã preenchendo o local onde ele se encontrava com parte do fundo.

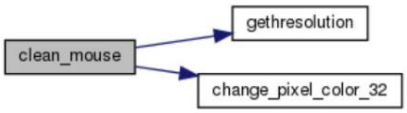


Figura 28 *Call graph* da função *clean_mouse()*

- *draw_menu()*: esta função desenha no ecrã o menu correspondente ao estado do programa.

Figura 29 *Call graph* da função *draw_menu()*

- *create_menu()*: função que inicializa todos os parâmetros da *struct* Menu.

Figura 30 *Call graph* da função *create_menu()*

- *create_numbers()*: função que faz *load* de todas as imagens necessárias para exibir o *counter*.
- *numbers_to_xpm()*: função que nos retorna o *xpm* correspondente ao número passado como argumento.
- *divide_time()*: função que separa o tempo nos seus diversos parâmetros e atualiza a *struct time*_now.

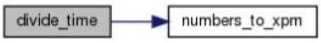


Figura 31 *Call graph* da função *divide_time()*

- *visualize_counter()*: função que mostra o *counter* no ecrã.

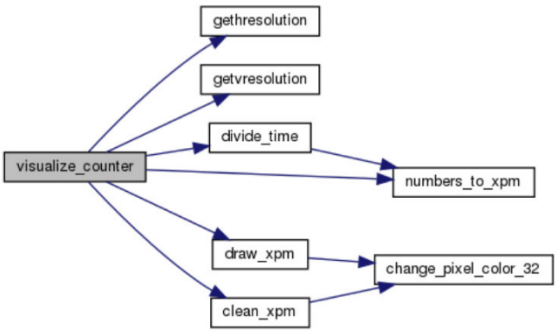


Figura 32 *Call graph* da função *visualize_counter()*

Neste módulo, são utilizadas algumas estruturas de dados, tais como *packet*, *Time*, *Menu* e *Mouse*.

Peso: 5%

Contribuição: Mariana Truta (50%), Rita Peixoto (50%)

