U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Relatório SaveXmas 2020

Trabalho realizado por:

Grupo 4, Turma 2
Beatriz Aguiar, up201906230
Margarida Vieira, up201907907

No âmbito da Unidade Curricular:

Laboratório de Computadores

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Índice

Interface do jogo

É ao nível da interface do jogo nos menus que a utilização, sobretudo, do mouse ganha mais expressão.

Utilizando este periférico, o utilizador tem a possibilidade de interagir e executar um conjunto de ações, mediante o menu onde se encontra e o botão que premir.

Uma vez que, tal como supramencionado, as ações a realizar dependem da interface atual onde se encontra o utilizador, estas serão explicadas com mais detalhe em cada uma das secções que se seguem abaixo.

Init Menu

No menu inicial, o utilizador tem a possibilidade de selecionar o modo de jogo *Singleplayer*, cuja implementação será explorada mais detalhadamente no decorrer deste relatório, bastando, para tal, premir o respectivo botão.

Para além disso, damos também a possibilidade ao utilizador de consultar as instruções do jogo, clicando no botão *Instructions* e, ainda, de sair do jogo, ao premir o botão *Exit*.

Por último, é feito também o display da horas atual, *feature* para a qual recorremos à utilização do *RTC - Real Time Clock*, cuja implementação, à semelhança de outros aspetos acima, será explicada a posteriori.

Figura 1. Init Menu

Choose Player Menu

Neste menu, o jogador tem a possibilidade de escolher qual o personagem que quer encarnar no jogo, podendo optar por uma de duas opções, as quais designamos de "*Covid Santa*" – *fig. 2*, e "*Happy Santa*" – *fig. 3*.

Para tal, basta fazer *swipe left*, ao clicar com o rato na seta para a esquerda, ou *swipe right*, ao clicar com o rato na seta para a direita, até que surja na janela o personagem que pretende escolher.

Figura 2. Choose Player Menu - Covid Santa

Figura 3. Choose Player Menu - Happy Santa

Instructions Menu

O nome deste menu é autoexplicativo. Aqui, o utilizador tem a possibilidade de consultar as instruções do jogo, caso tenha dúvidas acerca das suas regras e funcionamento ou, simplesmente, se assim o desejar.

Para navegar neste menu e ler o texto respeitante às instruções, o utilizador tem apenas que movimentar o rato, sem necessidade de premir qualquer botão.

Assim que desejar regressar ao menu inicial, basta-lhe clicar no "*home button*", no canto superior esquerdo.

Figura 2. Instructions Menu

Level Completed Menu

Esta interface é mostrada ao utilizador quando ele completa todos os propósitos do jogo com sucesso, ou seja, consegue apanhar todos os presentes e chegar ao iglo antes do cronómetro terminar a contagem decrescente, de quatro minutos.

Neste caso, tal como é mostrado no ecrã, o jogador recebe um total de três estrelas.

Terminado o jogo, o jogador tem agora a possibilidade de voltar ao menu incial, premindo o botão *Main Menu*, ou de fechar a aplicação, premindo o botão *Exit*.

Figura 3. Level Completed Menu

Level Failed Menus

Figura 4. Level Failed – two stars

Figura 5. Level Failed - one star

Esta interface é mostrada ao jogador quando ele não consegue completar todos os objetivos do jogo.

Se o jogador tiver conseguido apanhar todos os presentes sem, no entanto, ter chegado ao iglo antes do cronómetro terminar a contagem decrescente, é-lhe mostrada no ecrã a informação de que recebeu duas estrelas – *fig. 6*.

Qualquer resultado não contemplado no cenário acima tem como consequência a atribuição de apenas uma estrela ao jogador – *fig. 7*.

Finalmente, em ambos os casos, o jogador tem a possibilidade de voltar ao menu incial, premindo o botão *Main Menu*, ou de fechar a aplicação, premindo o botão *Exit*.

Pause Menu

Em modo de jogo, sempre que o jogador prime a tecla ESC, é direcionado para este menu, significando que colocou o jogo em pausa.

Pode permanecer neste menu o tempo que desejar, retomar o jogo exatamente no mesmo ponto onde estava imediamente antes de ter sido pausado, clicando no botão *Resume*, voltar ao menu principal, se clicar no botão *Main Menu* ou, ainda, fechar a aplicação, premindo o botão *Exit*.

Figura 6. Pause Menu

Dispositivos usados

Overview dos dispositivos usados

Dispositivo	***Breve descrição da utilização***	***Interrupções***
Timer	Controlar o display do conteúdo do buffer no ecrã. Controlar a atualização do cronómetro no estado de jogo.	Sim
Keyboard	Controlar todos os movimentos dos sprites no estado de jogo. Pausar o jogo, clicando na tecla ESC.	Sim
Mouse	Controlar a navegação pelos menus. Mover os sprites para a esquerda e para a direita no estado de jogo.	Sim
Video Card	Responsável pela gestão de toda a interface da aplicação, em modo gráfico.	N/A
RTC	Mostrar a hora atual no menu inicial.	Sim

Nas secções abaixo, serão mencionadas as *source files* onde o dispositivo se torna crucial, bem como as **principais** funções e variáveis onde se manifesta a sua implementação. No entanto, para um melhor entendimento do funcionamento do jogo, torna-se essencial a análise das restantes funções.

Timer

Este dispositivo desempenha um papel fundamental no jogo uma vez que é com base nas suas interrupções que é feito o display do conteúdo do buffer principal no ecrã.

A cada interrupção do timer é feita a atualização das coordenadas quer do cursor do rato, quando nos menus, quer dos sprites e do tempo respetivo ao cronómetro, quando no modo de jogo propriamente dito.

Também relativamente ao cronómetro mencionado acima, o jogador dispõe de um tempo limite de quatro minutos para completar todos os objetivos do jogo. O valor correspondente a esse tempo é guardado numa variável cuja atualização é feita também com base nas interrupções do timer.

- ***timer.c***
- ***game_ctrl.c***
  funções: ***timer_events_handler***
  variáveis: ***cronometer***
- ***game.c***
  variáveis: ***time_numbers***
  current_time

Keyboard

Este dispositivo atua ao nível do estado do jogo e dos sprites.

Por um lado, se o jogador premir a tecla ESC, o jogo será interrompido e colocado em pausa.

Por outro lado, é responsável pela movimentação dos sprites, podendo fazê-los saltar (teclas ↑ ou W), andar para a direita (teclas → ou D) ou andar para a esquerda (teclas ← ou A).

- ***kbc.c***
- ***keyboard.c***
- ***game_ctrl.c***
  funções: ***keyboard_events_handler***
  move_santa_keyboard

- ***game.c***

  variáveis: ***keyboard_event***

Mouse

As interrupções deste dispositivo permitem recolher os dados necessários à montagem de *packets* que, por sua vez, contém toda a informação relevante para a atualização das coordenadas quer do sprite do cursor, nos menus, quer dos sprites das personagens do jogo, se o jogador decidir movimentá-las, para a esquerda ou para a direita, utilizando este periférico.

Para além disso, ainda relativamente aos menus, é com base na informação retirada dos *packets* do mouse que o programa sabe se foi ou não premido algum botão e, caso algum tenha sido efetivamente premido e após uma verificação das coordenadas do cursor, executa ações em conformidade.

- ***kbc.c***
- ***mouse.c***
- ***game_ctrl.c***

  funções: ***mouse_events_handler***

  mouse_events_handler_while_playing

  move_santa_mouse
- ***game.c***

Video Card

Este dispositivo é responsável por tudo o que diz respeito aos gráfico do jogo, sendo, como tal, a peça chave que dá vida e permite a visualização das *features* que já foram sendo mencionadas.

Optou-se pelo modo 0x14C, com uma resolução horizontal de 1152 por uma resolução vertical de 864 píxeis, por ser o mais adequado ao tipo de background de jogo que se pretendia alcançar.

De forma a evitar uma pobre qualidade de imagem, implementou-se a técnica de *double buffering* - com o buffer auxiliar limpo, o programa efetua a renderização das frames necessárias para o mesmo e, uma vez todo o conteúdo no buffer auxiliar, este é copiado para o buffer principal para que seja efetivamente mostrado no ecrã.

Relativamente às movimentações das personagens, utilizou-se animação de sprites, com deteção de obstáculos a partir das suas coordenadas e sabendo, a priori, todas as coordenadas suscetíveis de constituirem um obstáculo.

- ***video_gr.c***
- ***game_ctrl.c***

  funções: ***update_display***

  update_display_player

  update_display_instructions

  update_playing_display
- ***game.c***

  variáveis: ***main_buf***

  aux_buf

Real Time Clock (RTC)

Este dispositivo é utilizado para obter a hora atual, a qual é mostrada no menu inicial.

- ***rtc.c***
- ***game_ctrl.c***
- ***game.c***

  variáveis: ***curr_time***

  rtc_time

Estruturação do código

Módulo	Peso relativo (%)
macros	4
xpms	3
utils	1
kbc	2
keyboard	6

Módulo	Peso relativo (%)
mouse	7
video_gr	5
rtc	4
game_ctrl	40
game	30

macros

Esta *header file* foi criada com o propósito de reunir as macros essenciais ao jogo.

A utilização de macros permite tornar o código muito mais intuitivo e de fácil interpretação. Para além disso, na eventualidade de se ter que alterar um dos seus valores, evita-se ter que o fazer em todos os sítios onde este é utilizado.

A maior parte das macros foram importadas das aulas práticas laboratoriais, nomeadamente as que dizem respeito aos dispositivos implementados nas mesmas.

Para além disso, foram acrescentadas outras, nomeadamente ***STEP_KB***, ***STEP_JUMP_X***, ***STEP_JUMP_Y***, ***STEP_TXT***, que remetem para os deslocamentos dos sprites, ou ainda ***TIME_LIMIT*** e ***NUM_PRESENTS***, que registam, respetivamente, o tempo limite de que o jogador dispõe para completar o jogo e o número de presentes a ser apanhados, entre outras.

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

xpms

Esta *header file* reúne a referência aos vários xpms utilizados no jogo.

CONTRIBUIÇÃO: 100% Beatriz | 0% Margarida

REFERÊNCIAS: https://www.gameart2d.com/santa-claus-free-sprites.html
https://craftpix.net/freebies/xmas-games-gui/
https://www.gameart2d.com/winter-platformer-game-tileset.html

utils

Este módulo reúne funções úteis, a destacar a ***util_sys_inb*** que procede à adaptação da função ***sys_inb***, para permitir a leitura de dados do tipo *uint8_t.*

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

timer

O código relativo a este dispositivo foi reutilizado das aulas práticas laboratoriais, não tendo sido efetuadas quaisquer modificações relevantes.

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

kbc

Sendo o KBC o controlador comum ao keyboard e ao mouse, este módulo reúne, portanto, funções genéricas que podem ser extendidas a ambos os dispositivos, sem necessidade de adaptação a cada um deles, nomeadamente ***kbc_read*** e ***kbc_write***.

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

keyboard

O código relativo a este dispositivo foi reutilizado das aulas práticas laboratoriais, não tendo sido efetuadas quaisquer modificações relevantes.

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

mouse

O código relativo a este dispositivo foi reutilizado das aulas práticas laboratoriais, não tendo sido efetuadas quaisquer modificações relevantes.

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

video_gr

Grande parte do código relativo a este módulo foi reutilizado das aulas práticas laboratoriais.

No entanto, para além desse, procedeu-se ainda à implementação de duas funções responsáveis pela gestão do *double buffering* – ***clear_buffer***, que limpa o buffer recebido como parâmetro, e ***page_flipping***, que renderiza para o buffer principal – variável global ***main_buf***, o conteúdo do buffer auxiliar – variável global ***aux_buf***.

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

rtc

Este módulo atua ao nível da variável global ***curr_time***, cujo papel é guardar dados relativos à hora atual.

Os valores desta variável, no formato BCD, pré-definido pela invocação da função ***rtc_set_BCD***, são preenchidos com recurso à função ***rtc_read_time***. Posteriormente, são convertidos para o formato binário, através da função ***BCD_to_binary*** e, finalemente, para o formato decimal, através da função ***binary_to_decimal***.

De resto, todo este processo só ocorre se a função ***rtc_reading_allowed*** detetar que tal é possível e é gerido pelo *handler* de interrupções, ***rtc_ih***, que, para tornar o código o mais flexível possível, recebe como parâmetro o tipo de interrupção a ser *handled*.

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

game_ctrl

Este módulo é, sem dúvida, o motor do jogo, funcionando, como de resto o próprio nome inidca, como o controlador do mesmo.

Começou por se definir variáveis do tipo *enum* para guardar o estado do jogo – ***game_state*** (do tipo *enum* ***game_states*** – *fig. 7*), do personagem a ser mostrado naquele momento na janela do *Choose Player Menu* descrito acima – ***char_state*** (do tipo *enum* ***palyer_states*** – *fig. 9*) e do personagem – ***santa_state*** (do tipo *enum* ***santa_states*** – *fig. 8*). Para além disso, definiu-se ainda uma variável para registar o tipo de evento ocorrido mediante uma interrupção do keyboard – ***keyboard_event*** (do tipo *enum* ***keyboard_events*** – *fig.10*).

Enumerator
INITMENU
PAUSE
CHOOSE_PLAYER
INSTRUCTIONS_MENU
PLAYING
WON
TIME_OUT
IGLO
INSTRUCTIONS
EXIT

Figura 7. enum game_states

Enumerator
STANDING_RIGHT
STANDING_LEFT
WALKING_RIGHT
WALKING_LEFT
JUMPING_RIGHT
JUMPING_LEFT
FALLING_RIGHT
FALLING
FALLING_LEFT
DEAD

Figura 8. enum santa_states

Enumerator
HAPPY_SANTA
COVID_SANTA

Figura 9. enum player_states

Enumerator
ESC
LEFT
RIGHT
UP
RELEASED

Figura 10. enum keyboard_events

Quando ocorre uma interrupção, esta é processada pelo *handler* do dispositivo correspondente e, de seguida, encaminhada para o *handler* deste módulo, para que sejam desencadeadas ações em conformidade. Aqui, destacam-se as funções – ***timer_events_handler*** – *fig. 11*, ***set_keyboard_events*** – *fig. 12*, ***keyboard_events_handler*** – *fig. 13*, ***mouse_events_handler*** – *fig. 14* e ***mouse_events_handler_while_playing*** – *fig. 15*.