3.13 Call Graph

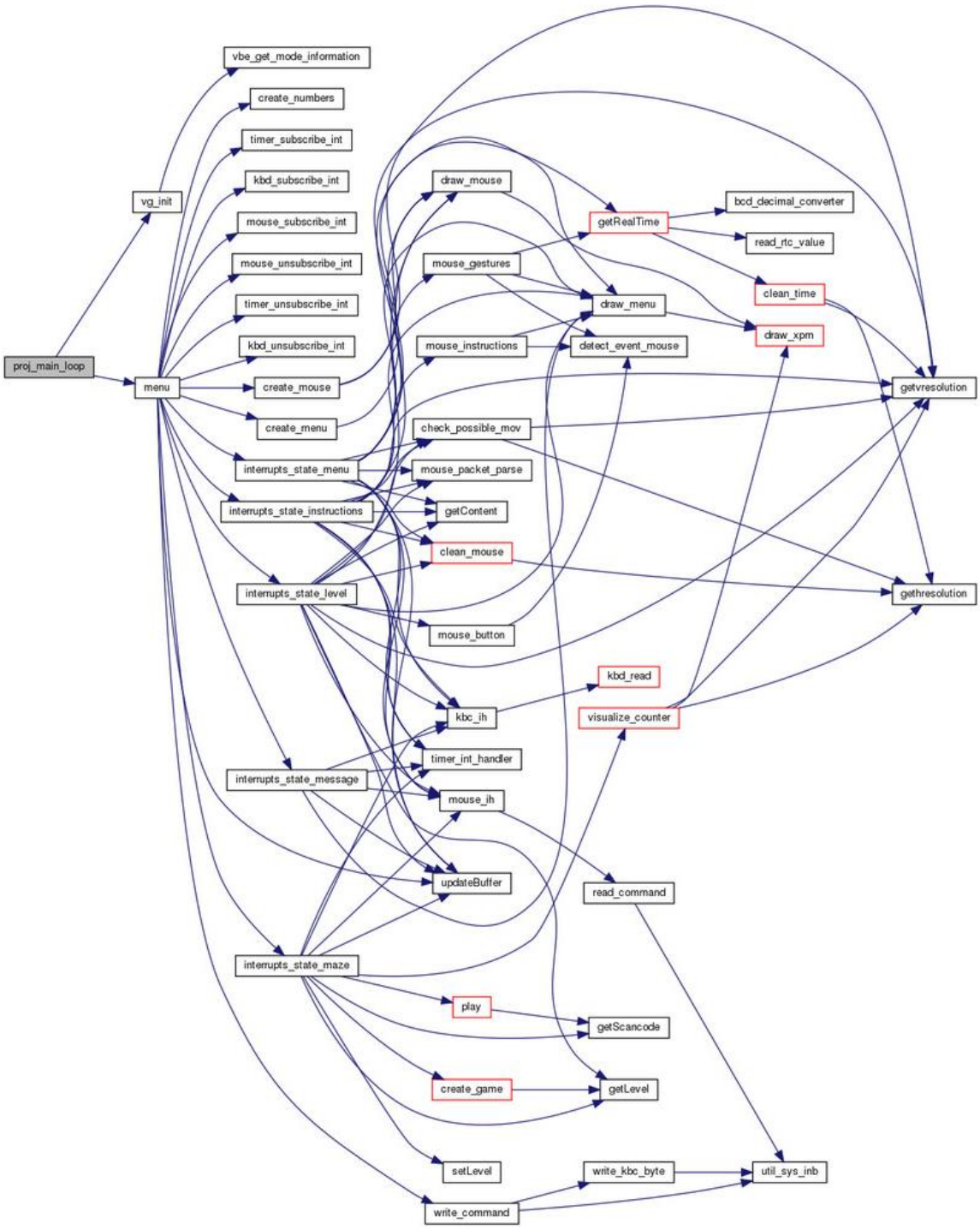


Figura 33 *Call graph* do projeto

4. Detalhes de implementação

Ao longo do nosso projeto, tivemos a oportunidade de identificar a importância do uso de estruturas de dados para guardar algumas informações necessárias, por exemplo, o movimento do rato e da bola. Inicialmente tivemos problemas neste aspeto já que a partilha de dados entre módulos estava a ser bastante complicada, mas foram resolvidos a partir do momento em que criamos várias *structs* e *enums* para guardar todas as variáveis necessárias para o jogo funcionar. Desta forma, apenas bastava colocar o apontador para um dos objetos como argumento da função e esta poderia modificar qualquer membro da estrutura.

```
enum state_game{
  state_menu,
  state_level,
  state_maze,
  state_instructions,
  state_message
};

typedef struct
{
    uint16_t x, y;
    void *parent;
    char c;
    char directions;
} Node;

typedef struct
{
    uint16_t x, y;
} Wall;

typedef struct {
    uint16_t x, y;
} Blank_space;


typedef struct {

  int x, y;
  xpm_image_t img;
} Mouse;


enum type{
  initial_stat,
  play_stat,
  quit_stat,
  level_stat,
  instructions_stat,
  instructions_button_stat,
  message_completed_stat,
  message_failed_stat,
  message_all_passed
};
```

```
typedef struct {

  uint16_t x, y;
  xpm_image_t img_initial;
  xpm_image_t img_play;
  xpm_image_t img_quit;
  xpm_image_t img_level;
  xpm_image_t img_instructions;
  xpm_image_t img_instructions_button;
  xpm_image_t img_message_complete;
  xpm_image_t img_message_failed;
  xpm_image_t img_all_passed;
  enum type menu_type;
} Menu;


enum state_t{
  SOMETHING_ELSE,
  BUTTON_RELEASED
};

struct Screen_pos{
  double xmin;
  double xmax;
  double ymin;
  double ymax;
};

typedef struct {

  uint16_t x, y, speed;
  xpm_image_t img;
} Ball;

typedef struct {

  uint16_t x, y;
  xpm_image_t img;
  bool st_catch;
} Star;

typedef struct {

  xpm_image_t zero, one, two, three, four, five, six, seven, eight, nine, colon, slash;
} Numbers;

typedef struct {
  int m1, m2, s1, s2;
  int w1, w2, w3, w4;
  xpm_image_t m1_xpm, m2_xpm, s1_xpm, s2_xpm;
} Time;

typedef struct {

  int second, minute, hour, day, month, year;
  int wt;
} RealTime;
```

Figura 34 Estruturas de dados definidas no ficheiro *auxiliar_structs.h*

O facto de termos decidido colocar o nosso labirinto no centro do ecrã e manter o resto do jogo em *full screen*, de forma a tornar tudo mais visualmente agradável, implicou que fossem usados um *x_centrado* e um *y_centrado*, já que mudar diretamente a forma como se desenhava os vários objetos na função *change_pixel_color()* implicaria que todo o jogo tivesse de ficar centrado, o que não era o pretendido.

Um aspeto importante do nosso trabalho consiste no facto das colisões com as paredes no labirinto serem detetadas pixel a pixel, isto é, uma função verifica se o movimento pretendido pelo jogador leva a que a bola se desloque até a um local onde a cor do pixel

é a cor das paredes do labirinto. No movimento do rato, as colisões são detetadas a partir da posição relativa dos botões, sendo a imagem anterior ao movimento do cursor limpa com a parte do fundo respetiva à posição do mesmo.

5. Conclusão

Consideramos que LCOM foi uma unidade curricular que exigiu muito trabalho e dedicação da nossa parte e não foi de todo fácil.

Os *handouts* do *Labs* muitas vezes não facilitaram o nosso trabalho, porque têm a informação muito dispersa e são muito longos e, no início não tendo conhecimentos sobre o que é lecionado torna-se um grande desafio.

A adaptação à cadeira foi um bocado tardia, notando-se uma gigante evolução durante o projeto. No entanto, o facto de não termos tido *feedbackb* dos *Labs*, antes do projeto, dificultou ligeiramente o mesmo uma vez que se tivéssemos sido alertadas para alguns erros das nossas resoluções, talvez teríamos perdido menos tempo a fazer *debug* de código anteriormente feito.

Gostaríamos de salientar que o *Lab5* revelou-se ser o mais complicado na altura, no entanto, com este projeto conseguimos realmente compreendê-lo na sua totalidade.

O projeto, além de ser desafiador, tornou-se de elevadíssimo interesse, principalmente por vermos que, apesar de cada vez que acrescentávamos algo nos deparássemos com 1001 problemas, também surgiam 1001 ideias para o melhorar. Também nos obrigou a refletir um pouco sobre organização do nosso código e a sua eficiência para tornar o programa coerente e fluído.

Por fim, acrescentamos que, quando desenvolvemos a especificação do nosso projeto, não tínhamos ainda a noção do que eramos ou não capazes de fazer e por isso acabou por se tornar um pouco vaga. É importante realçar também que este trabalho explorou ao máximo as nossas capacidades e conhecimentos acerca do que é lecionado na cadeira.