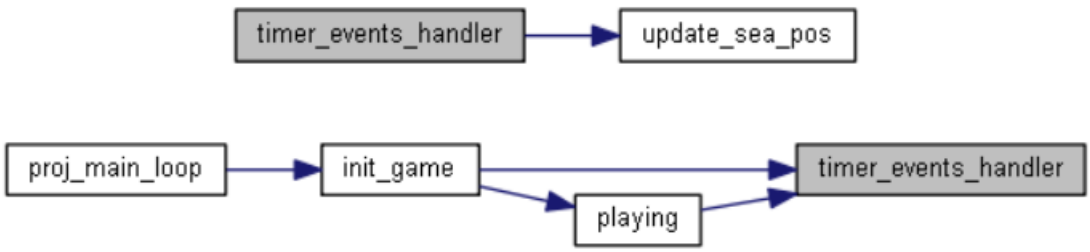

Figura 11. timer_events_handler call graph

Figura 12. set_keyboard_events call graph

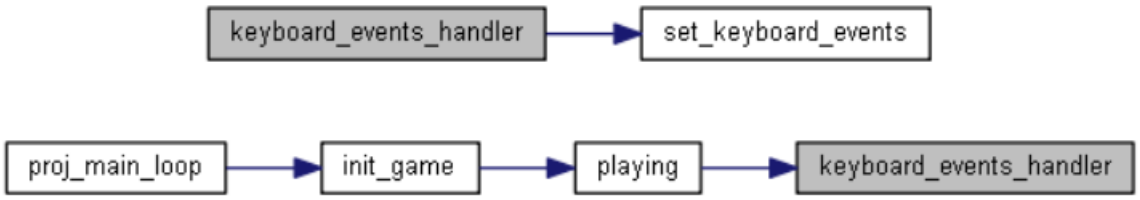

Figura 13. keyboard_events_handler call graph

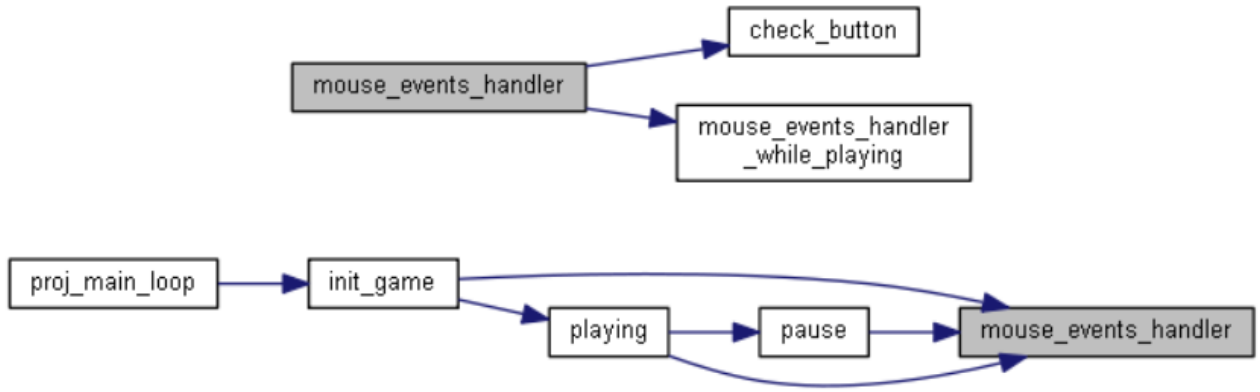

Figura 14. mouse_events_handler call graph

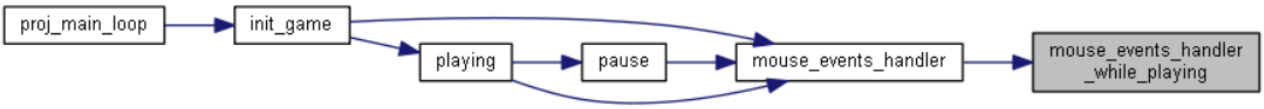

Figura 15. mouse_events_handler_while_playing call graph

Quando as ações a ser desencadeadas passam por atualizar os dados de determinado(s) sprite(s), esta funcionalidade é garantida pelas funções – ***update_cursor*** – *fig. 16*, ***update_time*** – *fig. 17*, ***update_cronometer*** e ***update_santa_pos***.

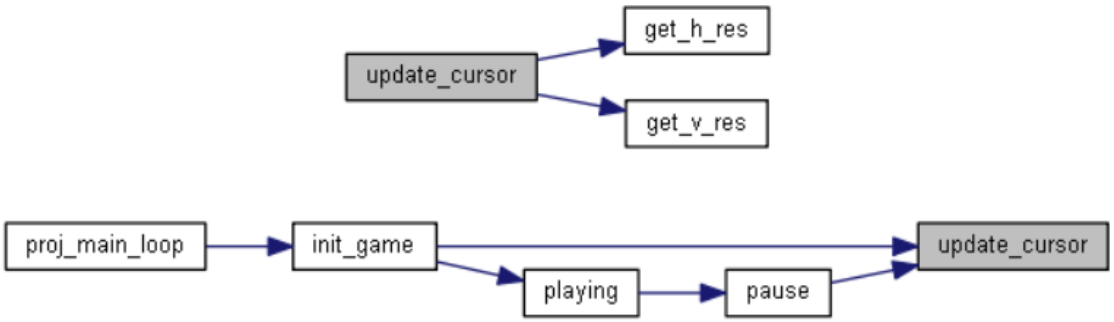

Figura 16. update_cursor call graph

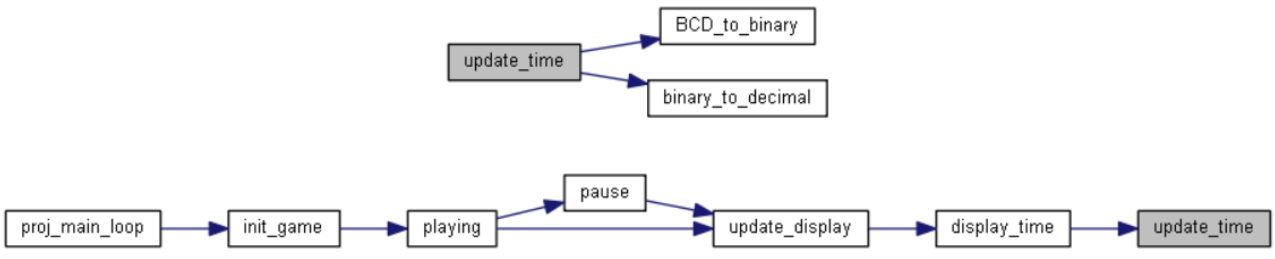

Figura 17. update_time call graph

As funções ***check_collision*** – *fig. 18* e ***check_victory*** – *fig. 19*, merecem especial destaque, uma vez que implementam toda a deteção de colisões e se daí resultará a vitória ou derrota do jogador. Para além destas, há que mencionar ainda a ***check_button***, que permite verificar se foi premido algum botão e a ***check_present***, que verifica se algum presente foi apanhado pelo jogador retornando, em caso afirmativo, o identificador do mesmo.

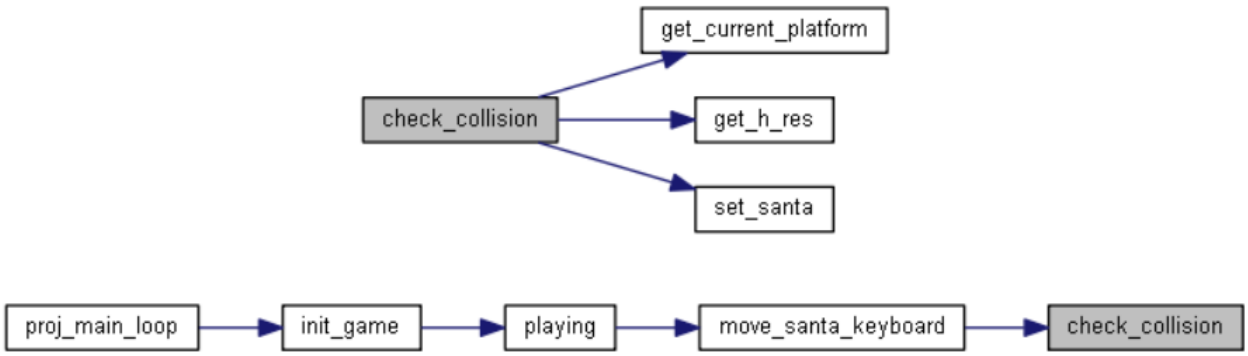

Figura 18. check_collision call graph

Figura 19. check_victory call graph

As funções ***move_santa_keyboard*** – *fig. 21* e ***move_santa_mouse*** – *fig. 20*, sempre aliadas à ***check_collision*** – *fig. 18*, implementam a movimentação do sprite do personagem, através da devida atualização das suas coordenadas, quando este é movido com recurso ao keyboard, no caso da primeira, ou ao mouse, no caso da segunda.

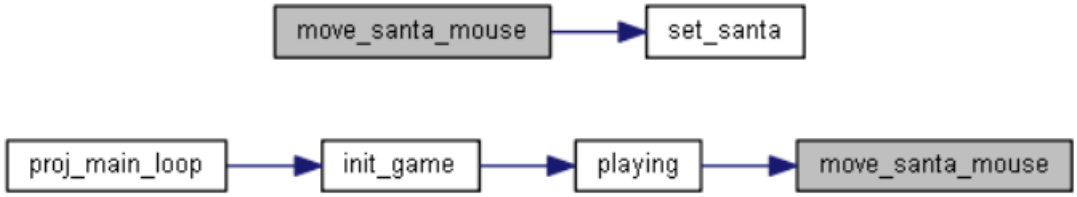

Figura 20. move_santa_mouse call graph

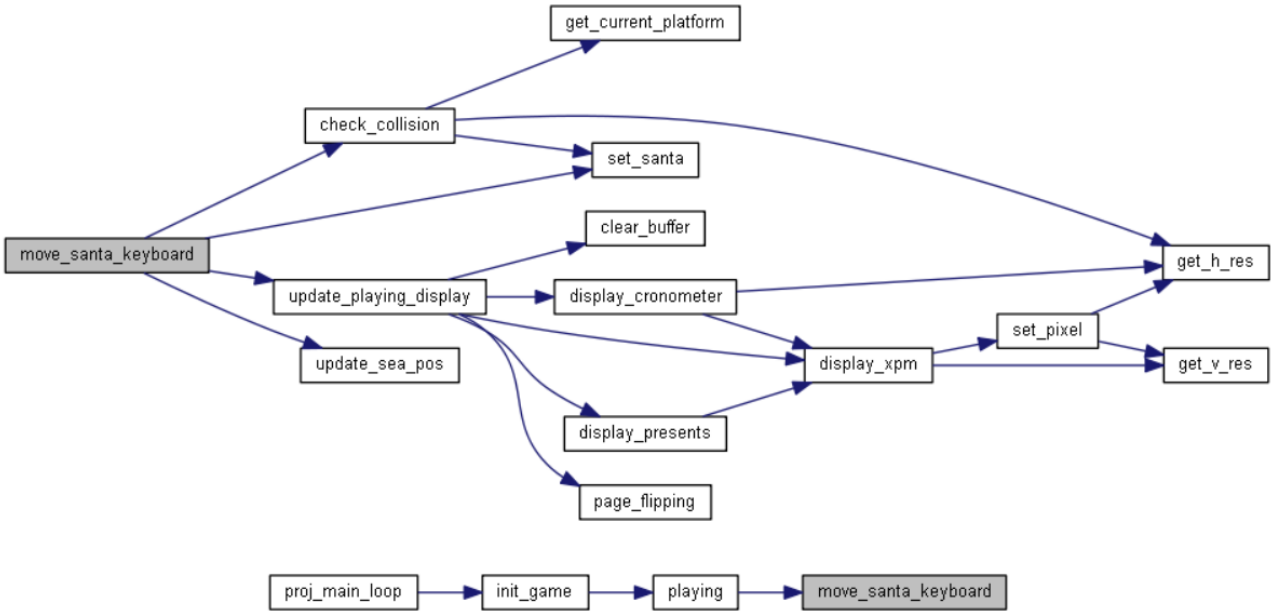

Figura 21. move_santa_keyboard call graph

Por último, há que mencionar ainda as funções que concretizam, efetivamente, o display de tudo o que, dada a interface atual, tem que ser mostrado no ecrã – ***update_display*** *– fig. 22*, ***update_display_player*** *– fig. 23*, ***update_display_instructions*** *– fig. 24* e ***update_playing_display*** *– fig. 25*.

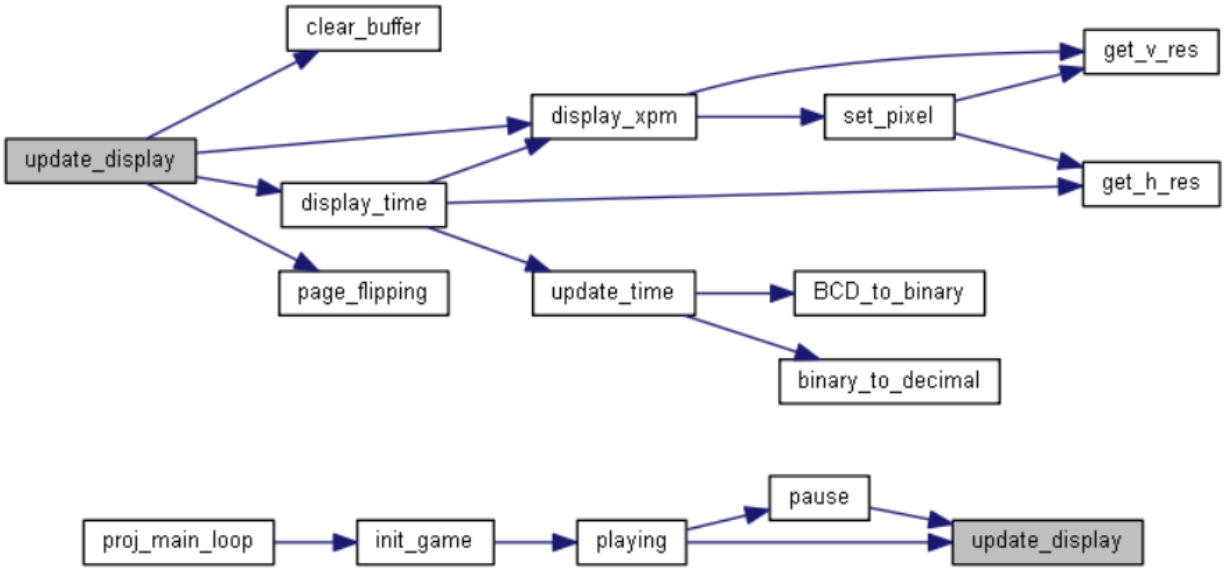

Figura 22. update_display call graph

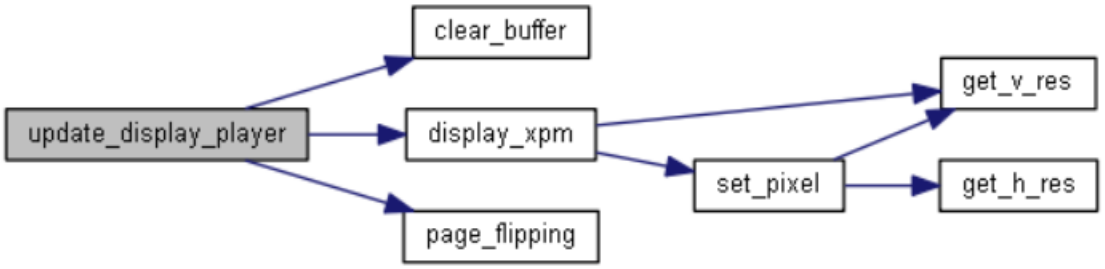

Figura 23. update_display_player call graph

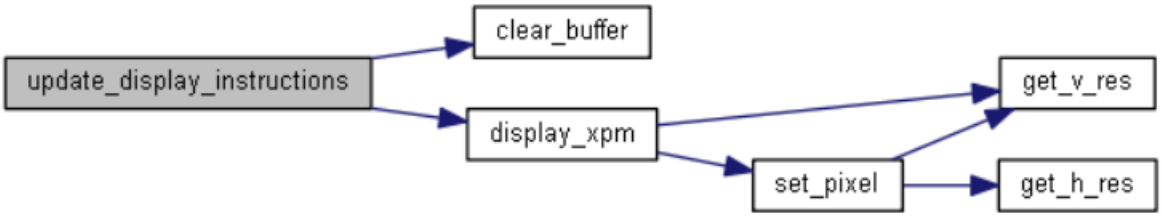

Figura 24. update_display_instructions call graph

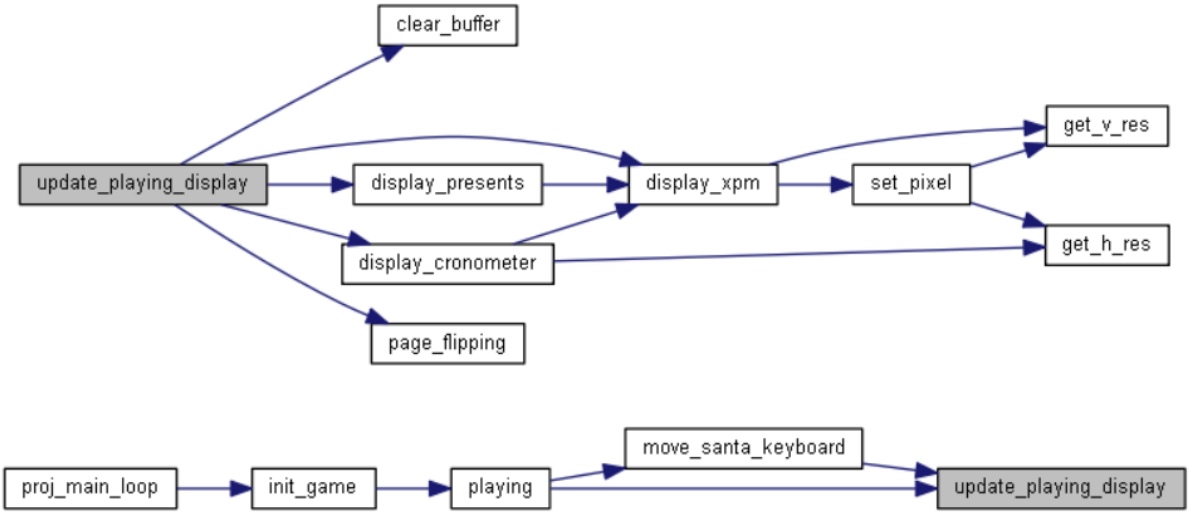

Figura 25. update_playing_display call graph

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

game

Este módulo é responsável por unir todas as peças implementadas nos módulos acima.

Aqui são efetuadas todas as alocações de memória necessárias, são chamadas as funções ***enable***, ***disable***, ***subscribe_int*** e ***unsubscribe_int*** respetivas a cada dispositivo, é feito o ***set_up*** – *fig. 26* do jogo e a inicialização de todas as variáveis de estado anteriormente mencionadas.

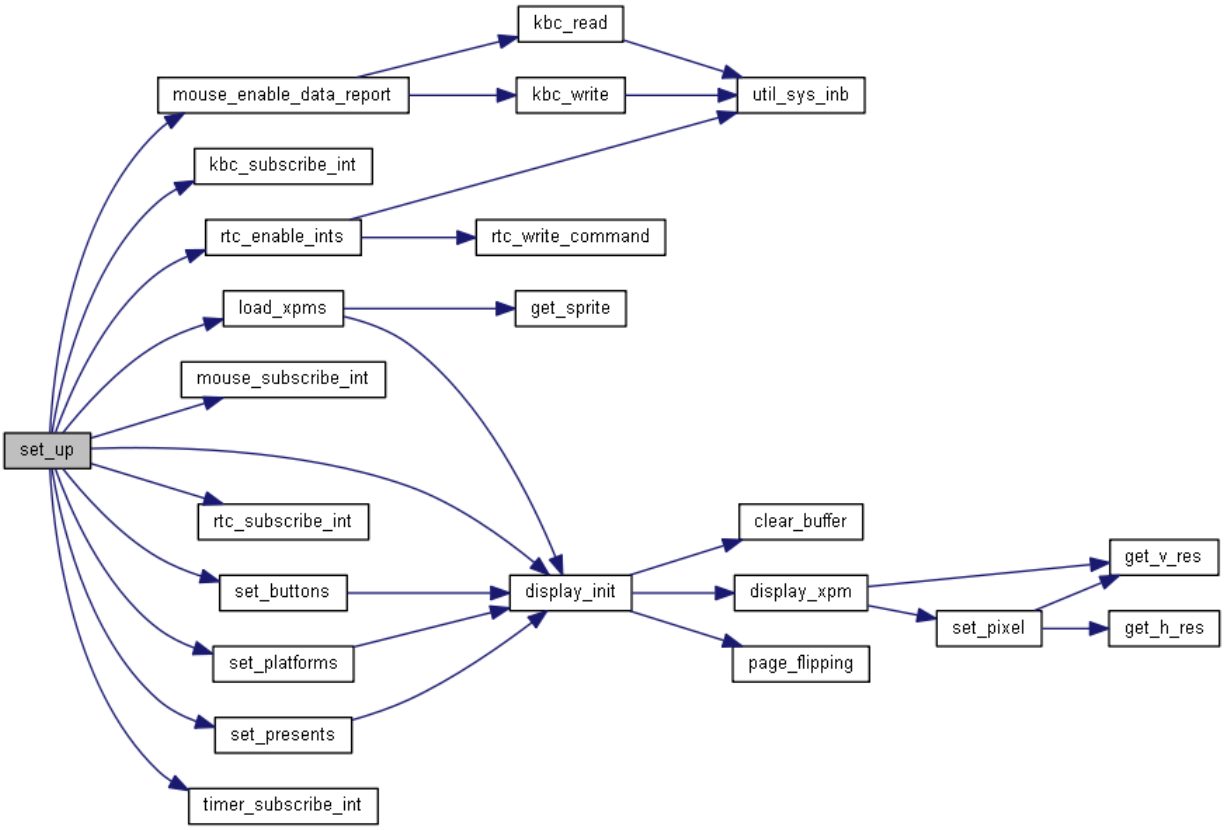

Figura 26. set_up call graph

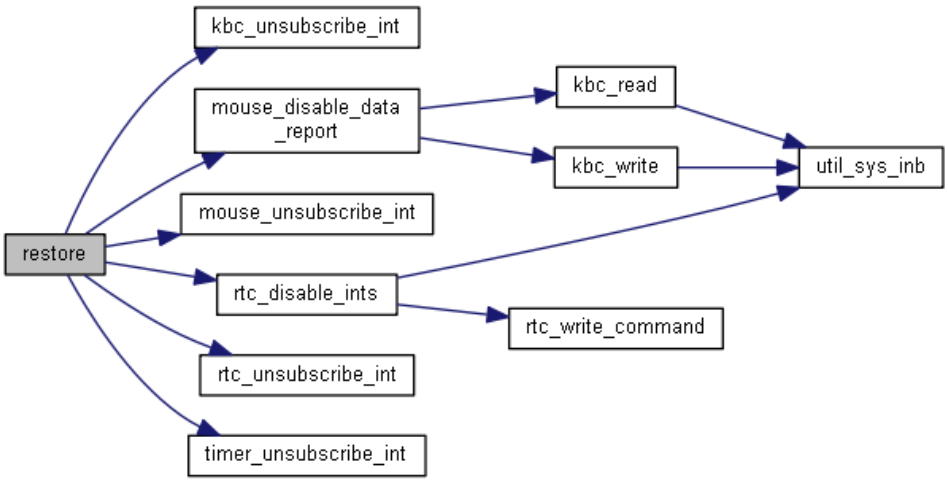

Figura 27. restore call graph

É imperativo referir, ainda, as funções ***init_game*** – *fig. 28*, ***playing*** – *fig. 29* e ***pause*** – *fig. 30*, que implementam os ciclos de *driver receive* e gerem todas as funcionalidades do jogo, ao invocarem as respetivas funções do módulo anterior.

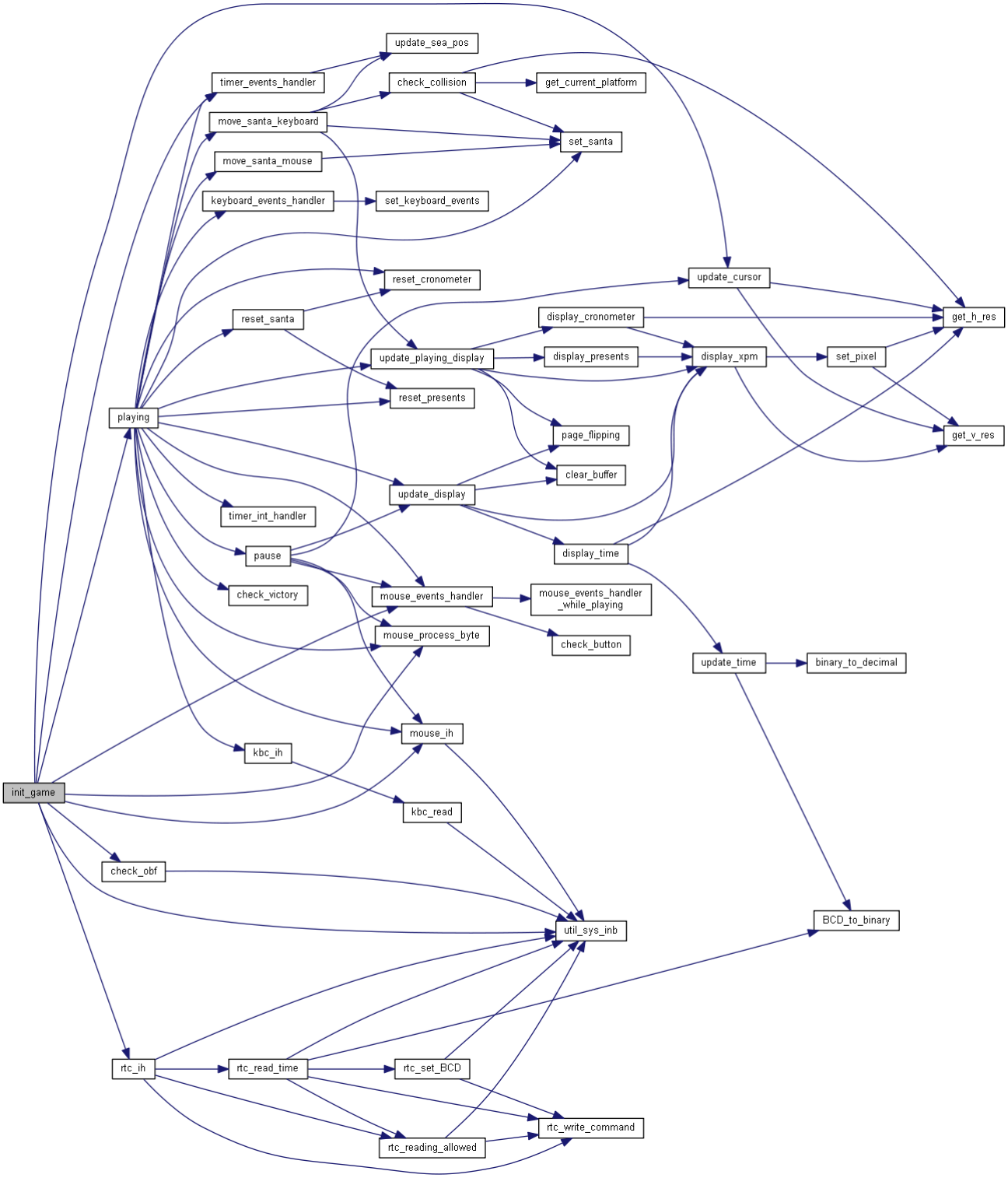

Figura 28. init_game call graph

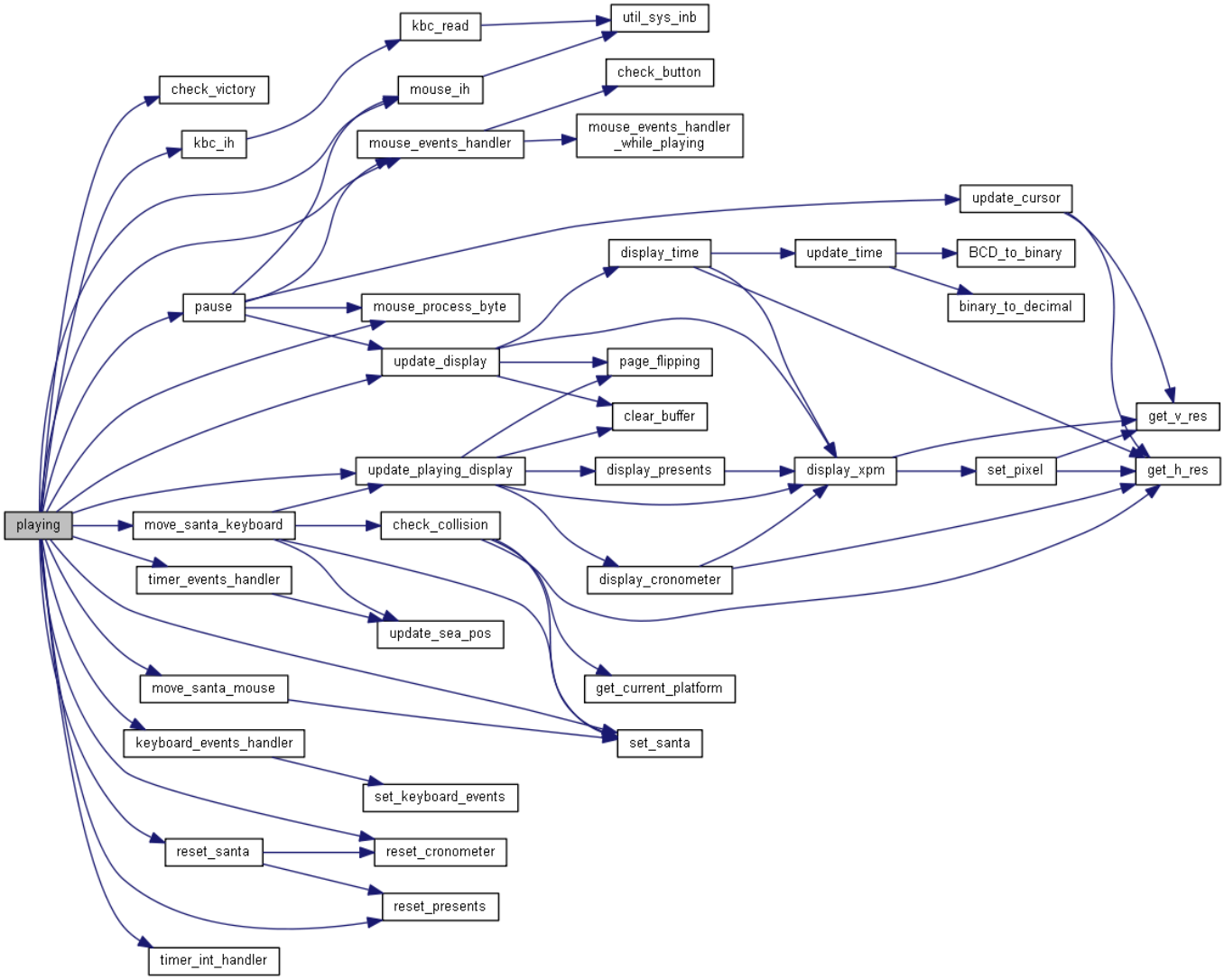

Figura 29. playing call graph

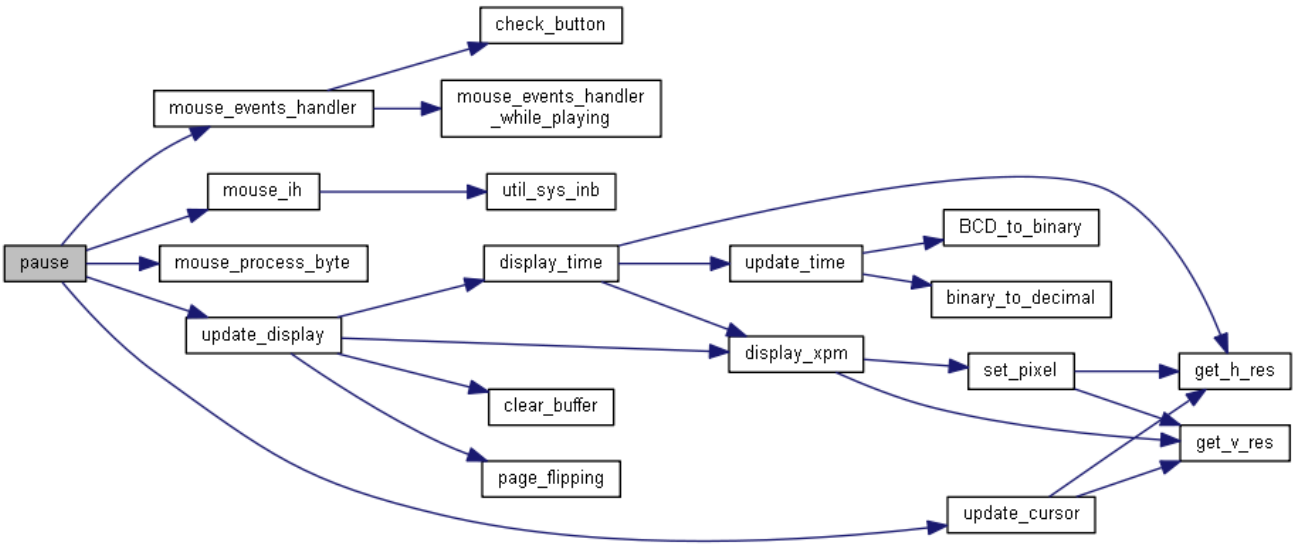

Figura 30. pause call graph

CONTRIBUIÇÃO: 50% Beatriz | 50% Margarida

Detalhes de implementação

Tudo começa com as funções ***set_display*** *e* ***set_up***. A primeira é responsável pela inicialização do modo gráfico, bem como do mapeamento da memória física para a memória virtual do processo atual. A segunda é responsável por permitir e subscrever as interrupções de todos os dispositivos, pela alocação de memória para as diferentes variáveis do programa e por invocar a função ***load_xpms***, esta que, por sua vez, trata de carregar os vários xpms e sprites para as variáveis correspondentes. A ***set_up*** invoca ainda as funções ***set_buttons***, ***set_platforms*** e ***set_presents***, responsáveis por inicializar os diferentes objetos com as coordenadas apropriadas. Achamos por bem definir o tipo *struct* ***backgrounds_***, que permitisse compilar os diferentes xpms dos backgrounds numa só estrutura e, assim, tornar o código mais simples e compacto. Para além desse, definimos ainda o tipo *struct* ***button*** para representar os diferentes botões através das suas coordenadas, largura e altura, o tipo *struct* ***platform***, para representar as diferentes plataformas através das suas coordenadas e largura e o tipo *struct* ***present***, caracterizado pelas suas coordenadas e por um parêmetro booleano, responsável por registar se o presente já fora ou não apanhado pelo jogador.

Antes de dar seguimento ao fio condutor do programa, gostaríamos de destacar a implementação de *state machines* que vieram a revelar-se cruciais para que conseguíssemos simplificar o código e torná-lo o mais intuitivo possível, principalmente no que à deteção de colisões diz respeito.

Começamos por definir a variável ***game_state***, do tipo *struct* ***game_states***, inicializada com o valor ***INITMENU*** e responsável por guardar o estado do jogo.

A função ***game_init*** inicializa o programa. O seu ciclo *driver receive* deteta as interrupções dos diferentes dispositivos, invocando os *handlers* correspondentes, que desencadearão uma resposta em função do valor atual da variável. A esta função cabe estabelecer a ligação entre o utilizador e os menus, pelo que são apenas consideradas aqui as interrupções do timer, do rtc e do mouse. O primeiro é responsável por gerir o que é mostrado no ecrã e o último atua ao nível da lógica do jogo, uma vez que é com base nas suas interrupções que a variável ***game_state*** é alterada.

No caso do utilizador carregar no botão que permite aceder às instruções, o ***game_state*** passa a registar o valor ***INSTRUCTIONS***. Aqui, gostaríamos, particulamente, de destacar o efeito de *swipe up/down* da navegação pelo texto das instruções, possível através da deteção de movimentos verticais com o rato, sem que haja necessidade de pressionar algum botão.

Se, por outro lado, for premido o botão para iniciar o jogo propriamente dito, a variável ***game_state*** passa a registar o valor ***CHOOSE_PLAYER*** e o utilizador é reencaminhado para um outro menu que lhe permite escolher qual o personagem com que pretende jogar. Por esta altura, entra em jogo uma outra variável do tipo *enum* ***player_states***, ***char_state***, inicializada por *default* com o valor ***COVID_SANTA***, que regista qual o personagem atualmente na "montra" e, portanto, qual o personagem potencialmente a ser escolhido. A *feature* de destaque deste menu é o efeito de *swipe left/right* da "montra" dos personagens.

Assim que o utilizador carrega no botão *Start* deste menu, a variável ***game_state*** muda para ***PLAYING*** e o programa é encaminhado para a função ***playing***. O jogo está oficialmente pronto a começar.

A função ***playing*** implementa um dos três ciclos de *driver receive* que, à semelhança do que foi descrito para a função ***game_init***, deteta as interrupções dos dispositivos e invoca os respetivos *handlers*. No entanto, ao contrário do que acontecia na última, na função ***playing*** entra também em jogo, passo a redundância, o keyboard.

Em primeiro lugar, há que referir o efeito de movimento dos presentes e do mar, que torna o jogo visualmente mais atrativo, bem como o facto do cronómetro, no canto superior direito do ecrã, só começar a contagem decrescente quando o sprite do personagem começa efetivamente a mover-se.

Dado que o jogador pode mover o personagem com recurso ao rato ou ao teclado, surgiu a necessidade de criar duas funções distintas – ***move_santa_mouse*** e ***move_santa_keyboard***, que funcionam aliadas à ***check_collisions***.

No entanto, é impossível falar desta última sem mencionar a variável ***santa_state***, do tipo *struct* ***santa_states***, inicializada com o valor ***STANDING_RIGHT*** e que guarda o estado do personagem, como, aliás, o próprio nome indica. O funcionamente da ***check_collisions*** é indissociável desta variável, uma vez que, não só o seu valor vai influenciar as ações a serem realizadas pela função, como a própria função está encarregue de alterar o seu valor sempre que

verificadas as condições para tal. Uma manifestação direta do uso de uma *state machine* para controlar o sprite do personagem diz respeito, por exemplo, ao facto de, ao saltar, o jogador não ter necessidade de carregar simultaneamente na seta para a esquerda ou direita. O próprio programa encarregar-se-á de detetar para qual dos lados está virado o sprite do personagem e efetuará o salto com incrementos das coordenadas horizontais no sentido respetivo.

A deteção de colisões é feita com recurso a uma outra função, ***get_current_platform*** que, por retornar o número, previamente definido, da plataforma onde se encontra o sprite do personagem, permite à ***check_collisions*** saber quais as coordenadas a verificar. Para além dessa, é invocada uma outra função, ***check_present*** que, em paralelo com a detação de colisões, trata de apurar se o jogador apanhou algum presente. Isto é possível mediante a comparação das coordenadas dos diferentes presentes com as coordenadas do sprite do personagem, ambas conhecidas. Há que mencionar a função ***set_santa*** que atualiza a variável ***current_santa*** com o valor do parâmetro recebido, o que permite alternar facilmente o sprite a ser mostrado no ecrã e, por esse motivo, contribui também para tornar o jogo visualmente mais atrativo e realista.

A par de tudo isto, não nos podemos esquecer do timer, que desempenha um papel essencial, uma vez que é responsável pela cronometragem do tempo do jogo.

Quando o programa deteta que a tecla ESC foi pressionada, entra na função ***pause***, que surgiu face à necessidade de regressar ao menu sem que, no entanto, fossem alteradas as variáveis do jogo, nomeadamente o cronómetro e a posição do sprite da personagem.

Gostaríamos de salientar, em relação às funções ***game_init***, ***playing*** e ***pause***, que optamos por implementar três ciclos *driver receive*, um em cada uma, apenas por uma questão de legibilidade e estruturação do código, uma vez que esta divisão foi aquela que, intuitivamente, se nos afigurou fazer mais sentido.

Antes do programa terminar, é invocada a função ***restore***, que procede à invocação das funções ***disable*** e ***unsubscribe*** de todos os dispositivos, bem como à libertação da memória dinâmica previamente alocada.

Conclusões

Ao longo do semestre, tanto durante as aulas laboratoriais, como autonomamente em casa, sempre nos esforçamos por desenvolver o nosso código de forma inteligente, quer através da modulação estratégica do mesmo, numa abordagem *bottom to top*, que tornasse o código reutilizável para o projeto, quer através da realização de comentários e definição de macros, que permitissem um rápido entendimento do que estava a ser implementado.

Não obstante as dificuldades que tivemos que enfrentar, principalmente por se tratar de um tipo de programação *low level* ao qual não estávamos acostumadas, sempre nos esforçamos por ultrapassar essas dificuldades, procurando superá-las e superar-nos a nós mesmas.

No entanto, gostaríamos de mencionar alguns pontos em relação aos quais, na nossa perspetiva, a unidade curricular poderia melhorar.

Cremos que a UC peca, desde logo, pela falta de apoio inicial que fornece aos alunos, uma vez que se espera destes uma adapatação demasiado rápida ao tipo de conteúdos lecionados. A solução passaria, por exemplo, por disponibilizar um maior número de aulas para ambientar os discentes, quer ao uso de máquinas virtuais quer, se possível, à utilização da ferramenta git. Tais ensinamentos revelar-se-iam certamente muito uteis, não só no âmbito da UC como, e sobretudo, na vida enquanto fututos engenheiros informáticos.

Para além disso, acreditamos que a estratégia de avaliação contínua através dos labs teria resultado melhor, uma vez que teria permitido aos alunos confirmarem a sua implementação do código e, assim, apurar a sua aptidão para ser reutilizado no projeto. Uma outra alternativa seria a anterior, aliada à avaliação através de testes de programação, que reconhecemos ser a melhor estratégia para distinguir os alunos que desenvolveram efetivamente métodos de estudo e trabalho, daqueles que não.

Por último, ainda em relação aos aspetos negativos, não poderíamos deixar de expressar o nosso desagrado relativamente à demora na entrega das notas dos testes de programação. Consideramos que tal não é correto para com os alunos que, constantemente, se esforçam por cumprir os prazos que lhes são impostos e, nesse sentido, mereciam ter conhecimento do resultado do seu trabalho.

Focando agora nos aspetos positivos, gostaríamos de agradecer ao docente das aulas práticas, Mário Cordeiro, que sempre se mostrou recetivo a ajudar e a responder a todas as dúvidas que lhe eram colocadas. Para além disso, consideramos que a presença de um monitor nas aulas laboratoriais é uma ótima ideia uma vez que, por se tratar de uma figura mais perto da nossa faixa etária, a comunicação torna-se, na maioria das vezes, mais fácil.

Independentemente de qualquer aspeto menos positivo, é com sinceridade que afirmamos que gostamos da UC e daquilo que ela nos acrescentou e nos permitiu aprender. Consideramos, aliás, que foi a cadeira mais desafiante, naquilo que a palavra tem de positivo, até agora e, em consequência, irrefutavelmente enriquecedora.

Assim sendo, não nos resta concluir mais nada a não ser que este projeto revelou ser, efetivamente, a melhor maneira de "encerrar" o processo de aprendizagem.